Predicción del rendimiento: Análisis de la escalabilidad de aplicaciones paralelas

Memoria presentada por Javier Panadero para optar al grado de Doctor por la Universitat Autònoma de Barcelona, bajo la dirección de los doctores Dr. Emilio Luque y Dr. Álvaro Wong.

Barcelona, 2015

Predicción del rendimiento: Análisis de la escalabilidad de aplicaciones paralelas

Tesis doctoral presentada por Javier Panadero para optar al grado de doctor por la Universitat Autònoma de Barcelona. Trabajo realizado en el Departamento de Arquitectura de Computadores y Sistemas Operativos de la Escuela de Ingeniería, bajo el programa de doctorado en Computación de Altas Prestaciones.

Barcelona, 2015

Director

Dr. Emilio Luque

Co-Director

Dr. Àlvaro Wong

Resumen

Ejecutar aplicaciones paralelas de paso de mensajes sobre un elevado número de recursos de forma eficiente no es una tarea trivial. Debido a la compleja interacción entre la aplicación paralela y el sistema HPC, a medida que se aumenta el número de procesos de la aplicación, dependiendo del sistema, puede llegar un punto donde la aplicación presente su límite de escalabilidad, momento a partir del cual se empezarán a producir ineficiencias en el sistema, conllevando a un uso ineficiente del mismo. Este problema se vuelve especialmente crítico en aplicaciones paralelas las cuales tienen que ser ejecutadas frecuentemente, utilizando un elevado número de recursos sobre un largo período de tiempo

Con el fin de intentar solventar estos problemas y hacer un uso eficiente del sistema, como principal contribución de esta tesis, se propone la metodología *P3S (Prediction of Parallel Program Scalability)*, la cual permite predecir el rendimiento de la escalabilidad fuerte de aplicaciones paralelas de paso de mensajes en un determinado sistema, utilizando un tiempo de análisis limitado y un conjunto reducido de recursos del sistema.

La metodología P3S, se basa en analizar el comportamiento repetitivo de las aplicaciones paralelas de paso de mensajes. Este tipo de aplicaciones están compuestas por un conjunto de fases identificables, las cuales se van repitiendo a lo largo de toda la aplicación, independientemente del número de procesos de la aplicación. La metodología consta de tres etapas. Una primera etapa de caracterización, donde se caracterizan las fases relevantes de la aplicación paralela, a partir de la información obtenida de la ejecución de un conjunto de firmas de la aplicación, para un número reducido de procesos. Una segunda etapa de modelización de la aplicación, donde se genera el modelo lógico de escalabilidad de cada fase relevante de la aplicación paralela, mediante el cual construir la traza lógica escalable de la aplicación. Esta traza permitirá predecir el comportamiento lógico de la aplicación, a medida que se aumenta su número de procesos. Finalmente, la última etapa de la metodología consiste en predecir el rendimiento de la aplicación para un número específico de procesos. Para ello, la traza lógica escalable será precisada para el número de procesos para el cual se desea predecir el rendimiento de la aplicación, con el objetivo de predecir el tiempo de cómputo y comunicación de cada fase relevante de la aplicación, para ese número de procesos, con la finalidad de obtener el tiempo predicho de la aplicación.

Gracias a la curva de escalabilidad de la aplicación, proporcionada por la metodología P3S, los usuarios pueden seleccionar los recursos más adecuados para ejecutar su aplicación en el sistema objetivo, lo que permite tener la seguridad de utilizar los recursos del sistema de forma eficiente.

Palabras Clave: Predicción de escalabilidad, Aplicaciones paralelas de paso de mensajes, Escalabilidad fuerte, Sistemas HPC

Resum

Executar aplicacions paral·leles de pas de missatges sobre un elevat nombre de recursos de manera eficient no és una tasca trivial. A causa de la complexa interacció entre l'aplicació paral·lela i el sistema HPC, a mesura que s'augmenta el nombre de processos de l'aplicació, depenent del sistema, pot arribar un punt on l'aplicació presenti el seu límit d'escalabilitat, moment a partir del qual es començaran a produir ineficiències en el sistema, comportant a un ús ineficient del mateix. Aquest problema es torna especialment crític en aplicacions paral·leles les quals han de ser executades freqüentment, utilitzant un elevat nombre de recursos sobre un llarg període de temps

Per tal d'intentar solucionar aquests problemes i fer un ús eficient del sistema, com a principal contribució d'aquesta tesi, es proposa la metodologia *P3S (Prediction of Parallel Program Scalability)*, la qual permet predir el rendiment de l'escalabilitat forta d'aplicacions paral·leles de pas de missatges en un determinat sistema, utilitzant un temps d'anàlisi limitat i un conjunt reduït de recursos del sistema.

La metodologia P3S, es basa en analitzar el comportament repetitiu de les aplicacions paral·leles de pas de missatges. Aquest tipus d'aplicacions estan compostes per un conjunt de fases identificables, les quals es van repetint al llarg de tota l'aplicació, independentment del nombre de processos de l'aplicació. La metodologia consta de tres etapes. Una primera etapa de caracterització, on es caracteritzen les fases rellevants de l'aplicació paral·lela, a partir de la informació obtinguda de l'execució d'un conjunt de signatures de l'aplicació, per a un nombre reduït de processos. Una segona etapa de modelització de l'aplicació, on es genera el model lògic d'escalabilitat de cada fase rellevant de l'aplicació paral·lela, mitjançant el qual construir la traça lògica escalable de l'aplicació. Aquesta traça permetrà predir el comportament lògic de l'aplicació, a mesura que s'augmenta el seu nombre de processos. Finalment, l'última etapa de la metodologia consisteix en predir el rendiment de l'aplicació per a un nombre específic de processos. Per a això, la traça lògica escalable serà precisada per al nombre de processos per al qual es vol predir el rendiment de l'aplicació, amb l'objectiu de predir el temps de còmput i comunicació de cada fase rellevant de l'aplicació, per aquest nombre de processos , amb la finalitat d'obtenir el temps predit de l'aplicació.

Gràcies a la corba d'escalabilitat de l'aplicació, proporcionada per la metodologia P3S, els usuaris poden seleccionar els recursos més adients per executar la seva aplicació al sistema destí, el que permet tenir la seguretat d'utilitzar els recursos del sistema de manera eficient.

Paraules Clau: Predicció d'escalabilitat, Aplicacions paral·leles de pas de missatges, Escalabilitat forta, Sistemes HPC

Abstract

Executing message-passing applications using an elevated number of resources is not a trivial task. Due to the complex interaction between the message-passing applications and the HPC system, depending on the system, many applications may suffer performance inefficiencies, when they scale to a large number of processes. This problem is particularly serious when the application is executed many times over a long period of time.

With the purpose of avoiding these problems and making an efficient use of the system, as main contribution of this thesis, we propose the methodology *P3S (Prediction of Parallel Program Scalability)*, which allows us to analyze and predict the strong scalability behavior for message-passing applications on a given system. The methodology strives to use a bounded analysis time, and a reduced set of resources to predict the application performance.

The P3S methodology is based on analyzing the repetitive behavior of parallel message-passing applications. Such applications are composed of a set of phases, which are repeated through the whole application, independently of the number of application processes. The methodology is made up of three stages. A first characterization step, where the relevant phases of the parallel application are characterized, from the execution of a set of small-scale application signatures. A second stage of modeling of the application, where the application logical scalability model is generated for each relevant application phase, whereby to construct the scalable logical trace of the application. This trace will be used to predict the logical behavior of the application, as the number of the application processes increases. Finally, the last stage of the methodology consist of predicting the application performance for a specific number of processes. In order to do that, the scalable logical trace will be specified for the number of processes to predict the application performance, with the objective to predict the computational and communication time of each relevant phase for this number of processes, in order to obtain the performance prediction.

The output of the P3S methodology will be the predicted curve of application speedup. Using this information, the users can select the most appropriate resources to execute his application on the target system, in order to use the system resources efficiently.

Keywords: Scalability prediction, Message-passing applications, Strong scalability, HPC systems

Agradecimientos

Primeramente, me gustaría mostrar mi más sincera gratitud al director de mi tesis, el Dr. Emilio Luque, por el tiempo que me dedicara durante estos tres años, ya que sin sus acertados consejos y ayuda, no hubiera sido posible finalizar este trabajo de investigación. Su constante dedicación y perseverancia en el mundo de la investigación, son cualidades dignas de mencionar y hacen de él un modelo a seguir.

Al co-director de mi tesis, el Dr. Álvaro Wong, por su supervisión y ayuda en el día a día, orientándome durante todo el desarrollo de este trabajo y dedicándome gran parte de su tiempo. También me gustaría mencionar las muestras de amistad que me ofreciera desde el primer día.

A la Dra. Dolores Rexachs por recibirme en este mundo científico, dirigiendo mi primera tesina de máster. Gracias a su estímulo constante aprendí a no rendirme jamás.

A mis compañeros de viaje durante estos tres años: Josefina, Sandra, Arindam, Marcela, Claudio, Francisco, Joe, Cecilia, Liu, Alejandro, Albert, Ferran, Pilar, Cesar, Tomás, Jorge y Laura.

A Daniel Ruiz y Remo Suppi, por su ayuda técnica durante estos tres años, facilitándome todo el soporte técnico que necesité para la realización de esta tesis.

Al Centro de Supercomputación de Galicia, por ofrecerme la posibilidad de realizar una estancia de investigación en su centro, y en especial al grupo de aplicaciones por ayudarme en todo lo que necesite durante mi estancia.

Al proyecto TIN2011-24384 otorgado por Ministerio de Ciencia e Innovación, el cual financió este trabajo de investigación.

A todos mis amigos y conocidos que confiaron en mí, en especial a Jesús López, quién me enseñó que uno tiene que hacer lo que realmente le gusta, y a Ferrán Mateo, Antonio Iglesias, Miguel Molina, Ignasi Mares y Víctor Gaspar, por sus muestras de amistad.

Y por último, y no menos importante, a mis padres por su paciencia durante estos tres años.

Índice general

Índice de figuras xv

Índice de tablas xxi

1. Introducción 1

1.1. Motivación . . . 1
1.2. Objetivos . . . 4
1.3. Contribuciones . . . 7
1.4. Organización de la tesis . . . 8

2. Sistemas paralelos 11

2.1. Introducción . . . 11
2.2. Modelos arquitecturales . . . 11
2.3. Arquitecturas paralelas . . . 13
2.3.1. Multiprocesadores . . . 13
2.3.2. Multicomputadores . . . 14
2.3.2.1. MPP (Massively parallel processing) . . . 15
2.3.2.2. COW(Clusters of workstation) . . . 15
2.4. Modelos de programación paralela . . . 15
2.4.1. Modelo de memoria compartida . . . 16
2.4.2. Modelo de paso de mensajes . . . 16
2.5. Paradigmas de programación paralela . . . 17
2.5.1. Master/Worker . . . 17
2.5.2. SPMD (Single Program Multiple Data) . . . 18
2.5.3. Divide and Conquer . . . 18
2.5.4. Data Pipeline . . . 19

2.6. Algoritmos de comunicación 19
2.6.1. Perfect Shufle 19
2.6.2. Butterfly 20
2.6.3. Patrón Lineal 21
2.6.4. Patrón en Anillo 21
2.6.5. Patrón en Malla 21
2.7. Métricas de rendimiento 21
2.7.1. Tiempo de ejecución 22
2.7.2. Speedup 22
2.7.3. Eficiencia 22
2.7.4. Ley de Amdahl 23
2.7.5. Ley de Gustafson 23
2.7.6. Escalabilidad 24
2.7.6.1. Escalabilidad Fuerte 24
2.7.6.2. Escalabilidad Débil 24
2.7.7. Isoeficiencia 25

3. Estado del Arte **27**
3.1. Introducción 27
3.2. Métodos analíticos 28
3.3. Métodos computacionales 30
3.4. Métodos experimentales 33

4. Metodología P3S **39**
4.1. Introducción 39
4.2. Caracterización de la aplicación 42
4.3. Modelizado lógico de la aplicación 42
4.4. Predicción del rendimiento de la aplicación 45

5. Modelo lógico de aplicaciones paralelas para el análisis y predicción de la escalabilidad **51**
5.1. Introducción 51
5.2. Caracterización de la aplicación 51
5.3. Modelo Lógico de la aplicación 54

5.3.1. Modelización del patrón de comunicación 58
5.3.1.1. Generación de las ecuaciones locales 60
5.3.1.2. Modelización de las ecuaciones globales 64
5.3.2. Modelización del peso de la fase 70
5.3.3. Modelización del patrón de cómputo 71
5.4. Validación experimental 78
5.4.1. Modelo lógico para la aplicación BT 80
5.4.2. Modelo lógico para la aplicación CG 83
5.4.3. Modelo lógico para la aplicación Sweep3D 86
5.4.4. Modelo lógico para la aplicación N-Body 87

6. Modelo de predicción del tiempo de cómputo 91
6.1. Introducción 91
6.2. Metodología para predecir el tiempo de cómputo 91
6.3. Validación experimental 97

7. Modelo de predicción del tiempo de comunicación 105
7.1. Introducción 105
7.2. Selección de parámetros de ejecución y carga de la traza física escalada . 109
7.3. Selección del número de procesos Black Hole 115
7.4. Procesado de la traza física 117

8. Predicción del rendimiento de la aplicación 127
8.1. Introducción 127
8.2. Predicción del tiempo de ejecución de la aplicación 127
8.3. Predicción de la escalabilidad de la aplicación 129

9. Resultados experimentales 131
9.1. Introducción 131
9.2. Predicción del rendimiento de la aplicación 133
9.2.1. Predicción del rendimiento para la aplicación N-Body 134
9.2.2. Predicción del rendimiento para la aplicación Sweep3D 139
9.2.3. Predicción del rendimiento para la aplicación SP 140
9.2.4. Predicción del rendimiento para la aplicación BT 142
9.2.5. Predicción del rendimiento para la aplicación CG 144

9.3. Predicción de la curva de escalabilidad de la aplicación 146
9.3.1. Predicción de la escalabilidad para la aplicación N-Body 147
9.3.2. Predicción de la escalabilidad para la aplicación Sweep3D 149
9.3.3. Predicción de la escalabilidad para la aplicación SP 149
9.3.4. Predicción de la escalabilidad para la aplicación para CG 151
9.3.5. Predicción de la escalabilidad para la aplicación para BT 153

10. Conclusiones **157**
10.1. Introducción 157
10.2. Conclusiones finales 157
10.3. Trabajo futuro y líneas abiertas 159
10.4. Lista de publicaciones 161

Anexos **165**

A. Herramienta PAS2P **165**
A.1. Introducción 165
A.2. Descripción de la herramienta PAS2P 166
A.2.1. Generación de la firma 166
A.2.2. Ejecución de la firma 169
A.3. Integración del módulo de instrumentación con la librería de contadores hardware PAPI 170
A.4. Paralelización del módulo de Análisis 171
A.4.1. Carga de datos 172
A.4.2. Modelo abstracto de la aplicación 173
A.4.3. Identificación de patrones 175
A.4.4. Evaluación Experimental del módulo paralelo 178
A.5. Identificación de patrones de la aplicación utilizando el número de instrucciones 183

B. Validación experimental de la modelización lógica de la aplicación **189**
B.1. Introducción 189
B.2. Modelo lógico de la aplicación BT 190
B.3. Modelo lógico de la aplicación CG 193
B.4. Modelo lógico de la aplicación Sweep3D 196

B.5. Modelo lógico de la aplicación SP 197
B.6. Modelo lógico de la aplicación LU 200

Bibliografía **201**

Lista de acrónimos **209**

Lista de abreviaturas de variables utilizadas **211**

Índice de figuras

1.1. Repetitividad de las aplicaciones paralelas de paso de mensajes 4
1.2. Descripción general de la metodología 6

2.1. Taxonomía de Flynn 12
2.2. Arquitectura Multiprocesador 13
2.3. Arquitectura Multicomputador 14
2.4. Paradigma Master/Worker 18
2.5. Patrón de comunicación *Perfect Shufle* 20
2.6. Patrón de comunicación Butterfly 20
2.7. Escalabilidad fuerte 25
2.8. Escalabilidad débil 26

3.1. Tabla resumen con las principales características de los métodos de predicción 28

4.1. Visión global de la metodología P3S 40
4.2. Fragmento de la aplicación científica de paso de mensajes NPB CG para 256 procesos, donde se muestra visualmente la repetitividad de las fases de la aplicación 41
4.3. Generación de la firma de la aplicación a partir de las fases relevantes de la aplicación 42
4.4. Relación de fases por similitud funcional 44
4.5. Traza Lógica Escalada (LT4NC) para la fase 1 del proceso 0 45
4.6. Traza física Escalada (ST4NP) para la fase 1 del proceso 0 46
4.7. Error introducido en los modelos de regresión a medida que nos alejamos de los puntos reales 47

4.8. Procesado de la traza física escalada (ST4NP) para N procesos en el sistema paralelo 49

5.1. Caracterización de las fases de la aplicación 52
5.2. Ejemplo de traza de la firma para el proceso 1 53
5.3. Organización de la estructura de datos utilizada para analizar las trazas físicas generadas por las diferentes firmas de la aplicación. 54
5.4. Modelo Lógico de escalabilidad de la aplicación 55
5.5. Relación de fases por similitud funcional 57
5.6. Evolución de la frecuencia de repetición de las fases de la firma 58
5.7. Relación de las fases de la firma de la aplicación para distinto número de procesos 59
5.8. Modelización de la Ecuación General de comunicación (GE_{F_i}) 60
5.9. Etapas del algoritmo de generación de la Ecuación General de comunicación (GE_{F_i}) 61
5.10. Obtención de la ecuación local de comunicación para la primera comunicación de la fase 62
5.11. Ejemplo de aplicación paralela de paso de mensajes con paradigma SPMD 63
5.12. Generación de comunicaciones nulas 64
5.13. Generación de la Ecuación General a partir de sus ecuaciones locales . . 65
5.14. Modelización de los parámetros de las Ecuaciones Generales 66
5.15. Obtención del número total de comunicaciones de la fase 67
5.16. Patrón de la primera comunicación de la fase 1 de la aplicación BT . . . 68
5.17. Patrón de comunicación predicho a partir de la Ecuación General para 49 procesos 69
5.18. Modelo de regresión para la fase 1 del NPB BT 71
5.19. Linealización del modelo de regresión 72
5.20. Distribución del número de instrucciones de cada fase de la aplicación, a medida que se aumenta su número de procesos 73
5.21. Distribución del número de instrucciones 74
5.22. Distribución del número de instrucciones de cada proceso 75
5.23. Traza Lógica Escalada (LT4NC) para la fase 1 del proceso 0 78

5.24. Tiempos de ejecución de las firmas de la aplicación versus tiempos de ejecución de la aplicación para BT 81
5.25. Patrón de comunicación predicho para la aplicación BT para 1,024 procesos 82
5.26. Tiempos de ejecución de las firmas de la aplicación versus tiempos de ejecución de la aplicación para CG 84
5.27. Patrón de comunicación predicho para la aplicación CG para 4,096 procesos 85
5.28. Tiempos de ejecución de las firmas de la aplicación versus tiempos de ejecución de la aplicación para Sweep3D 87
5.29. Tiempos de ejecución de las firmas de la aplicación versus tiempos de ejecución de la aplicación para N-Body 88
5.30. Patrón de comunicación predicho para la aplicación NBody para 4,096 procesos 88

6.1. Traza física Escalada (ST4NP) para la fase 1 del proceso 0 92
6.2. Modelo de regresión de Cómputo (CRM) 93
6.3. Distribución del número de instrucciones de cada proceso 94
6.4. Diagrama de flujo para predecir el tiempo de computo para cada fase de la aplicación 95

7.1. Estructura de la herramienta Synthetic Signature (SS) 106
7.2. Proceso medible envía un mensaje de control al proceso BH solicitándole un mensaje 109
7.3. Activación de las comunicación ruido por parte de los procesos BHN . . 109
7.4. Diagrama general de flujo del método de ejecución de la herramienta SS 110
7.5. Mapping real de la aplicación en el sistema 111
7.6. Mapping del rango de procesos en el sistema 112
7.7. Modificación del destino de los eventos de comunicación de la traza física escalada 113
7.8. Tabla de correspondencia entre el proceso real y el proceso lógico 114
7.9. Modificación del destino real del evento de comunicación por su destino lógico 115

7.10. Tabla de correspondencia entre el proceso medible y su proceso Black Hole (BH) 116
7.11. Modificación del destino del mensaje por un proceso BH 117
7.12. Asignación de procesos BH a las comunicaciones externas 118
7.13. Digrama de flujo del procedimiento de ejecución de una iteración de la herramienta SS para procesar las trazas ST4NP 119
7.14. Medición del tiempo de comunicación de los eventos MPI 120
7.15. Envío de comunicaciones externas al proceso Back Hole (BH) 121
7.16. Mensaje de control informando de una recepción externa 122
7.17. Ejemplo de ejecución de la herramienta SS - Iteración 1 123
7.18. Ejemplo de ejecución de la herramienta SS - Iteración 2 124

8.1. Visión global de la metodología P3S 128
8.2. Curva de rendimiento de la aplicación 129

9.1. Topología de la red de interconexión del cluster CAPITA 132
9.2. Speedup de la aplicación N-Body para 100,000 partículas 148
9.3. Speedup de la aplicación Sweep3D para el workload input.80 con 11 iteraciones 150
9.4. Speedup de la aplicación SP Clases C 151
9.5. Speedup de la aplicación CG Clases C 153
9.6. Speedup de la aplicación BT Clases C 154

A.1. Etapas de la metodología PAS2P 167
A.2. Interposición Librería PAS2PLib 167
A.3. Generación de la firma de la aplicación 168
A.4. Ejemplo de traza de la firma para el proceso 1 169
A.5. Modulo de instrumentación integrado con librería PAPI 170
A.6. Ejemplo de traza de la firma con información de bajo nivel para el proceso 0171
A.7. Visión general de la paralización del módulo analizado 172
A.8. Carga del Log de traza 172
A.9. Generación del ID Relación 173
A.10.Insertar tiempos lógicos utilizando el algoritmo de ordenación paralelo . 174
A.11.Traza lógica de la aplicación utilizando el módulo paralelo 176

A.12.Paralización del algoritmo de identificación de fases 176
A.13.Paso1: Los procesos másters del nodo reciben los datos de los procesos workers del nodo 178
A.14.Paso2: Procesos Masters reciben datos de los procesos master del grupo de nodos 178
A.15.Análisis de la traza usando Sweep.250.30 181
A.16.Análisis de la traza usando LU Clase C 182
A.17.Análisis de la traza usando SP Clase C 184
A.18.Análisis de la traza usando CG Clase C 185
A.19.Fases de la aplicación delimitadas entre dos eventos de comunicación . . 186
A.20.Fases obtenidas mediante PAS2P 186
A.21.Instrucciones totales de la fase 187

ÍNDICE DE FIGURAS

Índice de tablas

5.1. Ecuaciones locales para la primera comunicación de la fase 1 de la aplicación BT 69
5.2. Tabla de predicción de instrucciones (Scalable Computational Instruction Count (SCIC)) 77
5.3. Características del Cluster CAPITA 79
5.4. Modelo lógico de las fases de las aplicaciones paralelas (SLT) 83
5.5. Traza lógica escalada (LT4NC), obtenida aplicando las ecuaciones generales de la traza lógica escalable (STL) de tabla 5.4 para un número específico de procesos 86
5.6. Resumen del error de predicción para el resto de fases y aplicaciones analizadas 90

6.1. Características del Cluster CAPITA 98
6.2. Incremento del peso para la fase 1 de BT 99
6.3. Tabla de predicción de instrucciones (SCIC) para la fase 1 de BT con el workload D 101
6.4. Ejecuciones de la firma para BT usando la clase B 101
6.5. Error de predicción para las fases de la aplicación BT 102
6.6. Comparación de los modelos de regresión de cómputo (CRM) 103
6.7. Error de predicción para las fases de las aplicaciones utilizadas 104

9.1. Características del Cluster CAPITA 132
9.2. Modelo de la fase 1 de la aplicación N-Body (Ecuaciones utilizadas para generar la traza lógica escalable (SLT) para n procesos) 134

9.3. Tabla de predicción de instrucciones (SCIC) de la fase 1 de la aplicación N-Body 135
9.4. Paramétros de la fase predichos (Traza lógica Escalada (LT4NC)) aplicando las ecuaciones de la tabla 9.2 para 512 procesos 135
9.5. Error de predicción del tiempo de cómputo para las fase 1 de la aplicación N-Body para 512 procesos 136
9.6. Traza Física Escalada (ST4NC) para 512 procesos 137
9.7. Predicción de la aplicación N-Body para 512 procesos usando la metodología P3S y la firma de PAS2P. 137
9.8. Recursos utilizados por la metodología P3S y por la firma de PAS2P . . 138
9.9. Tiempos de predicción utilizando la herramienta *SS* y la firma PAS2P . 138
9.10. Predicción de la aplicación Sweep3D para 512 procesos usando la metodología P3S y la firma de PAS2P. 140
9.11. Predicción de la aplicación SP para 484 procesos usando la metodología P3S y la firma de PAS2P. 142
9.12. Predicción de la aplicación BT para 484 procesos usando la metodología P3S y la firma de PAS2P. 144
9.13. Predicción de la aplicación CG para 512 procesos usando la metodología P3S y la firma de PAS2P. 146
9.14. Predicción del tiempo de ejecución de la aplicación N-Body con 100,000 partículas de 16 a 512 procesos 147
9.15. Predicción del tiempo de ejecución de la aplicación Sweep3D con input.80 11 iteraciones de 16 a 512 procesos 148
9.16. Predicción del tiempo de ejecución de la aplicación SP Clase C de 16 a 484 procesos 150
9.17. Predicción del tiempo de ejecución de la aplicación CG Clase C de 16 a 512 procesos 152
9.18. Predicción del tiempo de ejecución de la aplicación BT Clase C de 16 a 484 procesos 153

A.1. Características de los clústers utilizados 179

B.1. Modelo de la aplicación para BT (SLT) 191

B.2. Parámetros predichos de las fases de la aplicación BT (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento para n = 1024 192
B.3. Modelo de la aplicación para CG (SLT) 194
B.4. Parámetros predichos de las fases de la aplicación CG (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento n = 4096. 195
B.5. Modelo de la aplicación para Sweep3D (SLT) 196
B.6. Parámetros predichos de las fases de la aplicación Sweep3D (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento para n = 4096. 196
B.7. Modelo de la aplicación para SP (SLT) 198
B.8. Parámetros predichos de las fases de la aplicación SP (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento para n = 1024. 199
B.9. Modelo de la aplicación para LU (SLT) 200
B.10. Parámetros predichos de las fases de la aplicación LU (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento para n = 4096 procesos. 200

ÍNDICE DE TABLAS

1

Introducción

1.1. Motivación

Durante los últimos años, los sistemas de Cómputo de Altas Prestaciones *(High Performance Computing (HPC))* han incrementado el número de unidades de procesamiento (cores) significativamente [1]. Actualmente, estos sistemas poseen una gran potencia de cómputo y todo parece indicar que esta tendencia continuará aumentando durante los próximos años, debido a las constantes mejoras tecnológicas, lo que hace posible el incremento tanto del número de cores por procesador, como del número total de procesadores en estos sistemas.

Los usuarios de estos sistemas, desean obtener el máximo partido a toda esta potencia de cómputo a la hora de ejecutar sus aplicaciones, ya sea disminuyendo el tiempo de ejecución de sus aplicaciones (escalabilidad fuerte), o ejecutando sus aplicaciones con una carga mayor de trabajo (escalabilidad débil) [2].

Ejecutar aplicaciones paralelas de paso de mensajes sobre un elevado número de recursos de forma eficiente no es una tarea trivial. Debido a la compleja interacción entre la aplicación paralela y el sistema HPC, a medida que se aumenta el número de procesos de la aplicación y el número de cores utilizados, dependiendo del sistema, puede llegar un punto donde la aplicación presente su límite de escalabilidad, momento a partir del cual se empezarán a producir ineficiencias en el sistema, conllevando a un uso ineficiente del mismo. Este problema se vuelve especialmente crítico en aplicaciones paralelas las cuales tienen que ser ejecutadas diariamente, utilizando un elevado número de recursos sobre un largo período de tiempo, como podrían ser los simuladores de

predicción meteorológica [3] o los simuladores de predicción de mareas oceánicas [4].

Con el fin de hacer un uso eficiente del sistema utilizando un elevado número de recursos, un aspecto a tener en cuenta antes de ejecutar la aplicación paralela, es conocer su rendimiento en el sistema en el cual será ejecutada a medida que cambia su carga de trabajo *(Workload)* o los recursos disponibles. Es importante conocer esta información, ya que el número ideal de procesos para ejecutar una aplicación paralela puede variar de un sistema a otro, debido a diferencias hardware, como podrían ser: la red de interconexión del sistema, la cantidad de memoria principal de los nodos de cómputo, o la arquitectura de los procesadores. Además, utilizar un elevado número de recursos no siempre implica un mayor rendimiento [5]. La velocidad de ejecución de un algoritmo, que se define como el trabajo dividido por el tiempo de respuesta, teóricamente debería incrementarse de forma lineal con el tamaño del sistema, obteniendo un *Speedup* ideal. En la realidad esta definición de velocidad no se cumple estrictamente, debido a las sobrecargas de comunicación y sincronización de los sistemas paralelos que se incrementan al aumentar el número de procesos de la aplicación.

El desconocimiento de esta información a la hora de ejecutar la aplicación en el sistema, podría llevar a realizar un uso ineficiente del mismo, causando problemas a distintos niveles, como podrían ser: la disminución del trabajo realizado en el sistema *(througput)*, ya que si no se utilizan eficientemente los recursos podríamos reducir drásticamente el número disponible de éstos; no poder planificar los trabajos de las colas de ejecución de manera eficiente; no conseguir el *Speedup* esperado de la aplicación; o todavía más importante, el problema económico, ya que no utilizar eficientemente el sistema, supone un elevado coste económico y/o energético. Este último punto es especialmente crítico en entornos HPC sobre Cloud, donde el usuario paga únicamente por el tiempo consumido durante la ejecución de su aplicación.

Con el fin de intentar solventar estos problemas y hacer un uso eficiente del sistema, tanto usuarios como administradores de sistemas utilizan modelos predictivos de rendimiento, los cuales están centrados en predecir el rendimiento de la aplicación en un determinado sistema, antes de ser ejecutada, utilizando como métrica de rendimiento el tiempo de ejecución de la aplicación. Gracias a estos modelos de predicción, los usuarios pueden seleccionar la cantidad de recursos más adecuada para ejecutar su aplicación, lo que permite tener la seguridad de utilizar los recursos de forma eficiente.

Actualmente, existen tres técnicas para predecir el rendimiento de aplicaciones paralelas de paso de mensajes en sistemas HPC, las cuales se pueden clasificar en tres grandes grupos: simulación, modelos analíticos y medición.

Las técnicas de simulación consisten en modelizar tanto la aplicación paralela como el comportamiento del sistema, con el objetivo de reproducir una abstracción del mismo. Presentan una manera fácil de predecir el rendimiento y son muy útiles en entornos de producción donde no se tiene acceso al sistema, para evaluar distintas alternativas de como ejecutar la aplicación de la forma más eficiente. Por el contrario, este tipo de técnica de predicción requiere un gran esfuerzo previo para modelizar y desarrollar el sistema completo. Además, en muchos casos, si se desea obtener una alta calidad de predicción requieren un elevado tiempo de simulación.

Los modelos analíticos son representaciones matemáticas del entorno de ejecución (aplicación paralela + computador paralelo). Requieren un elevado nivel matemático por parte del usuario, dando como resultado modelos de una gran complejidad, que en muchos casos no resultan útiles en la práctica. Con el fin de hacer útiles y aplicables estos modelos, se simplifican descartando variables del sistema. Estas simplificaciones dan como resultado modelos poco confiables y con un elevado error de predicción.

Las técnicas de medición consisten en monitorizar el sistema real cuando está sometido a una carga de trabajo, ya sea real (la propia aplicación) o sintética (*benchmarks*), con el objetivo de medir ciertos parámetros del sistema que ayuden a predecir el rendimiento de la aplicación. Debido a la complejidad de las aplicaciones paralelas de paso de mensajes, junto a las restricciones de configuración de los *benchmarks*, hace que el uso de estos no sea adecuado para predecir el rendimiento de la aplicación, ya que no representa su comportamiento con un alto nivel de precisión, siendo más apropiado utilizar la propia aplicación para predecir su rendimiento. A la hora de utilizar la propia aplicación para analizar y predecir su rendimiento, se debe considerar el tiempo y los recursos requeridos para realizar la dicha evaluación.

En este trabajo se propone una nueva metodología, llamada *Prediction of Parallel Program Scalability* (P3S), la cual permite predecir el rendimiento de la escalabilidad fuerte de aplicaciones paralelas de paso de mensajes, en un determinado sistema, utilizando un tiempo de análisis acotado y un número de recursos limitado.

1.2. Objetivos

El presente trabajo de investigación está enmarcado en el estudio de la predicción de la escalabilidad fuerte de aplicaciones científicas paralelas de paso de mensajes. La escalabilidad fuerte tiene como objetivo disminuir el tiempo de ejecución de una aplicación aumentando su número de procesos. En escalabilidad fuerte, el tamaño del problema (*workload*) se mantiene constante a medida que la aplicación escala. A la hora de estudiar la escalabilidad fuerte en aplicaciones paralelas, el rendimiento de la aplicación se convierte en uno de los aspectos más importante a considerar.

El objetivo principal de esta investigación es proponer una metodología para analizar y predecir el comportamiento de la escalabilidad fuerte en aplicaciones paralelas de paso de mensajes, en un determinado sistema, utilizando un tiempo de análisis acotado y un número de recursos limitado del sistema.

La metodología propuesta, denominada P3S, parte del comportamiento repetitivo de las aplicaciones paralelas de paso de mensajes. Se conoce que este tipo de aplicaciones están compuestas por un conjunto de fases, las cuales se van repitiendo a lo largo de toda la aplicación [6], tal y como se muestra en la figura 1.1, donde se presenta un fragmento de la aplicación científica de paso de mensajes NAMD [7]. Como se puede apreciar en la figura, encontramos dos fases claramente diferenciadas, las cuales se van repitiendo a lo largo del tiempo de ejecución de la aplicación.

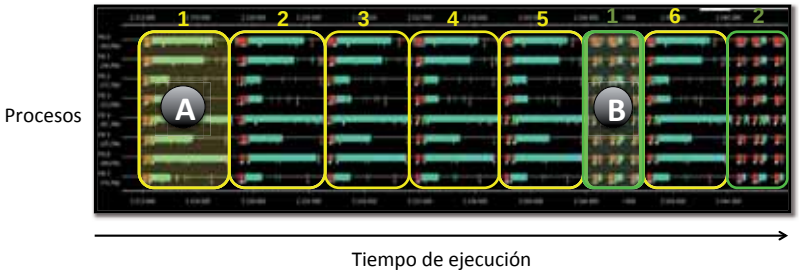


Figura 1.1: Repetitividad de las aplicaciones paralelas de paso de mensajes

Las fases de las aplicaciones paralelas han sido escritas por los desarrolladores utilizando patrones específicos de comunicación y cómputo, los cuales siguen unas determinadas reglas de comportamiento a medida que su número de procesos aumenta. Conforme la aplicación incrementa su número de procesos, el número total de fases de la aplicación permanece constante, pero los patrones cambian su comportamiento

siguiendo estas reglas, siendo las fases funcionalmente constantes, es decir, realizan el mismo trabajo repartido entre un número diferente de procesos. Esto es debido a que son los mismos segmentos de código de la aplicación, escalados para un número mayor de procesos.

Para obtener los patrones de las fases de una forma transparente y automática, la metodología P3S utiliza la herramienta PAS2P [8]. Dicha herramienta identifica las fases de la aplicación y nos permite crear la firma de la aplicación. Esta firma contiene únicamente las fases relevantes, es decir, segmentos de código ejecutables que tienen impacto en el rendimiento de la aplicación, y su frecuencia de repetición (peso). La ejecución de la firma nos permite caracterizar y analizar la aplicación de manera eficiente, ya que nos permite centrarnos en analizar únicamente las fases relevantes de la aplicación, cada una de las cuales puede tener un comportamiento diferente, cubriendo aproximadamente un 95 % del código de la aplicación, en el 1 % del tiempo de ejecución de la aplicación, ya que únicamente ejecutamos sus fases relevantes.

La figura 1.2 muestra una descripción global de la metodología P3S con el objetivo de ofrecer al lector un visión general de la misma. Tal y como se muestra en la figura, el principal enfoque de la metodología consiste en caracterizar, analizar, y finalmente modelizar los patrones de comunicación, cómputo y peso para cada fase relevante de la aplicación de forma transparente, es decir, sin modificar el código fuente de la aplicación, a partir de un conjunto de firmas de la aplicación para un número diferente y reducido de procesos. Mediante la ejecución de este conjunto de firmas, será posible obtener información rápida sobre el comportamiento de cada fase relevante de la aplicación, a medida que la aplicación escala. Una vez analizados los patrones de cada fase relevante de la aplicación, se modelizarán con el objetivo de obtener sus ecuaciones generales de comportamiento, las cuales nos permitirán predecir estos patrones a medida que se incrementa el número de procesos de la aplicación.

A partir de las ecuaciones generales de comportamiento para cada fase relevante de la aplicación, se genera la traza lógica escalable de la aplicación *(Scalable Logical Trace (SLT))*, la cual es independiente del sistema paralelo, sólo depende de la forma en que la propia aplicación ha sido desarrollada, es decir, únicamente posee los parámetros intrínsecos de la aplicación. Esta traza podrá ser parametrizada para el número de procesos para el cual se desee predecir el rendimiento de la aplicación.

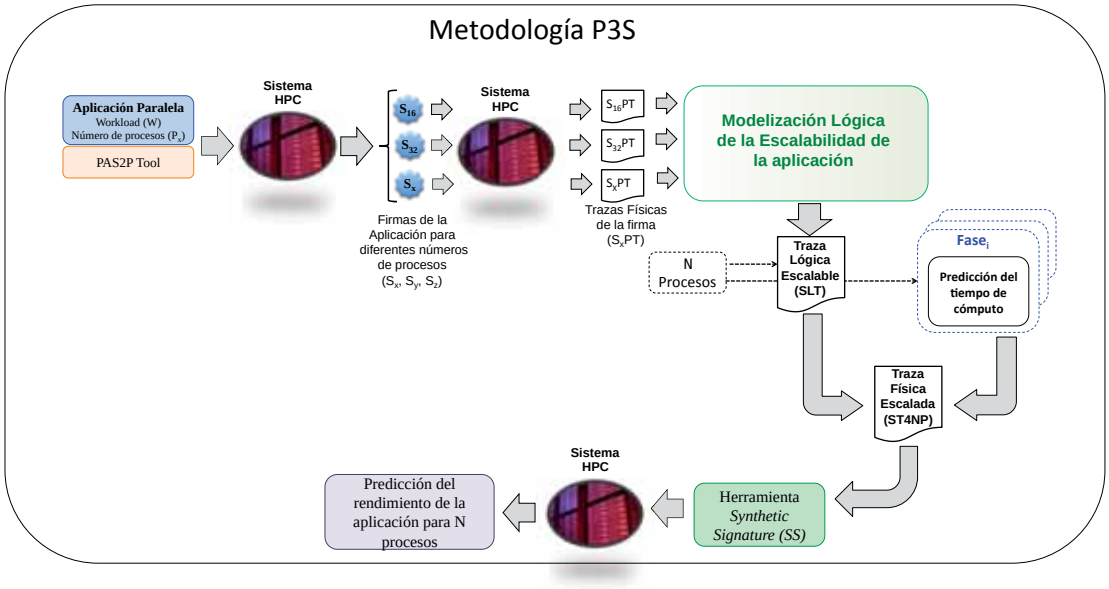


Figura 1.2: Descripción general de la metodología

La SLT contiene únicamente el conjunto mínimo de parámetros intrínsecos necesarios de cada fase relevante de la aplicación, utilizados para predecir el tiempo de cómputo y comunicación a medida que la aplicación escala. Los parámetros utilizados son: el tipo de evento de comunicación (tipo de primitiva MPI), el origen y destino de cada evento de comunicación, su volumen de comunicación, el número de instrucciones de cómputo entre dos eventos de comunicación consecutivos y el peso de la fase.

Una vez la SLT ha sido generada, tiene que ser complementada con el tiempo predicho de cómputo de cada fase, para el número de procesos para el cual se desea predecir su rendimiento *(N)*, con el objetivo de generar la traza física escalada *(Scaled Trace for N Processes (ST4NP))* de la aplicación para N procesos, la cual será utilizada para predecir el rendimiento de la aplicación. Esta nueva traza es dependiente de la máquina, ya que contiene el tiempo de cómputo predicho, además de toda la información de la SLT.

Finalmente, el último paso de la metodología antes de obtener el rendimiento de la aplicación, es evaluar el comportamiento prestacional de la ST4NP en el sistema destino, utilizando un número de recursos reducido. Para ello, se utiliza una herramienta, llamada *Synthetic Signature (SS)*, la cual recibe como entrada la ST4NP. Una vez leída la traza, la herramienta SS procesará sus eventos de comunicación y cómputo en el sistema

paralelo, con el objetivo de obtener el rendimiento de la aplicación. Puesto que la traza física contiene el origen y destino de los mensajes, sus volúmenes de comunicación y el tiempo de cómputo entre primitivas, el objetivo es medir el tiempo de comunicación de los mensajes de cada fase, con el fin de obtener el tiempo de ejecución total de la fase. Además de los eventos de comunicación, la herramienta SS emulará también el tiempo de cómputo entre los eventos de comunicación, con el objetivo de mantener la frecuencia cómputo-comunicación de cada fase, ya que si sólo ejecutáramos los eventos de comunicación, no estaríamos representando el comportamiento real de la fase, tal y como sucede en aplicación, afectando a los tiempos de envío y recepción de los eventos de comunicación. Puesto que uno de los requisitos de la metodología es utilizar un número de recursos limitado, la ST4NP será procesada en el sistema por partes, mediante un proceso iterativo en el cual se ejecutan rangos de procesos en función del número de cores del nodo del sistema, hasta que todos los procesos de la ST4NP han sido procesados, con el fin de obtener el tiempo predicho de cada fase de la aplicación.

Una vez que todos los eventos de comunicación de los diferentes procesos han sido medidos, podemos calcular el tiempo de ejecución predicho de cada fase relevante de la aplicación, ya que disponemos del tiempo de cómputo y comunicación de cada evento de la fase. Con el objetivo de obtener el tiempo de predicción de la aplicación, el tiempo de ejecución predicho de cada fase será multiplicado por su peso predicho (el número de veces que se repite la fase), dando como resultado el tiempo total de la aplicación.

A lo largo de este trabajo se explicará en detalle la metodología propuesta, la cual será validada aplicándola con *benchmarks* y aplicaciones científicas de paso de mensajes.

1.3. Contribuciones

A partir de la metodología P3S, los fundamentos de la cual fueron introducidos brevemente en la sección anterior, se presentan las principales contribuciones surgidas de este trabajo:

- Como contribución principal se ha propuesto la metodología P3S [9], mediante la cual analizar y predecir el comportamiento de la escalabilidad fuerte de las aplicaciones paralelas de paso de mensajes en un determinado sistema, tanto en computadores paralelos como configurando un sistema HPC en Cloud, utilizando un conjunto limitado de recursos y un tiempo de análisis acotado.

La metodología, además de ser utilizada para predecir el rendimiento de la aplicación considerando la escalabilidad fuerte, puede ser aplicada para diferentes propósitos como:

- Estudiar el comportamiento de las diferentes fases de la aplicación con el objetivo de detectar posibles fases, las cuales pueden convertirse en cuellos de botella potenciales a medida que la aplicación escala.
- Analizar el impacto del *mapping* y proponer al usuario estrategias de *mapping* que permitan minimizar los efectos de las latencias entre los diferentes enlaces de comunicación.
- Utilizar la información proporciona por la metodología en la planificación de tareas o en la reserva de recursos *(scheduling)*, con el objetivo de hacer un uso eficiente de los recursos del sistema.

- Se ha propuesto una metodología para obtener la curva de escalabilidad de la aplicación paralela en un sistema concreto. El modelo se basa en caracterizar, analizar y finalmente modelizar los patrones de comunicación, cómputo y peso para cada fase relevante de la aplicación, a medida que escala, de una forma transparente sin necesidad de analizar el código fuente de la aplicación, generando una traza lógica escalada para N procesos (SLT) [10].

- Se ha propuesto una metodología [11] para predecir el tiempo de cómputo de cada fase de la aplicación, a medida que la aplicación escala, utilizando un conjunto reducido de recursos del sistema, mediante la cual generar la traza física escalada para N procesos.

- Se ha desarrollado una herramienta [12] que reproduce el comportamiento de la aplicación a partir de la traza física generada, y permite medir los eventos de comunicación de cada fase de la aplicación, utilizando un conjunto limitado de recursos del sistema.

1.4. Organización de la tesis

El trabajo presentado en esta tesis se ha dividido en los siguientes capítulos:

- Capítulo 2: Presenta una visión general de las bases teóricas necesarias que fundamentan el desarrollo de este trabajo de investigación. También se detallan las métricas utilizadas para análisis de rendimiento, destacando los conceptos de eficiencia y escalabilidad, fundamentales para definir el rendimiento de los sistemas paralelos.

- Capítulo 3: Presenta una visión general actual del estado del arte sobre predicción de rendimiento de aplicaciones paralelas de paso de mensajes.

- Capítulo 4: Presenta la metodología P3S propuesta en esta tesis para predecir la escalabilidad de aplicaciones paralelas de paso de mensajes.

- Capítulo 5: Presenta en detalle el modelo propuesto en la primera etapa de la metodología, para modelizar el comportamiento lógico de escalabilidad de cada fase relevante de la aplicación paralela, a partir de un conjunto reducido de firmas de la aplicación, obteniendo las reglas generales que definen el comportamiento lógico de cada fase.

- Capítulo 6: Presenta en detalle el modelo de predicción del tiempo de cómputo, utilizado en la segunda etapa de la metodología para predecir el tiempo de cómputo de cada fase de la aplicación para N procesos.

- Capítulo 7: Presenta en detalle la herramienta *Synthetic Signature (SS)*, utilizada en la tercera etapa de la metodología, para predecir el tiempo de comunicación de cada fase de la aplicación.

- Capítulo 8: Presenta la última etapa de la metodología, consistente en obtener el rendimiento de la aplicación y su curva de escalabilidad.

- Capítulo 9: Presenta la validación experimental de la metodología propuesta, mediante la utilización de diferentes aplicaciones paralelas de cómputo científico.

- Capítulo 10: Finalmente, este último capítulo presenta las conclusiones generales de este trabajo. Además, se exponen las principales contribuciones de esta investigación para la comunidad científica, y se presenta el trabajo futuro y las líneas abiertas.

2

Sistemas paralelos

2.1. Introducción

En los últimos años han surgido una gran cantidad de problemas que requieren gran capacidad de cómputo para ser resueltos. Un ejemplo de estos problemas podría ser: el alineamiento de genomas, la predicción meteorológica o la simulación de fenómenos naturales. Debido a la gran cantidad de cómputo que requieren este tipo de problemas, no pueden ser resueltos en computadores convencionales en un tiempo razonable. Una solución a este problema es utilizar computadores paralelos. La utilización de este tipo de computadores presenta nuevos retos y desafíos que se suman a los ya existentes en computadores convencionales, como pueden ser: el tipo de arquitectura paralela a utilizar, el diseño de algoritmos paralelos, o la necesidad de nuevas métricas de rendimiento como la escalabilidad o la eficiencia de los algoritmos paralelos.

Este capítulo presenta una visión general de las bases teóricas necesarias para el desarrollo y ejecución de aplicaciones paralelas, haciendo especial énfasis en los conceptos que serán utilizados para el desarrollo de la metodología propuesta.

2.2. Modelos arquitecturales

Una de las taxonomías más conocidas y utilizada en computadores es la taxonomía de Flynn [13], propuesta por Michael J. Flynn en 1972. A pesar de ser una taxonomía muy antigua, es la base de muchas otras más modernas. La figura 2.1 ilustra esta taxonomía, la cual se divide en cuatro grandes grupos en función del número de instrucciones concurrentes y el flujo de datos disponibles en la arquitectura:

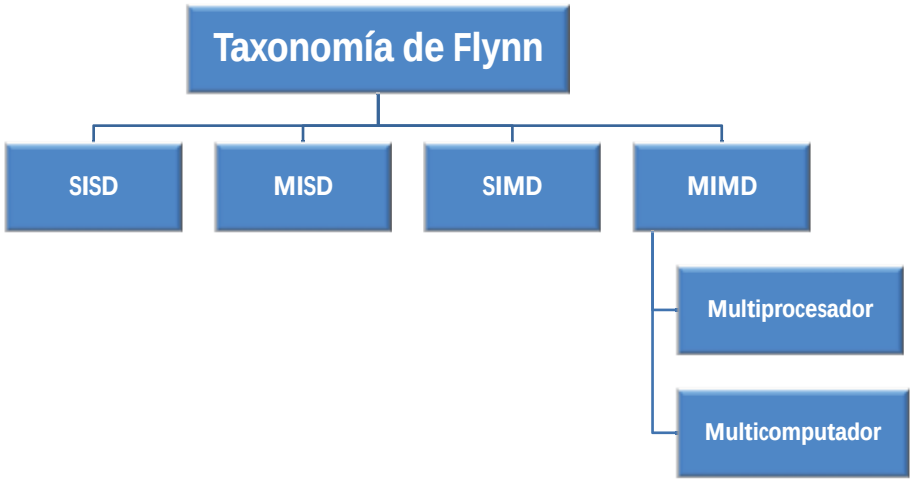


Figura 2.1: Taxonomía de Flynn

- *SISD (Single Instruction Single Data)*: Arquitectura tradicional que no explota el paralelismo en las instrucciones ni en el flujo de datos. Se caracteriza por tener un único flujo de instrucciones con un único flujo de datos. Son los sistemas que corresponden a la arquitectura Von Neumann.

- *MISD (Multiple Instruction Single Data)*: Arquitectura paralela donde el mismo conjunto de datos es tratado de forma diferente por los procesadores. Son útiles en casos donde sobre el mismo conjunto de datos se deben realizar muchas operaciones. Hasta el momento, no se ha fabricado ninguna máquina de propósito general, sólo prototipos a nivel de investigación.

- *SIMD (Single Instruction Multiple Data)*: Arquitectura paralela que explota varios flujos de datos dentro de un único flujo de instrucciones con el objetivo de realizar operaciones que pueden ser paralelizadas de manera natural, por ejemplo, un procesador vectorial.

- *MIMD (Multiple Instruction Multiple Data)*: Arquitectura que presenta un paralelismo funcional y/o de datos. No sólo se distribuyen datos sino también tareas a procesar entre los distintos procesadores. Varios flujos de instrucciones son aplicados a distintos flujos de datos. Esta categoría no es muy concreta, ya que puede abarcar una gran cantidad de arquitecturas posibles.

La metodología propuesta en esta tesis está enfocada a predecir el comportamiento de la escalabilidad de aplicaciones paralelas de paso de mensajes, las cuales se ejecutan sobre este tipo de arquitectura.

2.3. Arquitecturas paralelas

Nuestro marco de trabajo está centrado en los computadores paralelos MIMD *(Multiple Instruction Multiple Data)*. Los computadores paralelos pueden ser clasificados en dos tipos de arquitecturas según la distribución de la memoria: multiprocesadores y multicomputadores.

2.3.1. Multiprocesadores

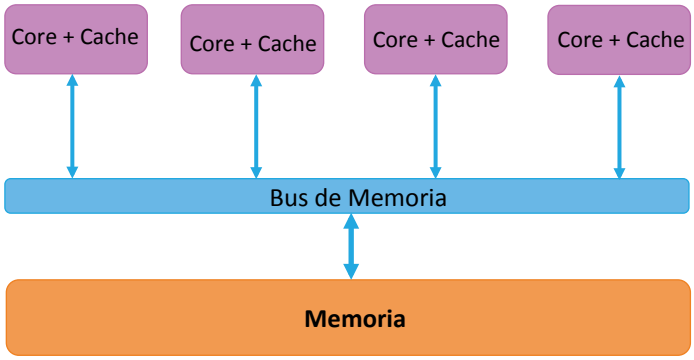


Figura 2.2: Arquitectura Multiprocesador

A diferencia de los multicomputadores, donde los procesadores que conforman el sistema no comparten memoria, los multiprocesadores tienen una visión global del espacio de memoria, es decir, utilizan un único espacio de memoria que comparte todo el sistema. La figura 2.2 ilustra un ejemplo de este tipo de arquitectura, la cual a su vez puede clasificarse en:

- *Sitemas UMA (Uniform Memory Access)*: En este tipo de arquitectura el tiempo de acceso a cualquier posición de la memoria tendrá la misma latencia. Corresponden a este modelo las arquitecturas PVP (Parallel Vector Processors) y SMP (Symmetric Multi Processors)

- *Sistemas NUMA (Non Uniform Memory Access)*: En este tipo de arquitectura el tiempo de acceso dependerá de la localización de los datos. Utilizan una capa DSM

(Distributed Shared Memory) que puede estar implementada mediante hardware o software.

En los últimos años, este tipo de arquitectura ha evolucionado a tal nivel, que los sistemas actuales dedicados al uso personal incluyen varios procesadores por nodo, donde cada procesador contiene un conjunto de cores.

2.3.2. Multicomputadores

Es el tipo de arquitectura utilizada en el marco de este trabajo de investigación. Este tipo de arquitectura no comparte memoria común entre nodos. Cada nodo de procesamiento posee su propios cores y memoria. Esta memoria será dedicada exclusivamente para el uso de los cores del nodo de procesamiento. También pueden disponer de dispositivos de Entrada/Salida (E/S). Los nodos de cómputo, intercambian información (datos) entre ellos mediante el intercambio de mensajes (primitivas de comunicación de envío y recepción), que circulan por redes de interconexión.

A la hora de desarrollar las aplicaciones que se ejecutarán en este tipo de arquitectura, se utiliza el modelo de programación de paso de mensajes (MPI) [14]. La figura 2.3 ilustra un ejemplo de este tipo de arquitectura, la cual puede clasificarse de forma más concreta en dos grupos: MPP (Massively Parallel Processors) y COW (Cluster of Workstations) [15].

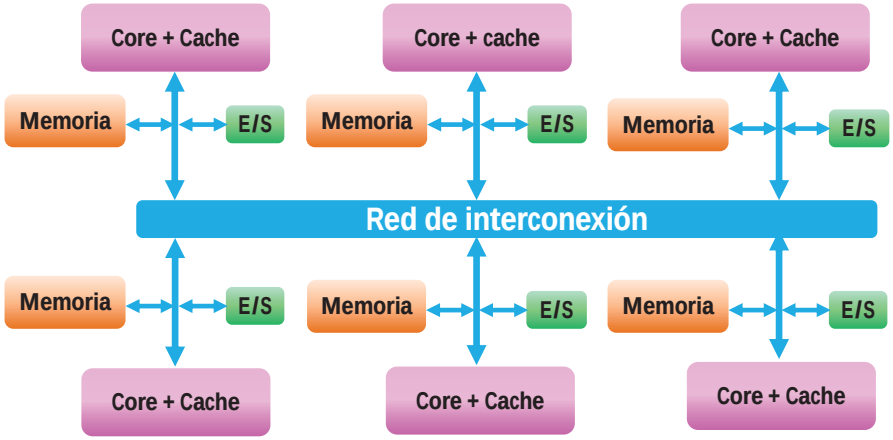


Figura 2.3: Arquitectura Multicomputador

2.3.2.1. MPP (Massively parallel processing)

Este tipo de arquitectura fue diseñada con el objetivo de evitar los cuellos de botella en el bus de memoria. Se basa en distribuir la memoria entre los procesadores de modo que se asemeje a una red (cada procesador con su memoria distribuida asociada es similar a un computador dentro de una red de procesamiento distribuido). Debido a la distribución dispersa de los recursos, esta arquitectura es también conocida como débilmente acoplada *(Loosely Coupled)*. Con el fin de tener acceso a la memoria fuera de su nodo de cómputo, los nodos utilizan un esquema de paso de mensajes. Este sistema reduce el tráfico del bus de memoria, debido a que cada sección de memoria observa únicamente aquellos accesos que le están destinados, en lugar de observar todos los accesos.

2.3.2.2. COW(Clusters of workstation)

Se basan en agrupar un conjunto de *Workstations* de propósito general interconectadas a través de una red de interconexión estándar. La principal motivación por la cual surgieron los *Cluster of Workstation (COW)*, fue intentar reducir el coste de los computadores paralelos, ya que existe una gran diferencia entre el precio de una *Workstation* de propósito general y el precio de un supercomputador. No obstante, las *Workstation* han ido aumentando significativamente su capacidad de procesamiento, debido a los avances tecnológicos de los últimos años, convirtiéndose en una buena alternativa a la hora de utilizar computación paralela.

2.4. Modelos de programación paralela

Las aplicaciones paralelas consisten en una o más tareas que pueden comunicarse y cooperar entre ellas con el objetivo de resolver un determinado problema. En cuanto al diseño e implementación de aplicaciones paralelas, no hay una metodología claramente definida, esto es debido a la fuerte dependencia de la arquitectura del sistema en el cual se ejecute la aplicación. Debido a este motivo, una de las decisiones más importantes a la hora de implementar una aplicación paralela es decidir el modelo de programación paralela a utilizar. Los modelos de programación paralela se definen como una abstracción de la arquitectura entre el uso de hardware y la memoria. A continuación, se

detallan los dos modelos más utilizados a la hora de implementar aplicaciones paralelas, el modelo de memoria compartida y el modelo de paso de mensajes.

2.4.1. Modelo de memoria compartida

Modelo de programación [16] donde los programadores ven sus programas como una colección de procesos accediendo a variables locales y un conjunto de variables compartidas. Cada proceso accede a los datos compartidos mediante operaciones de lectura o escritura asíncrona. Por tanto, como más de un proceso puede realizar operaciones de acceso a memoria a los mismos datos compartidos en el mismo instante tiempo, es necesario implementar mecanismos para resolver problemas de exclusiones mutuas.

En este modelo de programación se ve la aplicación como una colección de tareas, que normalmente son asignadas a *threads* de ejecución de forma asíncrona. Una de las implementaciones más utilizadas para programar aplicaciones paralelas utilizando este modelo es OpenMP *(Open Specifications for Multi Processing)* [17]. OpenMP se basa en primitivas de librería para controlar la paralelización de bucles y otras secciones del código susceptibles a ser paralelizables.

Actualmente están surgiendo nuevos modelos de programación de memoria compartida, los cuales están diseñados específicamente para ser ejecutados sobre aceleradores sujetos a la arquitectura SIMD *(Single Instruction Multiple Data)*. Dos de los lenguajes que están teniendo más impacto en la comunidad científica son CUDA [18] y OpenCL [19].

2.4.2. Modelo de paso de mensajes

Modelo de programación [20] paralelo ampliamente extendido por la comunicad científica. Se encuentra implementado en la mayoría de plataformas actuales de cómputo paralelo. Se basa en una colección de tareas con variables locales privadas que pueden ser enviadas y recibidas entre los procesos de la aplicación por medio del intercambio de mensajes, que son enviados a través de una red de interconexión. Este es un modelo de programación explícito, pues el programador debe implementar explícitamente el esquema de distribución de datos, las comunicaciones entre procesos y su sincronización.

Las librerías más utilizadas de paso de mensajes son MPI [14] y PVM [21], estando esta última cayendo en desuso en los últimos años. En ambas librerías, al realizar el

intercambio de información, el proceso fuente inicia la comunicación enviando un mensaje de datos utilizando una interfaz de programación de paso de mensajes (funciones standards de envío y recepción de la librería de paso de mensajes) y el proceso receptor recibe la información.

La metodología propuesta en esta tesis está orientada a predecir el comportamiento de la escalabilidad de aplicaciones paralelas de paso de mensajes, desarrolladas utilizando la librería de paso de mensajes MPI.

2.5. Paradigmas de programación paralela

A la hora de elegir un paradigma de programación paralela, el programador debe considerar las características de la aplicación, de tal forma que el paradigma seleccionado permita ofrecer las máximas prestaciones a la aplicación.

Actualmente, existen diferentes clasificaciones de paradigmas de programación paralela. En este trabajo se ha seleccionado la propuesta por Buyya [22], ya que engloba a la mayoría de aplicaciones paralelas existentes. Buyya propone clasificar los paradigmas de programación paralela en 4 grupos: (*Master/Worker*, *Divide and Conquer*, *Single Instruction Multiple Data (SPMD)* y *Data Pipeline*). A continuación, se detalla cada uno de estos grupos.

2.5.1. Master/Worker

Este paradigma está compuesto por dos elementos: el *Master* y los *Workers*. El *Master* es el encargado de dividir y distribuir la carga de trabajo. Por su parte, los *Workers* son entidades que reciben la información, la computan y devuelven el resultado al *Master*. La figura 2.4 ilustra este paradigma, el cual permite distribuir la carga de forma estática o dinámica. La repartición estática distribuye la totalidad de la carga entre los *Workers*, mientras que la repartición dinámica asigna *chunks* (trozos) de la carga de trabajo a los *Workers* y, cuando estos finalizan su cómputo devuelven su resultado al *Master*, el cual vuelve a enviar más *chunks* al los *Workers*. Este proceso se realiza hasta que la carga total de la aplicación ha sido computada. Este paradigma presenta un patrón de comunicación fácilmente expandible, debido a que los *Workers* únicamente comunican con el *Master*. Por contra, al escalar la aplicación, el *Master*

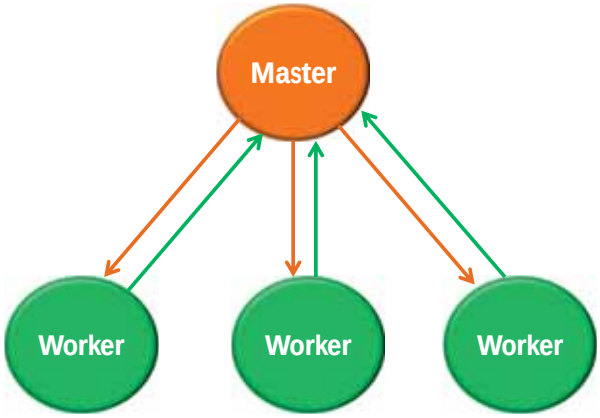


Figura 2.4: Paradigma Master/Worker

puede presentar un cuello de botella debido al elevado número de comunicaciones que tiene que gestionar con los workers.

2.5.2. SPMD (Single Program Multiple Data)

Este paradigma de programación está compuesto por dos características principales a evaluar, la aplicación y la carga de trabajo. El paradigma SPDM presenta un comportamiento determinístico, que viene determinado por el reparto de los datos y un proceso iterativo de intercambio de información entre los procesos. Cada proceso de la aplicación ejecutará el mismo segmento de código sobre un determinado conjunto de datos de entrada. Existe una necesidad de sincronismo entre los procesos, que vendrá determinada por el proceso más lento y podrá ser resuelta mediante mensajes de control por paso de mensajes. Debido a que el patrón de comunicación de este modelo de programación es fácilmente escalable, es uno de los más utilizados a la hora de desarrollar aplicaciones científicas de paso de mensajes.

2.5.3. Divide and Conquer

Este paradigma de programación permite sub-dividir la aplicación en diversas tareas, cada una de las cuales puede ser computada de manera independiente en paralelo. Al analizar la aplicación, las tareas son agrupadas con el objetivo de obtener el resultado final de la aplicación. La estructura de ejecución de la aplicación posee forma de árbol,

ya que el proceso está basado en dividir, computar y unir, siendo las hojas del árbol donde se realiza el cómputo de las sub-tareas.

2.5.4. Data Pipeline

Este tipo de paradigma se puede relacionar con el proceso de *pipeline* implementado en los procesadores. Se basa en atribuir para cada proceso una etapa de *pipeline* diferente, asociando diferentes fases con diferentes tareas.

2.6. Algoritmos de comunicación

Los algoritmos de comunicación de una aplicación, determinan como se comunican o enlazan los procesos de una aplicación entre ellos en tiempo de ejecución. Estos algoritmos están escritos utilizando reglas generales de comportamiento, las cuales indican como tiene que evolucionar el algoritmo a medida que se incrementa el número de procesos de la aplicación.

A continuación, se presentan los algoritmos de comunicación más extendidos en aplicaciones paralelas de cómputo científico. Cabe comentar que no son los únicos existentes a la hora de programar aplicaciones paralelas, incluso un programador puede diseñar su propio algoritmo paralelo si así lo requiere el tipo de aplicación que este desarrollando.

2.6.1. Perfect Shufle

El algoritmo de comunicación denominado *Perfect Shufle* es utilizado con mucha frecuencia en aplicaciones paralelas debido al tipo de permutación de este. Deriva su nombre del proceso de barajar naipes, ya que el proceso de permutación es el mismo al proceso de barajar un mazo de cartas convencional, donde se corta la baraja en dos partes iguales y después se va tomando un naipe de cada parte para conseguir el mazo barajado. De esta forma, se consigue que todas las cartas que eran adyacentes antes de la operación, estén separadas, al menos, por otra carta. La figura 2.5 ilustra la permutación sufrida por 8 elementos después de efectuar sobre ellos un *Perfect Shufle*.

La ecuación 2.1 muestra la regla de comunicación de este algoritmo, donde x es el nodo emisor del cual se quiere calcular el nodo receptor del mensaje y b es el número de bit del nodo.

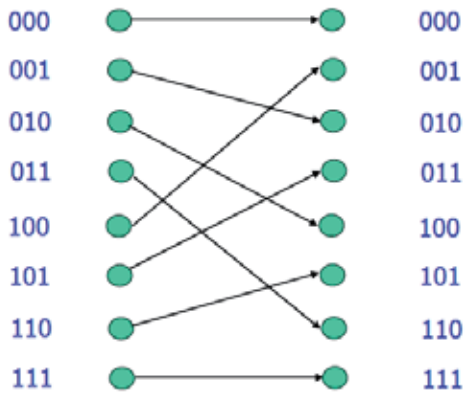


Figura 2.5: Patrón de comunicación *Perfect Shufle*

$$S(x) = \{b_{n-1}, b_{n-2}, ..., b_1, b_n\} \tag{2.1}$$

2.6.2. Butterfly

Otro algoritmo bastante utilizado es patrón *Butterfly*. Este algoritmo de comunicación es muy parecido al *Perfect Shufle*, estando basado en permutaciones de bits. Tiene la interesante propiedad de ser simétrico, es decir, el nodo receptor es el inverso del nodo emisor. La figura 2.6 muestra un ejemplo de este tipo de patrón.

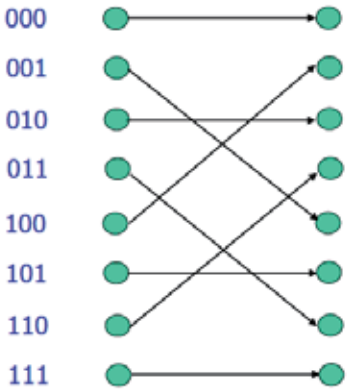


Figura 2.6: Patrón de comunicación Butterfly

Este patrón de comunicación viene dado por la ecuación 2.2, donde x es el nodo del cual se quiere calcular su comunicación y b es el número de bit del nodo.

$$B(x) = \{b_1, b_{n-1}, ..., b_2, b_n\} \tag{2.2}$$

2.6.3. Patrón Lineal

Es uno de los patrones más sencillos de comunicación, se trata de un patrón en el que los nodos se conectan con su nodo inmediatamente superior, es decir, el nodo de cómputo 1 se conectara con el nodo 2, este a su vez se conectará con el nodo 3 y así sucesivamente hasta llegar al último nodo de cómputo. No es un patrón muy utilizado ya que debido a su forma de interconexión presenta una escalabilidad reducida.

2.6.4. Patrón en Anillo

Es similar al patrón de comunicación lineal a diferencia que el último nodo de cómputo se vuelve a conectar con el primero formando un anillo. Al igual que el patrón lineal, presenta una escalabilidad reducida, dando como resultado que no sea muy utilizado en aplicaciones científicas las cuales tienen que ser ejecutadas con un elevado número de procesos.

2.6.5. Patrón en Malla

Patrón de comunicación donde un nodo está conectado con todos sus nodos vecinos. Este patrón de comunicación es muy utilizado en aplicaciones del tipo *SPMD*, debido a que puede ser escalado fácilmente.

2.7. Métricas de rendimiento

Uno de los principales desafíos en el cómputo de altas prestaciones es conseguir aplicaciones paralelas eficientes, las cuales sean capaces de aprovechar de la mejor manera posible los recursos que ofrecen los sistemas. Es importante tener métricas e índices que nos indiquen si la aplicación está haciendo un uso eficiente del sistema. Las métricas utilizadas a la hora de medir las prestaciones son variadas, como el tiempo de ejecución de la aplicación, el *speedup*, la eficiencia, la redundancia o la isoeficiencia. Otro factor que se está empezando a considerar en los últimos años y que está adquiriendo cada vez más peso en los centros de computación, es el consumo energético. Prueba de ello

es el ranking *Green500* [23], el cual contiene una lista de los supercomputadores con el consumo energético más eficiente.

A continuación, se detallan las principales métricas de prestaciones utilizadas en este trabajo de investigación.

2.7.1. Tiempo de ejecución

Es el índice de prestaciones más intuitivo. Es un parámetro absoluto, pues permite medir la rapidez de la aplicación paralela sin necesidad de compararla con otra aplicación. En aplicaciones paralelas, el tiempo de ejecución se define como el intervalo de tiempo que transcurre desde el comienzo de la ejecución de la aplicación en el sistema paralelo, hasta que el último proceso culmina su ejecución.

2.7.2. Speedup

El *speedup* es la principal métrica que nos indica la ganancia que obtenemos mediante la paralelización. Se calcula como el cociente entre el tiempo de cálculo en una máquina secuencial y el mismo cálculo en una máquina paralela con un determinado número de procesadores. La ecuación 2.3 muestra la fórmula para realizar el cálculo siendo p el número de procesos con los cuales se ha ejecutado la aplicación paralela.

$$Speedup(p) = \frac{T.Serie}{T.paralelo(p)} \tag{2.3}$$

Obviamente, el *speedup* de una aplicación variará de una máquina paralela a otra y, en función de como se haya realizado la paralelización. Cuanto más alto sea el *speedup*, más ganancia habremos obtenido con nuestra paralelización.

2.7.3. Eficiencia

Se define como el cociente entre el *speedup* y el número de procesos en los que ejecutamos la aplicación. La ecuación 2.4 muestra la ecuación utilizada para realizar el cálculo, siendo p el número de procesos con los cuales se ha ejecutado la aplicación.

$$Eficiencia(p) = \frac{Speedup(p)}{p} \tag{2.4}$$

Su valor estará comprendido entre 0 y 1. Nos interesa que la eficiencia tenga un valor lo más cercano a uno que sea posible, aunque generalmente obtendremos un valor

menor. En el hipotético caso de que la eficiencia fuera uno, significará que la paralelización ha sido total, repartiendo equilibradamente la carga y el tiempo de proceso entre todos los procesadores. Llegar a esta situación es casi imposible. La dificultad de repartir equitativamente el tiempo de proceso estriba en gran medida en como sea nuestro algoritmo y lo fácil que sea de paralelizar. Suele ser muy habitual que haya partes de nuestro algoritmo que no podamos paralelizar de ninguna manera. Además, el hecho de paralelizar siempre supone un *overhead* presente en todos los algoritmos paralelos, que es el responsable de no poder lograr un *speedup* lineal. Al diseñar algoritmos paralelos, se trabaja para minimizar la sobrecarga y poder lograr así un aumento del *speedup*. Todo esto hace que nuestra paralelización nunca vaya a ser perfecta, en el sentido de que frecuentemente obtendremos un rendimiento inferior a 1.

2.7.4. Ley de Amdahl

La ley de Amdahl [24] evalúa la máxima eficiencia que un algoritmo paralelo puede lograr con respecto a su versión serie. La eficiencia del algoritmo paralelo, según se deduce de la ley de Amdahl, está sujeta a la proporción paralelizable del algoritmo serie, y no depende de la cantidad de procesos que intervengan en el cálculo. El aumento de las prestaciones de un computador tiene que ver con la identificación de sus cuellos de botella, tal y como se deduce de la ley de Amdahl que, además, plantea las dificultades del aprovechamiento eficiente del procesamiento paralelo.

2.7.5. Ley de Gustafson

El científico John L. Gustafson hizo una revisión de la ley de Amdahl en 1988, lo que dio lugar a lo que se conoce como la Ley de Gustafson [25]. Guftanson consideraba que la ley de Amdahl daba un limite para el descenso del tiempo de ejecución del algoritmo paralelo, dependiendo de la fracción serie del algoritmo, la cual no puede ser paralizada, pero esto por contra podría aprovecharse para aumentar el volumen del problema manteniendo el mismo tiempo serie, es decir, el volumen del problema realizado, aumentaba la fracción paralela, disminuyendo el influjo de la porción serie.

2.7.6. Escalabilidad

La escalabilidad [26] nos da una idea del comportamiento del sistema cuando incrementa el número de procesos o/y el tamaño del problema *(workload)* de una aplicación paralela. Se define como la capacidad de un determinado algoritmo de mantener sus prestaciones cuando aumenta el número de procesos o/y el tamaño del problema *(workload)*. En definitiva, la escalabilidad suele indicarnos la capacidad de una aplicación paralela de utilizar de forma eficiente un incremento en los recursos computacionales. Una aplicación paralela escalable suele ser capaz de mantener su eficiencia constante cuando aumentamos su número de procesos, incluso a base de aumentar el tamaño del problema. Si un algoritmo no es escalable, aunque se aumente el número de procesos, no se conseguirá incrementar la eficiencia, con lo cual no conseguiremos sacar partido a todos los cores del sistema.

La escalabilidad se puede clasificar en dos tipos: Escalabilidad Fuerte y Escalabilidad Débil.

2.7.6.1. Escalabilidad Fuerte

En este modelo de escalabilidad [27] el tamaño del problema *(workload)* a resolver se mantiene constante. El objetivo es disminuir el tiempo de ejecución de la aplicación aumentando el número de procesos. La figura 2.7 muestra un ejemplo de escalabilidad fuerte. La escalabilidad fuerte está estrechamente relacionado con la ley de Amdahl.

La metodología propuesta en esta tesis está centrada en predecir el comportamiento de la escalabilidad fuerte de las aplicaciones paralelas de paso de mensajes en un sistema concreto.

2.7.6.2. Escalabilidad Débil

En este tipo de escalabilidad [28] se aumenta el número de procesos de la aplicación manteniendo el tamaño de problema para cada proceso constante, consiguiendo con ello que el tiempo de cómputo por proceso sea constante. La figura 2.8 muestra un ejemplo de escalabilidad débil. Este tipo de escalabilidad fue descrito analíticamente por Gustafson. Observando la ley de Gustafson se desprende que incrementando el tamaño de un sistema nos permite tratar un tamaño de problema mayor en un tiempo de ejecución similar.

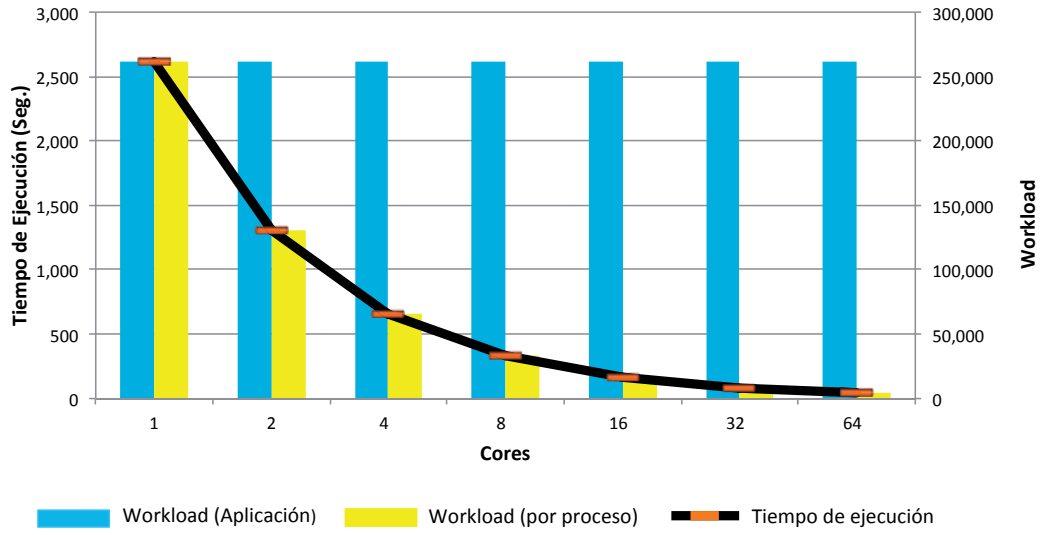


Figura 2.7: Escalabilidad fuerte

2.7.7. Isoeficiencia

El modelo de isoeficiencia parte de un sistema paralelo compuesto por N procesadores y se basa en dos consideraciones:

- La eficiencia de un sistema paralelo decrece cuando se incrementa el número de procesadores.
- Para un número de procesadores dado, instancias más grandes de un problema obtienen mejores valores de *speedup* y eficiencia.

Si se incrementan simultáneamente estos dos factores se podrá mantener la eficiencia constante. Debido a estas consideraciones, esta métrica permite analizar la escalabilidad, ya que indica cuanto tiene que aumentar W *(Workload)* para poder incluir más procesos (p) sin que la eficiencia del sistema se resienta. La ecuación 2.5 muestra la ecuación de isoeficiencia.

$$Isoeficiencia(W,p) = \frac{W}{p * T.Paralelo(W,p)} \tag{2.5}$$

En un sistema escalable la eficiencia se puede mantener fija cuando el tamaño del problema crece. Es útil saber a que velocidad debe crecer el problema. Para los distintos

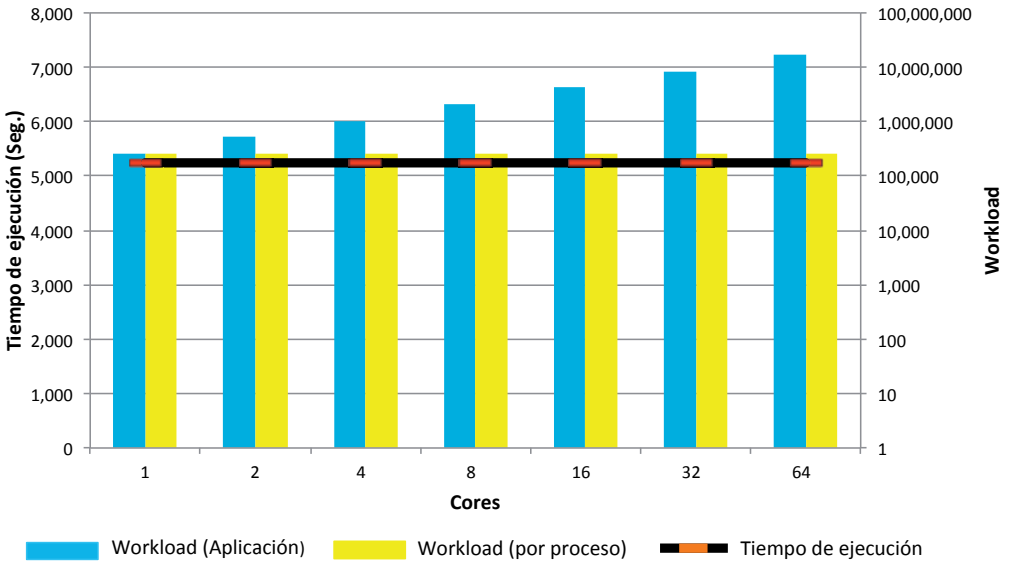


Figura 2.8: Escalabilidad débil

sistemas, esta velocidad de crecimiento (grado de escalabilidad) puede ser muy disímil. El análisis de isoeficiencia permite analizar el rendimiento de un programa paralelo sobre pocos procesadores y predecir su comportamiento sobre un número mayor.

El análisis de isoeficiencia es complejo en la práctica y no puede ser aproximado con alto nivel de precisión mediante el uso de la ecuación 2.5, pues debido a la compleja interacción entre aplicación y sistema, se requiere conocer diversos factores, tanto del comportamiento de la aplicación a medida que escala, como del sistema determinado para el cual se desea calcular la eficiencia de la aplicación.

Con el objetivo de tener un análisis más preciso de la escalabilidad de una aplicación paralela en un sistema determinado, este trabajo de investigación propone una metodología para predecir el comportamiento de la escalabilidad fuerte a medida que la aplicación escala, en un determinado sistema, a partir de la modelización del comportamiento de las fases relevantes de la aplicación.

3

Estado del Arte

3.1. Introducción

Dentro de la línea de investigación de predicción de escalabilidad de aplicaciones paralelas de paso de mensajes, hay dos desafíos importantes que tienen que ser resueltos con el objetivo de poder predecir el rendimiento de este tipo de aplicaciones, a medida que se incrementa su número de procesos. El primer desafío, consiste en modelizar la aplicación paralela con el objetivo de conocer su comportamiento y predecir su evolución a medida que la aplicación escala. El segundo desafío, consiste en proponer metodologías y técnicas para predecir el rendimiento de las aplicaciones en sistemas destino de forma eficiente, es decir, utilizando un tiempo de análisis acotado y un número de recursos limitado.

A lo largo de este capítulo se presentan los trabajos relacionados más relevantes en el área de modelización y predicción de aplicaciones paralelas de paso de mensajes, los cuales son comparados con la metodología propuesta en esta tesis.

Los métodos de predicción de rendimiento están basados en: modelos experimentales (basados en medición), modelos computacionales (basados en simulación) y modelos analíticos (basados en modelos teóricos). En los tres métodos se hace necesario disponer del modelo de la aplicación para poder llevar a cabo la predicción. La selección del método a utilizar influye en el tiempo de análisis, los recursos necesarios para resolver el problema y el nivel de precisión requerido en la predicción. La figura 3.1 muestra una tabla resumen con las principales características de los tres métodos.

La evaluación mediante modelos experimentales (basados en medición) es posible

	Precisión	Tiempo utilizado para realizar la predicción	Complejidad para generar el modelo de la aplicación	Complejidad para aplicar/utilizar el método	Necesidad de disponer del sistema paralelo para realizar la predicción	Necesidad de utilizar todos los recursos del sistema para predecir su rendimiento	Permiten predecir la aplicación para un elevado número de procesos
Modelos Experimentales (Medición)	Alta	Medio	Media	Media	Si	No	Si
Modelos Computacionales (Simulación)	Media	Alto	Alta	Baja	No	No	Si
Modelos Analíticos	Baja	Bajo	Alta	Alta	No	No	No

Figura 3.1: Tabla resumen con las principales características de los métodos de predicción

solamente si disponemos del sistema físico, de forma total o parcial, para el cual se desea obtener la predicción del rendimiento de la aplicación. Es la técnica más fiable y precisa, ya que se basa en monitorizar la aplicación en el sistema real y tomar ciertas mediciones que nos ayuden a predecir el rendimiento de la aplicación. En caso de no disponer del sistema real, se ha de recurrir a modelos computacionales (simulación) o modelos analíticos. Los modelos computacionales presentan una manera sencilla de predecir el rendimiento, resultando muy útiles en entornos donde no se tiene acceso al sistema. Por lo general, requieren un gran esfuerzo previo para modelizar y desarrollar el sistema completo (aplicación + computador paralelo). Además, en muchos casos, si se desea obtener una alta calidad de predicción, se requiere un elevado tiempo de simulación. Puesto que la mayoría de veces los resultados tienen que ser obtenidos con rapidez, hay que recurrir a modelos analíticos, los cuales son representaciones matemáticas del entorno de ejecución (aplicación + computador paralelo). Este tipo de modelos son complejos de obtener y en muchos casos no son útiles en la realidad, debido a la complejidad del modelo, por lo que requieren un alto grado de simplificación. Estas simplificaciones dan como resultado modelos de predicción rápidos y orientativos, los cuales pueden presentar un elevado error de predicción. Tanto los métodos de simulación como los métodos analíticos, resultan más eficientes si están basados en medidas previas de ciertos parámetros de la aplicación en el sistema real.

3.2. Métodos analíticos

En este área existen dos enfoques principales, los modelos matemáticos de regresión y *machine learning*. Estos métodos están ampliamente extendidos en la modelización de aplicaciones paralelas, ya que nos permiten modelizar la aplicación sin necesidad de

conocer sus detalles internos de comportamiento. Por otro lado, son fáciles de utilizar, por lo que resulta atractivo para el usuario, ya que le permite predecir de forma sencilla el rendimiento de la aplicación. Por contra, son métodos poco precisos, ya que simplifican el sistema a predecir, tanto aplicación como computador paralelo. A continuación, se presentan los trabajos más relevantes en esta área.

Un trabajo interesante donde se predice la escalabilidad de aplicaciones paralelas mediante la utilización de modelos matemáticos de regresión, tanto lineales como logarítmicos, es el propuesto por Barnes [29]. Su propuesta se centra en realizar un conjunto de ejecuciones de la aplicación, para un reducido número de procesos, con el objetivo de generar los modelos de regresión, mediante los cuales predecir el rendimiento de la aplicación, a medida que escala. A fin de mejorar la predicción, el modelo propuesto separa el tiempo de cómputo y el de comunicación. Una vez separados, genera los modelos de regresión para predecir el tiempo de cada uno de ellos. Esta propuesta utiliza modelos de caja negra, donde el comportamiento de la aplicación es desconocido, se basan en los parámetros de entrada de la aplicación para generar el modelo de la aplicación, los cuales tienen que ser conocidos por el usuario y conocer su relevancia e influencia en la aplicación.

Lee et al [30] propone predecir el rendimiento de aplicaciones paralelas mediante la utilización de dos técnicas claramente diferenciadas. Estas técnicas están basadas en modelos, los cuales son generados a partir de los parámetros de entrada de la aplicación. La primera técnica se basa en modelos de regresión polinomiales, mientras que la segunda técnica se basa en algoritmos de redes neuronales.

Ipek et al [31] presenta un enfoque diferente, basado en redes neuronales multicapa. A partir de un entrenamiento previo, donde la aplicación es ejecutada con diferentes configuraciones, se crea el modelo de la aplicación de manera automática, el cual permite extrapolar el rendimiento de la aplicación a medida que se incrementa su número de procesos. Este tipo de métodos presentan como inconveniente que necesitan un alto conjunto de muestras para obtener una alta calidad de predicción.

Rodríguez et al [32] [33] propone a través de una metodología, un entorno de análisis del rendimiento que permite obtener, de un modo sencillo, modelos analíticos muy precisos de aplicaciones en sistemas paralelos. La metodología proporciona un mecanismo automático de generación de modelos analíticos, a partir de la información de rendimiento obtenida tras la ejecución de las aplicaciones en un sistema concreto. En

particular, esta metodología ha sido diseñada para que el usuario concentre sus esfuerzos en el diseño experimental y en la valoración de los resultados, de tal forma que no sean necesarios conocimientos detallados del algoritmo de la aplicación o de la arquitectura subyacente. La metodología propuesta consta de dos fases interrelacionadas. La primera fase consiste en instrumentar el código, para obtener información de rendimiento acerca del comportamiento de la aplicación durante su ejecución, en múltiples situaciones experimentales, mientras que la segunda fase consiste en analizar estos valores de rendimiento mediante técnicas estadísticas, y construir automáticamente, a partir de las métricas y parámetros de rendimiento seleccionados por el usuario, un modelo analítico preciso del comportamiento de la aplicación. Esta fase ha sido implementada mediante una librería en el entorno estadístico R. En particular, el procedimiento de modelizado ha sido desarrollado mediante técnicas de selección de modelos basadas en el criterio de información de Akaike, una herramienta objetiva que permite cuantificar la idoneidad de un modelo particular, respecto de un conjunto finito de modelos.

3.3. Métodos computacionales

La simulación de aplicaciones paralelas tiene una larga historia en el campo de HPC, en particular en el contexto de aplicaciones paralelas de paso de mensajes. Existen dos formas de realizar la simulación claramente diferenciadas, la simulación *off-line* y la simulación *on-line*. La simulación *off-line* consiste en ejecutar la aplicación paralela en un sistema real con el objetivo de obtener su traza de ejecución, la cual representa el modelo de la aplicación, y será utilizada como entrada del simulador con el objetivo de simular la aplicación en diferentes sistemas. La alternativa a este tipo de simulación es la simulación *on-line*, la cual ejecuta la aplicación en un sistema anfitrión que intenta emular el comportamiento del sistema destino. Para ello, partes del código son interceptadas y enviadas al simulador con el objetivo de obtener la predicción en el sistema destino. Este tipo de simuladores necesitan un elevado número de recursos, ya que necesita los recursos requeridos por la aplicación más los requeridos por el simulador.

Debido a que los simuladores *off-line* son los más extendidos, en este apartado se describen los simuladores *off-line* más representativos en el área. Uno de los simuladores que ha adquirido mayor popularidad en los últimos años ha sido BigSim [34], debido a que permite predecir el rendimiento de aplicaciones paralelas de paso de mensajes

en supercomputadores, como por ejemplo el Blue-Gene/L, utilizando un conjunto limitado de recursos. BigSim proporciona un entorno de emulación y otro de simulación. El entorno de emulación es útil para estudiar el comportamiento de las aplicaciones paralelas, las cuales son ejecutadas sobre un gran número de cores, con el objetivo de detectar posibles cuellos de botella. Este entorno no proporciona información útil sobre el rendimiento de la aplicación en el sistema. En caso de necesitar predecir el rendimiento de la aplicación, se utiliza el entorno de simulación. Una vez ejecutada la aplicación en el entorno de emulación, se obtiene una traza por proceso, el conjunto de las cuales representa el modelo de la aplicación, y serán utilizadas como entrada del simulador con el objetivo de predecir su rendimiento. El simulador está basado en la simulación discreta de eventos y permite ser ejecutado en paralelo, con lo que se obtiene un tiempo de simulación aceptable. Aunque el simulador proporciona un nivel de predicción aceptable para cientos de procesos, a medida que se incrementa el número de procesos de la aplicación, el error de predicción va aumentando. Como trabajo futuro se está intentando mejorar la calidad de predicción mediante la incorporación de un simulador de instrucciones, ya que actualmente el simulador utiliza métodos heurísticos para predecir el tiempo secuencial de cómputo entre las primitivas de comunicación.

Otro simulador que ha adquirido gran protagonismo debido a su versatilidad es SimGrid [35]. Este simulador permite simular aplicaciones paralelas sobre un amplio conjunto de plataformas: *clusters, grid, cloud o peer-to-peer*. El simulador está compuesto por 4 módulos: MSG, SMPI, SIMDAG y SIMIX. El primer módulo permite describir mediante su propio lenguaje de programación el comportamiento de aplicaciones, ya sean reales o sintéticas, con el objetivo de simular la aplicación y predecir su rendimiento. El Módulo SMPI, permite trazar aplicaciones reales MPI, las cuales se están ejecutando en el sistema real, con el objetivo de generar un Log que contenga la traza de la aplicación, para posteriormente ser simulado en diferentes sistemas destino. El Módulo SIMDAG es el encargado de transformar las trazas generadas, tanto por el módulo MSG como por el SMPI, en un grafo de dependencias, el cual representa el modelo de la aplicación, con el objetivo de simular la aplicación correctamente. Finalmente, el módulo SIMIX es el encargado de llevar a cabo la simulación.

Al igual que en el resto de métodos de predicción, la simulación también se puede combinar con otros métodos de predicción con el objetivo de reducir el tiempo de simulación. El simulador LogGOPSim [36] permite simular el comportamiento de aplicaciones

paralelas de paso de mensajes y predecir su rendimiento a medida que se incrementa su número de procesos. Para realizar estas predicciones utiliza modelos de regresión matemáticos, generados a partir de un conjunto de puntos de entrada obtenidos mediante simulación. Este simulador se basa en el modelo LogP [37], uno de los más extendidos a la hora de simular la red de interconexión por su simplicidad, ya que ignora la congestión de red, siendo a la vez preciso. El simulador presenta resultados muy precisos con un reducido número de procesos. A medida que se incrementa el número de recursos va perdiendo precisión.

Otro simulador que combina simulación con métodos de medición con el fin de mejorar el proceso de simulación es FASE [38]. Este simulador fue diseñado con el objetivo de seleccionar el mejor sistema para ejecutar una aplicación paralela. FASE intenta mejorar la calidad de simulación mediante la caracterización del comportamiento de los parámetros críticos de rendimiento de la aplicación, los cuales son obtenidos mediante la ejecución de la aplicación en un sistema real. FASE se compone de dos módulos principales, el módulo de aplicación y el módulo de simulación. El módulo de aplicación caracteriza la aplicación mediante herramientas de rendimiento con el objetivo de modelizar su comportamiento y obtener una representación de su ejecución. El modelo de la aplicación creado previamente, se convierte en un conjunto de trazas que sirven de entrada al módulo de simulación. Este segundo módulo se utiliza para predecir el rendimiento de la aplicación en sistemas destino. Para ello, el módulo permite crear sistemas virtuales con los componentes que tendría el sistema real. Esta etapa puede llevar un tiempo considerable, dependiendo del nivel de complejidad con el que el sistema real se quiera modelizar, así como de la fidelidad con la cual se quiera representar.

Otra ventaja de los simuladores que merece la pena ser comentada, es que debido a la abstracción que presentan, permiten predecir el comportamiento de aplicaciones en sistemas propietarios no estándares, donde la mayoría de las herramientas de medición no están diseñadas para correr sobre este tipo de sistemas. Un ejemplo de ello es el simulador jitSim [39], el cual capaz de predecir la escalabilidad de aplicaciones paralelas en sistemas que contengan instalado el sistema operativo Jitter [40], propiedad de IBM y utilizado en sistemas HPC. Para lograr predecir la escalabilidad de las aplicaciones, al simulador le tiene que ser proporcionado la traza de la aplicación, obtenida en un sistema real que contenga el sistema operativo Jitter, la topología de la red en la cual se va a simular la aplicación, la arquitectura del sistema y el tiempo de latencia entre hops.

Una vez se le ha proporcionado toda la información necesaria, es capaz de predecir el comportamiento de la aplicación a medida que escala.

3.4. Métodos experimentales

Es el método de predicción más utilizado en entornos HPC, ya que ofrece una alta calidad de predicción al monitorizar el sistema real, cuando está sometido a una carga de trabajo, ya sea real (la propia aplicación) o sintética (*benchmarks*), con el objetivo de medir ciertos parámetros del sistema que ayuden a predecir el rendimiento de la aplicación.

En cuanto al uso de *benchmarks* para predecir el rendimiento de la aplicación, existen trabajos [41] [42] [43] centrados en la creación de *benchmarks* configurables, los cuales pueden ser parametrizados con el objetivo de representar el comportamiento de la aplicación. Presentan la ventaja que son códigos compactos, portables y fácilmente modificables, los cuales representan comportamientos genéricos de aplicaciones paralelas, representativos de campos de aplicación, enmascarando la complejidad de la aplicación paralela. Debido a la complejidad de las aplicaciones paralelas de paso de mensajes, hace que el uso de estos *benchmarks* no sea adecuado para predecir el rendimiento de la aplicación, ya que no representan el comportamiento de la aplicación con un alto nivel de precisión. Actualmente, es difícil encontrar un *benchmark* que represente todas las fases significativas de una aplicación paralela. El trabajo propuesto en esta tesis se diferencia de estas propuestas, ya que se basa en generar el modelo lógico de cada una de las fases relevantes de la aplicación, el cual representa el comportamiento de la fase. A partir del modelo lógico de cada fase, se genera la traza lógica escalable de la aplicación, la cual es equivalente al modelo de la aplicación, y será utilizada para predecir el rendimiento de la aplicación en el computador paralelo.

Debido a las ventajas que ofrece utilizar un modelo representativo del comportamiento de la propia aplicación para predecir su rendimiento en sistemas destino, existen una amplia variedad de métodos propuestos, los cuales han sido desarrollados como herramientas, con el objetivo de ser transparentes y automáticos tanto para usuarios como administradores de sistemas.

Una de las herramientas de predicción de rendimiento más extendida y utilizada debido a su larga trayectoria, lo que ha dado como resultado una herramienta fiable

y robusta, es Scalasca [44]. Esta herramienta permite analizar el comportamiento de aplicaciones paralelas de paso de mensajes de manera post-mortem, con el objetivo de identificar cuellos de botella potenciales, en particular los relacionados con la comunicación y sincronización. La herramienta ha sido especialmente diseñada para analizar el comportamiento de aplicaciones paralelas las cuales son ejecutadas en supercomputadores, como podrían ser el BlueGene o Cray, pero puede ser utilizada también en sistemas HPC de pequeño o mediano tamaño. Scalasca se basa en trazar la aplicación en tiempo de ejecución, con el objetivo de crear unas trazas con información de cómputo y comunicación, las cuales representan el modelo de la aplicación. Estas trazas serán posteriormente analizadas para detectar ineficiencias, las cuales pueden convertirse en cuellos de botella potenciales a medida que se aumenta el número de procesos de la aplicación. Nuestra metodología se diferencia de esta herramienta, ya que genera una traza compacta de la aplicación, la cual contiene únicamente las fases relevantes de la aplicación y representa el 95 % de su código. Debido al reducido tamaño de la traza, la cual representa el modelo de la aplicación paralela, la aplicación puede ser analizada de manera eficiente, centrándose únicamente en las fases representativas que tienen impacto en el rendimiento de la aplicación. Por otro lado, la metodología propuesta utiliza un conjunto limitado de recursos del sistema para predecir su rendimiento.

Un problema que cobra cada vez mayor importancia, debido a la llegada del exascale, es el tamaño del log de traza de la aplicación, el cual es obtenido en tiempo de ejecución, y será utilizado posteriormente tanto para analizar la aplicación como para predecir su rendimiento en sistemas destino. Existen trabajos [45] [46] [47] [48] los cuales han centrado una parte importante de sus esfuerzos en proponer técnicas de compresión, con el objetivo de reducir el tamaño del log de traza de la aplicación, el cual es utilizado para predecir el rendimiento de la misma en plataformas destino. La metodología P3S se diferencia de estas propuestas, ya que no es necesario aplicar técnicas de compresión a la hora de generar el log de traza. Esto es debido, a que la metodología P3S utiliza un conjunto de firmas de la aplicación, para un reducido número de procesos, para modelizar la aplicación y generar la traza física escalada, la cual será utilizada para predecir el rendimiento de la aplicación. La firma de la aplicación genera una traza únicamente de las fases relevantes de la aplicación. Además, contiene solamente los parámetros de la fase necesarios para modelizar su evolución a medida que la aplicación escala. Estos factores hacen que la traza de la firma posea un tamaño del orden de

Kilobytes. Por otra parte, la traza física escalada, generada para el número de procesos para el cual se desea predecir el rendimiento de la aplicación, a pesar de ser generada por proceso, al contener únicamente las fases relevantes de la aplicación, hace que pueda ser perfectamente asumible ser creada para miles de procesos, ya que el tamaño de la traza por proceso es del orden de Kilobytes.

Uno de los retos a afrontar a la hora de predecir el rendimiento de la aplicación paralela en sistemas destino, es simplificar su complejidad, manteniendo su comportamiento real. La herramienta ScalaBenchGen [49] propone crear automáticamente un *benchmark* de la aplicación, mediante el cual predecir su rendimiento. Dicho *benchmark*, al ser generado a partir de las trazas de ejecución de la aplicación, mantiene el comportamiento real de la aplicación, reflejando los patrones de comunicación y cómputo de la aplicación con un elevado nivel de precisión. Por otro lado, presenta la ventaja de ser un código compacto, portable y fácilmente entendible para el usuario.

Otra propuesta similar es la herramienta FACT [50]. Dicha herramienta genera el *Program Slicing* de la aplicación paralela, el cual representa el modelo de la aplicación paralela. El *Program Slicing* está basado en reducir la aplicación a su mínima expresión, conservando las propiedades clave de la aplicación, fundamentales para predecir su rendimiento.

Siguiendo con la línea de simplificación de la complejidad de aplicaciones paralelas manteniendo su comportamiento real, Xu et al [51] propone predecir el rendimiento de la aplicación paralela mediante el uso de *Skeletons*, los cuales son programas de corta duración, que representan el comportamiento de la aplicación, generados automáticamente a partir de las trazas de la aplicación obtenidas en un sistema base. Mediante la ejecución del *Skeleton* de la aplicación, es posible predecir de manera rápida el rendimiento de la aplicación en sistemas destinos.

El trabajo propuesto en esta tesis se diferencia de estas propuestas, ya que para predecir el rendimiento de la aplicación, no es necesario trazar la aplicación para el número de procesos a predecir, sino que a partir de la ejecución de un conjunto de firmas para un número reducido de procesos, será posible proyectar el comportamiento de la aplicación para generar su modelo lógico para un número mayor de procesos, mediante el cual predecir el rendimiento de la aplicación. Por otro lado, para predecir el rendimiento de la aplicación, no necesitamos el mismo número de recursos que requeriría la aplicación para ser ejecutada.

En muchos casos, a la hora de predecir el rendimiento de aplicaciones paralelas no se tiene acceso a todos los nodos que se utilizan en producción, únicamente a un conjunto de nodos de test destinados a ese propósito, por lo que muchas herramientas de predicción combinan la medición con algún otro método para obtener el rendimiento de la aplicación a medida que se incrementa su número de procesos. Un ejemplo de ello es el trabajo propuesto por Calotoiu [52], el cual propone una herramienta de predicción mediante la cual detectar posibles *bugs* de escalabilidad, los cuales se producen al aumentar el número de procesos de la aplicación. Para ello, la herramienta realiza un conjunto de ejecuciones de la aplicación, aumentando el número de procesos, mediante las cuales identifica los *kernels* de la aplicación (los segmentos más representativos de código de la aplicación), los cuales determinan el rendimiento de la aplicación. Identificado el conjunto de *kernels*, estos son modelizados con el objetivo de poder predecir su rendimiento. Una vez se ha generado el modelo se comprueba su precisión, en caso de no ser la requerida por el usuario, se realiza un proceso iterativo en el cual se va refinando el modelo. Finalmente, se aplican métodos de regresión para cada uno de los *kernels*, con el objetivo de predecir su rendimiento e identificar los cuellos de botella a medida que la aplicación escala. Este trabajo es similar a nuestra metodología, ya que nosotros también identificamos las fases relevantes de la aplicación, las cuales son modelizadas con el objetivo de predecir su rendimiento, y posteriormente aplicamos métodos de regresión para predecir el volumen de comunicación y el tiempo de cómputo de las fases relevantes a medida que el número de procesos aumenta. Entre otras diferencias, destacamos que nuestra metodología utiliza el sistema real para predecir el tiempo comunicación de cada fase, utilizando un conjunto reducido de recursos del sistema, sin necesidad de aplicar métodos de regresión.

Otra herramienta propuesta para predecir el rendimiento de aplicaciones paralelas en sistemas donde no se tiene acceso a los nodos de cómputo es SWAPP [53]. Esta herramienta asume que el sistema destino no está disponible para ejecutar aplicaciones, hasta que no se tiene la certeza que se hará un uso eficiente de los recursos, únicamente es posible ejecutar *benchmarks* con la finalidad de predecir el rendimiento que tendrá la aplicación en el sistema. Esta herramienta se basa en realizar proyecciones entre la aplicación, la cual ha sido ejecutada en un sistema base, y los resultados obtenidos de la ejecución de los *benchmarks* en el sistema destino. Con el objetivo de ser más precisos y simplificar el modelo, la herramienta modeliza el cómputo y la comunicación

por separado para realizar la proyección. Este trabajo se diferencia del propuesto en esta tesis, ya que en nuestro caso si tenemos acceso a un conjunto limitado de nodos del sistema, en los cuales podemos medir parámetros de la aplicación para predecir su rendimiento. Por otro lado, la metodología P3S no ejecuta *benchmarks*, sino que utiliza las fases relevantes de la propia aplicación para predecir su rendimiento a medida que escala.

Otras aproximaciones muy utilizadas son combinar la medición con la simulación. Un ejemplo de ello es la herramienta PHANTOM, propuesta por Wenguang [54], la cual puede ser utilizada para predecir el rendimiento de aplicaciones paralelas de paso de mensajes, suponiendo que sólo se tiene acceso de 1 nodo del sistema, en el cual se ejecutarán procesos representativos de la aplicación, en lugar de ejecutar la aplicación completa, con el objetivo de medir el tiempo de cómputo. Finalmente, una vez se ha predicho el tiempo de cómputo de los procesos representativos de la aplicación, se realiza una simulación de las comunicaciones, mediante el uso de un simulador de aplicaciones de paso de mensajes, con el fin de predecir el rendimiento de la aplicación.

Otro framework [55] para predecir el rendimiento que combina medición con simulación es el desarrollado en el PERC (Performance Evaluation Research Center), en colaboración con la UPC (Universidad Politécnica de Catalunya) y el TACC (Texas Advanced Computing Center). Este framework es capaz de modelizar y predecir el comportamiento de aplicaciones paralelas y, predecir su escalabilidad. El framework está compuesto por el conjunto de herramientas: MAPS [56], MetaSim Tracer [57] y DIMEMAS [58]. Gracias a este conjunto de herramientas, es capaz de predecir tanto la escalabilidad fuerte como débil de aplicaciones paralelas. Para lograr predecir la escalabilidad, se basa en conocer tanto el tiempo de cómputo, como los accesos a cada nivel de la jerarquía de memoria que realiza la aplicación en un sistema determinado. Por otra parte, también se requiere modelizar la red de interconexión del sistema destino para el cual se desea predecir el rendimiento de la aplicación, con el objetivo de simular el tiempo de comunicación. Para lograr realizar esas tareas, el framework utiliza la herramienta MetaSim para obtener un conjunto de firmas de la aplicación. La firma contiene los accesos a cada nivel de la jerarquía de memoria. A partir de este conjunto de firmas, se obtiene el comportamiento de los accesos a memoria a medida que la aplicación escala. Mediante el análisis y modelizado de este conjunto de datos se obtienen las ecuaciones de predicción del tiempo de acceso a memoria y del tiempo

de cómputo, las cuales serán utilizadas para predecir el rendimiento de la aplicación a medida que la aplicación escala. Finalmente, para predecir el tiempo de comunicación se utiliza el simulador DIMEMAS, el cual ha sido configurado con el modelo de la red de interconexión del sistema destino.

Este tipo de herramientas se diferencia de nuestra propuesta, ya que la metodología P3S no utiliza simuladores para predecir el tiempo de comunicación, sino que se realiza una ejecución por partes de la traza física escalada, sobre un conjunto reducido de nodos del sistema para el cual se desea obtener la predicción del rendimiento de la aplicación, utilizando la propia red de interconexión.

A lo largo de este capítulo se han presentado los principales trabajos relacionados en el área de predicción del rendimiento de aplicaciones paralelas, los cuales se han dividido en tres grandes áreas. Los trabajos presentados tienen su propios objetivos, con el fin de cubrir unos desafíos concretos. La metodología P3S ha sido desarrollada con el objetivo de cubrir los retos que no abarcaban los trabajos propuestos.

4

Metodología P3S

4.1. Introducción

En este capítulo se presenta la metodología *Prediction of Parallel Program Scalability (P3S)*, la cual permite predecir el comportamiento de la escalabilidad fuerte de aplicaciones paralelas de paso de mensajes en un determinado sistema, tanto en sistemas HPC, como HPC sobre Cloud, de una manera rápida y eficiente, utilizando un conjunto reducido de recursos.

La metodología P3S, tal y como se muestra en la figura 4.1, donde se presenta una descripción general de la metodología, está dividida en tres etapas. Una primera etapa de caracterización, donde se caracteriza la aplicación a partir de la información obtenida de la ejecución de un conjunto reducido de firmas de la aplicación (S_j), obtenidas con la herramienta PAS2P, con diferente número de procesos (j <Número de procesos a predecir). Una segunda etapa de modelización de la aplicación, donde se genera el modelo lógico de la aplicación paralela a partir de la caracterización previa y, finalmente, una tercera etapa de predicción del rendimiento, donde a partir del modelo lógico de la aplicación, se predice el tiempo de cómputo y comunicación para un número N de procesos, con el objetivo de predecir el comportamiento de la escalabilidad de la aplicación en el sistema utilizado.

La metodología P3S, se basa en analizar el comportamiento repetitivo de las aplicaciones paralelas de paso de mensajes. Este tipo de aplicaciones están compuestas por un conjunto de fases identificables, las cuales se van repitiendo a lo largo de toda la aplicación [6], tal y como se muestra en la figura 4.2, donde se presenta un fragmento

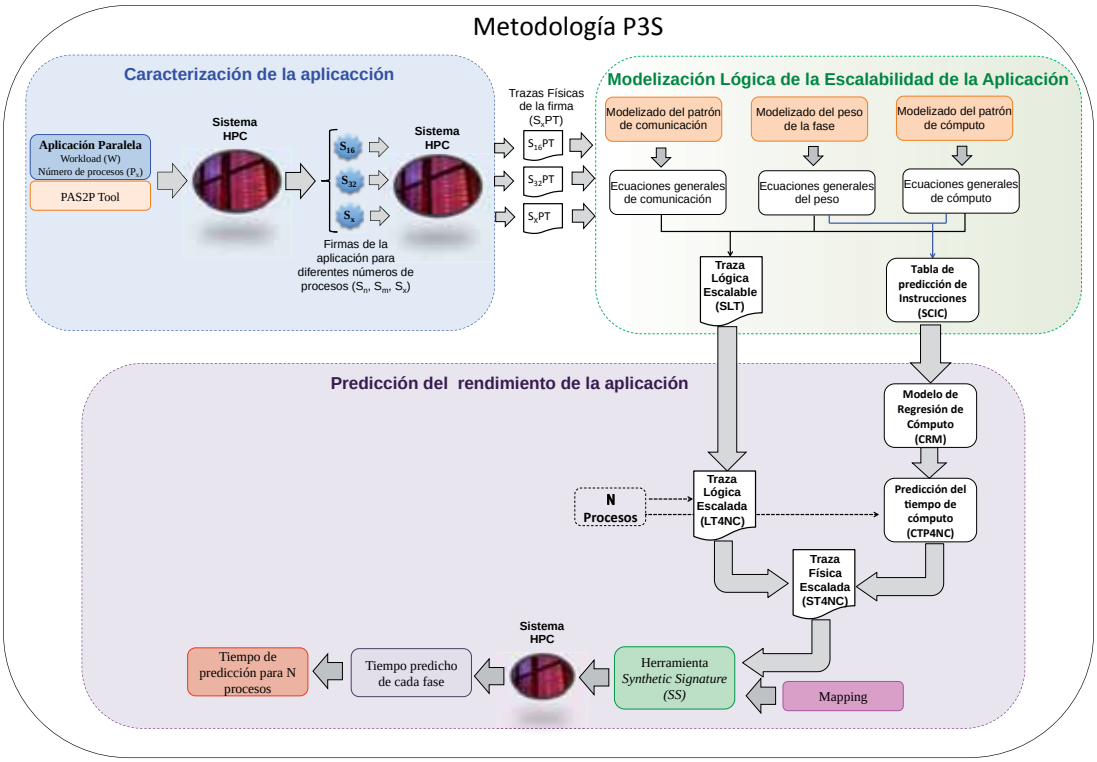


Figura 4.1: Visión global de la metodología P3S

de la aplicación científica de paso de mensajes NPB CG [59]. Como se puede apreciar en la figura, encontramos dos fases claramente diferenciadas, una de las cuales (fase 2) se va repitiendo a lo largo del tiempo de ejecución de la aplicación.

Las diferentes fases de las aplicaciones paralelas corresponden a algoritmos escritos por los desarrolladores utilizando patrones específicos de comunicación y cómputo, los cuales siguen unas determinadas reglas de comportamiento a medida que su número de procesos aumenta. Analizando el comportamiento de la aplicación, cuando se va incrementando el número de procesos, el número total de fases de la aplicación permanece constante, pero los patrones cambian su comportamiento siguiendo estas reglas de comportamiento, siendo las fases funcionalmente constantes, es decir, realizan el mismo trabajo, pero repartido entre un número diferente de procesos, puesto que son los mismos segmentos de código de la aplicación, escalados para un número mayor de procesos.

Para obtener los patrones de las fases de una forma transparente y automática, la metodología P3S utiliza la herramienta PAS2P [8]. Dicha herramienta identifica las

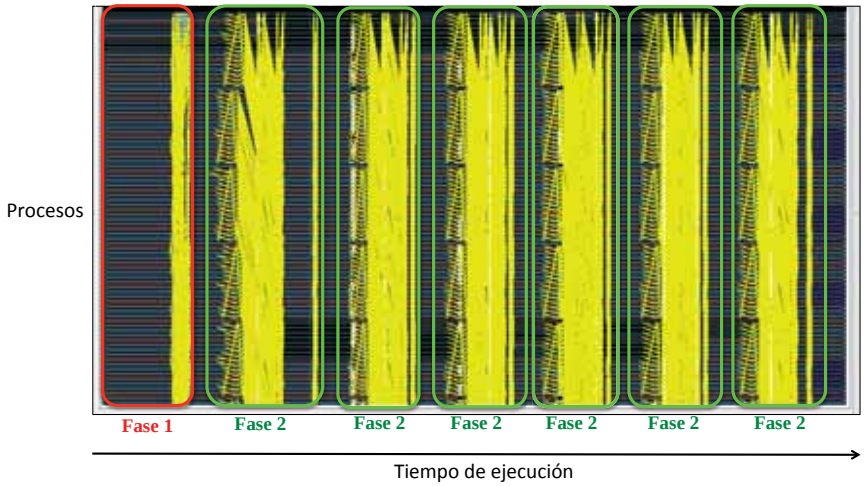


Figura 4.2: Fragmento de la aplicación científica de paso de mensajes NPB CG para 256 procesos, donde se muestra visualmente la repetitividad de las fases de la aplicación

fases relevantes de la aplicación y nos permite crear la firma de la aplicación. Para este trabajo hemos considerado que una fase es relevante, si su tiempo total de ejecución (todas las veces que la fase se repite durante la ejecución de la aplicación) es mayor al 1 % del tiempo de ejecución de la aplicación.

Tal y como se muestra en la figura 4.3, la firma de la aplicación contiene únicamente las fases relevantes de la aplicación, es decir, segmentos de código ejecutables que tienen impacto en el rendimiento de la aplicación, y su frecuencia de repetición (peso). La ejecución de la firma nos permite caracterizar y analizar la aplicación de manera eficiente, ya que nos permite centrarnos en analizar únicamente las fases relevantes de la aplicación, cada una de las cuales puede tener un comportamiento diferente, cubriendo aproximadamente un 95 % del código de la aplicación, en el 1 % del tiempo de ejecución de la aplicación, ya que únicamente ejecutamos sus fases relevantes un número M estadísticamente significativo de veces ($M << wi$), siendo wi el peso de la fase.

En el anexo A de este trabajo se encuentra detallado el funcionamiento de la herramienta PAS2P, así como los módulos de la cual se compone. Además, se presentan las nuevas funcionalidades que se han añadido a la herramienta con el objetivo de adaptarla a las necesidades de la metodología P3S.

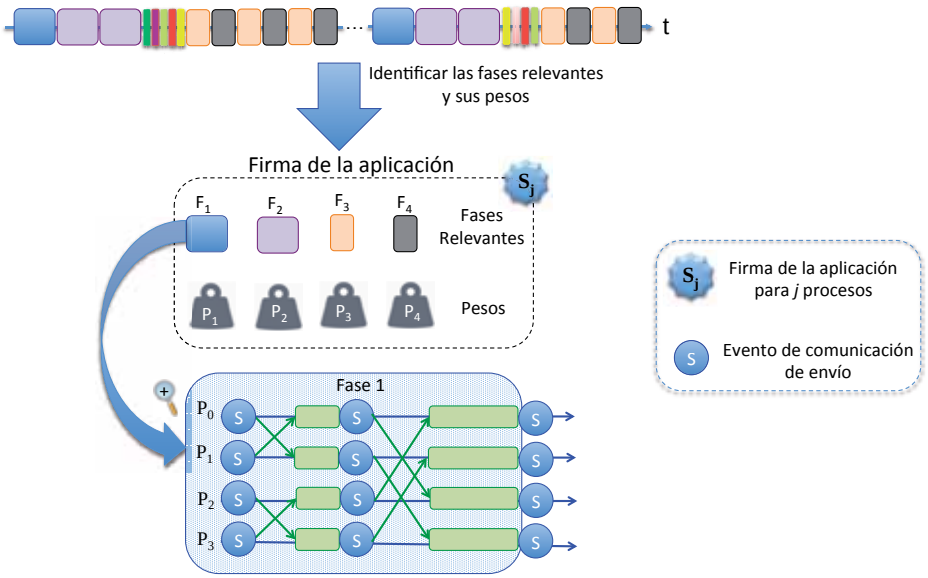


Figura 4.3: Generación de la firma de la aplicación a partir de las fases relevantes de la aplicación

4.2. Caracterización de la aplicación

Tal y como se muestra en la figura 4.1, la primera etapa de la metodología P3S consiste en caracterizar y analizar los patrones de comunicación, cómputo y peso para cada fase relevante de la aplicación de forma transparente, es decir, sin modificar el código fuente de la aplicación. La metodología asume que el usuario o administrador del sistema no tiene porque disponer del código fuente de la aplicación, únicamente del binario compilado en el sistema destino. Con el objetivo de caracterizar los patrones de cada fase relevante de la aplicación, se ejecuta en el sistema paralelo un conjunto de firmas de la aplicación para un número diferente y reducido de procesos. Mediante la información obtenida a partir de la ejecución de este conjunto de firmas, será posible obtener información rápida sobre el comportamiento de cada fase relevante de la aplicación, a medida que la aplicación escala.

4.3. Modelizado lógico de la aplicación

Una vez caracterizados y analizados los patrones de cada fase relevante de la aplicación paralela, se realiza la segunda etapa de la metodología, la cual consiste en modelizar

los patrones de comportamiento para cada fase relevante de la aplicación, con el objetivo de obtener sus ecuaciones generales de comportamiento, las cuales nos permitirán proyectar la estructura de comunicación y cómputo de cada fase relevante a medida que se incrementa el número de procesos de la aplicación. Durante esta etapa se modeliza el patrón de comunicación, el patrón de cómputo y el comportamiento del peso para cada fase de la aplicación.

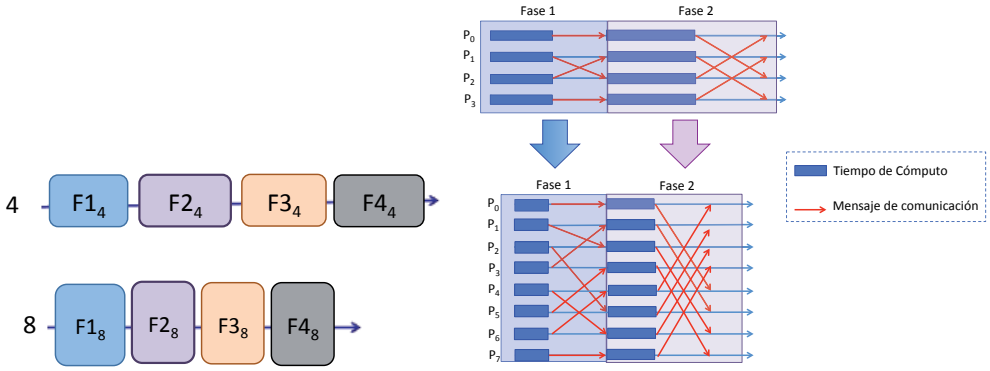
Actualmente, la metodología está enfocada a aplicaciones paralelas de paso de mensajes con patrones regulares, las cuales han sido desarrolladas utilizando patrones que siguen reglas de comportamiento a medida que la aplicación escala. Para aplicaciones con patrones irregulares, los cuales a partir de un cierto número de procesos cambian sus reglas de comportamiento, presentando reglas totalmente diferentes, requiere realizar una modificación de la metodología actual para identificar las nuevas reglas de comportamiento.

A fin de modelizar las ecuaciones generales de comportamiento de cada fase, se busca la relación por similitud funcional entre las diferentes ejecuciones de la firma.

Consideramos que dos fases de una misma aplicación, para diferente número de procesos, tendrán similitud funcional, cuando las dos fases realicen el mismo trabajo (cómputo), distribuido entre diferente número de procesos, cambiando únicamente la estructura del patrón de comunicación de la fase (origen, destino y volumen de comunicación), es decir, las dos fases realicen la misma funcionalidad pero utilizando un número distinto de procesos y repartiendo la carga de trabajo entre ellos de distinta forma.

Se utiliza similitud funcional para identificar las fases similares en firmas con distinto número de procesos, ya que cuando el número de procesos de la aplicación aumenta, las fases cambian su comportamiento y se hace necesario reconocerlas y relacionarlas para poder analizarlas y modelizarlas. Analizando el comportamiento de las fases a medida que se incrementa el número de procesos de la aplicación, sabemos que su comunicación (número de mensajes y destino), el volumen de comunicación y el tiempo de cómputo pueden variar, pero el trabajo total realizado por la fase (cómputo) será el mismo, repartido entre un número mayor de procesos, ya que es el mismo segmento de código y estamos trabajando en escalabilidad fuerte, donde el *workload* de la aplicación se mantiene constante a medida que se aumenta el número de procesos de la aplicación.

Para relacionar las fases, se utiliza un algoritmo basado en conocer la secuencia de fases en tiempo de ejecución, ya que no depende del número de procesos, sino de la



(a) Secuencia lógica de la firma por fases para 4 y 8 procesos

(b) Comportamiento de las fases 1 y 2 para 4 y 8 procesos

Figura 4.4: Relación de fases por similitud funcional

secuencia lógica en la cual la aplicación ha sido desarrollada. La figura 4.4 muestra un ejemplo del procedimiento. Como podemos ver en la figura 4.4.a, si ejecutamos la firma de una aplicación para 4 y 8 procesos, el número de fases de la aplicación se mantiene constante, ya que es la secuencia lógica de código escrita en la aplicación, independientemente del número de procesos para el cual haya sido ejecutada. Si ahora nos centramos en la figura 4.4.b, donde se muestra en detalle la fase 1 y la fase 2 de la aplicación para 4 y 8 procesos, vemos que aunque las fases poseen diferente comportamiento entre ellas, ya que tienen diferente patrón de comunicación y diferente tiempo de cómputo, se trata de las mismas fases, ya que se encuentran situadas en la misma posición lógica en tiempo de ejecución, la cuales han escalado de 4 a 8 procesos siguiendo sus reglas generales de comportamiento, por lo tanto, están realizando el mismo trabajo tanto para 4 como para 8 procesos, repartido entre diferente número de procesos, ya que ambas corresponden al mismo bloque de código.

Una vez se han relacionado las fases relevantes entre las diferentes ejecuciones de la firma y se han modelizado sus ecuaciones generales de comportamiento, tal y como veremos en el siguiente capítulo, se genera la traza lógica escalable de la aplicación *(Scalable Logical Trace (SLT))* y la tabla de predicción de instrucciones *(Scalable Computational Instruction Count (SCIC))* .

La traza lógica escalable (SLT) contiene las ecuaciones generales de comportamiento de cada fase de la aplicación, es decir, las ecuaciones generales que describen el patrón

de comunicación, patrón de cómputo y el patrón del peso de la fase. Dicha traza es independiente del sistema paralelo para el cual se desea obtener la predicción, sólo depende de la forma en que la propia aplicación ha sido desarrollada, es decir, únicamente posee las ecuaciones generales con parámetros intrínsecos de la aplicación. La SLT se genera por proceso en lugar de generar una traza global con el objetivo de poder tratar cada proceso de forma independiente.

En cuanto a la tabla de predicción de instrucciones (SCIC), se genera a partir de las ecuaciones generales de cómputo y peso de cada fase de la aplicación, y contiene una visión global del número total de instrucciones y el número total de instrucciones por proceso para cada fase de la aplicación, a medida que la aplicación escala. La SCIC será utilizada para predecir el tiempo de cómputo de cada fase de la aplicación.

4.4. Predicción del rendimiento de la aplicación

Una vez que tanto la SLT como la SCIC han sido generadas, el último paso de la metodología es predecir como escala la aplicación cuando va aumentando su número de procesos. Para ello, es necesario predecir el tiempo de ejecución (rendimiento), para un número específico de procesos. Con el objetivo de obtener el rendimiento de la aplicación para un número específico de procesos, la SLT, la cual posee como parámetro de entrada el número de procesos para el cual se desea predecir el rendimiento de la aplicación, se desarrolla para un número N de procesos, obteniendo una nueva traza lógica, denominada traza lógica escalada *(Logical Trace for N cores (LT4NC))*, la cual contiene el comportamiento de comunicación, cómputo y peso de cada fase, para ese

Peso Fase 1: 2,800

Proceso	Fase	Tipo de primitiva	Origen	Destino	Volumen de comunicación (Bytes)	Cómputo (Número de instrucciones)
0	1	MPI_Irecv	0	1	4,000	756
0	1	MPI_Send	0	1	4,000	456
0	1	MPI_Wait	0	1	4,000	456,746,733
0	1	MPI_Irecv	0	2	2,000	975
0	1	MPI_Send	0	2	2,000	875
0	1	MPI_Wait	0	2	2,000	357,876,543

Figura 4.5: Traza Lógica Escalada (LT4NC) para la fase 1 del proceso 0

Peso Fase 1: 2,800

Proceso	Fase	Tipo de primitiva	Origen	Destino	Volumen de comunicación (Bytes)	Número de instrucciones de cómputo	Tiempo de cómputo (ns)
0	1	MPI_Irecv	0	1	4,000	756	4,000
0	1	MPI_Send	0	1	4,000	456	2,345
0	1	MPI_Wait	0	1	4,000	456,746,733	83,593,535
0	1	MPI_Irecv	0	2	2,000	975	7,533
0	1	MPI_Send	0	2	2,000	875	5,366
0	1	MPI_Wait	0	2	2,000	357,876,543	45,326,854

Figura 4.6: Traza física Escalada (ST4NP) para la fase 1 del proceso 0

número concreto de procesos. Para ello, las ecuaciones generales de comportamiento de la SLT, han sido concretadas para el número de procesos para el cual se desea predecir el rendimiento de la aplicación. La figura 4.5 muestra un ejemplo de traza lógica escalada (LT4NC) para la fase relevante 1 de una aplicación paralela. Como se puede apreciar en la figura, la LT4NC contiene información de cada fase describiendo: el tipo de evento de comunicación (tipo de primitiva MPI), el origen y destino de cada evento de comunicación, el volumen de comunicación, el número de instrucciones de cómputo entre dos eventos consecutivos de comunicación y el peso de la fase.

La metodología hace especial hincapié en generar una traza lógica reducida, con el objetivo de poder ser manejada en memoria evitando los accesos al disco. La LT4NC contiene únicamente el conjunto mínimo de parámetros intrínsecos necesarios de cada fase, utilizados posteriormente para predecir el tiempo de comunicación y cómputo.

Generada la LT4NC, considerando el modelo de la aplicación (LT4NC y SCIC), a partir de la SCIC, se predice el tiempo de cómputo para cada fase de la aplicación, con el objetivo de generar la traza física escalada para el número de procesos a predecir *(Scaled Trace for N processes (ST4NP))*. Tal y como se muestra en la figura 4.6, esta nueva traza es dependiente de la máquina, ya que además de toda la información lógica que contenía la LT4NC, contiene los tiempos físicos predichos de cómputo para cada fase de la aplicación. Mediante esta traza física, será posible predecir el rendimiento de la aplicación.

Con el objetivo de predecir el tiempo de cómputo de cada fase, se ha propuesto un método basado en modelos matemáticos de regresión, llamado *Computational Regression Model (CRM)*, el cual usa como datos de entrada para generar el modelo, los

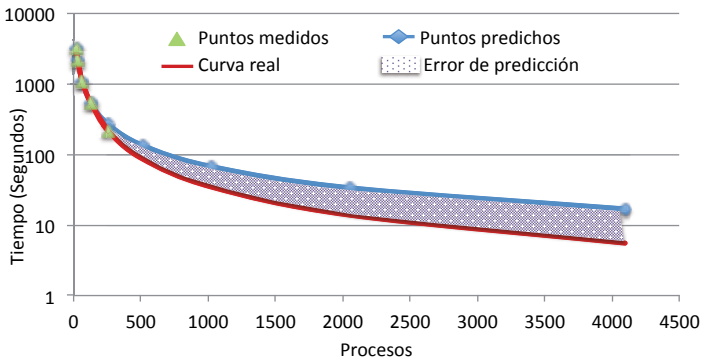


Figura 4.7: Error introducido en los modelos de regresión a medida que nos alejamos de los puntos reales

tiempos de cómputo de cada fase del conjunto inicial de firmas ejecutadas.

Debido a la simplificación de los modelos matemáticos de regresión para poder ser útiles y aplicables en la realidad, estos modelos están limitados en el alcance de la predicción, tal y como se muestra en la figura 4.7, obteniendo un alto error de predicción a medida que nos alejamos de los puntos medidos utilizados para generar el modelo. Este factor limitante, hace que estos modelos no sean adecuados para predecir puntos lejanos, resultando útiles únicamente para predecir puntos cercanos a los utilizados para generar el modelo de regresión.

Con el objetivo de mejorar el alcance de predicción del modelo de cómputo sin aumentar su complejidad, se propone un método mediante el cual medir un punto lejano, sin necesidad de ejecutar la aplicación para un elevado número de procesos, con el fin de ajustar el modelo de regresión de cómputo (CRM), añadiendo este nuevo punto al conjunto inicial de puntos de entrada para generar el modelo de regresión.

El método propuesto está basado en el principio de escalabilidad fuerte, donde el *workload* de la aplicación se mantiene constante y el trabajo por proceso va disminuyendo, distribuyéndose entre un número mayor de procesos, a medida que la aplicación escala.

Con el fin de obtener el punto lejano mediante el cual ajustar el modelo, se realiza un cambio de dominio en el *workload* de la aplicación, utilizando un *workload* menor que el *workload* original, y que sea equivalente al trabajo por proceso que le tocará realizar. Este nuevo *workload* será ejecutado sobre un reducido número de procesos, con

el objetivo de emular el tiempo de cómputo para el *workload* original ejecutado sobre un elevado número de procesos, ya que al ir escalando la aplicación con el *workload* original, el trabajo por proceso irá disminuyendo progresivamente. De esta forma, es posible incorporar medidas de puntos distantes sin utilizar un elevado número de procesos.

Utilizando este método, somos capaces de mejorar la calidad de predicción del CRM, ya que podemos ajustarlo introduciendo en el modelo inicial este nuevo punto lejano, obteniendo una nueva función de regresión, mediante la cual será posible predecir puntos distantes con una alta calidad de predicción.

Este método está limitado a aplicaciones paralelas cuyas fases mantienen constante el número de instrucciones a medida que la aplicación escala. Con el objetivo de validar esta restricción del modelo, se hace uso de la tabla de predicción de instrucciones *Scalable Computational Instruction Count (SCIC)*, la cual nos da información global del número total de instrucciones de cada fase a medida que la aplicación escala.

Una vez que se ha generado el CRM para cada fase de la aplicación, se genera una tabla específica con los tiempos de cómputo predichos de cada fase, para el número de procesos para el cual se desea predecir el rendimiento de la aplicación, denominada, Tabla de Predicción de Cómputo para N Cores *(Computational time for N cores (CT4NC))*. Mediante esta tabla y la LT4NC se genera la traza física escalada (ST4NP) para el número de procesos de la aplicación para el cual se desea predecir su rendimiento. Como se comentó anteriormente, esta traza es dependiente de la máquina, ya que además de toda la información de la LT4NC, contiene el tiempo de cómputo de cada fase, tal y como se muestra en la figura 4.6.

Finalmente, el último paso de la metodología antes de obtener el rendimiento de la aplicación, es procesar la ST4NP en un conjunto reducido de nodos (recursos) del sistema destino, para obtener los tiempos de comunicación de cada fase de la aplicación. Puesto que conocemos el origen y destino de los mensajes, sus volúmenes de comunicación y el tiempo de cómputo entre primitivas, el objetivo es medir el tiempo de comunicación de los mensajes de cada fase. Para ello, se ha diseñado una herramienta, llamada *Synthetic Signature (SS)*, la cual recibe como entrada la ST4NP. Una vez leída la traza, la herramienta SS procesará los eventos de comunicación y cómputo en el sistema paralelo con el objetivo de obtener el tiempo de comunicación de cada fase, a fin de de obtener la predicción del rendimiento de la aplicación.

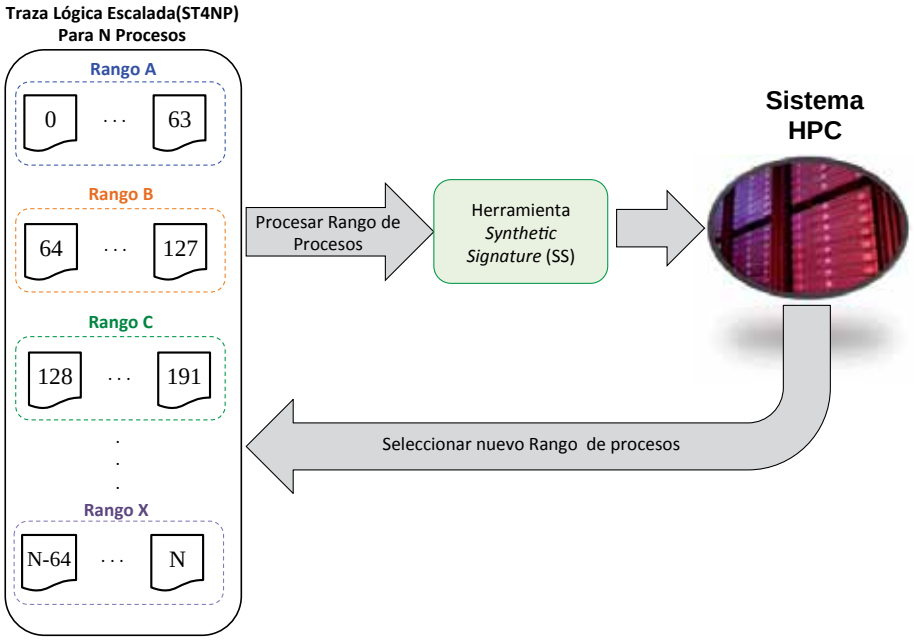


Figura 4.8: Procesado de la traza física escalada (ST4NP) para N procesos en el sistema paralelo

Aunque la ST4NP contiene el tiempo predicho de cómputo de cada fase, y únicamente se requiere conocer su tiempo de comunicación para obtener el tiempo de ejecución predicho de la fase, además de los eventos de comunicación, la herramienta SS también considera el tiempo de cómputo de cada fase, con el objetivo de mantener la relación cómputo-comunicación, ya que si sólo ejecutáramos los eventos de comunicación, no estaríamos representando el comportamiento de la fase tal y como sucede en aplicación real, afectando a los tiempos de envío y recepción de los eventos de comunicación.

Puesto que uno de los requisitos de la metodología es utilizar un número de recursos limitado del sistema, la herramienta SS procesará la ST4NP en el sistema por partes, utilizando un conjunto limitado y reducido de nodos del sistema. Para ello, se realiza un proceso iterativo conducido por traza, donde la ST4NP es procesada en el sistema destino por rangos de procesos, tal y como se muestra en la figura 4.8. El tamaño del rango de procesos se define en función del número de cores de los nodos del sistema. Debido a que ejecutamos rangos de procesos, este proceso se repite de forma iterativa hasta que todos los procesos que componen la ST4NP han medido sus eventos de comunicación.

Para obtener los tiempos de comunicación correctos, es necesario realizar una etapa

de caracterización previa de la red de interconexión del sistema, a fin de poder mapear los procesos a medir, manteniendo las distancias físicas de comunicación entre los procesos, tal y como se ejecutaría en la aplicación real utilizando todos los recursos del sistema.

Una vez que todos los procesos han sido medidos, obtendremos el tiempo predicho para cada fase relevante de la aplicación, ya que disponemos del tiempo de cómputo y comunicación de cada fase. Con el objetivo de obtener el tiempo de predicción de la aplicación, el tiempo de ejecución predicho de cada fase será multiplicado por su peso predicho (el número de veces que se repite la fase), dando como resultado el tiempo predicho total de la aplicación.

A lo largo de los siguientes capítulos de este trabajo, se presentan en detalle cada una de las diferentes etapas de las cuales consta la metodología propuesta. El capítulo 5 presentará en detalle el modelo propuesto en la primera etapa de la metodología, para modelizar el comportamiento de escalabilidad de cada fase relevante de la aplicación paralela. El capítulo 6 presentará en detalle el modelo de predicción del tiempo de cómputo, utilizado en la segunda etapa de la metodología para predecir el tiempo de cómputo de cada fase de la aplicación para N procesos. El capítulo 7 presentará en detalle la herramienta *Synthetic Signature (SS)*, utilizada en la tercera etapa de la metodología, para predecir el tiempo de comunicación de cada fase de la aplicación, y finalmente, el capítulo 8 presentará en la última etapa de la metodología, consistente en obtener el rendimiento de la aplicación.

5

Modelo lógico de aplicaciones paralelas para el análisis y predicción de la escalabilidad

5.1. Introducción

A lo largo de este capítulo, se presentan las etapas la metodología P3S *(Prediction of Parallel Program Scalability)* relacionadas con la caracterización y modelización del comportamiento de la escalabilidad de la aplicación paralela de paso de mensajes.

Además, se presenta la validación experimental del modelo lógico de la aplicación.

5.2. Caracterización de la aplicación

Esta etapa consiste en caracterizar las fases relevantes de la aplicación, con el objetivo de obtener información para modelizar su comportamiento, a medida que la aplicación escala. El objetivo de esta etapa es recolectar información acerca del patrón de comunicación, el patrón de cómputo y el comportamiento del peso de cada fase relevante de la aplicación.

Para llevar a cabo esta caracterización, tal y como se muestra en la figura 5.1, se utiliza la herramienta PAS2P [8], la cual nos permite obtener las fases relevantes de la aplicación de una manera transparente y totalmente automática. Para este trabajo se ha considerado que una fase es relevante, cuando su tiempo total de ejecución (todas

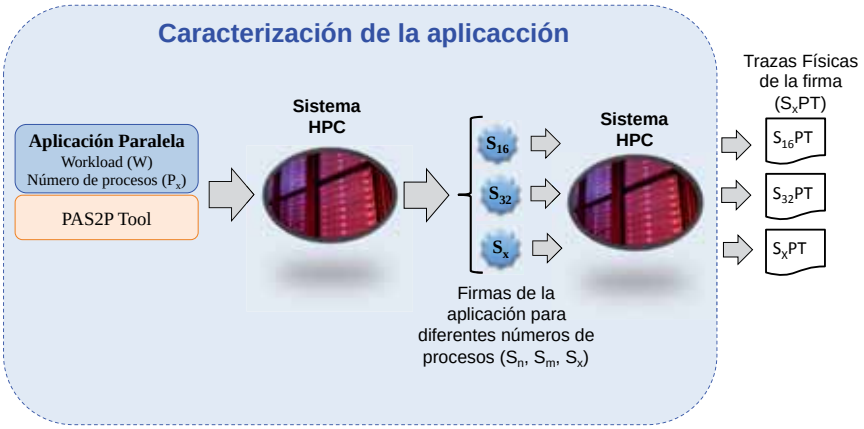


Figura 5.1: Caracterización de las fases de la aplicación

las veces que la fase se repite durante la ejecución de la aplicación) es mayor al 1 % del tiempo total de ejecución de la aplicación.

Una vez que PAS2P ha identificado las fases relevantes de la aplicación, es decir, segmentos de código ejecutables que tienen impacto en el rendimiento de la aplicación, se crea la firma de la aplicación, la cual contiene las fases relevantes y su frecuencia de repetición (peso). Puesto que la firma únicamente contiene la fases relevantes de la aplicación, puede ser ejecutada en el sistema en un tiempo acotado aproximado del 1 % del tiempo total de ejecución de la aplicación, permitiéndonos caracterizar y analizar la aplicación de manera eficiente, ya que nos podemos centrar en analizar únicamente las fases relevantes de la aplicación, las cuales cubren aproximadamente un 95 % del código de la aplicación.

En el anexo A de este trabajo se encuentra detallado el funcionamiento de la herramienta PAS2P, así como los módulos de la cual se compone. Además, se presentan las nuevas funcionalidades que se han añadido a la herramienta con el objetivo adaptarla a las necesidades de la metodología P3S.

La ejecución de la firma nos permite extraer información del comportamiento de cada una de las fases relevantes de la aplicación. Esta información es guardada en una traza por proceso, tal y como se muestra en la figura 5.2, denominada, traza física de la firma *(Signature Physical Trace (S_iPT))*. Como se puede observar en la misma figura, la S_iPT proporciona información del ID de la fase, el tipo de primitiva MPI, el origen y destino de cada mensaje, el volumen de comunicación en bytes, el tiempo de comunicación de cada primitiva MPI, el tiempo de cómputo entre cada primitiva MPI,

Peso Fase 1: 2,800

Fase ID	Tipo de primitiva	Origen	Destino	Volumen de comunicación (Bytes)	Tiempo de cómputo (ns)	Tiempo de comunicación (ns)	Número de instrucciones	Número de ciclos	Fallos de cache L2
1	MPI_Irecv	1	2	4,000	4,000	9	756	256	32
1	MPI_Send	1	2	4,000	2,345	12	456	156	13
1	MPI_Wait	1	2	4,000	83,593,535	989	357,812,845	9,871,583	3,178
1	MPI_Irecv	1	4	4,000	4,533	9	975	345	45
1	MPI_Send	1	4	4,000	2,366	14	875	256	65
1	MPI_Wait	1	4	4,000	83,598,394	983	357,876,543	9,876,567	3,466

Figura 5.2: Ejemplo de traza de la firma para el proceso 1

el número de instrucciones y ciclos del tiempo de cómputo, los fallos de caché del último nivel de la jerarquía de memoria y el peso de la fase.

Con el fin de obtener información de la evolución del comportamiento de cada fase relevante la aplicación, a medida que la aplicación escala, se lleva a cabo un conjunto de ejecuciones de la firma de la aplicación para diferentes números de procesos. El número de procesos que se selecciona para ejecutar las firmas es un número reducido, con el objetivo de poder ser ejecutadas en un conjunto reducido de nodos, normalmente entre 2 y 4 nodos.

La idea de ejecutar un conjunto de firmas para diferentes números de procesos, es comparar las fases relevantes de la aplicación, a medida que se aumenta el número de procesos de la aplicación, con el objetivo de obtener información de cómo va variando su patrón de comunicación, cómputo y peso de cada fase relevante.

A partir de la información obtenida de la ejecución de este conjunto de firmas de la aplicación, será posible modelizar el comportamiento de las fases de la aplicación a medida que la aplicación escala.

Con el objetivo de analizar computacionalmente de una forma eficiente las S_iPT's, obtenidas de la ejecución del conjunto de firmas de la aplicación, las S_iPT's son guardadas en una estructura de árbol, llamada, árbol de firmas *(Signature Tree (ST))*, la cual se muestra en la figura 5.3. Mediante la utilización de este árbol, será posible analizar la información por fase y número de procesos de forma eficiente.

A fin de reducir el tamaño del árbol en memoria, se define el concepto de grupo. Un grupo es una secuencia de primitivas que se repiten a lo largo de la S_iPT, y tienen un volumen de comunicación, número de instrucciones, tiempo de cómputo y tiempo de comunicación similar (mínimo un 98 % de similitud), únicamente se diferencian por

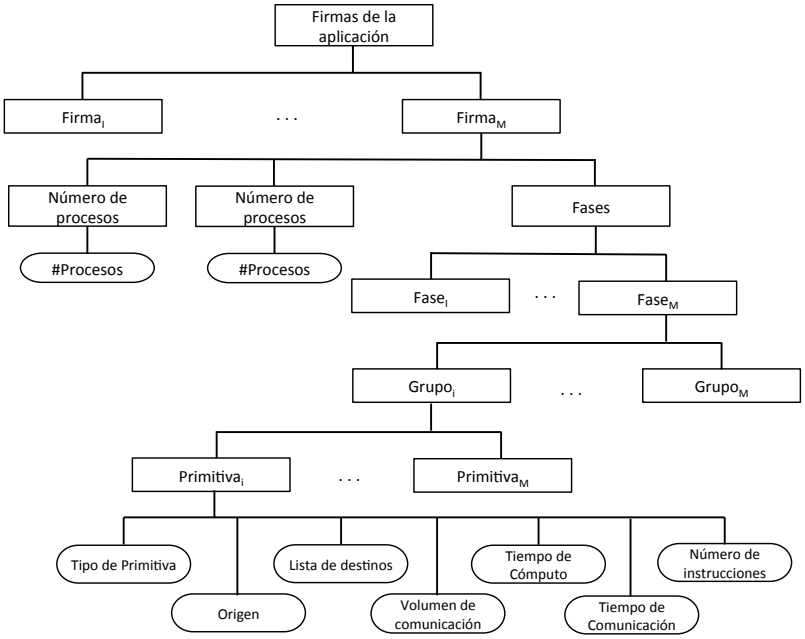


Figura 5.3: Organización de la estructura de datos utilizada para analizar las trazas físicas generadas por las diferentes firmas de la aplicación.

tener el destino de la comunicación diferente. Para esta variable, se ha creado una lista de destinos, la cual contiene todos los destinos en orden de aparición en la S_iPT. A continuación, se muestra un ejemplo para la tabla 5.2. Como se puede apreciar en la tabla, la secuencia de primitivas Irecv, Isend y Wait se repite dos veces en la traza. Como los campos son similares, tenemos un grupo de tres primitivas y su lista de destinos es {2,4}.

5.3. Modelo Lógico de la aplicación

Una vez que las fases de la aplicación han sido obtenidas, son analizadas y modelizadas con el objetivo de generar sus ecuaciones generales de comportamiento, las cuales nos permitirán proyectar el comportamiento de las fases a medida que la aplicación escala. El objetivo de esta fase de la metodología es modelizar las reglas generales de comportamiento del patrón de comunicación, el patrón de cómputo y el peso de cada fase, mediante las cuales generar la traza lógica escalable de la aplicación *(Scalable Logical Trace (SLT))* y la tabla de predicción de instrucciones *(Scalable Computational Instruction Count (SCIC)*. Esta etapa de modelizado lógico de la aplicación, tal y

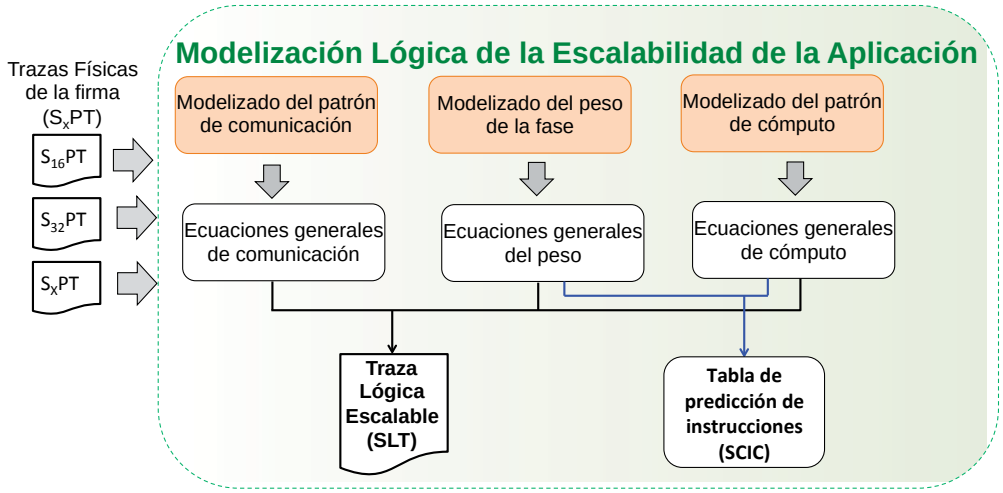


Figura 5.4: Modelo Lógico de escalabilidad de la aplicación

como se muestra en la figura 5.4, se agrupa en lo que denominados modelizado lógico escalable de la aplicación *(Scalable Behavior Prediction (SBP))*.

Con el objetivo de modelizar la evolución del comportamiento de cada fase, a medida que se incrementa el número de procesos de la aplicación, para generar sus ecuaciones generales de comportamiento, todas las fases del conjunto inicial de firmas ejecutadas, para un número de procesos diferente, serán relacionadas por similitud funcional.

Consideramos que dos fases de una misma aplicación, para diferente número de procesos, tendrán similitud funcional, cuando las dos fases realicen el mismo trabajo (cómputo), distribuido entre diferente número de procesos, cambiando únicamente la estructura del patrón de comunicación de la fase (origen, destino y volumen de comunicación), es decir, las dos fases realicen la misma función pero de distinta forma.

Se utiliza similitud funcional ya que cuando el número de procesos de la aplicación aumenta, las fases cambian su comportamiento y es necesario reconocerlas y relacionarlas para poder analizarlas y modelizarlas. Analizando el comportamiento de las fases a medida que se incrementa el número de procesos de la aplicación, sabemos que su comunicación (número de mensajes y destino), el volumen de comunicación y el tiempo de cómputo pueden variar, pero el trabajo realizado por la fase será el mismo, repartido entre un número mayor de procesos, ya que es el mismo segmento de código y estamos trabajando en escalabilidad fuerte, donde el *workload* de la aplicación se mantiene constante a medida que se aumenta el número de procesos de la aplicación.

Para relacionar las fases entre las diferentes ejecuciones de la firma, se utiliza un algoritmo el cual se basa en conocer la secuencia de fases en tiempo de ejecución, ya que no depende del número de procesos sino de la forma lógica en la cual la aplicación ha sido desarrollada.

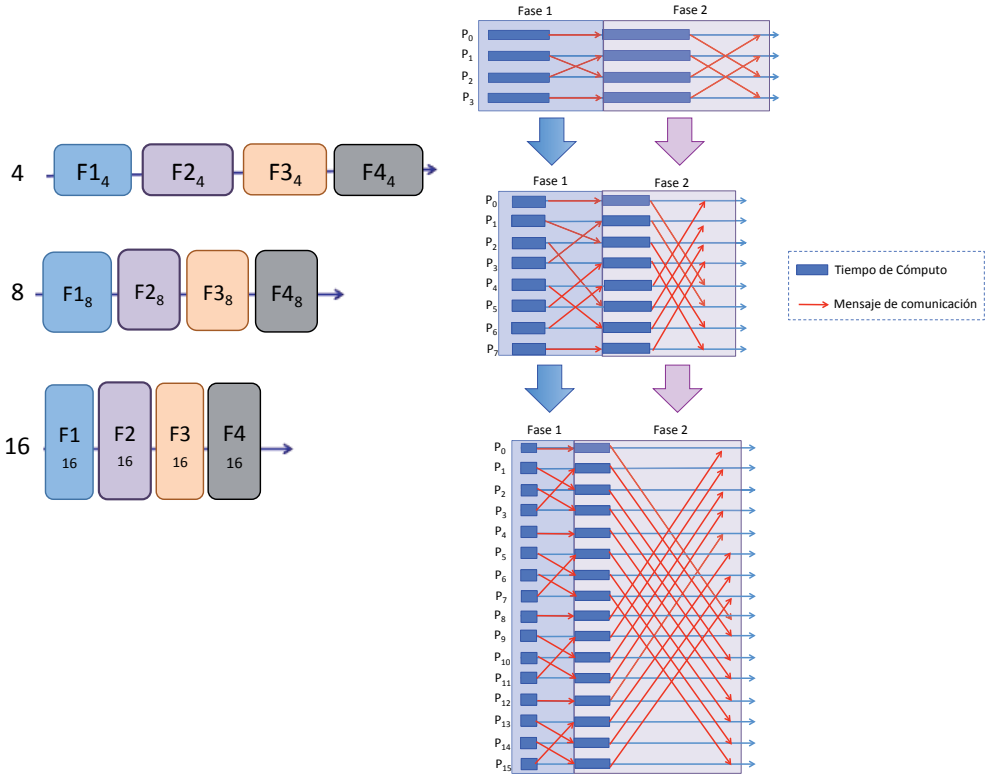
La figura 5.5 muestra un ejemplo del procedimiento. Como podemos ver en la figura 5.5.a, si ejecutamos la firma de una aplicación para 4, 8 y 16 procesos, el número de fases de la aplicación se mantiene constante, ya que es la secuencia lógica de código escrita en la aplicación, independientemente del número de procesos para el cual haya sido ejecutada. Si ahora nos centramos en la figura 5.5.b, donde se muestra en detalle la fase 1 y la fase 2 para las distintas ejecuciones de la firma, vemos que aunque las fases poseen diferente comportamiento entre ellas para 4, 8 y 16 procesos, ya que tienen diferente patrón de comunicación y diferente tiempo de cómputo, se trata de la misma fase, ya que se encuentran situadas en la misma posición lógica en tiempo de ejecución, la cuales han escalado de 4 a 16 procesos siguiendo sus reglas generales de comportamiento, por lo tanto, están realizando el mismo trabajo repartido entre diferente número de procesos, ya que es el mismo segmento de código.

Aunque el número total de fases se mantiene constante a medida que la aplicación escala, su frecuencia de repetición (peso) puede variar, tal y como se muestra en la figura 5.6, debido a la evolución de los lazos internos de la fase, dando como resultado que el número de veces que se repite la fase de forma consecutiva aumente.

A causa de esta evolución en el comportamiento de la firma, se hace necesario identificar cuando una nueva fase aparece, es decir, cuando hay un cambio de fase, con el fin de poder relacionar las fases de las firmas ejecutadas para un número de procesos diferente.

Con el objetivo de identificar cada vez que sucede un cambio de fase, el algoritmo compara por similitud las fases de la firma. Para ello, compara la fase actual, con la fase posterior, con el fin de identificar si son iguales. Para realizar esta comparación, el algoritmo utiliza los mismos criterios que utiliza PAS2P para identificar las fases que contiene una aplicación:

- El tamaño de la fase (número de ticks) tiene que ser el mismo.
- Se compara si dos eventos tienen el mismo tipo de comunicación y similar volumen de comunicación (5 % de diferencia).



(a) Secuencia lógica de la firma para fases para 4, 8 y 16 procesos

(b) Comportamiento de las fases 1 y 2 para 4, 8 y 16 procesos

Figura 5.5: Relación de fases por similitud funcional

- El tiempo de cómputo entre dos eventos tiene que ser similar (90 % de similitud o más).

Si se cumple este conjunto de criterios entre las 2 fases, se identifican como fases iguales, y se sigue comparando hasta que todas las fases han sido comparadas. En caso de no cumplir alguno de los criterios de comparación, se identifican como fases distintas, y se etiqueta la fase posterior con un identificador secuencial, el cual identifica cuando se realiza un cambio de fase.

Identificado cuando sucede un cambio de fase en la firma, se relacionan las fases de las distintas firmas con los mismos id secuenciales, tal y como se muestra en la figura 5.7, con el objetivo de analizarlas y modelizar su evolución, ya que se trata de la misma fase para distinto número de procesos. Podemos asegurar que se trata de la misma fase,

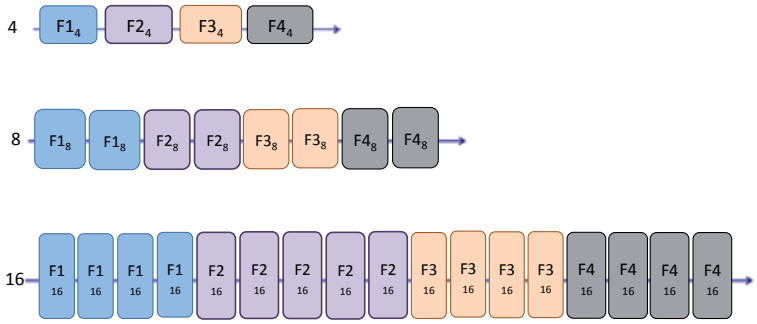


Figura 5.6: Evolución de la frecuencia de repetición de las fases de la firma

ya que aunque su número de repeticiones varíe a medida que la aplicación escala, se encuentra situada en la misma posición lógica de la firma, ya que como hemos comentado previamente, es la forma lógica en que la aplicación ha sido escrita, independientemente del número de procesos para los cuales fue ejecutada la aplicación.

Una vez las fases han sido relacionadas y antes de proceder a modelizar su evolución, se comprueba analíticamente que las fases están correctamente relacionadas. Para ello, aplicamos la ecuación 5.1 a las fases de las diferentes firmas con el mismo identificador, mediante la cual se obtiene el número total de instrucciones que realiza la fase. Puesto que como hemos comentado antes, la fase realiza el mismo trabajo repartido entre diferente número de procesos, si se trata de la misma fase evolucionada para un número mayor de procesos, el número total de instrucciones se debería mantener prácticamente constante para las fases de las diferentes firmas.

$$NumeroTotalInstruccionesFase = \#Procesos * Peso * \#instrucciones \quad (5.1)$$

5.3.1. Modelización del patrón de comunicación

Relacionadas las fases relevantes de la aplicación, se modeliza el patrón de comunicación, el cual está formado por las ecuaciones generales de comportamiento y las ecuaciones generales del volumen de comunicación para cada comunicación de la fase. La ecuación general de comportamiento calcula el destino del mensaje a partir de su origen, mientras que la ecuación general del volumen de comunicación, predice el tamaño del mensaje.

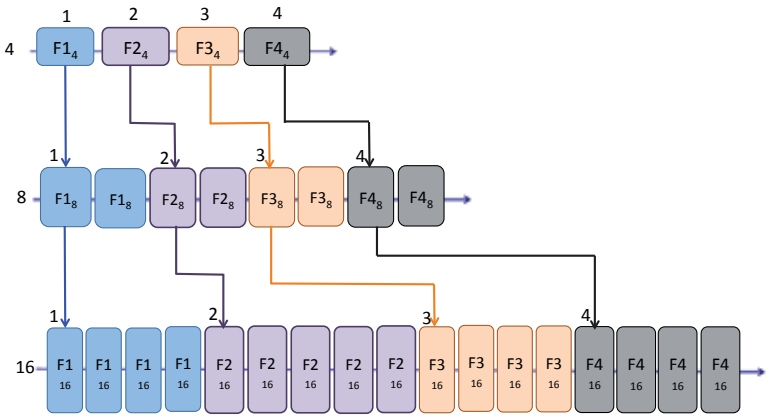


Figura 5.7: Relación de las fases de la firma de la aplicación para distinto número de procesos

Las ecuaciones generales del volumen de comunicación se modelizan mediante modelos matemáticos de regresión, los cuales permitirán predecir el volumen de cada mensaje de la fase a medida que la aplicación escala. Se ha optado por usar este tipo de modelos matemáticos para predecir el volumen de comunicación, ya que debido al comportamiento determinista de la evolución del patrón de comunicación, permiten representar sus reglas generales mediante ecuaciones sencillas y con una alta calidad de predicción.

En cuanto a las ecuaciones generales de comunicación (predecir el destino del mensaje a partir del origen), se ha propuesto un algoritmo mediante el cual modelizarlas.

El algoritmo se basa en obtener una ecuación por cada comunicación de la fase, denominada ecuación local, la cual representa el comportamiento de la comunicación. El conjunto de ecuaciones locales de cada fase, formarán el patrón de comunicación de la fase. A partir del conjunto de ecuaciones locales de una misma fase para distinto número de procesos, se modelizarán las ecuaciones generales de comunicación, una por cada comunicación de la fase, las cuales serán utilizadas para predecir el patrón de comunicación de la fase para un número mayor de procesos.

La figura 5.8 muestra un ejemplo del procedimiento. Como se puede apreciar, ejecutamos la firma de la aplicación para 16, 32, 64 y 128 procesos. A partir de la ejecución de la firma obtenemos 4 fases relevantes. Para cada fase relevante generamos las ecuaciones locales que representan sus comunicaciones, se ha considerado para este ejemplo que cada fase tiene únicamente una comunicación y por lo tanto una única ecuación local. Cada ecuación local está identificada de manera única por medio del identificador

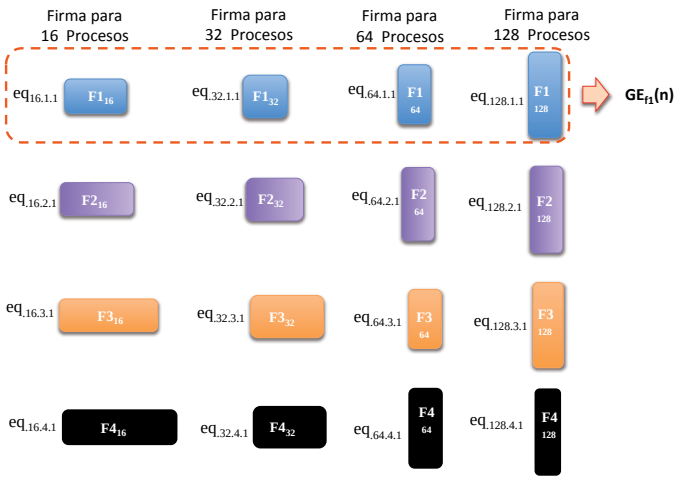


Figura 5.8: Modelización de la Ecuación General de comunicación (GE_{F_i})

($eq._{\#procesos.\#fase.\#comunicacion}$). Una vez que se han obtenido las ecuaciones locales, se analizan con el objetivo de modelizar las ecuaciones generales de comunicación para cada fase (GE_{F_i}), tal y como se muestra en la figura para la fase 1, la cual nos permitirá predecir la comunicación para un número mayor de procesos.

A continuación, en las siguientes secciones se explica en detalle el algoritmo propuesto para modelizar las ecuaciones generales (GE_{F_i}) de cada fase .

5.3.1.1. Generación de las ecuaciones locales

Como se muestra en la figura 5.9, esta etapa está compuesta de dos fases. Una primera fase de análisis, que tiene como entrada el árbol de firmas (ST) generado en la etapa de caracterización, en la cual se analiza la información de las fases obtenida en la etapa de caracterización de la aplicación, con el fin de obtener información acerca del patrón de comunicación de cada fase, y una segunda fase de modelización, donde las ecuaciones locales para cada fase son modelizadas y generadas.

Durante la fase de análisis, para cada fase relevante de la aplicación, se identifican las dependencias entre procesos, el tipo de patrón: Estático (Malla, Anillo, etc) o Dinámico (Intercambio, permutación, etc) y la matriz de distancia entre procesos. La matriz de distancia contiene la distancia lógica entre el origen y el destino escrita en los algoritmos de comunicación de cada fase de la aplicación, sin considerar la distancia física donde los procesos fueron mapeados en el sistema. Toda esta información es pro-

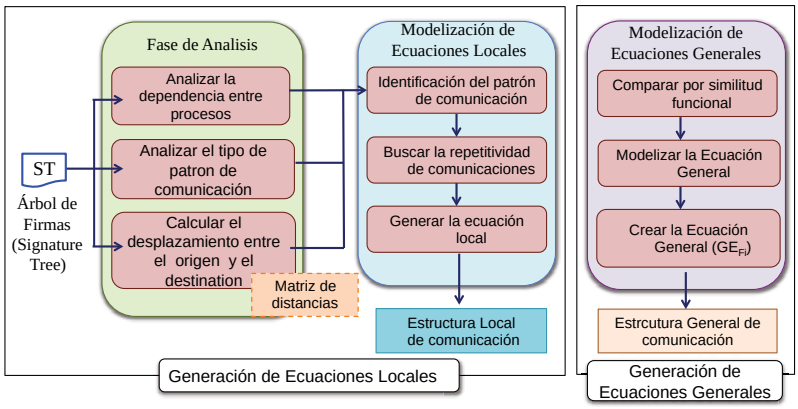


Figura 5.9: Etapas del algoritmo de generación de la Ecuación General de comunicación (GE_{F_i})

porcionada a la segunda fase de modelización, con el objetivo de generar las ecuaciones locales de comunicación de cada fase. En esta etapa, las ecuaciones locales que identifican el comportamiento del patrón de comunicación de cada fase son obtenidas. Para ello, se utiliza un algoritmo de identificación. Este algoritmo se basa en el principio de que la aplicación ha sido bien desarrollada y los patrones de comunicación siguen un comportamiento determinista, donde a medida que la aplicación escala, los patrones de comunicación siguen unas determinadas reglas de comportamiento, las cuales no tienen sentencias condicionales que hagan impredecible el patrón a medida que la aplicación escala. Por otro lado, el algoritmo considera que los patrones de comunicación han estado escritos utilizando comunicaciones punto a punto, no considerando las comunicaciones colectivas. En caso de no cumplir alguna de estas condiciones, la metodología no podrá ser aplicada.

El algoritmo obtiene una ecuación local por cada evento de comunicación, la cual identifica la comunicación para todos los procesos que componen la fase. Para ello, el algoritmo analiza el origen y destino de todas las primitivas send de una misma comunicación, como se muestra en la figura 5.10 para el primer send (Comunicación 1), con el objetivo de generar una ecuación que represente esta comunicación. Esta ecuación indicará a cada proceso que compone la fase, el origen y destino de su comunicación. Una vez que se ha generado esta ecuación, se busca la repetitividad de los destinos entre los procesos que componen la comunicación, con el objetivo de generar una ecuación más sencilla, simplificando el análisis y modelización de las ecuaciones generales. Una

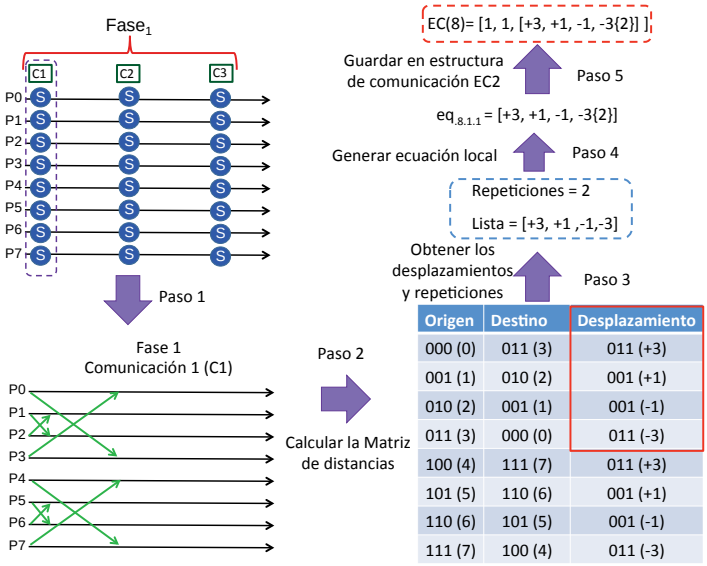


Figura 5.10: Obtención de la ecuación local de comunicación para la primera comunicación de la fase

vez hemos identificado esta información, las ecuaciones locales son generadas. La salida de este módulo es una estructura que contiene las ecuaciones locales de cada fase.

El algoritmo usa dos estructuras diferentes, ya que el camino para predecir el patrón de comunicación es diferente dependiendo del tipo de patrón (dinámico o estático). Por esa razón, la información que tiene que ser guardada para generar las ecuaciones generales es diferente. Si el patrón es dinámico, la obtención del proceso destino se basa en un intercambio o permutación de ciertos bits del identificador del proceso origen, los cuales son llamamos bits implicados. Para este tipo de patrón de comunicación, se utiliza la estructura *EC1*. En caso de ser un patrón estático, para obtener el proceso destino, el algoritmo se basa en obtener la distancia entre procesos y la repetitividad de comunicaciones. Para este tipo de patrones, se utiliza la estructura *EC2*.

La estructura *EC1* tiene como parámetros el número de fase (#Fase), el número de comunicación de la fase (#Comm),el cual es un contador secuencial que identifica cada comunicación dentro de la fase de manera única, el tipo de algoritmo (Intercambio o Permutación) y la lista de bits implicados. La estructura *EC2* tiene como parámetros el número de fase, el número de comunicación de la fase, una lista que contiene las distancias de comunicación entre el origen y el destino, y el número de repeticiones.

1. EC1(p) = {#Fase, #Comm, Tipo , Lista de bits implicados }

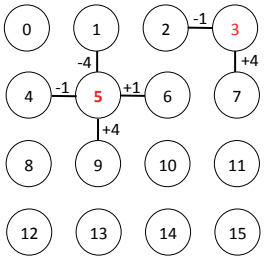
2. EC2(p) = {#Fase, #Comm, lista[Distancias de comunicación{#repeticiones}] }

En la Fig. 5.10 mostramos un breve ejemplo del procedimiento. Tenemos una fase con 8 procesos (p = 8) y tres comunicaciones. Estas tres comunicaciones componen el patrón de comunicación de la fase, el cual es estático ya que es una malla 4x2, identificado durante la fase de análisis del algoritmo. En la misma fase de análisis también obtenemos la dependencia entre los procesos. Como podemos observar en la figura, existe una dependencia entre el grupo de procesos del 0 al 3 y otra entre el grupo de procesos del 4 al 7, siendo estos dos grupos independientes entre ellos. Esta dependencia será utilizada en la siguiente etapa (etapa de modelización) para buscar la repetitividad entre procesos. Debido al tipo de patrón de la fase, se utilizará la estructura *EC2*. Si nos centramos en la primera comunicación de la fase (paso 1), generamos su matriz de distancias entre el origen y el destino (paso 2). Entonces, se busca la repetitividad, en este caso, la secuencia {+3,+1,+1,-3} se repite dos veces (paso 3), la primera para los procesos del 0 a 3 y la segunda para los procesos del 4 a 7. Una vez tenemos la secuencia y su repetitividad, generamos la ecuación local que representa esta comunicación (paso 4) y finalmente, se guarda en la estructura de salida (EC2).

Dependiendo del tipo de aplicación y su patrón de comunicación, puede ser posible que las fases de la aplicación no contengan el mismo número de comunicaciones para todos los procesos. La figura 5.11 muestra un ejemplo de una aplicación *SPMD (Single Program Multiple Data)*, donde el proceso 5 tiene 4 comunicaciones mientras que el proceso 3 tiene 2 comunicaciones, debido a que es un proceso borde de la malla. Con el objetivo de obtener ecuaciones locales correctas, las fases tienen que contener el mismo

Traza para el proceso 5

Tipo de primitiva	Origen	Destino
MPI_Send	5	6
MPI_Send	5	9
MPI_Send	5	4
MPI_Send	5	1



Traza para el proceso 3

Tipo de primitiva	Origen	Destino
MPI_Send	3	7
MPI_Send	3	2

Figura 5.11: Ejemplo de aplicación paralela de paso de mensajes con paradigma SPMD

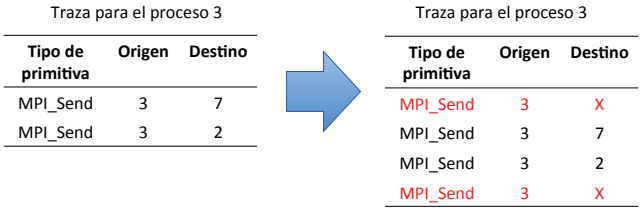


Figura 5.12: Generación de comunicaciones nulas

número de comunicaciones para todos los procesos, ya que si no los algoritmos no pueden relacionar las comunicaciones correctamente.

Para solventar este problema, el algoritmo selecciona entre los procesos de las trazas S_xPT, obtenidas durante la etapa de caracterización, el proceso con mayor número de comunicaciones de la fase, y completa los procesos con menor número de comunicaciones, con comunicaciones nulas, hasta que todos los procesos tienen el mismo número de comunicaciones. Estas comunicaciones no serán ejecutadas a la hora de predecir el rendimiento de la aplicación, su único objetivo es que todas las fases tengan el mismo número de eventos para que puedan ser relacionadas.

Con el objetivo de completar las trazas, el algoritmo se basa en la idea que la aplicación está escrita de una manera determinista, y los envíos siguen un orden lógico con un comportamiento específico. Mostramos un ejemplo siguiendo con la figura 5.11. El proceso 5 tiene 4 comunicaciones, con un vector de distancia [+1,+4,-1,-4], mientras que el proceso 3 tiene 2 comunicaciones con un vector de distancia [+4,-1]. Comparando los vectores, las comunicaciones que no aparecen en la traza del proceso 3 son la primera y la última (+1,-4). Entonces, el algoritmo completa la traza del proceso 3 con comunicaciones nulas en la primera y cuarta posición de la traza, las cuales están identificadas con X en el destino del mensaje, tal y como se muestra en la figura 5.12.

5.3.1.2. Modelización de las ecuaciones globales

Una vez que la estructura local de comunicación ha sido generada, el siguiente paso del algoritmo es modelizar las ecuaciones generales de comportamiento, las cuales serán utilizadas para predecir el patrón de comunicación a medida que aumenta el número de procesos de la aplicación.

Con el objetivo de modelizar la evolución del patrón de comunicación de cada fase, tal y como se muestra en la fig. 5.13, se propone un método que consiste en comparar

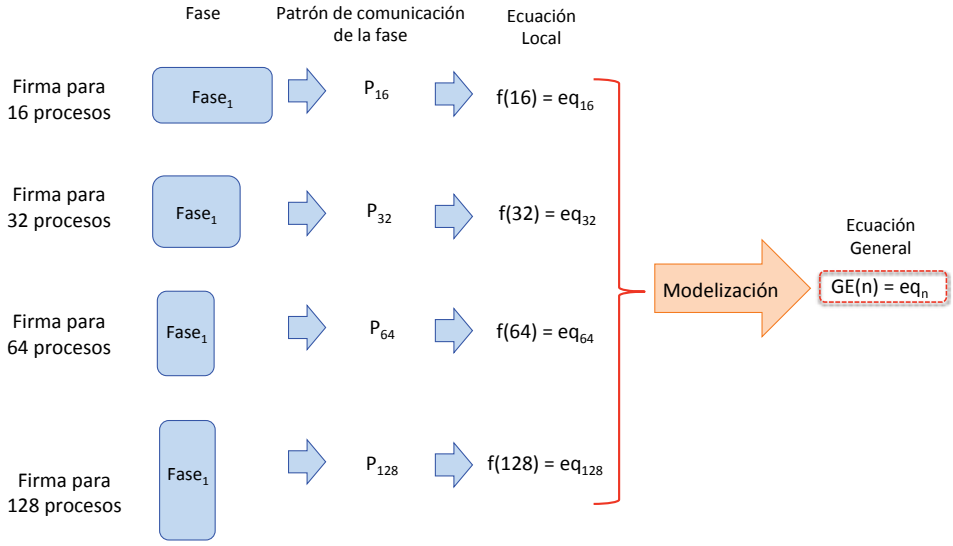


Figura 5.13: Generación de la Ecuación General a partir de sus ecuaciones locales

las ecuaciones locales de una fase, a medida que la aplicación escala, con el fin de modelizar una ecuación general que represente la evolución del comportamiento del patrón de comunicación y nos permita predecirlo a medida que se incremente el número de procesos de la aplicación.

Las ecuaciones generales tienen la misma estructura que las ecuaciones locales, la diferencia radica en el hecho que tienen como parámetro de entrada el número de procesos (n) para el cual queremos predecir el patrón de comunicación. Además, los parámetros de las ecuaciones han sido modelizados como funciones. La fig 5.14 muestra un ejemplo, a partir de las ecuaciones locales para 16, 32 y 64 procesos, se modeliza la ecuación general de comportamiento que nos permitirá predecir el patrón de comunicación para un número mayor de procesos. Si nos fijamos en los parámetros de las ecuaciones locales, podemos observar que algunos de sus parámetros, debido al tipo de patrón de comunicación, son constantes para todas las ecuaciones locales, en cuyo caso, son puestos como constantes con el mismo valor en la ecuación general. Los parámetros que cambian su valor a medida que se incrementa el número de procesos, son modelizados mediante una función matemática que describe su comportamiento a medida que se escala el número de procesos. Estas funciones tienen como parámetro de entrada el número de procesos a predecir. En el ejemplo, los parámetros cuarto y quinto cambian su valor, debido a la evolución del patrón, estos parámetros son modelizados obteniendo como resultado una

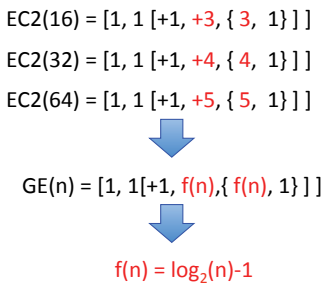


Figura 5.14: Modelización de los parámetros de las Ecuaciones Generales

función ($f(n) = log_2(n) - 1$) que representa la evolución del patrón. Finalmente, esta estructura se simplifica para manejar ecuaciones más sencilla de utilizar.

En aplicaciones paralelas escritas con patrones de comunicación dinámicos, a medida que se incrementa el número de procesos, puede suceder que el patrón de comunicación expanda comunicando con un número mayor de procesos, dando como resultado que aparezcan nuevas comunicaciones. Para predecir el número de comunicaciones, el algoritmo modeliza el comportamiento de cómo esas nuevas comunicaciones van apareciendo y que comportamiento siguen a medida que se incrementa el número de procesos de la aplicación. La figura 5.15 muestra un ejemplo. A partir de las trazas para 2, 4, 8 y 16 procesos, se desea predecir el patrón de comunicación para 64 procesos para una fase de la aplicación. Como se puede observar, a medida que se dobla el número de procesos de la aplicación, el patrón de comunicación de la fase se expande comunicando con un nuevo proceso. El objetivo es modelizar una ecuación general mediante la cual predecir el número de comunicaciones de la fase, a medida que la aplicación escala. En la figura 5.15 podemos observar que la ecuación $F(n) = log_2(n) + 1$, modelizada a partir del análisis de las firmas ejecutadas, donde n es el número de procesos para el cual se desea predecir el número de comunicaciones de la fase, representa el comportamiento del número total de comunicaciones de la fase, a medida que la aplicación escala. Puesto que se desea predecir el número de comunicaciones para 64 procesos, aplicamos la ecuación para 64 procesos (n=64) y obtenemos 7 comunicaciones. A diferencia de las ecuaciones generales donde el número de comunicaciones se mantiene constante, estas incorporan en el último parámetro de la ecuación EC1, además de las ecuaciones para predecir el patrón de comunicación, esta nueva ecuación general para predecir el número total de

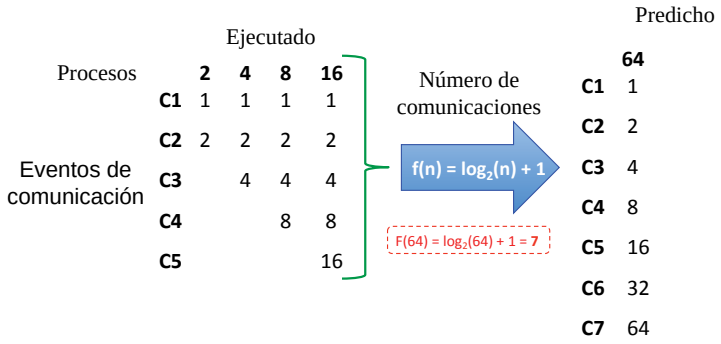


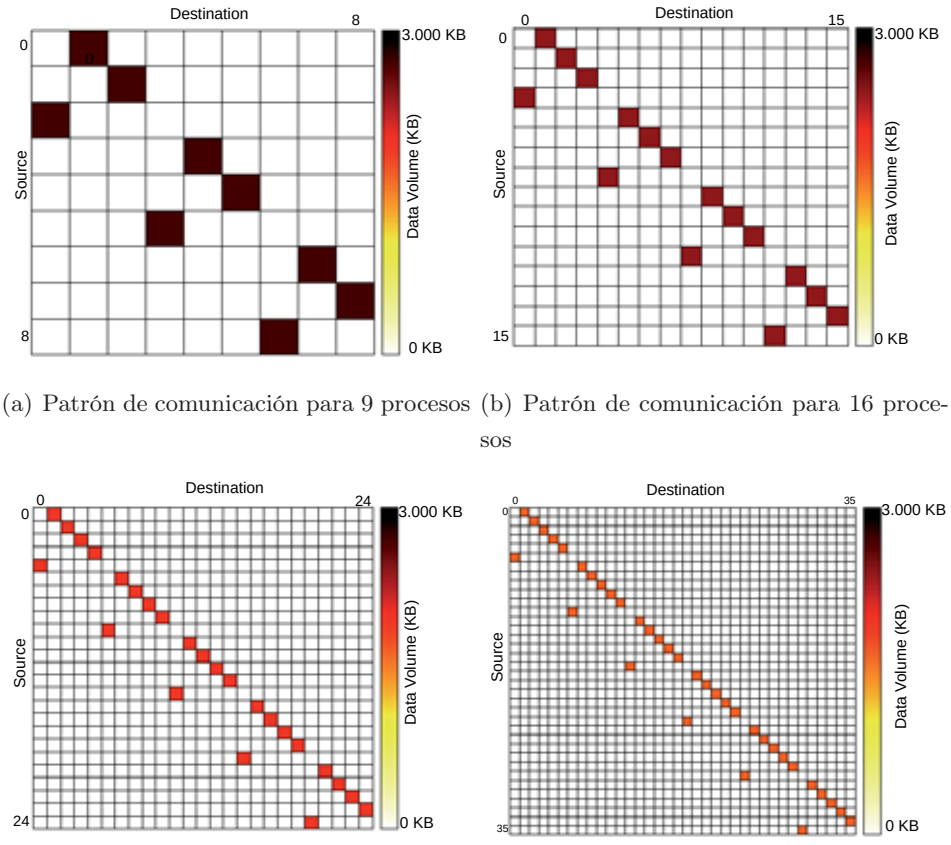
Figura 5.15: Obtención del número total de comunicaciones de la fase

comunicaciones de la fase.

A continuación, a modo de resumen de esta sección, se muestra un ejemplo de predicción del patrón de comunicación mediante la utilización de la ecuación general de comportamiento, la cual ha sido generada a partir de las ecuaciones locales.

Para realizar este ejemplo, se ha utilizado la aplicación BT, la cual está incluida en la suite de Benchmarks NPB NAS [59]. Con el objetivo de generar las ecuaciones generales de comportamiento, se han ejecutado las firmas para 9, 16, 25 y 36 procesos. A partir de la ejecución de este conjunto de firmas, se desea predecir el patrón de comunicación para 49 procesos.

Una vez ejecutadas las firmas, se han obtenido 6 fases relevantes. Para el ejemplo, nos centramos en la primera comunicación de la fase 1. La figura 5.16 muestra los patrones de comunicación para la primera comunicación de la fase 1 de las firmas ejecutadas, el comportamiento de las cuales está descrito en sus ecuaciones locales, mostradas en la tabla 5.1. A partir de la etapa de análisis hemos obtenido que el patrón de comunicación es estático, ya que se calcula a través de desplazamientos y no aparecen nuevas comunicaciones a medida que se aumenta el número de procesos de la aplicación, por lo que se utiliza la estructura EC2. Debido al comportamiento del patrón de comunicación de la aplicación, las ecuaciones locales están compuestas por dos términos. Si nos centramos en la ecuación local para 9 procesos, vemos que está compuesta por un primer término [+1 {+2,+3}] y un segundo término [-2{+1,+3}]. El primer término indica un desplazamiento lógico +1, el cual se repetirá 2 veces de forma consecutiva, puesto que el primer parámetro de la lista de repeticiones es un 2. El segundo termino de la ecuación local indica que después del desplazamiento anterior proseguirá un desplazamiento lógico de



(a) Patrón de comunicación para 9 procesos (b) Patrón de comunicación para 16 procesos

(c) Patrón de comunicación para 25 procesos (d) Patrón de comunicación para 36 procesos

Figura 5.16: Patrón de la primera comunicación de la fase 1 de la aplicación BT

-2, el cual se repetirá una única vez, ya que el primer término de la lista de repetición es 1. Siguiendo esta ecuación tenemos una primera secuencia [+1,+1,-2]. Puesto que el segundo parámetro de la lista de repetición es un 3 tanto para el primer como segundo término, la secuencia [+1,+1,-2] se repitira 3 veces, obteniendo la secuencia final [+1,+1,-2, +1,+1,-2,+1,+1,-2], la cual representa el patrón de comunicación para los 9 procesos, tal y como se puede ver en la figura 5.16.a.

A partir de las ecuaciones locales, se modeliza la ecuación general de comportamiento, la cual está descrita en la ecuación 5.2. Esta ecuación tiene como parámetro de entrada el número de procesos (n) para el cual se desea predecir el patrón de comunicación.

Tabla 5.1: Ecuaciones locales para la primera comunicación de la fase 1 de la aplicación BT

Número de Procesos	Fase	Comunicación. Local	Ecuación
9	1	1	[+1 {+2,+3}], [-2{+1,+3}]
16	1	1	[+1 {+3,+4}], [-3{+1,+4}]
25	1	1	[+1 {+4,+5}], [-4{+1,+5}]
36	1	1	[+1 {+5,+6}], [-5{+1,+6}]

$$GE(n) = [+1\{\sqrt(n) - 1, \sqrt(n)\}], [-\sqrt(n) + 1\{+1, \sqrt(n)\}] \tag{5.2}$$

Una vez tenemos la ecuación general, si la aplicamos para 49 procesos, obtenemos la expresión 5.3. Si expandimos esta expresión, obtendremos el patrón de comunicación predicho para 49 procesos, tal y como se muestra en la figura 5.17.

$$GE(49) = [+1\{+6, +7\}], [-6\{+1, +7\}] \tag{5.3}$$

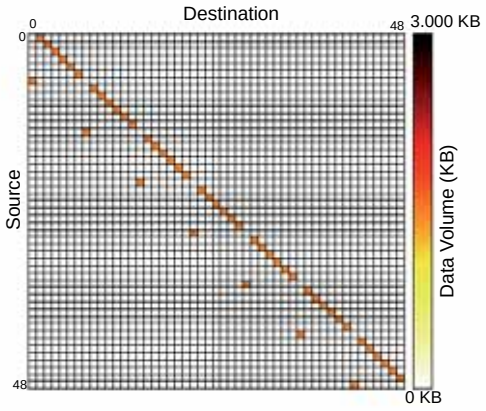


Figura 5.17: Patrón de comunicación predicho a partir de la Ecuación General para 49 procesos

5.3.2. Modelización del peso de la fase

Una vez que el patrón de comunicación ha sido modelizado, el siguiente paso es modelizar el comportamiento del peso de cada fase, a medida que se incrementa el número de procesos de la aplicación.

Con el objetivo de modelizar el comportamiento del peso de cada fase, se utilizan modelos matemáticos de regresión. Debido al comportamiento determinista de las aplicaciones paralelas, ya que como se comentó anteriormente, las aplicaciones son desarrolladas lógicamente de una manera específica, las cuales siguen reglas de comportamiento a la hora de escalar, existe una dependencia lineal entre el incremento del número de procesos de la aplicación y el peso de la fase. Por esta razón, los modelos matemáticos de regresión lineal son apropiados para modelizar el peso, ya que nos permiten modelizar el peso de la fase mediante una ecuación lineal del tipo $y = a + bx_0$, usando como variable independiente el número de procesos de la aplicación, la cual representa el comportamiento exacto de la evolución del peso de la fase a medida que la aplicación escala, obteniendo un $R - square = 1$.

Puesto que las aplicaciones científicas no pueden ser ejecutadas para cualquier número de procesos, sino que siguen reglas de ejecución con el objetivo de programar sus algoritmos de forma eficiente, puede ser posible que los métodos de regresión lineales no ajusten adecuadamente, obteniendo un índice de correlación *R-square* alejado de 1, siendo más apropiado otros métodos de regresión.

La baja calidad de predicción obtenida en el modelo de regresión lineal, se debe a la distancia entre las muestras (número de procesos) utilizadas para generar el modelo de regresión. La fig. 5.18 muestra un ejemplo para la fase 1 de la aplicación NPB BT [59], donde por limitaciones de la aplicación, los usuarios únicamente pueden ejecutar la aplicación utilizando un número cuadrático de procesos. A partir de las ejecuciones de la firma para 16, 25, 36, 49, 64, 81, 100 y 121 procesos, se modeliza su peso utilizando el modelo de regresión lineal. Como se puede apreciar en la figura, la distancia de los puntos de entrada no es uniforme, causando que la ecuación de regresión generada para predecir el peso de la fase tenga un índice de correlación $R - Square = 0{,}98253$, siendo más apropiado en este caso utilizar un modelo de regresión potencial.

La teoría de modelos matemáticos de regresión nos dice que utilizar modelos de regresión con un índice inferior a 0.99, podría ocasionar obtener un error de predicción

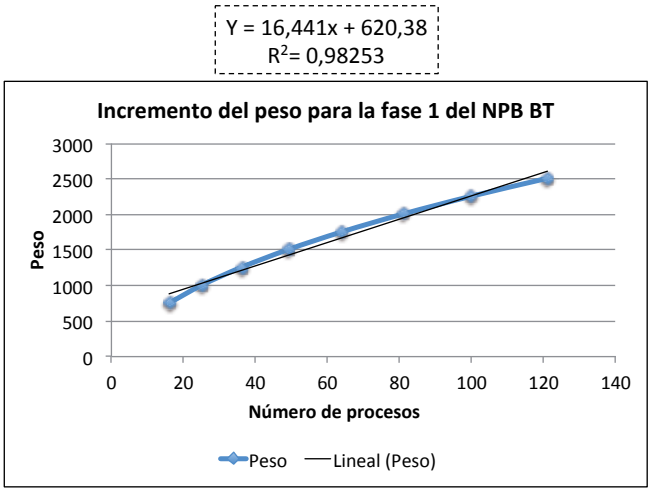


Figura 5.18: Modelo de regresión para la fase 1 del NPB BT

elevado, el cual se irá incrementando a medida que nos vayamos alejando de los puntos utilizados para generar el modelo.

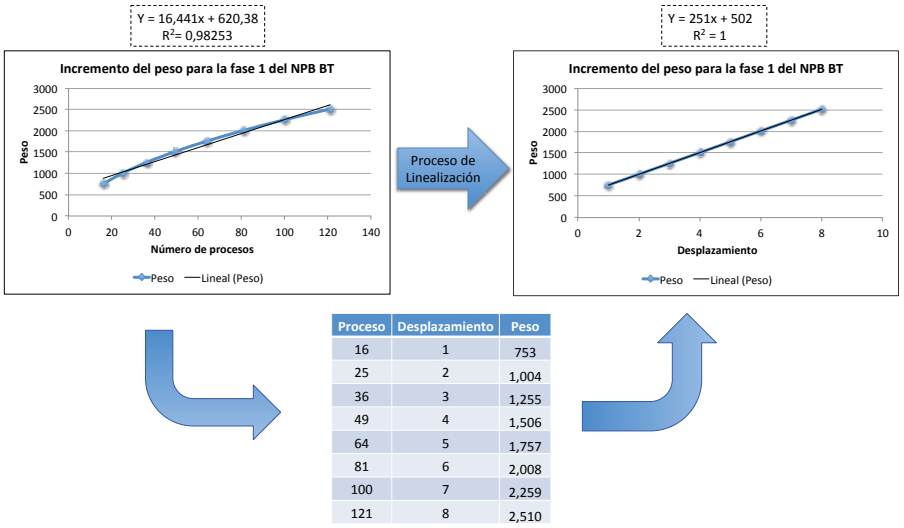
Con el fin de poder modelizar una función de regresión lineal con un índice de correlación $R - square = 1$, se lleva a cabo un proceso de linealización basado realizar un cambio de dominio de la variable independiente (número de procesos). El objetivo de este proceso es obtener una distancia equidistante entre los puntos de entrada que generan el modelo.

Con el objetivo de realizar este cambio de dominio, como se puede apreciar en la figura 5.19, se cambia el valor de la variable independiente por un índice secuencial, llamado *desplazamiento*, el cual aumenta su valor una unidad a medida que incrementamos el número de procesos de la aplicación. Mediante este cambio de dominio se consigue una distribución equidistante de los puntos, obteniendo un índice de correlación $R - Square = 1$, mediante el uso de modelos de regresión lineal.

5.3.3. Modelización del patrón de cómputo

El último paso para generar la traza lógica escalable (SLT) es modelizar el patrón de cómputo de la fase, el cual está compuesto por el número de instrucciones de cómputo ejecutadas entre eventos de comunicación de la fase.

El método propuesto para modelizar el patrón de cómputo se basa en el hecho que cuando consideramos la escalabilidad fuerte, el *workload* de entrada de la aplicación se



Proceso	Desplazamiento	Peso
16	1	753
25	2	1,004
36	3	1,255
49	4	1,506
64	5	1,757
81	6	2,008
100	7	2,259
121	8	2,510

Figura 5.19: Linealización del modelo de regresión

mantiene constante a medida que se incrementa el número de procesos de la aplicación. Dicho *workload*, es distribuido entre todos los procesos de la aplicación, disminuyendo el número de instrucciones ejecutadas por proceso, a medida que la aplicación aumenta el número de procesos, siendo el número total de instrucciones ejecutadas por todos los procesos prácticamente constante, ya que el trabajo a realizar es el mismo distribuido entre un número diferente de procesos.

Este concepto puede ser extrapolado y aplicado a las diferentes fases de la aplicación, manteniéndose constante el número total de instrucciones ejecutadas en cada fase, a medida que se incrementa el número de procesos de la aplicación, como se muestra en la figura 5.20. Esto es debido, a que las fases son secuencias de código que implementan un algoritmo o una parte del algoritmo (etapa de un algoritmo científico), que realizan un trabajo específico, los cuales distribuyen el *workload* a computar entre el número de procesos de la aplicación. Puesto que estamos trabajando con escalabilidad fuerte, el *workload* de entrada que recibe cada fase a medida que escala la aplicación, siempre es el mismo, distribuido entre un número diferente de procesos, lo que hace que el número total de instrucciones se mantenga prácticamente constante.

Con el objetivo de predecir el número de instrucciones de cada proceso de la fase, se modeliza como las instrucciones son distribuidas entre los procesos de la aplicación, a medida que aumenta su número de procesos. Dependiendo de la aplicación, no todos

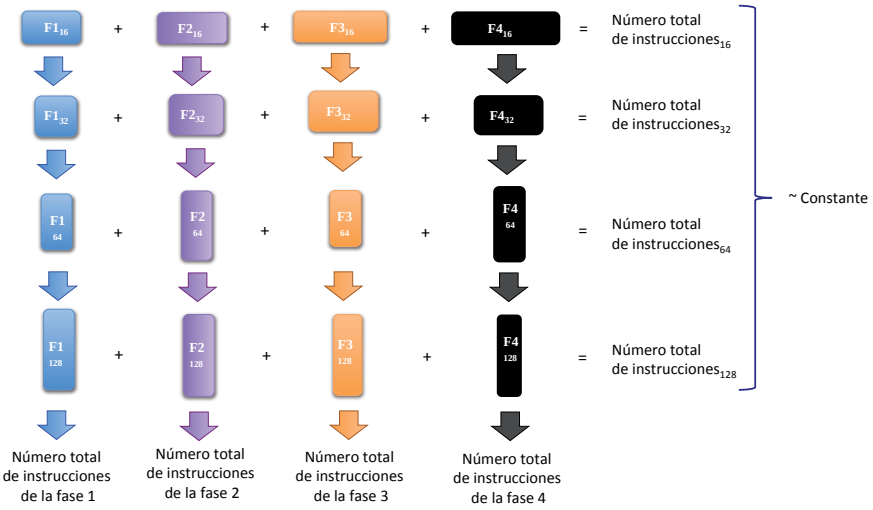


Figura 5.20: Distribución del número de instrucciones de cada fase de la aplicación, a medida que se aumenta su número de procesos

los procesos realizan el mismo trabajo, por lo que es necesario agrupar los procesos que tengan el mismo comportamiento con el objetivo de poder modelizar su evolución. A fin de modelizar este comportamiento, los procesos con similar número de instrucciones (95 % de similitud) son agrupados en grupos, llamados *Instruction Groups* (IG_i). El número total de instrucciones de cada IG_i se mantendrá constante a medida que se incrementa el número de procesos de la aplicación, siendo la suma de todos los IG_i de la fase el número total de instrucciones de la fase.

La figura 5.21 muestra un ejemplo de una fase con 4 procesos los cuales tienen diferente número de instrucciones. Los procesos 0 y 1 tienen el mismo número de instrucciones, mientras que los procesos 2 y 3 tienen otro número de instrucciones distinto. Si escalamos la aplicación para 8 procesos, podemos observar que los procesos 0 y 1 distribuyen sus instrucciones entre los procesos del 0 al 3, mientras que los procesos 2 y 3 distribuyen sus instrucciones entre los procesos del 4 al 7, siguiendo las reglas de su patrón de cómputo. Por lo tanto, para este ejemplo tenemos 2 Instruction Groups diferentes, IG_1 e IG_2, los cuales irán evolucionando su comportamiento a medida que se incremente el número de procesos de la aplicación, siguiendo sus reglas de comportamiento escritas en la aplicación para esta fase.

Una vez se han generado los *Instruction Group* de la fase, son modelizados. La suma del número total de instrucciones de cada IG_i ($TotalIG$), será el número total de

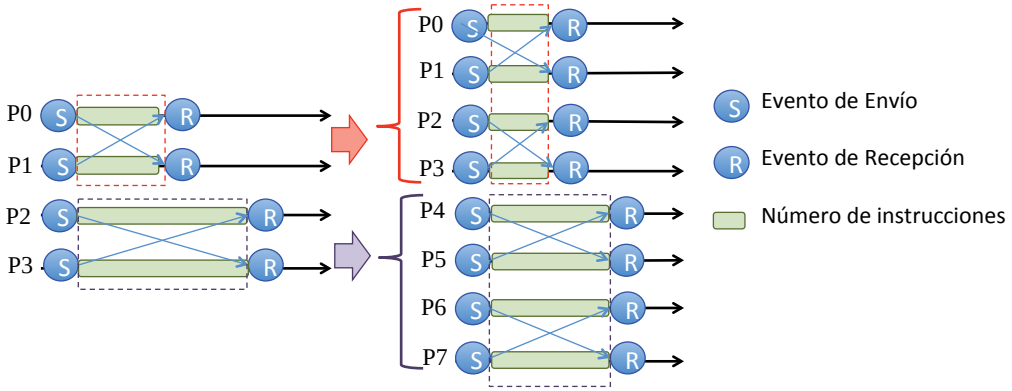


Figura 5.21: Distribución del número de instrucciones

instrucciones de la fase, como se muestra en la ecuación 5.4, donde x es el número total de grupos (IG_i's).

$$TotalInstruccionesFase = \sum_{i=1}^{x} (TotalIG_i) \tag{5.4}$$

Como se muestra en la ecuación 5.5, el número total de instrucciones de cada IG_i ($TotalIG$), es la suma del número total de instrucciones de cada proceso $"NI_i"$ perteneciente al grupo, multiplicado por el peso de la fase (W_x), siendo n el número total de procesos pertenecientes al grupo.

$$TotalIG = \sum_{i=0}^{n} (NI_i) * W_x \tag{5.5}$$

Siguiendo con la misma ecuación, el objetivo es predecir el número de instrucciones de cada proceso del IG_i a medida que la aplicación escala, es decir, el término $"NI_i"$. Debido a que el término $TotalIG$ es prácticamente constante, ya que es el número total de instrucciones del grupo, el cual se mantiene constante a medida que la aplicación escala, y el comportamiento de la evolución del peso fue modelizado en la etapa anterior, la variable $"NI_i"$ puede ser despejada, obteniendo la predicción del número de instrucciones para cada proceso del grupo IG_i , tal y como se muestra en la ecuación 5.6.

$$NI_i = \frac{\frac{TotalIG_i}{n}}{W_x} \tag{5.6}$$

Una vez que se ha obtenido el número total de instrucciones de cada proceso, tienen que ser distribuidas entre los segmentos de cómputo, situados entre los eventos de comunicación de la fase. Con el objetivo de repartir estas instrucciones, se aplican reglas de similitud funcional, las cuales indican que proporción del número total de instrucciones tiene que ser asignado a cada segmento de cómputo, como se muestra en la figura 5.22. Como podemos observar en la figura, a partir de la ejecución para 16 y 32 procesos, el proceso 0 (P_0) disminuye su número de instrucciones, pero sus segmentos de cómputo mantienen el porcentaje relativo del número total de instrucciones del proceso (NI_i). Si predecimos el número de instrucciones para el proceso 0 (P_0) con la aplicación escalada para 4,096 procesos, el proceso seguirá las mismas reglas de comportamiento a la hora de distribuir sus instrucciones que para 16 y 32 procesos, por lo que se reparten aplicando sus reglas de comportamiento, en función del número total de instrucciones del proceso, tal y como se muestra en la figura.

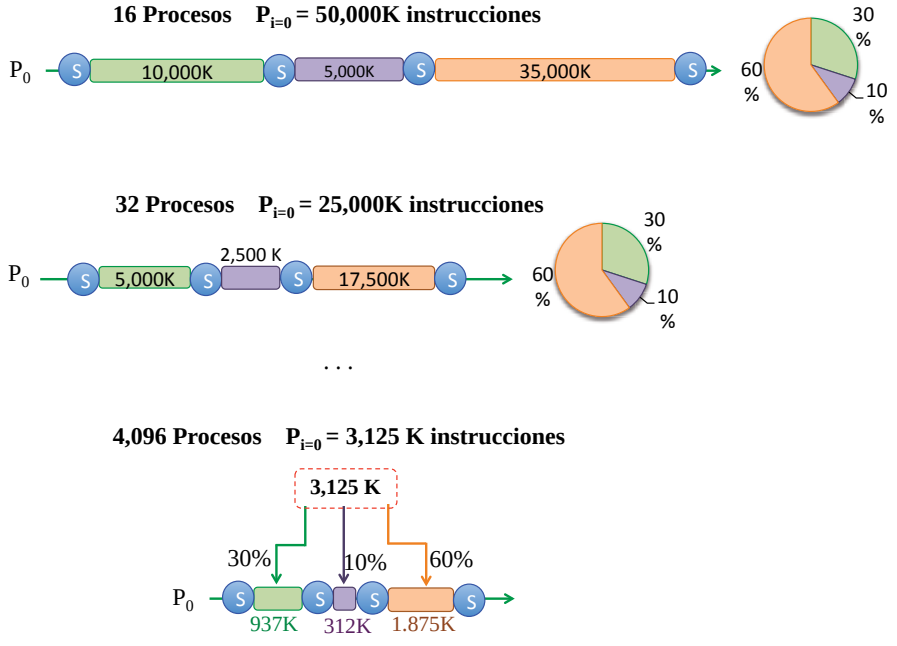


Figura 5.22: Distribución del número de instrucciones de cada proceso

Una vez se ha realizado la modelización del patrón de comunicación, el peso de la fase y el patrón de cómputo (número de instrucciones de cada segmento de cómputo), y se han obtenido sus ecuaciones generales de comportamiento para cada fase de la aplicación, se genera la traza lógica escalable de la aplicación *(Scalable Logical Trace*

(SLT)) y la tabla de predicción de instrucciones *(Scalable Computational Instruction Count (SCIC))* .

La SLT contiene las ecuaciones generales de comportamiento de cada fase de la aplicación. Dicha traza es independiente del sistema paralelo, sólo depende de la forma en que la propia aplicación ha sido desarrollada, es decir, únicamente posee los parámetros intrínsecos de la aplicación. La SLT se genera por proceso en lugar de generar una traza global con el objetivo de poder tratar cada proceso de forma independiente.

En cuanto a la SCIC, se genera a partir de las ecuaciones generales de cómputo y peso de cada fase de la aplicación, y contiene una visión global de cada fase con respecto al número total de instrucciones predichas, a medida que la aplicación aumenta el número de procesos. Esta tabla contiene toda la información necesaria: número de instrucciones por proceso, número total de procesos de la fase y, peso de la fase y su desplazamiento, para poder predecir el número total de instrucciones de la fase.

Como se muestra en la tabla 5.2, la tabla contiene dos partes claramente diferenciadas, una parte con datos medidos y otra parte con datos predichos. La parte medida es obtenida a partir de la ejecución del conjunto inicial de firmas, mientras que la parte predicha es generada a partir de las ecuaciones generales obtenidas en la etapa de modelización. El objetivo de tener una parte medida y una predicha es validar el número total de instrucciones con el fin de tener una idea de la calidad y precisión del modelo lógico de la aplicación.

En caso de obtener más de un IG_i durante la etapa de modelización del patrón de cómputo, se generan tantas tablas de predicción de instrucciones como IG_i's se hayan generado. La suma del número total de instrucciones total de cada SCIC será el número total de instrucciones de la fase.

La SCIC será utilizada a la hora de generar el modelo de regresión de cómputo, mediante el cual predecir el tiempo de cómputo para cada fase de la aplicación.

Finalmente, una vez que tanto la SLT como la SCIC han sido creadas, la SLT tiene que ser concretada para un número específico *(N)* de procesos. Para ello, la SLT posee como parámetro de entrada el número de procesos para el cual se desea predecir el rendimiento de la aplicación. Una vez se ha concretado la traza para un determinado numero de procesos, se obtiene una nueva traza lógica específica para ese número de procesos, denominada *(Logical Trace for N cores (LT4NC))*, la cual contiene el comportamiento de comunicación, cómputo y peso de cada fase para ese número de procesos. Para ello,

Tabla 5.2: Tabla de predicción de instrucciones (Scalable Computational Instruction Count (SCIC))

Número de Instrucciones	Número de Procesos	Peso	Desplazamiento del Peso	Número total de Instrucciones
Valores medidos (Conjunto inicial de Firmas)				
1,539,358,893	16	1,255	3	30,910,326,571,440
820,993,294	25	1,506	4	30,910,397,519,100
488,835,328	36	1,757	5	30,919,812,166,656
314,154,733	49	2,008	6	30,910,312,489,336
213,799,791	64	2,259	7	30,910,318,583,616
Valores predichos (Generados con el modelo lógico de la fase (SLT))				
1,539,358,780	16	1,255	3	30,910,324,302,400
820,993,140	25	1,506	4	30,910,391,721,000
488,835,164	36	1,757	5	30,919,801,793,328
314,154,767	49	2,008	6	30,910,,315,834,664
213,799,772	64	2,259	7	30,910,315,836,672
152,035,395	81	2,510	8	30,910,316,157,450
111,953,349	100	2,761	9	30,910,319,658,900
.....	.....	.....	.....	.....
1,877,266	1,600	10,291	39	30,910,315,829,644
1,744,266	1,681	10,542	40	30,910,315,829,644
.....	.....	.....	.....	.....

las ecuaciones generales de comportamiento de la SLT, han sido concretadas para el número de procesos para el cual se desea predecir el rendimiento de la aplicación. La figura 5.23 muestra un ejemplo de traza lógica escalada para la fase relevante 1 de una aplicación paralela. Como se puede apreciar en la figura, la LT4NC contiene información acerca de: el tipo de evento de comunicación (primitiva MPI), el origen y destino de cada evento de comunicación, el volumen de comunicación, el número de instrucciones de cómputo entre cada evento de comunicación y el peso de la fase.

A la hora de generar la LT4NC, la metodología hace especial hincapié en generar una traza lógica reducida, con el objetivo de que pueda ser manejada en memoria evitando los accesos al disco. La traza contiene únicamente el conjunto mínimo de parámetros intrínsecos necesarios de cada fase, utilizados para predecir el tiempo de comunicación y cómputo a medida que la aplicación escala.

Peso Fase 1: 2,800

Proceso	Fase	Tipo de primitiva	Origen	Destino	Volumen de comunicación (Bytes)	Cómputo (Número de instrucciones)
0	1	MPI_Irecv	0	1	4,000	756
0	1	MPI_Send	0	1	4,000	456
0	1	MPI_Wait	0	1	4,000	456,746,733
0	1	MPI_Irecv	0	2	2,000	975
0	1	MPI_Send	0	2	2,000	875
0	1	MPI_Wait	0	2	2,000	357,876,543

Figura 5.23: Traza Lógica Escalada (LT4NC) para la fase 1 del proceso 0

5.4. Validación experimental

En esta sección se valida el método propuesto para modelizar las fases relevantes de las aplicaciones científicas de paso de mensajes, mediante el cual generar la traza lógica escalable (SLT). Con el objetivo de evaluar la calidad del método, se ha seleccionado un amplio conjunto de aplicaciones científicas de paso de mensajes con diferentes patrones tanto de comunicación como de cómputo. Además, a la hora de seleccionar las aplicaciones científicas, se ha tenido en cuenta que puedan ser escaladas para un elevado número de procesos y que posean algoritmos representativos en el mundo de las aplicaciones científicas de paso de mensajes.

A continuación, se realiza una descripción general de las aplicaciones seleccionadas, las cuales serán utilizadas a lo largo de todo el trabajo a fin de evaluar las diferentes partes de la metodología:

- *BT (Block Tridimensional)* [59] y *SP (Scalar Pentadiagonal)* [59]: Resuelven sistemas sintéticos no linéales de PDEs, utilizando tres algoritmos diferentes que incluyen: Block tridiagonal, Scalar Pentadiagonal y Symmetric Succesive Over-Relaxation (SSOR).

- *CG (Conjugate Gradient)* [59]: Estima el valor propio más pequeño de una gran matriz escasamente simétrica, utilizando la iteración inversa con el método del gradiente conjugado, como una subrutina para resolver el sistema de ecuaciones lineales.

- *LU (Lower-Upper)* [59]: Resuelve un sistema de ecuaciones dinámico de fluidos (CFD), mediante la utilización del método Symmetric Gauss-Seide.

Tabla 5.3: Características del Cluster CAPITA

Cluster	Hardware	Software
CAPITA	8 nodos x 64 cores AMD Opteron 6262. 512 Cores en total 256GB de Memoria RAM por nodo. 2TB de Memoria Total Interconexión Infiniband QDR	Red Hat Linux OpenMPI 1.6.5 PAPI 5.1

- *Sweep3D* [60]: Aplicación de simulación avanzada y cómputo real. Es capaz de resolver ordenadas discretas de 1-grupo independientes del tiempo en coordenadas 3D cartesianas (XYZ). Esta geometría XYZ se representa por un grid rectangular. El cómputo y la memoria se incrementan substancialmente dependiendo de la carga de trabajo. El incremento de la carga de trabajo predomina en el número de iteraciones internas.

- *N-Body:* Aplicación que simula la interacción de un sistema dinámico de partículas bajo la influencia de diferentes fuerzas físicas, tales como la gravedad.

Como entorno experimental, se ha utilizado el Cluster CAPITA, las características del cual son mostradas en la tabla 9.1.

A la hora de llevar a cabo la validación experimental para el conjunto de aplicaciones seleccionadas, se ha seguido el siguiente workflow:

a) Para cada aplicación se han ejecutado cinco firmas para un reducido número de procesos. Cuatro firmas han sido utilizadas para generar el modelo y la última ha sido utilizada para validarlo antes de ser utilizado para predecir. Las firmas han sido ejecutadas con un mapping 1:1 (un proceso por core), ya que la pila de *software* del sistema está optimizada para producir un rendimiento máximo.

b) Una vez las firmas han sido ejecutadas, se generan las ecuaciones locales de comunicación para cada fase, con el objetivo de modelizar la ecuación general de comportamiento, mediante la cual predecir el patrón de comunicación.

c) Para cada fase se modeliza:

   - Las ecuaciones generales de comportamiento a partir de las ecuaciones locales.
   - Las ecuaciones de regresión del volumen de comunicación.
   - La ecuación de regresión del peso de cada fase.
   - Las ecuaciones para predecir el patrón de cómputo (número de instrucciones).

d) Una vez generado el modelo, se valida mediante la quinta firma ejecutada para comprobar si es correcto. Si los valores predichos son correctos, podrá ser utilizado para predecir el comportamiento de la aplicación a medida que escala. En caso de no ser correcto, se utiliza la quinta firma para mejorar el modelo y se ejecuta otra firma para un número mayor de procesos para volver a validarlo. Se considera que la ecuación general de comportamiento y la función de regresión del peso son correctas si son capaces de predecir el patrón de comunicación y el peso respectivamente sin error. Para el resto de parámetros de la aplicación (volumen de comunicación y patrón de cómputo), se considera que el modelo es correcto, si es capaz de predecir con un error inferior al 10 %. Una vez generado y validado el modelo, se genera la traza lógica escalable (STL).

e) Finalmente, la STL será utilizada para generar la traza lógica escalada (LT4NC) para un número N mayor de procesos.

Para validar la LT4NC generada de cada aplicación, se compara con la traza obtenida mediante la herramienta PAS2P. Para ello, se ejecuta la firma de la aplicación para el número de procesos para el cual se ha predicho la LT4NC. Debido a que la LT4NC es generada por proceso, nos centramos en mostrar la LT4NC para el proceso 0. Puesto que únicamente estamos interesados en esta etapa de la metodología en información lógica de la fase, las firmas utilizadas para validar la LT4NC son ejecutadas con un mapping x:1 (más de un procesos por core).

5.4.1. Modelo lógico para la aplicación BT

Con el objetivo de generar el modelo lógico de la aplicación (SLT), se ejecutaron las firmas de la aplicación para 16, 36, 64, 81 y 100 procesos, obteniendo un total de 6 fases relevantes. Se utilizaron las firmas de 16 a 81 procesos para generar la SLT y la última firma para validar el modelo. Mediante la SLT se generó la LT4NC para 1,024 procesos.

La figura 5.24 muestra los tiempos de ejecución para las diferentes firmas ejecutadas. Además, en la misma figura, se muestran también los tiempos de ejecución de la aplicación. Como se muestra la figura, utilizar la firma en lugar de la aplicación para caracterizar la aplicación, nos permite realizar una caracterización eficiente de la aplicación, en un reducido período de tiempo.

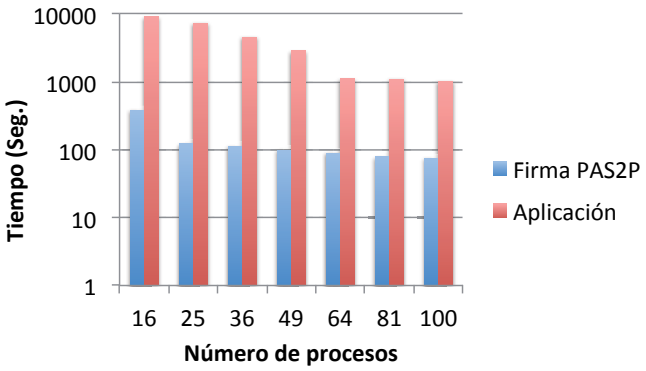


Figura 5.24: Tiempos de ejecución de las firmas de la aplicación versus tiempos de ejecución de la aplicación para BT

Una vez se han ejecutado las firmas, a partir del análisis de sus trazas se genera la SLT con las fases relevantes de la aplicación. La tabla 5.4 muestra las ecuaciones generales de comportamiento que definen el modelo lógico para la fase 1 y 2 de la aplicación. El modelo para el resto de fases puede encontrarse detallado en el anexo B de este trabajo.

Como se muestra en la tabla 5.4, debido al tipo de patrón de comunicación (estático), se utilizó el tipo de estructura EC2 para modelizar las ecuaciones generales de comportamiento y predecir el patrón de comunicación, a medida que la aplicación escala. Esta estructura ha sido simplificada y especificada para el proceso 0. Además, en la misma tabla se muestran las ecuaciones generales del volumen de comunicación, el peso y el patrón de cómputo (número de instrucciones). Todo este conjunto de ecuaciones tienen como parámetro de entrada el número de procesos a predecir.

La tabla 5.5 muestra la traza lógica escalada (LT4NC) generada a partir de la SLT y la obtenida mediante la ejecución de la firma con PAS2P para 1,024 procesos. Nos hemos centrado en mostrar las trazas para el proceso 0 de la fase 1 y 2. El resto de fases puede encontrarse en el anexo B de este trabajo. Como podemos apreciar en la tabla, el patrón de comunicación predicho de la LT4NC corresponde con el de la traza obtenida mediante PAS2P para las 2 fases. La figura 5.25, muestra el patrón de comunicación de la aplicación para 1,024 procesos, el cual fue obtenido a partir de las ecuaciones generales de comportamiento para las 6 fases de la aplicación.

El volumen de comunicación fue predicho con un error aproximado del 2 % para la

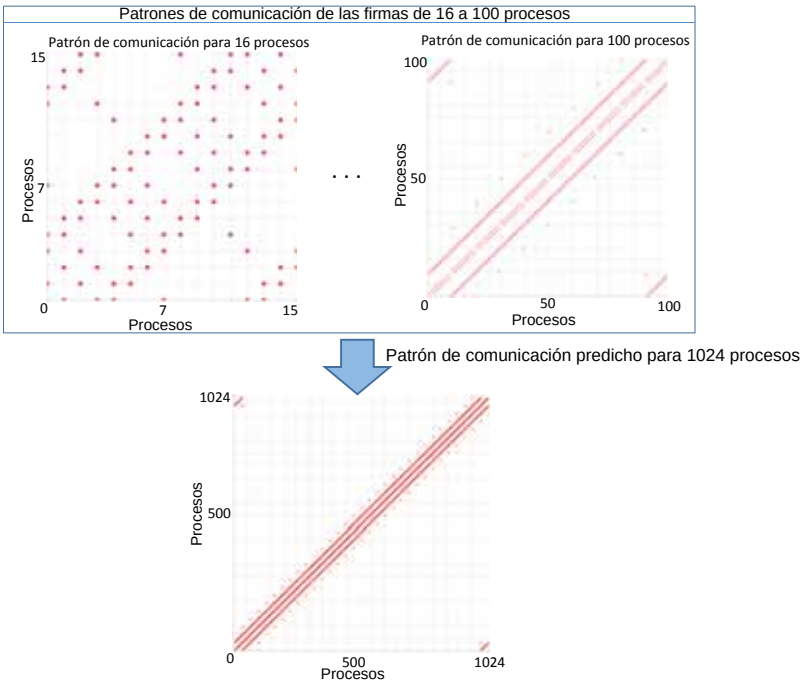


Figura 5.25: Patrón de comunicación predicho para la aplicación BT para 1,024 procesos

primera fase, y un 3 % para la segunda fase. El peso de la fase fue predicho con un modelo de regresión lineal con un error del 0 %, debido a su comportamiento determinista a la hora de incrementar el número de procesos de la aplicación. El número de instrucciones fue predicho con un error aproximado del 4 % para la primera fase y un 2.4 % para al segunda fase. Para ambas fases, todos los procesos ejecutaban el mismo número de instrucciones, por lo tanto únicamente se generó 1 IG. Los intervalos de cómputo entre las primitivas Isend - Irecv e Irecv - Wait de la traza lógica escalable tienen el número de instrucciones constante, a medida que la aplicación escala. Esto es debido a que son instrucciones condicionales de control, en lugar de cómputo. Por esa razón han sido predichas con un error de 0 %

Para las otras 4 fases de la aplicación, las cuales como se ha comentado anteriormente se encuentran detalladas en el anexo B de este trabajo, el patrón de comunicación fue predicho sin error para todas ellas. En la tabla 5.6 se presenta un resumen para estas fases con el error de predicción para el volumen de comunicación, el peso de la fase y el número de instrucciones. La tabla muestra el error máximo de predicción de todas las comunicaciones de la fase. Como se puede apreciar en la tabla, el peso de todas las fases

Tabla 5.4: Modelo lógico de las fases de las aplicaciones paralelas (SLT)

Fase ID	Ecuación General de Comunicación (Dest.)	Ecuación General del Volumen de comunicación	Ecuación General del Patrón de cómputo (Num. Insta.)
Modelo de la aplicación BT para las fases 1 y 2			
Fase 1	$1) f(n) = 1$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=699
	$2) f(n) = \sqrt{(n)} - 1$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=577
	$3) f(n) = \sqrt{(n)} - 1$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=((34,177,838,880,768/n)/w(n))
Ecuación General del peso para la fase 1 w(n)= (251*(sqrt(n)-1))+502			
Fase 2	$1) f(n) = n - \sqrt{(n)} + 1$	$y(n) = 7E + 06n^{(-0,997)}$	i(n)=699
	$2) f(n) = 17 + 2(\sqrt{(n)} - 9)$	$y(n) = 7E + 06n^{(-0,997)}$	i(n)=577
	$3) f(n) = 17 + 2(\sqrt{(n)} - 9)$	$y(n) = 7E + 06n^{(-0,997)}$	i(n)=((30,910,593,068,294/n)/w(n))
Ecuación General del peso para la fase 2 w(n)=(251*(sqrt(n)-1))+502			
Modelo de la aplicación Sweep3D para las fases 1 y 2			
Fase 1	1) if $log_2(n) mod 2 = 0 \rightarrow f(n) = \sqrt{(n)}$	$y(n) = 6E + 07n^{(-1)}$	i(n)=639
	if $log_2(n) mod 2! = 0 \rightarrow f(n) = \sqrt{(2 * n)}$		
	$2) f(n) = 1$	$y(n) = 6E + 07n^{(-1)}$	i(n)=((173,540,548,608,000/n)/w(n))
Ecuación General del peso para la fase 1 $w(n) = (200 * (log_2(n) - 3)) + 1000$			
Fase 2	$1) f(n) = 1$	$y(n) = 6E + 07n^{(-1)}$	i(n)=((172,958,266,163,200/n)/w(n))
	2) if $log_2(n) mod 2 = 0 \rightarrow f(n) = \sqrt{(n)}$	$y(n) = 6E + 07n^{(-1)}$	i(n)=((1,338,753,024,000/n)/w(n))
	if $log_2(n) mod 2! = 0 \rightarrow f(n) = \sqrt{(2 * n)}$		
Ecuación General del peso para la fase 2 $w(n) = (200 * (log_2(n) - 3)) + 1000$			
Modelo de la aplicación CG para la fase 1			
Fase 1	1..18)[$\#Comm.c(n) = log_2(n)/2$, $f(y)_{y=1..\#Comm.c(n)} = 2^{(y-1)}$]	$y(n) = 8E + 08n^{(-1)}$	3,6,9,12,15,18) (MPI_Wait Primitives)
			i(n)=((2,162,913,280,000/n)/w(n))
Ecuación General del peso para la fase 1 w(n)= 2500			
Modelo de la aplicación N-Body para la fase 1			
Fase 1	$1) f(n) = 1$	$y(n) = 1E + 07n^{(-1)}$	i(n)=420
	$2) f(n) = n - 1$	$y(n) = 1E + 07n^{(-1)}$	i(n)=((56,160,197,712,592/n)/w(n))
	$3) f(n) = n - 1$	$y(n) = 1E + 07n^{(-1)}$	i(n)=0
	$4) f(n) = 0$	$y(n) = 1E + 07n^{(-1)}$	i(n)= 13,366
Ecuación General del peso para la fase 1 w(n)= 9.375*n			

fue predicho sin error. El número de instrucciones fue predicho con un error máximo del 7.1 % para la fase 3, y el volumen de comunicación fue predicho con un error máximo próximo al 4 % para la fase 4.

5.4.2. Modelo lógico para la aplicación CG

Con la finalidad generar la SLT de la aplicación a partir de la cual generar la LT4NC para un número específico de procesos, se ejecutaron las firmas de la aplicación para 16, 32, 64, 128 y 256 procesos, obteniendo un total de 3 fases relevantes. La figura 5.26 muestra los tiempos de ejecución de las firmas ejecutadas. Se utilizaron las firmas de 16 a 128 procesos para generar el modelo lógico y la quinta firma para validarlo. A partir del modelo generado, se predijo la traza lógica escalada para 4,096 procesos.

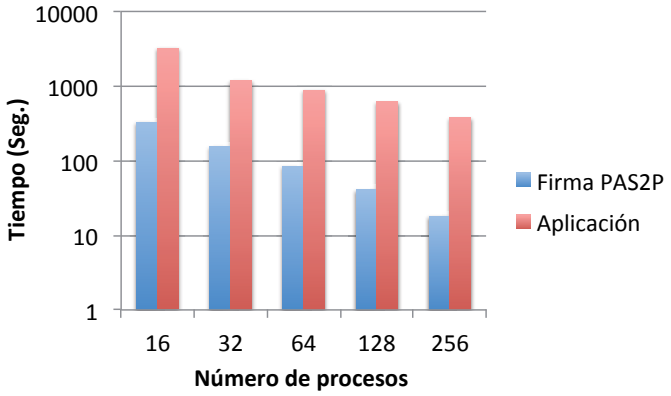


Figura 5.26: Tiempos de ejecución de las firmas de la aplicación versus tiempos de ejecución de la aplicación para CG

La tabla 5.5 muestra la traza lógica LT4NC y la obtenida al ejecutar la firma PAS2P para 4,096 procesos. Debido a la similitud entre las fases relevantes, nos hemos centrado en mostrar las trazas para el proceso 0 de la fase 1, encontrándose el resto de fases detallado en el anexo B de este trabajo.

La tabla 5.4 muestra las ecuaciones generales del modelo lógico de la fase 1 con las cuales se generó la LT4NC. Debido a que el tipo de patrón de comunicación es dinámico, ya que se expande comunicando con un número mayor de procesos, a medida que la aplicación escala, se utilizó la estructura EC1 para modelizar las ecuaciones generales del patrón de comunicación. Puesto que el patrón se expande comunicando con un número mayor de procesos, también se hace necesario, además de la ecuación general de comportamiento que predice el destino del mensaje, modelizar una ecuación general para predecir el número total de comunicaciones de la fase. En este caso, para predecir el patrón de comunicación, primero se aplica la ecuación para predecir el número total de comunicaciones de la fase, en este caso la ecuación $\#Comm.c(n) = log_2(n)/2$, la cual calcula el número de veces que la secuencia de la traza lógica escalable Isend, Irecv y Wait se repite en la fase. Aplicando esta ecuación para 4,096 procesos (n=4,096) como parámetro de entrada, obtenemos que la secuencia se repite 6 veces. Una vez conocemos el número de comunicaciones de la fase, se aplica la ecuación $f(y)_{y=1..\#Comm.c(n)} = 2^{(y-1)}$ para calcular el destino de cada secuencia, obteniendo la siguiente secuencia de destinos: 1, 2, 4, 8, 16 y 32.

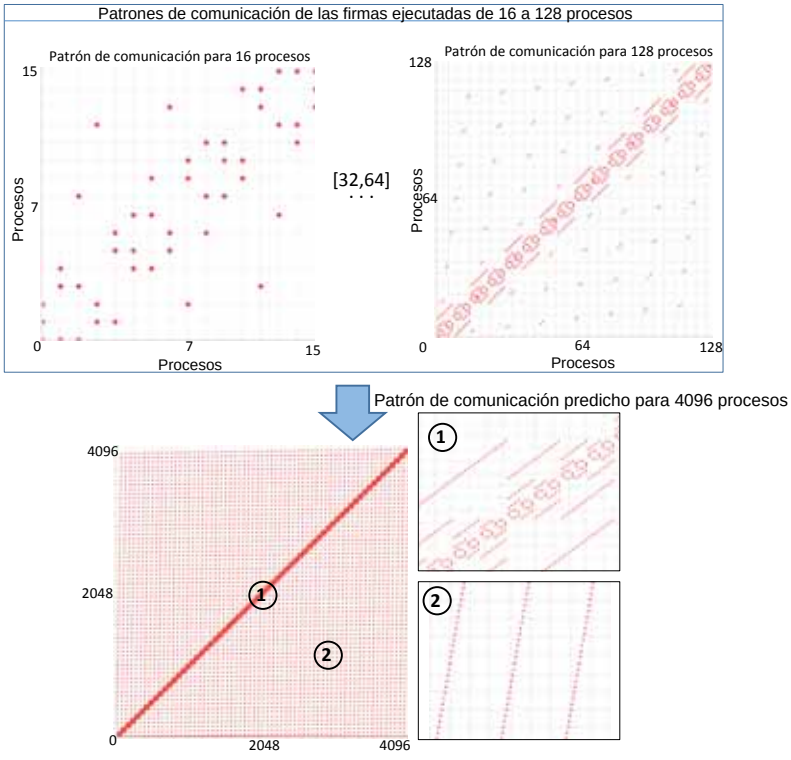


Figura 5.27: Patrón de comunicación predicho para la aplicación CG para 4,096 procesos

La figura 5.27 muestra el patrón de comunicación completo de la aplicación para 4,096 procesos, el cual fue generado a partir de las ecuaciones generales de las 3 fases relevantes de la aplicación. Debido al elevado número de procesos, se han hecho 2 ampliaciones para facilitar su visualización.

En cuanto a los demás parámetros del modelo lógico, el volumen de comunicación fue predicho con error aproximado del 4 %, el peso de la fase es constante a medida que la aplicación escala (2500), y finalmente el número de instrucciones fue predicho con un error aproximado del 0,18 %. Puesto que todos los procesos ejecutaban el mismo número de instrucciones, únicamente se ha generado 1 Instruction Group. Para los intervalos de cómputo posteriores a las primitivas Isend e Irecv de la traza lógica escalable, el número de instrucciones es constante a medida que se incrementa el número de procesos, ya que las instrucciones que hay entre las primitivas son instrucciones condicionales

Para las otras 2 fases de la aplicación, el patrón de comunicación fue predicho sin error para ambas fases. En la tabla 5.6, se presenta un resumen de estas 2 fases. Al igual que para la fase 1, el peso de las fases 2 y 3 es constante (2,500) a medida que

Tabla 5.5: Traza lógica escalada (LT4NC), obtenida aplicando las ecuaciones generales de la traza lógica escalable (STL) de tabla 5.4 para un número específico de procesos

Fase Ejecutada					Fase Predicha					Error de predicción	
Fase - Peso	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Insts.**	**Fase - Peso**	**Prim. MPI**	**Origen-Des.t**	**Volumen Com.**	**Num. Insts.**	**Volumen Com.**	**Num. Insts.**
Traza de la aplicación BT para el proceso 0 (n =1,024 procesos)											
	1)Isend	0-1	40,560	699		Isend	0-1	39,883	699	1.69 %	0 %
Fase 1-	2)Irecv	0 -31	40,560	577	Fase 1-	Irecv	0-31	39,883	577	1.69 %	0 %
8,283	3)Wait	0-31	40,560	3,879,431	8,283	Wait	0-31	39,883	4,029,554	1.69 %	3.73 %
	1)Isend	0-993	6,760	699		Isend	0-993	6,979	699	3.13 %	0 %
Fase 2-	2)Irecv	0-63	6,760	577	Fase 2 -	Irecv	0-63	6,979	577	3.13 %	0 %
8,283	3)Wait	0-63	6,760	3,733,902	8,283	Wait	0-63	6,979	3,644,336	3.13 %	2.39 %
Traza de la aplicación Sweep3D para el proceso 0 (n = 4,096 procesos)											
Fase 1-	1)Send	0-64	15,120	639	Fase 1-	Send	0-64	14,648	639	3.2 %	0 %
2,800	2)Recv	0-1	15,120	14,699,386	2,800	Recv	0-1	14,648	15,131,535	3.2 %	2.8 %
Fase 2-	1)Send	0-1	15,120	14,422,754	Fase 2-	Send	0-1	14,648	15,080,764	3.2 %	4.36 %
2,800	2)Recv	0-64	15,120	110,732	2,800	Recv	0 -64	14,648	116,730	3.2 %	5.1 %
Traza de la aplicación CG para proceso 0 (n = 4,096 procesos)											
	1)Isend	0-1	187,504	660		Isend	0-1	195,312	660	4 %	0 %
	2)Irecv	0-1	187,504	483		Irecv	0-1	195,312	483	4 %	0 %
	3)Wait	0-1	187,504	211,613		Wait	0-1	195,312	211,222	4 %	0.18 %
Fase 1-	...	...	...	...	Fase 1-	...	...	...	...	...	...
2,500	16)Isend	0-32	187,504	660	2,500	Isend	0-32	195,312	660	4 %	0 %
	17)Irecv	0-32	187,504	483		Irecv	0-32	195,312	483	4 %	0 %
	18)Wait	0-32	187,504	211,613		Wait	0-32	195,312	211,222	4 %	0.18 %
Traza de la aplicación N-Body para el proceso 0 (n = 4,096 processes)											
	ISend	0-1	2,342	420		ISend	0-1	2,441	420	4.0 %	0 %
Fase 1-	IRecv	0-4,095	2,342	362,127	Fase 1 -	IRecv	0-4,095	2,441	357,057	4.0 %	1.4 %
38,400	WaitAll	0-4,095	2,342	0	38,400	WaitAll	0-4,095	2,441	0	4.0 %	0 %
	WaitAll	0-0	2,342	13,366		WaitAll	0-0	2,441	13,366	4.0 %	0 %

la aplicación escala. El número de instrucciones fue predicho con un error máximo del 4 % para la fase 3, mientras que el volumen de comunicación fue predicho con un error máximo de 3 % para la fase 2.

5.4.3. Modelo lógico para la aplicación Sweep3D

Para la aplicación Sweep3D, se ejecutaron las firmas para 16, 32, 64, 128 y 256 procesos, obteniendo un total de 4 fases. A partir del modelo generado (STL) se predijo la LT4NC de 4,096 procesos. La figura 5.28 muestra el tiempo de ejecución de las firmas ejecutadas.

La tabla 5.5 muestra la LT4NC para 4,096 procesos y la traza lógica obtenida mediante PAS2P para las fase 1 y 2. El resto de fases está detallado en el anexo B de este trabajo.

Las ecuaciones del modelo lógico (SLT) para las fases 1 y 2 son mostradas en la tabla 5.4. Debido a que el tipo de patrón de comunicación de la aplicación es estático,

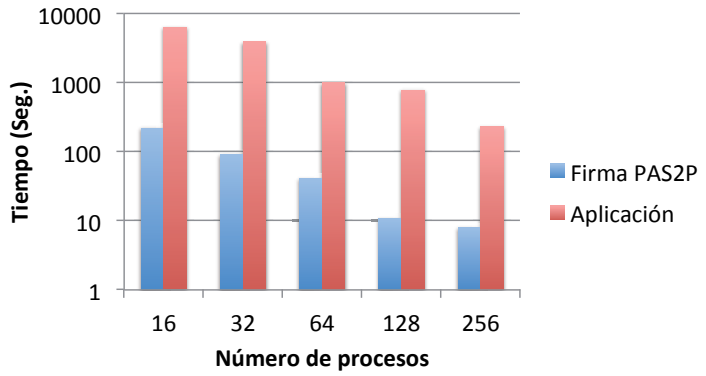


Figura 5.28: Tiempos de ejecución de las firmas de la aplicación versus tiempos de ejecución de la aplicación para Sweep3D

ya que corresponde a un patrón *wavefornt*, se utilizó la estructura EC2. Como se puede apreciar en la tabla 5.5, el patrón de comunicación corresponde con el ejecutado para ambas fases. El volumen de comunicación fue predicho con un error aproximado del 3.2 % para ambas fases. El peso de la fase fue predicho con un modelo matemático de regresión lineal con un error de predicción del 0 %. El número de instrucciones fue predicho con un error aproximado del 3 % para la primera fase y un 5 % para la segunda fase. Debido a que todos los procesos ejecutaban el mismo número de instrucciones, únicamente se creó 1 IG. Para la fase 1, el intervalo de cómputo posterior a la primitiva Send es constante, debido a que ejecuta el mismo número de instrucciones a medida que se incrementa el número de procesos de la aplicación.

Para las 2 fases restantes de la aplicación (fases 3 y 4), las cuales están detalladas en el anexo B, el patrón de comunicación fue predicho sin error para ambas fases. En la tabla 5.6, encontramos un resumen de estas 2 fases. Al igual que para las fases anteriores, el peso fue predicho sin error para las fases 3 y 4. El número de instrucciones fue predicho con un error máximo del 8.5 % para la fase 3, mientras que el volumen de comunicación, fue predicho con un error máximo del 3.20 % para las fases 3 y 4.

5.4.4. Modelo lógico para la aplicación N-Body

Para la aplicación N-Body, se ejecutó la firma de la aplicación para 16, 32, 64, 128 y 256 procesos, obteniendo 1 fase relevante. La figura 5.29 muestra los tiempos de ejecución de las firmas ejecutadas. Se utilizaron las firmas de 16 a 128 procesos para

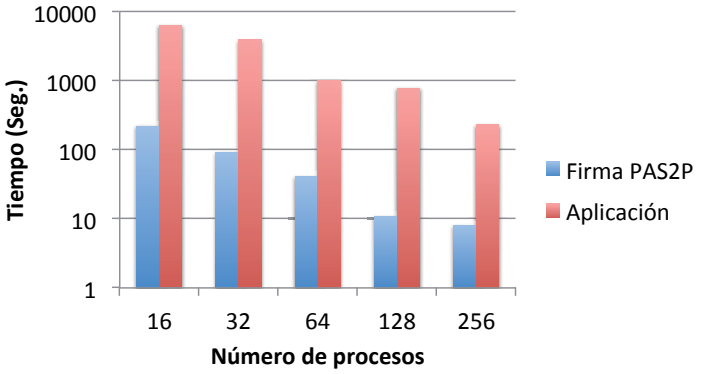


Figura 5.29: Tiempos de ejecución de las firmas de la aplicación versus tiempos de ejecución de la aplicación para N-Body

generar la SLT y la de 256 procesos para validar el modelo. A partir del modelo generado se predijo la Lt4NC para 4,096 procesos.

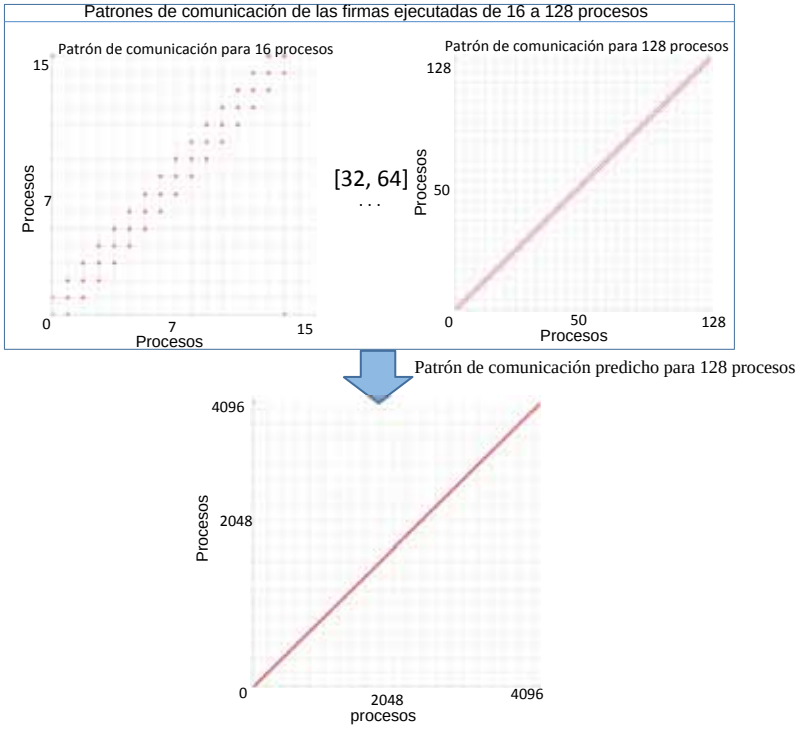


Figura 5.30: Patrón de comunicación predicho para la aplicación NBody para 4,096 procesos

La tabla 5.4 muestra las ecuaciones generadas para la fase relevante de la aplicación. Debido a que el patrón de comunicación es estático, ya que corresponde a un patrón *pipeline*, donde el proceso *'x'* comunica con el proceso *'x+1'*, hasta el último proceso, el cual comunica con el primer proceso de la aplicación, se utilizó la estructura EC2 para generar el modelo de las ecuaciones generales de comportamiento. La figura 5.30 muestra el patrón de comunicación para 4,096 procesos.

La tabla 5.5 muestra la LT4NC para 4,096 y la traza obtenida mediante PAS2P para el mismo número de procesos. La fase tiene 2 primitivas WaitAll consecutivas. La primera primitiva espera hasta que la primitiva Irecv libera su request, mientras que la segunda primitiva, espera hasta que el Isend libera su request. El segundo WaitAll es escrito en la aplicación con el objetivo de sincronizar la aplicación.

En cuanto a los otros parámetros del modelo, el volumen de comunicación fue predicho con un error aproximado del 4 %. El peso de la fase fue predicho con un error del 0 %. El número de instrucciones fue predicho con un error aproximado del 2 %. Debido a que todos los procesos tienen el mismo número de instrucciones, únicamente se generó 1 IG.

Tabla 5.6: Resumen del error de predicción para el resto de fases y aplicaciones analizadas

Fase ID	Volumen de. Comunicación	Número de Instrucciones	Peso de la fase
Resumen de fases para BT (Fases 3 a 6)			
Fase 3	7.1 %	1.18 %	0 %
Fase 4	1.69 %	4.04 %	0 %
Fase 5	3.13 %	0.95 %	0 %
Fase 6	1.69 %	0.10 %	0 %
Resumen de fases para CG (Fases 2 a 3)			
Fase 2	0.0 %	3.0 %	0 %
Fase 3	4.0 %	0.18 %	0 %
Resumen de fases para Sweep3D (Fases 3 a 4)			
Fase 3	3.2 %	8.5 %	0 %
Fase 4	3.2 %	5.4 %	0 %
Resumen de fases para SP (Fases 1 a 6)			
Fase 1	0.5 %	2.4 %	0 %
Fase 2	3.92 %	3.14 %	0 %
Fase 3	1.69 %	0.5 %	0 %
Fase 4	0.5 %	2.42 %	0 %
Fase 5	3.92 %	3.14 %	0 %
Fase 6	0.5 %	2.42 %	0 %
Resumen de fases para LU (Fases 1 a 2)			
Fase 1	0 %	0.8 %	0 %
Fase 2	0 %	7.20 %	0 %

6

Modelo de predicción del tiempo de cómputo

6.1. Introducción

Una vez que la traza lógica escalable (SLT) ha sido generada, y se ha concretado para un número específico de procesos, obteniendo la traza lógica escalada (LT4NC), tiene que ser completada con los tiempos de cómputo con el objetivo de generar la traza física escalada (ST4NP), la cual será utilizada para predecir el rendimiento de la aplicación (utilizando como métrica de rendimiento el tiempo de ejecución).

Con el objetivo de predecir el tiempo de cómputo, se utiliza un método por fase basado en modelos matemáticos de regresión. Una vez se ha predicho el tiempo de cómputo de cada fase, es añadido a la LT4NC con el objetivo de generar la ST4NP.

Tal y como se muestra en la figura 6.1, esta nueva traza es dependiente de la máquina, ya que además de toda la información lógica que contenía la LT4NC, contiene los tiempos físicos predichos de cómputo para cada fase de la aplicación.

Durante este capítulo se presentará y validará el método utilizado para predecir el tiempo de cómputo de cada fase.

6.2. Metodología para predecir el tiempo de cómputo

Con el objetivo de predecir el tiempo de cómputo de cada fase, se propone un método basado en modelos matemáticos de regresión, los cuales utilizan como datos de entrada

Peso Fase 1: 2,800

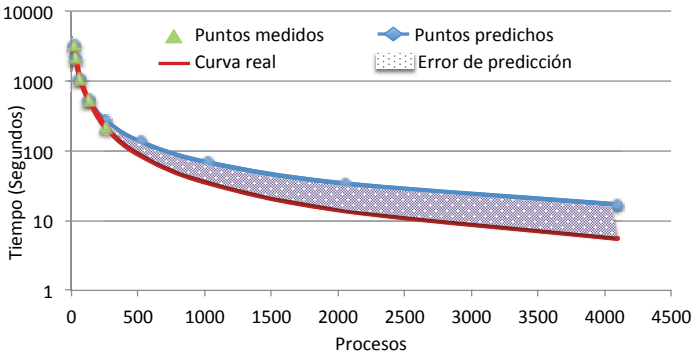
Proceso	Fase	Tipo de primitiva	Origen	Destino	Volumen de comunicación (Bytes)	Número de instrucciones de cómputo	Tiempo de cómputo (ns)
0	1	MPI_Irecv	0	1	4,000	756	4,000
0	1	MPI_Send	0	1	4,000	456	2,345
0	1	MPI_Wait	0	1	4,000	456,746,733	83,593,535
0	1	MPI_Irecv	0	2	2,000	975	7,533
0	1	MPI_Send	0	2	2,000	875	5,366
0	1	MPI_Wait	0	2	2,000	357,876,543	45,326,854

Figura 6.1: Traza física Escalada (ST4NP) para la fase 1 del proceso 0

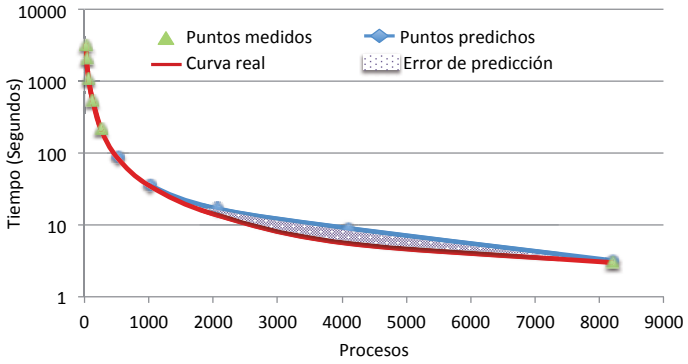
para generar el modelo, el tiempo de cómputo de cada fase del conjunto inicial de firmas, utilizadas en la etapa anterior para generar la SLT.

Aunque predecir el tiempo de cómputo por fase, en lugar del total de la aplicación, mejora considerablemente la calidad de predicción, ya que cada fase puede tener un comportamiento de cómputo diferente, el cual tiene que ser aproximado por una función de regresión específica, es importante considerar que los modelos de regresión pueden estar limitados por el alcance de la predicción, introduciendo un mayor error de predicción al predecir puntos distantes a los utilizados para generar el modelo. Estos errores de predicción se deben en gran parte tanto al conjunto limitado de muestras utilizadas para generar el modelo, como a la simplificación del mismo, el cual obvia parámetros de la aplicación y sistema, con el objetivo de generar modelos útiles en la práctica y rápidos de utilizar.

Con el objetivo de evitar este problema, y por lo tanto mejorar la calidad de predicción para un elevado número de procesos (puntos lejanos), proponemos un método el cual consiste en medir, para cada fase, un punto lejano, sin necesidad de ejecutar la aplicación para un elevado número de procesos, con el fin de ajustar el modelo de regresión de cómputo inicial (*Computational Regression Model initial (CRMi)*), utilizando este nuevo punto. Como se puede apreciar en la figura 6.2, si nos centramos en la primera figura 6.2.a, se puede observar como a partir del conjunto de puntos iniciales, obtenidos de la ejecución del conjunto inicial las firmas (de 16 a 128 procesos), se genera el modelo de predicción de cómputo inicial *(CRMi)* para una fase de una aplicación paralela. Si comparamos la curva real con la curva predicha, obtenida a partir de los puntos predichos mediante la utilización del modelo *CRMi*, podemos comprobar como a medida



(a) Modelo de regresión de Cómputo inicial (CRMi)



(b) Modelo de regresión de Cómputo ajustado(CRMc)

Figura 6.2: Modelo de regresión de Cómputo (CRM)

que nos alejamos de los puntos ejecutados, el error de predicción se va incrementando. Nótese que el eje vertical donde se representa el tiempo de cómputo está en escala logarítmica, por lo que a partir de 500 procesos, el modelo ya comienza a introducir un error de predicción considerable. Si por el contrario, tal y como se muestra en la figura 6.2.b, conseguimos medir un punto lejano, respecto de los puntos iniciales utilizados para generar el *CRMi*, como se puede apreciar, conseguiremos ajustar el modelo *CRMi*, obteniendo un nuevo modelo de regresión de cómputo, llamado modelo de regresión de cómputo ajustado (*Computational Regression Model Calibrated (CRMc)*), el cual mejora el alcance de predicción del modelo *CRMi*, permitiéndonos predecir puntos lejanos con un alto nivel de precisión.

El método propuesto para medir el punto lejano, se basa en realizar un cambio del *workload* de entrada de la aplicación, utilizando uno menor al original, el cual se

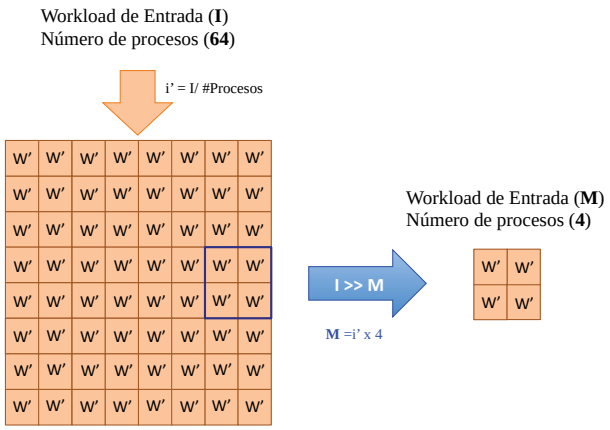


Figura 6.3: Distribución del número de instrucciones de cada proceso

ejecuta sobre un reducido número de procesos, con el fin de emular el mismo número de instrucciones y fallos de caché de cada proceso de la firma, que tendría la firma ejecutada con el *workload* original sobre un elevado número de procesos, con el objetivo de medir el tiempo de cómputo que tardan dichas instrucciones en ejecutarse.

Las fases de la firma son segmentos de código delimitados por dos eventos de comunicación, que ejecutan una función específica, por lo que si seleccionamos un nuevo *workload*, menor que el original, el cual es ejecutado sobre un número de procesos reducido (del orden de las centenas), y conseguimos obtener el mismo número de instrucciones y fallos de caché para cada proceso, que ejecutando la firma para el *workload* original sobre un elevado número de recursos (del orden de miles), conseguiremos emular el tiempo de cómputo de cada proceso para la fase. La figura 6.3 muestra un ejemplo del procedimiento. Tenemos una fase de una aplicación, la cual es ejecutada para 64 procesos con un *workload I*. Este *workload* es distribuido de forma uniforme entre todos los procesos de la fase, recibiendo cada uno un trabajo *i'*. Si ejecutamos la misma fase (mismo código) para 4 procesos, con un *workload M*, el cual tiene un tamaño total de *i'* x 4, los procesos están realizando el mismo trabajo (mismo número de instrucciones y fallos de caché), que ejecutando la firma de la aplicación para 64 procesos con el *workload I*, ya que los procesos de las 2 ejecuciones realizan la misma cantidad de trabajo (*i'*) .

A continuación, en la figura 6.4 se muestra el diagrama de flujo del método utilizado para predecir el tiempo de cómputo de cada fase de la aplicación.

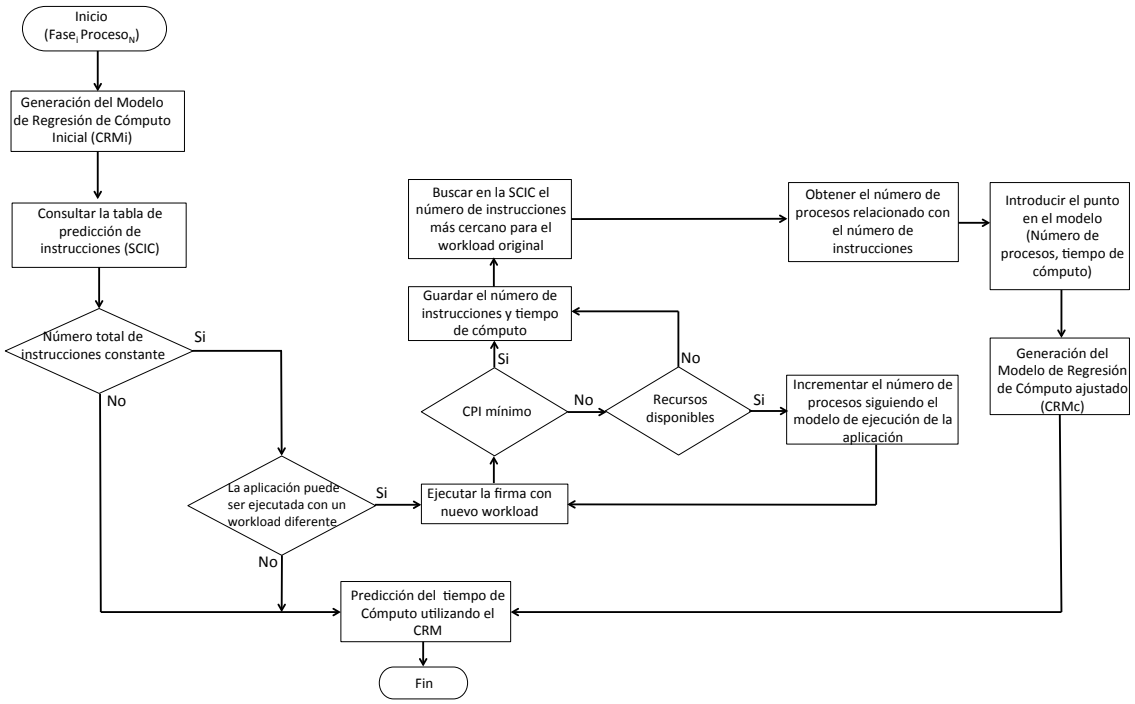


Figura 6.4: Diagrama de flujo para predecir el tiempo de computo para cada fase de la aplicación

1. A partir de los tiempos de cómputo del conjunto inicial de firmas ejecutadas, se genera el modelo de regresión de cómputo inicial *(CRMi)*. Este modelo será utilizado para predecir el tiempo de cómputo de la fase a medida que la aplicación escala.

2. A partir de la tabla de predicción de instrucciones (SCIC), se comprueba que el número total de instrucciones de la fase sea prácticamente constante a medida que la aplicación escala. Si esta premisa no se cumple, el método no puede ser aplicado y no podremos obtener un punto lejano, con el cual generar el modelo de regresión de cómputo ajustado *(CRMc)*. En estos casos, se utiliza el modelo *CRMi* generado en la etapa anterior. En caso de mantenerse constante el número total de instrucciones de la fase, se pasa al siguiente paso del algoritmo, con el objetivo de medir un punto lejano y generar el *CRMc*.

3. Con el objetivo de medir este nuevo punto, seleccionamos un nuevo *workload M*, menor al original, y ejecutamos la firma con esta nueva carga de trabajo para un

reducido número de procesos. Comenzamos ejecutando la firma para 16 procesos y, vamos escalando el número de procesos de la firma siguiendo el modelo de ejecución de la aplicación, hasta encontrar el CPI mínimo de la fase. Seleccionamos el número de procesos con el CPI mínimo por dos motivos principales, el primer motivo es que usando el CPI mínimo nos aseguramos que el *workload* de cada proceso es lo suficientemente pequeño como para obtener un punto lo suficientemente lejano para ajustar el *CRMi*. Por otro lado, esto nos permite situarnos en el punto de la curva donde los fallos de caché empiezan a ser prácticamente nulos y no tienen influencia en el modelo. Esto nos permite predecir puntos situados entre los puntos iniciales utilizados para generar el modelo y, el nuevo punto lejano, donde el efecto de la caché estará considerado en el modelo. Por otro lado, si predecimos puntos más allá del punto lejano, los efectos de la caché son prácticamente despreciables, por lo cual, nos permite predecir con gran precisión puntos para un elevado número de procesos, ya que a partir de ese punto la regresión tendrá un comportamiento lineal sin influencias de fallos de caché.

Dependiendo del tamaño del *workload* original, es posible que no se pueda obtener el CPI mínimo de la fase con el nuevo *workload M*, aún siendo menor al original, utilizando un número de recursos reducido. Puesto que una de las restricciones de la metodología es utilizar un reducido y limitado número de recursos, en caso de no haber conseguido obtener el punto con el CPI mínimo utilizando el máximo número de recursos disponibles para llevar a cabo la predicción del rendimiento de la aplicación, seleccionaremos el punto más lejano obtenido al llegar al límite de recursos, aún no siendo el punto ideal con el CPI mínimo. En este caso, el alcance de la predicción del modelo a partir del punto lejano estará limitada, debido a que aún existen fallos de caché a partir ese punto.

Una vez ejecutada la firma para este nuevo *workload M*, mediante la traza física de la firma *(SxPT)*, se obtiene el número de instrucciones y el tiempo de cómputo para la fase.

En algunos casos, por limitaciones de las aplicaciones paralelas, no es posible generar *workloads* de diferente tamaño al original. En estos casos, el punto lejano no puede ser obtenido y se tiene que utilizar el modelo *CRMi*, generado con los tiempos de cómputo del conjunto inicial de firmas ejecutadas.

4. Obtenido el número de instrucciones para el *workload M*, tiene que ser relacionado con el número de procesos para el *workload* original. Con el objetivo de predecir ese número de procesos, se utiliza la tabla de predicción de instrucciones (SCIC). Para ello, se busca en la SCIC el número de instrucciones por proceso más cercano al número de instrucciones obtenido mediante la ejecución de la firma para el *workload M*. Una vez obtenido, el número de procesos que corresponda a ese número de instrucciones será el utilizando para generar el *CRMc*.

5. Una vez que se ha obtenido el número de procesos para el *workload* original y el tiempo de cómputo, obtenido en el paso 3, se incorpora este nuevo punto al modelo inicial (*CRMi*), con el objetivo de generar el *CRMc*, obteniéndose un nuevo modelo de regresión con una nueva función de regresión.

6. Finalmente, se utiliza esta nueva función de regresión para predecir el tiempo de cómputo de la fase.

Una vez que se ha generado el modelo *CRM* (ya sea el modelo inicial *(CRMi)* o el modelo ajustado *(CRMc)*) de cada fase, se crea una tabla específica con los tiempos de cómputo predichos de cada fase, para el número de procesos para el cual se desea predecir el rendimiento de la aplicación, denominada Tabla de Predicción de Cómputo para N Cores *(Computational time for N cores (CT4NC))*. Mediante esta tabla y la traza lógica escalada (LT4NC), se genera la traza física escalada (ST4NP) para el número de procesos de la aplicación para el cual se desea predecir su rendimiento. Como se comentó anteriormente, esta traza es dependiente de la máquina, ya que contiene el tiempo de cómputo. Esta nueva traza contiene toda la información de la traza lógica escalada (CT4NC) más los tiempos de cómputo de cada fase.

6.3. Validación experimental

En esta sección se valida el método propuesto para predecir el tiempo de cómputo de cada fase de la aplicación. Con el objetivo de evaluar la calidad del método, se utiliza el mismo conjunto de aplicaciones científicas que el utilizado en la etapa de modelizado lógico de la aplicación: BT, SP, CG, LU, Sweep3D y N-Body. Además, se ha incorporado una nueva aplicación científica de paso de mensajes, QCDMPI [61] , la cual se describe a continuación:

Tabla 6.1: Características del Cluster CAPITA

Cluster	Hardware	Software
CAPITA	8 nodos x 64 cores AMD Opteron 6262. 512 Cores en total 256GB de Memoria RAM por nodo. 2TB de Memoria Total Interconexión Infiniband QDR	Red Hat Linux OpenMPI 1.6.5 PAPI 5.1

- *QCDMPI*: Aplicación científica que aplica el método de Montecarlo para simular sistemas *QCD* [62] y estudiar que aspectos perturban al sistema. La aplicación permite simular sistemas *QCD* para n-dimensiones.

A la hora de predecir el tiempo de cómputo para un elevado número de procesos, un punto importante a tener en cuenta a la hora de ejecutar las aplicaciones es su carga de trabajo (*workload*), ya que es necesario ejecutar la aplicación utilizando un *workload* que permita que la aplicación escale a medida que se incremente su número de procesos. Con el objetivo de cumplir esta premisa, las aplicaciones de la Suite NPB NAS (BT, CG, SP y LU) fueron ejecutadas utilizando el *workload* D, definido por defecto en la suite. La aplicación Sweep3D se ejecutó con un *workload* sweep.1500, definido por defecto en el conjunto de *workload*s de prueba de la aplicación. La aplicación N-Body fue ejecutada utilizando un *workload* de 3 millones de partículas y, finalmente, la aplicación QCDMPI, se ejecutó un *workload* QCD 3D con 2 millones de moléculas.

Según la aplicación y su capacidad para escalar, se ha predicho el tiempo de cómputo para diferente número de procesos. En este apartado se muestra la predicción del tiempo de cómputo para las aplicaciones BT y SP para 1,024 procesos, CG, LU y N-Body para 4,096 procesos y finalmente QCDMPI para 2,048 procesos.

Como entorno experimental, se ha utilizado el Cluster CAPITA, las características del cual son mostradas en la tabla 6.1.

A continuación, se muestra el desarrollo completo del método para predecir el tiempo de cómputo para la fase 1 de la aplicación BT. Como se comentó anteriormente, se ha predicho su tiempo de cómputo para 1,024 procesos.

Con el objetivo de generar el modelo de regresión de cómputo inicial *(CRMi)*, se ejecutó el conjunto de firmas de la aplicación para 16, 36, 64, 81 y 100 procesos, obteniendo un total de 6 fases relevantes.

Para el caso de la fase 1, a partir de las diferentes ejecuciones de la firma, se ha generado el *CRMi*, obteniendo la función predictora $y = 7 * 10^{10} * x^{-1,494}$, la cual será utilizada para predecir el tiempo de cómputo para un número mayor de procesos.

Tabla 6.2: Incremento del peso para la fase 1 de BT

Número de Procesos	Peso	Desplazamiento del Peso
4	251	1
9	753	2
16	1,255	3
25	1,506	4
36	1,757	5
49	2,008	6
64	2,259	7
81	2,510	8
100	2,761	9

Con el objetivo de generar el modelo de regresión de cómputo ajustado *(CRMc)*, se modelizó el comportamiento lógico de la fase, a medida que la aplicación aumenta su número de procesos. A partir de la modelización del patrón de cómputo, se obtuvo que todos los procesos tenían el mismo número de instrucciones, por lo tanto se generó un único Instruction Group (IG). Puesto que sólo tenemos un único IG, únicamente se generará una sola tabla de predicción de instrucciones (SCIC).

En relación a la modelización del peso de la fase, se utilizó un modelo de regresión lineal, generando la función predictora $y = 251 * x + 502$, donde la variable *'y'* es el peso predicho y la variable *'x'* es el desplazamiento. Se ha utilizado el desplazamiento, puesto que tal y como se muestra en la tabla 6.2, donde se presenta el incremento del peso para la fase 1 a medida que se aumenta el número de procesos de la aplicación, la distancia entre muestras (número de procesos de la firmas ejecutadas) no era uniforme, y se requirió hacer un proceso de linealización del modelo de regresión. Mediante este proceso de linealización se obtuvo un $R-square = 1$, representando el comportamiento exacto de crecimiento del peso de la fase, a medida que la aplicación escala.

A partir de la información lógica de la aplicación (patrón de cómputo y peso de la fase), se generó la tabla de predicción de instrucciones (SCIC), la cual es mostrada en la Tabla 6.3. Como se comentó en el capítulo anterior, dedicado al modelizado lógico de la aplicación, esta tabla está compuesta de 2 partes, una primera parte con datos medidos a partir del conjunto inicial de firmas ejecutadas en el sistema y, una segunda parte predicha, obtenida mediante el modelizado lógico de la aplicación. Como podemos comprobar, tanto en la parte medida como en la parte predicha, el número total de instrucciones se mantiene prácticamente constante a medida que la aplicación escala,

por lo que podemos buscar un punto lejano, con el objetivo de generar el modelo de regresión de cómputo ajustado *(CRMc)*. Para generar el *CRMc*, el punto lejano obtenido para un número elevado de procesos, será añadido a los puntos iniciales de las firmas ejecutadas como entrada del modelo.

Con el objetivo de obtener el punto lejano con el cual generar el *CRMc*, se ejecutó la firma de la aplicación utilizando un *workload* clase B. Este nuevo *workload*, viene definido en el conjunto de *workloads* de la aplicación, el cual es menor que el *workload* original clase D (tamaño de la matriz 16 veces menor). A fin de obtener el CPI mínimo de la fase, se realizaron diferentes ejecuciones de la firma de la aplicación. Se empezó ejecutando la aplicación para 16 procesos y se fue incrementando el número de procesos de la aplicación, siguiendo su modelo de ejecución, el cual permite ejecutar la aplicación utilizando un número de procesos cuadrático. En la tabla 6.4 se muestra la información de las ejecuciones realizadas para la fase 1 utilizando este nuevo *workload*. La tabla contiene información de cada ejecución de la firma sobre: el número de procesos para los cuales fue ejecutada la firma, el número de instrucciones por proceso, los ciclos de cómputo de la fase, los fallos de caché del último nivel de memoria (en el sistema utilizado para realizar la evaluación experimental corresponde al nivel de caché L2), el CPI (obtenido indirectamente de los ciclos y las instrucciones), y finalmente el tiempo de cómputo en nanosegundos.

Como podemos comprobar en la tabla, para 81 procesos obtenemos un CPI de 0.719. Conocemos que este es el CPI mínimo de la fase, ya que al aumentar el número de procesos a 100, obtenemos el mismo CPI que para 81 procesos, siendo los fallos de caché insignificantes. Una vez hemos obtenido el CPI mínimo de la fase, seleccionamos el número de instrucciones (1,759,558) y tiempo de cómputo (790,954 ns) de la última ejecución (100 procesos),

Una vez hemos obtenido el numero de instrucciones y el tiempo de cómputo, el siguiente paso es relacionar el número de instrucciones obtenido para la ejecución de la firma con el *workload* B, con el número de procesos para el *workload* D (*workload* original). Con el objetivo de realizar esta relación, se busca en la tabla 6.3 (tabla SCIC) el número de instrucciones más cercano al obtenido usando el *workload* B (1,759,558).

Como se puede apreciar en la tabla 6.3, el número más cercano de instrucciones es 1,744,266, el cual tiene una diferencia del 0.87 % con el número de instrucciones del

Tabla 6.3: Tabla de predicción de instrucciones (SCIC) para la fase 1 de BT con el workload D

Número de Instrucciones	Número de Procesos	Peso	Desplazamiento del Peso	Número total de Instrucciones
Valores medidos (Conjunto inicial de Firmas)				
1,539,358,893	16	1,255	3	30,910,326,571,440
820,993,294	25	1,506	4	30,910,397,519,100
488,835,328	36	1,757	5	30,919,812,166,656
314,154,733	49	2,008	6	30,910,312,489,336
213,799,791	64	2,259	7	30,910,318,583,616
152,035,493	81	2,510	8	30,910,336,081,830
111,953,973	100	2,761	9	30,910,491,945,300
Valores predichos (Generados con el modelo lógico de la fase)				
1,539,358,780	16	1,255	3	30,910,324,302,400
820,993,140	25	1,506	4	30,910,391,721,000
488,835,164	36	1,757	5	30,919,801,793,328
314,154,767	49	2,008	6	30,910,315,834,664
213,799,772	64	2,259	7	30,910,315,836,672
152,035,395	81	2,510	8	30,910,316,157,450
111,953,349	100	2,761	9	30,910,319,658,900
84,813,133	121	3,012	10	30,910,315,829,644
65,784,545	144	3,263	11	30,910,315,829,644
.....	.....	.....	.....	.....
1,877,266	1,600	10,291	39	30,910,315,829,644
1,744,266	**1,681**	**10,542**	**40**	**30,910,315,829,644**
1,623,539	1,764	10,793	41	30,910,315,829,644
1,513,701	1,849	11,044	42	30,910,315,829,644

Tabla 6.4: Ejecuciones de la firma para BT usando la clase B

Número de Procesos	Número de. Instrucciones	Número de ciclos	Fallos cache L2	CPI	Tiempo de Cómputo (nsec.)
16	34,356,426	27,621,512	20,241	0.803	17,263,445
25	17,836,224	14,324,977	10,275	0.803	8,953,111
36	9,915,863	7,861,580	5,479	0.792	4,913,488
64	4,027,761	3,104,163	1,467	0.770	1,940,102
81	2,392,021	1,721,033	865	0.719	1,075,646
100	**1,759,558**	**1,265,527**	**695**	**0.719**	**790,954**

Tabla 6.5: Error de predicción para las fases de la aplicación BT

Fases de BT (Predicción para 1.024 procesos)				
Fase ID	Tiempo real de cómputo (ns)	Tiempo predicho de cómputo (ns)	Error de Predicción (%)	Ecuación de regresión utilizada
Fase 1	1,941,784	1,982,934	2.11 %	$y = 9 * 10^10 * x^{-1,547}$
Fase 2	35,960,361	39,114,740	8.77 %	$y = 1 * 10^{11} * x^{-1,132}$
Fase 3	165,862,020	160,857,620	3.01 %	$y = 2 * 10^{11} * x^{-1,028}$
Fase 4	451,140	480,522	6.51 %	$y = 3 * 10^{10} * x^{-1,593}$
Fase 5	2,214,673	2,096,107	5.35 %	$y = 8 * 10^{10} * x^{-1,522}$
Fase 6	36,062,311	39,114,740	8.46 %	$y = 1 * 10^{11} * x^{-1,132}$

workload B (1,759,558). Este número de instrucciones corresponde a la ejecución de la firma utilizando el *workload* D para 1,681 procesos.

Obtenido el tiempo de cómputo (790,954 ns) mediante la ejecución del *workload* B para 100 procesos, y el número de procesos (1,681) para el *workload* original D, generamos el *CRMc* mediante los puntos del conjunto inicial de firmas ejecutadas, y este nuevo punto, obtenido a partir de la ejecución de la firma para el *workload* B. A partir del modelo *CRMc* generado, se obtiene la función predictora $y = 9 * 10^{10} * x^{-1,547}$, donde la variable *'y'* es el tiempo de cómputo y la variable *'x'* es el número de procesos a predecir.

Esta función predictora será utilizada para predecir el tiempo de cómputo para 1,024 procesos. Como podemos ver en la tabla 6.5, donde se muestra para cada fase de la aplicación: el tiempo medido de cómputo, obtenido mediante la ejecución de la firma de PAS2P en el sistema, el tiempo de cómputo predicho, obtenido mediante la función de regresión del modelo *CRMc*, y el error de predicción entre el tiempo de cómputo medido y el predicho, el error de predicción obtenido utilizando la función predictora del modelo *CRMc* para la fase 1, es aproximadamente del 2.11 %.

A continuación, se analiza el error de predicción utilizando el modelo *CRMi* y el modelo *CRMc*. La tabla 6.6 muestra el error de predicción utilizando los 2 modelos para la misma fase (Fase 1).

Como se aprecia en la tabla, utilizando el modelo *CRMi*, obtenemos un error de predicción del 14.65 %, debido a la distancia entre los puntos utilizados para generar el modelo y el punto a predecir. Mediante la utilización del modelo *CRMc*, el error de predicción se reduce a un 2 %, obteniendo una mejora en el modelo de aproximadamente el 85 %.

En la tabla 6.5, se muestra el error de predicción para resto de fases de la aplicación BT. Como se muestra en la tabla, para todas las fases de la aplicación, el error está por debajo del 9 %, obteniendo un error máximo de predicción para la fase 2 del 8.77 %.

En la tabla 6.7 se muestra el error de predicción para el resto de aplicaciones científicas utilizadas. Para las aplicaciones CG, LU y N-body se ejecutaron las firmas de la aplicación de 16 a 256 procesos y se predijo el tiempo de cómputo para 4,096 procesos. Para la aplicación QDIM, se ejecutaron las firmas de la aplicación de 16 a 256 procesos, y se predijo el tiempo de cómputo para 2,048 procesos, mientras que para la aplicación SP, se ejecutaron las firmas de 16 a 100 procesos y se predijo su tiempo de cómputo para 1,024 procesos.

Para todos los casos evaluados, el número de instrucciones de las fases se mantuvo constante a medida que la aplicación escalaba, por lo que se pudo obtener el punto lejano para todos los casos. Para todas las aplicaciones testeadas el error de predicción del tiempo de cómputo está por debajo del 9 %, obteniendo un error máximo de predicción del 8.99 % para la fase 3 de la aplicación CG.

Tabla 6.6: Comparación de los modelos de regresión de cómputo (CRM)

Modelo Utilizado	Tiempo de computo medido (nsec.)	Tiempo de cómputo Predicho (nsec.)	Error de Predicción	Ecuación de regresión utilizada
CRMc	1,941,754	1,982,934	2.11 %	$y = 9 * 10^10 * x^{-1,547}$
CRMi	1,941,754	2,226,947	14.65 %	$y = 7 * 10^{10} * x^{-1,494}$

Tabla 6.7: Error de predicción para las fases de las aplicaciones utilizadas

Fase ID	Tiempo real de cómputo (ns)	Tiempo predicho de cómputo (ns)	Error de Predicción (%)	Ecuación de regresión utilizada
Fases de CG (Predicción para 4.096 procesos)				
Fase 1	2,698,523	2,796,674	3.63 %	$y = 2 * 10^{10} * x^{-1,067}$
Fase 2	137,928	149,985	8.74 %	$y = 3 * 10^{7} * x^{-0,637}$
Fase 3	344,297	375,251	8.99 %	$y = 2 * 10^{7} * x^{-0,478}$
Fase 4	601,238	562,501	6.44 %	$y = 8 * 10^{7} * x^{-0,596}$
Fases de LU (Predicción para 4.096 procesos)				
Fase 1	51,549	49,647	3.83 %	$y = 2 * 10^{8} * x^{-0,998}$
Fase 2	34,645	32,573	6.36 %	$y = 4 * 10^{8} * x^{-1,132}$
Fases de SP (Predicción para 1.024 procesos)				
Fase 1	165,110,327	163,103,109	1.21 %	$y = 4 * 10^{11} * x^{-0,926}$
Fase 2	561,715	598,590	6.56 %	$y = 4 * 10^{10} * x^{-1,635}$
Fase 3	240,359	260,203	8.25 %	$y = 2 * 10^{10} * x^{-1,623}$
Fases de N-BODY (Predicción para 4.096 procesos				
Fase 1	449,150	430,921	4.23 %	$y = 2 * 10^{10} * x^{-0,975}$
Fases de QCDMPI (Predicción para 2.048 procesos)				
Fase 1	16,428,976	17,724,522	7.88 %	$y = 3 * 10^{13} * x^{-2,039}$
Fase 2	26,478,907	25,170,124	4.94 %	$y = 5 * 10^{10} * x^{-0,996}$

7

Modelo de predicción del tiempo de comunicación

7.1. Introducción

Una vez se ha generado la traza física escalada (ST4NP) para el número total de procesos para el cual se desea predecir el rendimiento de la aplicación *(NPROCS)*, el objetivo es procesarla en el sistema destino a fin de obtener el tiempo de comunicación para cada fase de la aplicación. Para ello, se ha desarrollado una herramienta, llamada *Synthetic Signature (SS)*, la cual recibe como entrada la ST4NP. Una vez leída la traza, la herramienta SS procesará sus eventos de comunicación y cómputo en el sistema paralelo, con el objetivo de obtener el rendimiento de la aplicación, utilizando un conjunto reducido de recursos.

Puesto que conocemos el origen y destino de los mensajes, sus volúmenes de comunicación y el tiempo de cómputo entre primitivas, el objetivo de la herramienta SS es medir el tiempo de comunicación de los mensajes de cada fase. Aunque la ST4NP contiene el tiempo predicho de cómputo y únicamente se requiere conocer el tiempo de comunicación de cada fase, para obtener su tiempo de ejecución, además de los eventos de comunicación, la herramienta SS considerará también el tiempo de cómputo, con el objetivo de mantener la relación cómputo-comunicación de cada fase, ya que si sólo ejecutáramos los eventos de comunicación, no estaríamos representando el comportamiento de la fase tal y como sucede en aplicación real, afectando a los tiempos de envío y recepción de los eventos de comunicación.

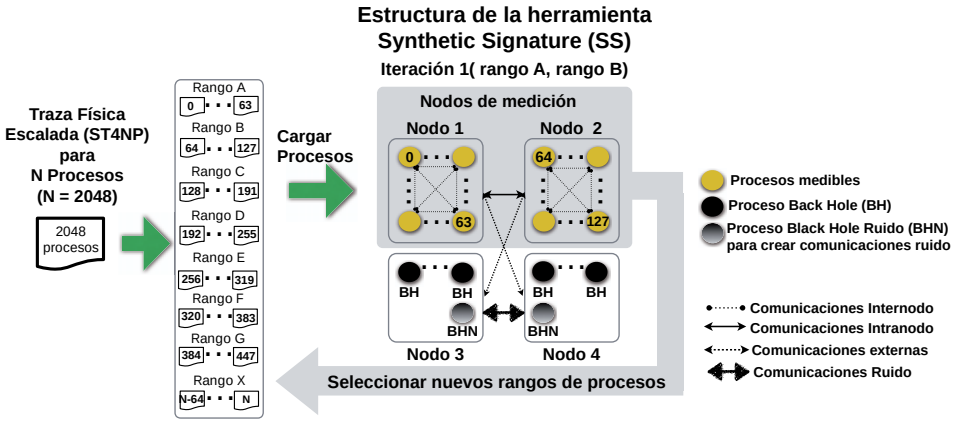


Figura 7.1: Estructura de la herramienta Synthetic Signature (SS)

Puesto que uno de los objetivos de la metodología es utilizar un número de recursos limitado, la ST4NP será procesada en el sistema por partes. Para ello, se realiza un proceso iterativo conducido por traza, donde la ST4NP es procesada en el sistema destino por rangos de procesos, en función del número de cores de los nodos del sistema. Debido a que ejecutamos rangos de procesos, este proceso se repite de forma iterativa hasta que todos los procesos que componen la ST4NP han medido sus eventos de comunicación.

Debido al requerimiento de utilizar un número de recursos limitados, tal y como se muestra en la figura 7.1, se ha definido una configuración del sistema con 4 nodos sobre el cual se ejecutará la herramienta SS. Dos nodos (nodo 1 y nodo 2), llamados *nodos de medición*, serán dedicados a procesar la traza ST4NP, para realizar las mediciones en el sistema destino, y los otros dos nodos (nodo 3 y nodo 4), llamados nodos *Black Hole*, serán dedicados a enviar o recibir las comunicaciones de cuyo proceso origen o destino no se encuentre entre el rango de procesos a ser medidos en la iteración actual.

A continuación, con el objetivo de explicar el funcionamiento de la herramienta SS, se han creado las siguientes definiciones, las cuales serán utilizadas a lo largo de todo el capítulo.

- *Rango de Procesos Escalados (Range of Scaled Traces Processes (RangesST))*: Conjunto de procesos que serán asignados en los nodos de medición con el objetivo de medir el tiempo de comunicación de sus eventos. El tamaño del rango de procesos vendrá definido en función del número de cores de los nodos de medición (nodos del sistema donde se va a predecir (*target*)). La figura 7.1 muestra un ejem-

plo de como se agrupan los procesos de la traza ST4NP en rangos, considerando que el sistema tiene nodos de 64 cores.

Con el objetivo de medir las comunicaciones en el sistema, utilizando un conjunto limitado de recursos, la herramienta SS considera de 3 tipos de procesos diferentes:

- *Proceso medible (Measurable Process (MeasurableP))*: Proceso que carga su ST4NP con el objetivo de medir sus eventos de comunicación. Se encuentran ubicados en los nodos de medición.
- *Proceso Black Hole (Black Hole Process (BH))*: Procesos ubicados en los nodos *Black Hole*, los cuales se comunican con los procesos medibles cuando el origen o el destino de un evento de comunicación de un proceso medible no se encuentra en el rango de procesos medibles. Las comunicaciones entre el proceso medible y el proceso BH no son medidas, únicamente ejecutadas con el objetivo de resolver las dependencias del evento de comunicación del proceso medible.
- *Procesos Ruido Black Hole (Black Hole Noise Processes (BHN))*: Par de procesos ubicados en los nodos *Black Hole*, con el objetivo de generar el mismo número de comunicaciones en tránsito que tendría la fase real, ejecutada con todos los recursos. Cada nodo *Black Hole* posee un proceso BHN.

Además de los tipos de procesos, tal y como se muestra en la figura 7.1, la herramienta SS distingue entre los siguientes tipos de comunicaciones:

- *Comunicación Intranodo (Intranode Communication)*: Comunicación entre 2 procesos medibles situados en el mismo nodo de medición (nodo 1 o nodo 2). Estas comunicaciones son medidas mediante timers proporcionados por el sistema operativo. La figura 7.1 muestra un ejemplo de esta comunicación.
- *Comunicación Internodo (Internode Communication)*: Comunicación utilizando la red de interconexión entre dos procesos medibles situados en nodos de medición distintos. La figura 7.1 muestra un ejemplo de esta comunicación.
- *Comunicación de Control (Control message)*: Existen dos tipos diferentes de mensajes de control. El primer tipo, mostrado en el paso A de la figura 7.2, ocurre cuando una comunicación de un proceso medible (MeasurableP), está esperando

por una recepción cuyo origen no se encuentra en el rango de procesos medibles RangesST. Con el objetivo de recibir el mensaje, el proceso MeasurableP envía un mensaje al proceso BH, especificándole el tipo de mensaje que espera recibir. Una vez el proceso BH recibe el mensaje, genera un mensaje con las mismas características del que espera recibir el proceso MeasurableP (tipo de mensaje, tamaño de buffer, tipo de dato y Tag del mensaje), el cual es enviado al proceso MeasurableP, tal y como se muestra en el paso B de la figura, con el objetivo de ejecutar el evento de recepción y resolver la dependencia de comunicación del evento.

El segundo tipo de mensaje de control, como se muestra en la figura 7.3, sucede cuando un proceso MeasurableP envía un mensaje a un proceso BH (Paso A), si es el primer mensaje que llega al nodo *Black Hole*, el proceso BH le envía un mensaje de control al proceso ruido Black Hole (BHN) (Paso B), con el objetivo de activar las comunicaciones ruido entre los procesos BHN (paso C).

- *Comunicaciones Externas (External Communications (EComms))*: Comunicación entre un Proceso MeasurableP y un Proceso BH, con el objetivo de resolver las dependencias de comunicación de la fase. Este tipo de comunicaciones son ejecutadas con el objetivo de mantener el comportamiento real de la fase, pero no medidas, serán medidas cuando tanto el origen como el destino del evento de comunicación se encuentren entre los rangos de procesos medibles.

- *Comunicaciones ruido (Noise Communications)*: Comunicaciones entre 2 procesos BHN. Estos procesos generan el mismo número de comunicaciones en tránsito que aparecerían en la aplicación si fuera ejecutada con todos los recursos. El objetivo de generar estas comunicaciones es serializar la entrada de comunicaciones al nodo, tal y como sucedería si ejecutáramos la aplicación utilizando todos los recursos del sistema.

Una vez se han presentado los componentes de la herramienta SS, se explica su funcionamiento para obtener el tiempo de comunicación. En la figura 7.4 se presenta el diagrama general de flujo del método utilizado para obtener el tiempo de comunicación de cada fase.

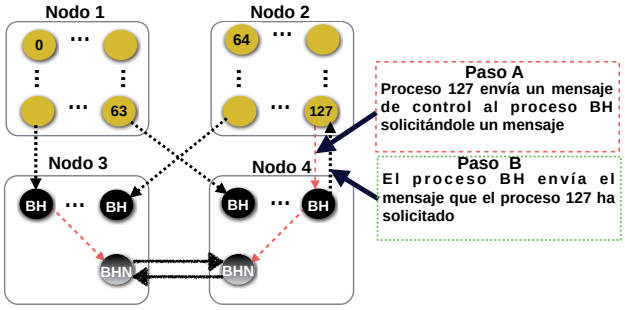


Figura 7.2: Proceso medible envía un mensaje de control al proceso BH solicitándole un mensaje

7.2. Selección de parámetros de ejecución y carga de la traza física escalada

Cómo se puede apreciar en el diagrama de la figura 7.4, la herramienta SS permite procesar la traza física escalada (ST4NP) por partes, de forma iterativa, utilizando un número reducidos de recursos (cores), o de forma completa, utilizando tantos cores como procesos contenga la ST4NP.

Con el objetivo de medir las comunicaciones utilizando un conjunto limitado de recursos, se ha de proporcionar a la herramienta SS, encargada de procesar la ST4NP, un conjunto de parámetros para poder realizar la ejecución correctamente. El primer parámetro es el número total procesos *(NPROCS)* para el cual ha sido generada la ST4NP.

Puesto que no todos los procesos son ejecutados al mismo tiempo, como segundo parámetro se ha de proporcionar a la herramienta SS el rango de procesos (RangesST)

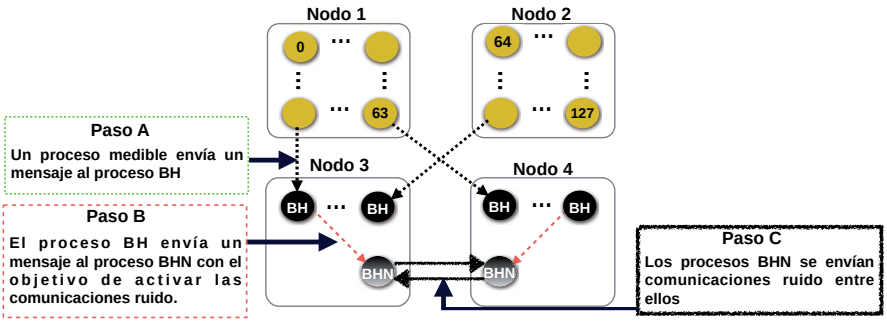


Figura 7.3: Activación de las comunicación ruido por parte de los procesos BHN

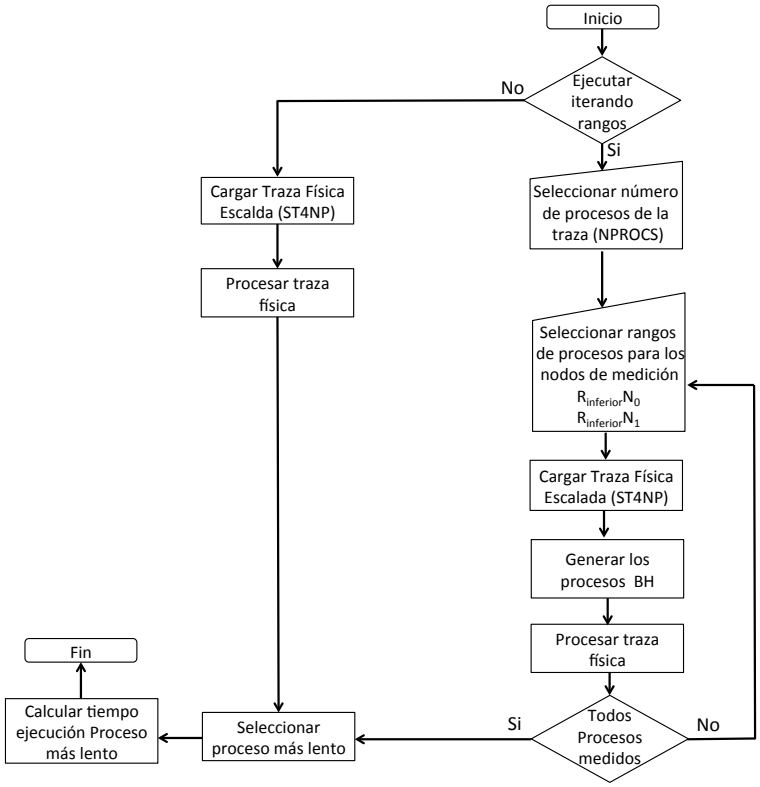


Figura 7.4: Diagrama general de flujo del método de ejecución de la herramienta SS

para los cuales se desea realizar la medición.

Para cada iteración, es necesario proporcionarle a la herramienta SS dos RangesST a ser medidos por los nodos de medición, el primer rango (RangesST 1), serán los procesos a ser medidos cargados en el nodo de medición 1, y el segundo rango (RangesST 2) serán los procesos a ser medidos cargados en el nodo de medición 2.

El tamaño de los RangesST se define en función del número de cores de los nodos de medición. Por ejemplo, si tenemos una ST4NP generada para 1024 procesos, si utilizamos un sistema con 64 cores por nodo, generaríamos rangos de 64 procesos cada uno, obteniendo un total de 16 RangesST (Proceso inicial del rango - Proceso final del rango): 0-63, 64-127, 128-192, ..., 959-1023.

A la hora de cargar los RangesST seleccionados en el sistema, es necesario seleccionar los nodos de medición considerando el mapping real de la aplicación, tal y como estaría mapeada la aplicación en el sistema si fuera ejecutada con todos los recursos del sistema.

Es necesario ser tan riguroso a la hora de seleccionar los nodos de medición, pues-

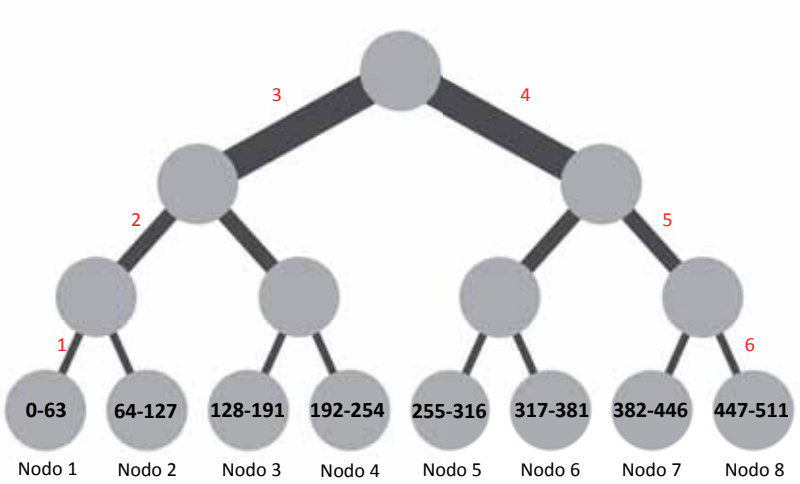


Figura 7.5: Mapping real de la aplicación en el sistema

to que si seleccionamos 2 nodos cualesquiera del sistema, estaríamos modificando las distancias físicas de la red de interconexión entre los procesos, no representando el comportamiento de la aplicación en el sistema, obteniendo un tiempo de comunicación diferente al de la aplicación original ejecutada con todos los recursos, ya que estamos modificando las distancias entre el origen y destino de los mensajes.

A continuación, se muestra un ejemplo de cómo mapear en el sistema los RangesST en los nodos de medición. La figura 7.5 muestra el mapping de una aplicación ejecutada para 512 procesos, la cual es mapeada en un sistema de 8 nodos, de 64 cores por nodo. Como se puede apreciar, se realiza un mapping secuencial asignando cada proceso de la aplicación a un core del sistema de forma incremental.

Si queremos medir las comunicaciones para los rangos de procesos de 0 a 63 (RangeST 1) y 447 a 511 (RangeST 2), los cuales, como se muestra en la figura se encuentran a una distancia de 6 saltos, es necesario seleccionar como nodos de medición, del nodo 1 al nodo 4 para el RangeST 1, y el nodo 5 al nodo 8 para el RangeST 2, ya que son el conjunto de nodos que respetan una distancia de 6 saltos entre el RangeST 1 y el RangeST 2. En la figura 7.6 se muestran los nodos seleccionados para ejecutar el rango de procesos, como se puede apreciar, se han seleccionado el nodo 1 y nodo 8. Con el objetivo de conocer la topología de la red de interconexión y poder seleccionar el conjunto de nodos medibles a utilizar, se utiliza la herramienta *Portable Network Locality (netloc)* [63].

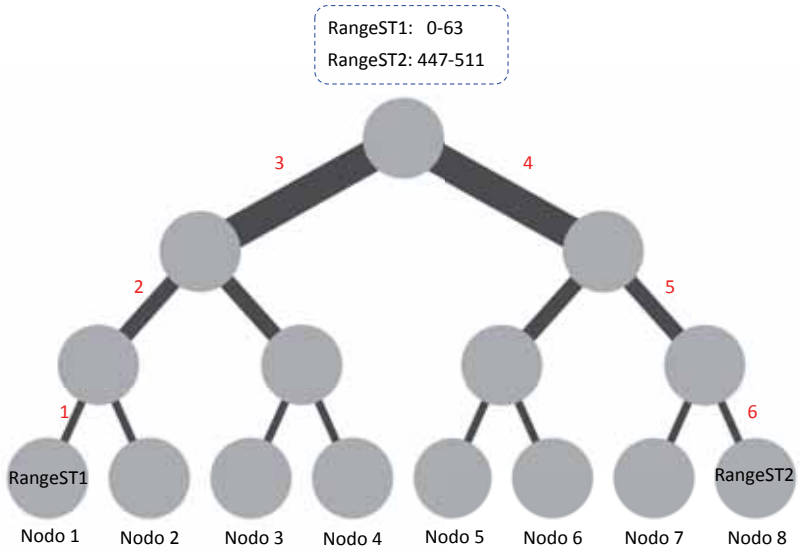


Figura 7.6: Mapping del rango de procesos en el sistema

Una vez los procesos han sido mapeados en el sistema, puede ser necesario modificar el identificador de proceso del origen y/o destino de sus eventos de comunicación, a fin de poder procesarlos correctamente la herramienta SS.

Este cambio se realiza porque al ejecutar la herramienta SS para medir los tiempos de comunicación, MPI identifica los procesos de manera secuencial, es decir, si tenemos una ST4NP generada para 512 procesos, y ejecutamos la herramienta SS con 192 procesos (del 0 al 191), 128 procesos MeasurableP, ubicados en los nodos de medición, y 64 procesos BH, ubicados en los nodos *Black Hole*, puesto que el proceso mayor del mundo MPI *(MPI_ COMM_ WORLD)*, creado al ejecutar la herramienta SS, es el 191, se hace necesario modificar el origen y/o destino de los eventos de comunicación de los procesos MeasurableP, por un identificador perteneciente al mundo *MPI_ COMM_ WORLD*, ya que si realizamos una comunicación a un proceso mayor al 191, no será reconocido por MPI, ya que no pertenece a su mundo, generando un error de comunicación en tiempo de ejecución.

La figura 7.7 muestra un ejemplo. Supongamos que deseamos predecir el rendimiento de una aplicación para 512 procesos. Para ello, utilizamos un sistema paralelo con nodos de 64 cores, sobre el cual ejecutamos la herramienta SS con 192 procesos, 128 procesos MeasurableP, y 64 procesos BH. Puesto que los nodos del sistema tienen 64 cores, creamos grupos de procesos (RangeST) de 64 procesos, obteniendo un total de 8 grupos:

Traza Física Escalada (ST4NP) del proceso 0 para la fase 1

ID Evento	Tipo de primitiva	Origen	Destino Real	Destino Lógico	Comunicación Externa
0	Send	0	63	63	
1	Irecv	0	63	63	
2	Wait	0	63	63	
3	Send	0	511	127	
4	Irecv	0	511	127	
5	Wait	0	511	127	
6	Send	0	64	64	X
7	Irecv	0	64	64	X
8	Wait	0	64	64	X

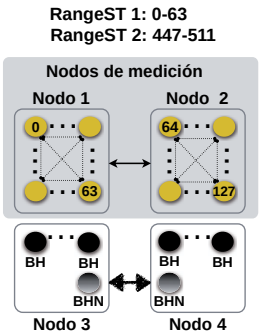


Figura 7.7: Modificación del destino de los eventos de comunicación de la traza física escalada

0-63, 64-127, 128-192, ..., 447-511.

En el primer nodo de medición (nodo 1) cargamos el RangeST 1, el cual contiene el grupo de procesos del 0 al 63, mientras que en el segundo nodo de medición (nodo 2), cargamos el RangeST 2, el cual contiene los procesos del 447 al 511.

Una vez tenemos cargada la traza ST4NP de los procesos MeasurableP, es necesario modificar el origen y/o destino real de sus eventos de comunicación por uno lógico perteneciente al *MPI_COMM_WORLD* (de 0 a 191). Para ello, el primer paso es crear una correspondencia entre el identificador real y lógico. Con el fin de crear esta correspondencia se genera la *tabla de correspondencias Real-Lógico*, la cual, tal y como se muestra en la figura 7.8, contiene la correspondía entre los identificadores reales y lógicos. El proceso lógico corresponde al proceso MPI donde se ha cargado la traza ST4NP del proceso MeasurableP, por ejemplo, siguiendo con la figura 7.8, la traza ST4NP del proceso 447 se ha cargado en el proceso 64 del mundo *MPI_COMM_WORLD*.

Una vez se ha generado la tabla de correspondencias, el siguiente paso es modificar el origen y/o destino de sus eventos de comunicación en caso de que sea necesario. Con el objetivo de modificar el origen o destino de los eventos de comunicación, se lleva a cabo un pre-procesado de las trazas ST4NP de los procesos medibles (MeasurableP). Siguiendo con la figura 7.7, se muestra un ejemplo para el proceso 0 de cómo se realiza esta modificación. Puesto que el origen de todas las comunicaciones es el proceso 0, el

Tabla de correspondencia Real - Lógico

Proceo Real	Proceso Lógico
447	64
448	65
449	66
450	67
451	68
452	69
...	...
510	126
511	127

Figura 7.8: Tabla de correspondencia entre el proceso real y el proceso lógico

cual pertenece al *MPI_ COMM_ WORLD*, únicamente es necesario modificar el destino de los eventos de comunicación.

Como vemos en la figura, los 3 primeros eventos de comunicación se comunican con el proceso 63, puesto que este proceso pertenece al *MPI_ COMM_ WORLD* creado por MPI, ya que está dentro del rango de procesos del 0 al 191, y pertenece al RangeST 1, no se modifica su destino, ya que el proceso 0 puede comunicarse con el procesos 63.

Los 3 siguientes eventos de comunicación (del 3 al 5), se comunican con el proceso 511, puesto que ese proceso no se encuentra dentro del mundo *MPI_ COMM_ WORLD*, pero fue seleccionado como proceso de medición, ya que se pertenece al RangeST 2, se consulta la tabla de correspondencia con el objetivo de modificar su destino real, por su destino lógico. Como podemos comprobar en la figura 7.8, donde se muestra la tabla de correspondencia, el destino lógico del proceso 511 es el proceso 127 (el proceso donde se encuentra cargada la traza ST4NP del proceso 511). Una vez se ha modificado el destino, el evento de envío con ID 3, podrá realizar la comunicación hacía el proceso 127, tal y como se muestra en la figura 7.9.

Finalmente, los 3 últimos eventos de comunicación, comunican con el proceso 64, puesto que este proceso no se encuentra entre los rangos RangeST 1 (0-63) o RangeST 2 (447-511), se marca el evento como comunicación externa (Ecomm), ya que tendrá que comunicar con los procesos BH. Puesto que aún no sabemos con que proceso BH comunicará, no se le asigna un proceso BH.

Mediante este cambio, conseguimos poder ejecutar los eventos de comunicación entre

los nodos de medición, con el objetivo de medir el tiempo de comunicación.

A fin de obtener los tiempos de comunicación correctos, el último paso es conocer el número de comunicaciones ruido que tendrán que ser ejecutadas en cada fase. Para ello, se realiza un pre-procesado de las trazas ST4NP de todos los procesos, donde se identifica el número total de comunicaciones en tránsito para cada fase ($TotComEnTransitoFase_i$), que tendría la aplicación si fuera ejecutada con todos los recursos.

Con el objetivo de generar el mismo número de comunicaciones en tránsito para la fase, utilizando menos recursos, tal y como se muestra en la ecuación 7.1, este número total de comunicaciones en tránsito tiene que ser restado al número total de comunicaciones Ecomms de los procesos medibles (MeasurableP) de la fase ($TotalECommsFase_i$), el cual fue obtenido durante el pre-procesado de la trazas ST4NP de los procesos medibles, obteniendo el número total de comunicaciones ruido que tendrán que ser ejecutadas en cada fase ($TotComRuidoFase_i$).

$$TotComRuidoFase_i = \#TotComEnTransitoFase_i - TotalECommsFase_i \quad (7.1)$$

7.3. Selección del número de procesos Black Hole

Dependiendo de los rangos de procesos (RangeST) seleccionados para ejecutar la herramienta SS, puede que el número de procesos *Black Holes* (BH) varíe. Esto es debido al número de comunicaciones externas (EComms) generadas en cada iteración del procedimiento, las cuales tienen que comunicarse con los procesos BH.

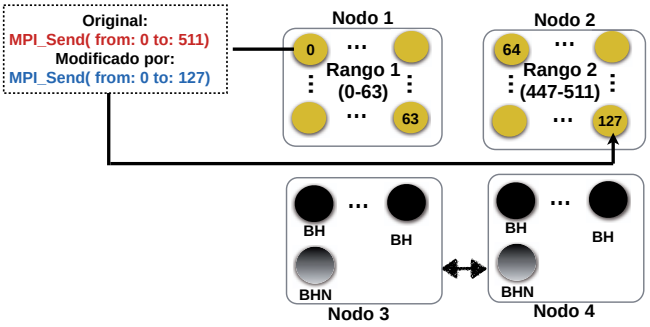


Figura 7.9: Modificación del destino real del evento de comunicación por su destino lógico

Tabla de correspondencia MeasurableP – Black Hole

Proceso medible con comunicaciones externas	Proceso Back Hole
0	128
2	129
4	130
8	131
16	132
32	133
...	...

Figura 7.10: Tabla de correspondencia entre el proceso medible y su proceso Black Hole (BH)

Con el objetivo de seleccionar cuantos procesos BH serán utilizados en cada iteración del algoritmo, se asigna un proceso BH a cada proceso medible (MeasurableP) que presente comunicaciones externas, siendo el número total de procesos BH utilizados en la iteración, igual al número total de procesos measurableP con comunicaciones externas. Por lo tanto, el número total de procesos BH ($NumeroDeProcesosBH$) vendrá asignado por la ecuación 7.2, siendo m el número total de procesos MeasurableP con comunicaciones externas.

$$NumeroDeProcesosBH = \sum_{i=1}^{m} (MeasurablePwithEComms_i) \tag{7.2}$$

Una vez se conoce el número total de procesos BH a utilizar durante la iteración, se crea una correspondencia entre el proceso MeasurableP con comunicaciones externas y el proceso BH con el cual interaccionará, para enviar o recibir sus comunicaciones externas. Para ello, se crea una tabla denominada *Tabla de correspondencia MeasurableP - Black Hole*, la cual, como se muestra en la figura 7.10, contiene el proceso MeasurableP y el proceso BH al cual enviará sus comunicaciones externas.

Finalmente, los eventos de comunicación de los procesos medibles, los cuales tienen comunicaciones externas (Ecomms), tienen que modificar el destino real de sus comunicaciones, con el identificador del proceso BH.

Puesto que en la etapa anterior se realizó un pre-procesado de las trazas ST4NP de los procesos MeasurableP, donde se identificaron los eventos de comunicación con

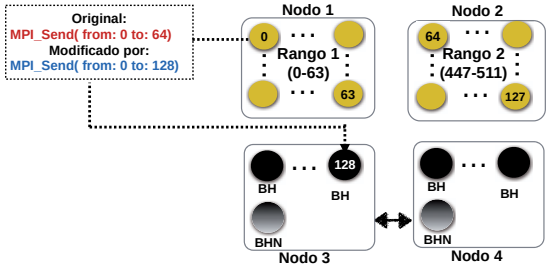


Tabla de correspondencia MeasurableP – Black Hole

Proceso medible con comunicaciones externas	Proceso Back Hole
0	128
2	129
4	130
8	131
16	132
32	133
...	...

Figura 7.11: Modificación del destino del mensaje por un proceso BH

comunicaciones externas de cada proceso MeasurableP, estos son seleccionados con el objetivo de modificar su destino. Para ello, se busca en la tabla de correspondencia el proceso MeasurableP, con el objetivo de obtener su proceso BH y modificar el destino del mensaje, tal y como se muestra en la figura 7.11 para el proceso 0, el cual comunica con el proceso BH que corresponde al proceso 128, por lo que cambia el destino del evento de comunicación del proceso 64 al 128.

La figura 7.12 muestra un ejemplo del procedimiento. Como se puede apreciar, para la traza física escalada (ST4NP) del proceso 0, tenemos 3 comunicaciones externas (Ecomms). Con el objetivo de seleccionar su proceso BH con el cual intercambiar información, se busca en la tabla de correspondencia utilizando el identificador del proceso medible como índice de la búsqueda. Puesto que el proceso 0 tiene asignado el proceso BH con identificador de proceso 128, se modifica el destino de las comunicaciones externas asignándole el proceso lógico 128.

Una vez se ha realizado esta asignación, la herramienta SS puede comenzar a procesar la traza física con el objetivo de medir el tiempo de comunicación de los eventos

7.4. Procesado de la traza física

La figura 7.13 muestra el diagrama de flujo del método utilizado para llevar a cabo el procesado de la traza ST4NP utilizando la herramienta SS.

Como se puede apreciar en la figura 7.13, el primer paso antes de medir las comunicaciones es ejecutar una primera etapa de calentamiento *(Warm-up)* de la red de

interconexión. Esta etapa consiste en procesar los eventos de comunicación de los procesos MeasurableP seleccionados en la iteración, obviando la emulación del tiempo de cómputo, con el objetivo de establecer la creación de los enlaces de comunicación a nivel de MPI y Sistema Operativo (SO).

Este calentamiento de la red se hace necesario ya que la primera vez que realizamos la comunicación entre 2 procesos, MPI tiene que crear el enlace de comunicación mediante el cual poder realizar la comunicación entre procesos, esta creación tiene como consecuencia que el tiempo de comunicación aumente la primera vez que realiza la comunicación entre dos procesos.

A la hora de ejecutar la aplicación utilizando todos los recursos, puesto que las fases se repiten un elevado número de veces, el tiempo de creación de los enlaces no tiene repercusión en el tiempo total de ejecución de la aplicación, ya que se crean solamente una única vez. Sin embargo, a la hora de medir las comunicaciones de la traza física escalada (ST4NP), puesto que el tiempo de comunicación de cada evento de la fase se mide un conjunto reducido de veces, a la hora de extrapolar este tiempo por el número de veces que se repite la fase (peso), puede ocurrir, que el tiempo de creación del enlace tenga una influencia considerable a la hora de obtener el tiempo total de predicción de la aplicación.

Una vez se ha realizado la etapa de *Warm-up*, con el objetivo de medir los eventos de comunicación de cada fase, los eventos de los procesos medibles (MeasurableP) son guardados en una cola y se comienzan a procesar hasta que la cola se queda sin eventos.

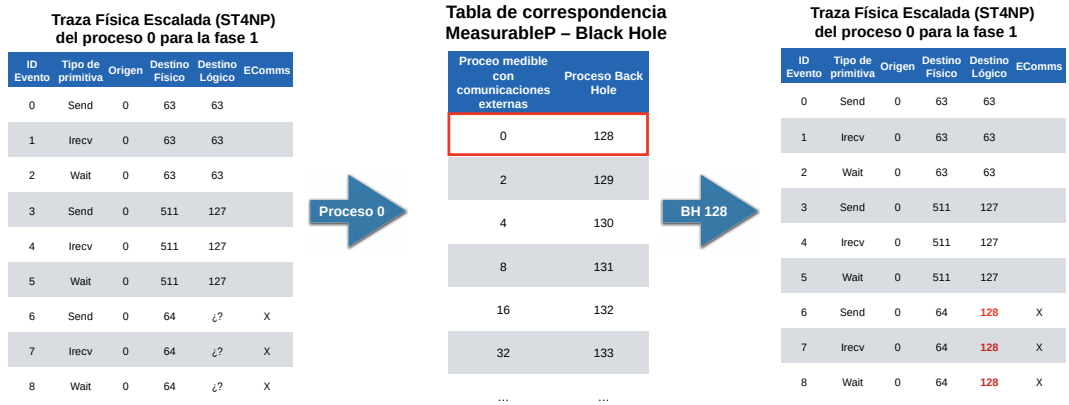


Figura 7.12: Asignación de procesos BH a las comunicaciones externas

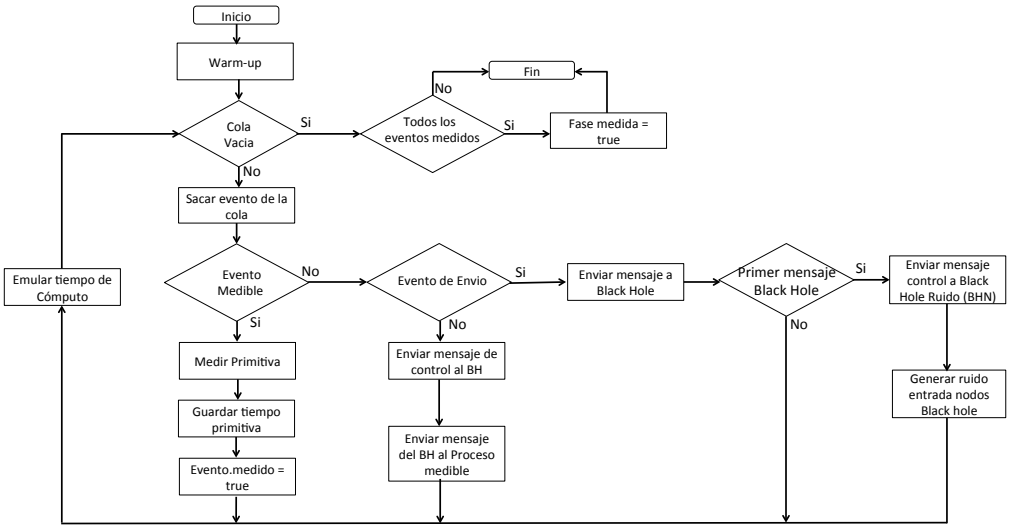


Figura 7.13: Digrama de flujo del procedimiento de ejecución de una iteración de la herramienta SS para procesar las trazas ST4NP

Extraído el evento de la cola, con el objetivo de poder ser procesado, se ha creado una estructura, llamada *procesa_ evento*, la cual contiene los siguientes campos: el tipo de mensaje del evento (tipo de primitiva MPI), el origen y destino de la comunicación, el volumen de comunicación del evento y el tiempo de cómputo entre el evento de comunicación actual y el siguiente evento de comunicación. Además, la estructura contiene 2 campos adicionales, uno llamado *medido*, el cual indica si el evento de comunicación ha sido medido, y otro llamado *tiempo_ comm*, el cual contiene el tiempo medido de comunicación del evento.

La herramienta SS realiza diversas comprobaciones con el objetivo de procesar correctamente el evento de comunicación de la fase.

Primeramente, se comprueba si el evento realiza una comunicación Intranode o Internode, es decir, si tanto el proceso origen como el proceso destino del evento están dentro de los rangos de medición, en caso afirmativo, el evento es medido, el atributo *medido* del evento es marcado como *true* y se guarda su tiempo de comunicación.

A la hora de medir los eventos de comunicación de la fase, tal y como se muestra en la figura 7.14, se mide el tiempo de ejecución de las primitivas MPI de cada proceso desde que comienza a ejecutarse la primitiva hasta que finaliza. Se ha seleccionado este método de medición en lugar de medir las comunicaciones entre el proceso origen y destino por dos motivos principales:

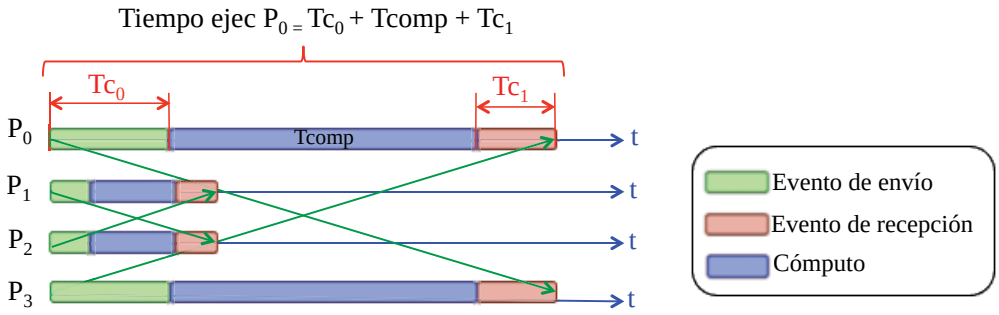


Figura 7.14: Medición del tiempo de comunicación de los eventos MPI

1. El primer motivo de medir cada proceso independientemente es evitar los problemas de la desincronización de relojes, ya que aunque los clusters HPC poseen métodos para que todos los relojes estén sincronizados, se puede presentar una pequeña desincronización que cause un error considerable al medir los tiempos de comunicación, debido al orden de magnitud con que se mide el tiempo de comunicación.

2. El segundo motivo es que al medir las primitivas MPI, además del tiempo de comunicación, se mide también el tiempo de preparación y gestión del mensaje, asociado al tipo de primitiva MPI.

Una vez se ha medido la comunicación, se emula el tiempo de cómputo entre el evento de comunicación actual y el siguiente evento de comunicación. Con el objetivo de emular el tiempo de cómputo, se utiliza la función del sistema operativo *nanosleep()*.

En caso de que el proceso destino del evento no se encuentre entre el rango de procesos a ser medidos en la iteración, es decir, realice una comunicación externa (Ecomm), se comprueba si es un evento de envío o un evento de recepción.

En caso de ser un evento de envío, se envía el mensaje al proceso BH con el objetivo de resolver la dependencia de la comunicación, pero no se instrumenta la medida hasta la siguiente iteración del algoritmo, donde los dos procesos (origen-destino), estén dentro de los rangos de medición. El objetivo es ejecutar todos los eventos de comunicación de la fase, ya que si los descartamos, no estaríamos representando el comportamiento real de la fase, desbalanceando el proceso, afectando al resto de procesos de la fase.

Si es el primer mensaje que llega a algún proceso BH, el proceso BH envía un mensaje de control al proceso *Black Hole de ruido* (BHN), con el objetivo de activar las

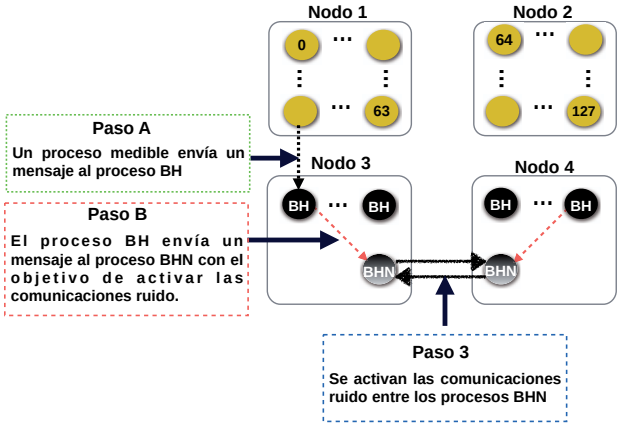


Figura 7.15: Envío de comunicaciones externas al proceso Back Hole (BH)

comunicaciones ruido entre los 2 procesos BHN, situados en los nodos *Black Hole*, con el objetivo de simular los mensajes en tránsito que circularían por la red en caso de ejecutar la aplicación utilizando todos los recursos. La figura 7.15 ilustra este procedimiento. Como se puede apreciar, un proceso medible (MeasurableP) envía una comunicación externa (Ecomm) a un proceso *Black Hole* (BH) (paso 1). Puesto que es la primera comunicación externa que llega a un proceso BH del nodo, el proceso BH que ha recibido el mensaje del proceso MeasurableP, envía un mensaje de control al proceso BHN (paso 2), el cual al recibir el mensaje, activa las comunicaciones ruido entre él y el otro proceso de ruido situado en el otro nodo *Black Hole* (paso 3).

En caso de ser un evento de recepción, el proceso BH debe enviar al proceso MeasurableP un mensaje a fin de resolver las dependencias del evento de comunicación. Puesto que el proceso BH no tiene información acerca de la ejecución de los procesos medibles, es necesario informarle para que pueda enviar el mensaje al proceso MeasurableP. Para ello, el proceso MeasurableP que espera recibir un mensaje del proceso BH, le envía un mensaje de control informándole que tiene que enviarle un mensaje, tal y como se muestra en el paso 1 de la figura 7.16. El mensaje de control lleva como información el tipo de mensaje que espera recibir el proceso (destino, tipo de mensaje, tamaño del buffer, tipo de dato y TAG del mensaje). Una vez que el proceso BH ha recibido el mensaje de control, lo procesa, y le envía un mensaje de vuelta del tipo que espera recibir el proceso MeasurableP (paso 2). En este caso, al igual que sucedía con los envíos externos, el evento de recepción es ejecutado pero no medido, hasta la

siguiente iteración del algoritmo, donde los dos procesos (origen-destino), estén dentro de los rangos de medición.

Este procedimiento se realiza de forma iterativa hasta que la cola de eventos está vacía, momento en el cual se comprueba si todos sus eventos poseen el atributo *medido* igual a *true*, en tal caso, se da la fase por medida. En caso contrario, el algoritmo deberá volver a ejecutarse con otra configuración diferente a fin de medir los eventos que faltan.

A continuación, se muestra el funcionamiento de la herramienta SS mediante un ejemplo. Como se muestra en la figura 7.17, tenemos una traza física escalada (ST4NP) generada para 512 procesos (NPROCS=512), la cual queremos procesar en un sistema con 64 cores por nodo.

Puesto que disponemos de nodos con 64 cores, se generan 8 rangos de procesos (RangeST) de 64 procesos cada uno.

Para ejecutar la herramienta SS y obtener el tiempo de comunicación de las fases, se ejecuta la herramienta SS utilizando 192 procesos, 128 procesos MeasurableP, ubicados en los nodos de medición, y 64 procesos BH, de los cuales, 32 procesos BH fueron ubicados en el primer nodo *Black Hole* (nodo 3), y los 32 restantes en el otro nodo *Black Hole* (nodo 4).

Para este ejemplo, tenemos una primera iteración donde se carga el RangeST 1, el cual va del proceso del 0 al 63, en el primer nodo de medición (nodo 1), y el RangeST 2, el cual va del proceso 64 al 127, en el segundo nodo de medición (nodo 2).

Centrando el ejemplo en el proceso 0, vemos que el destino de los 3 primeros eventos de comunicación (64), pertenece al RangeST 2, ubicado en el nodo 2, el cual va del

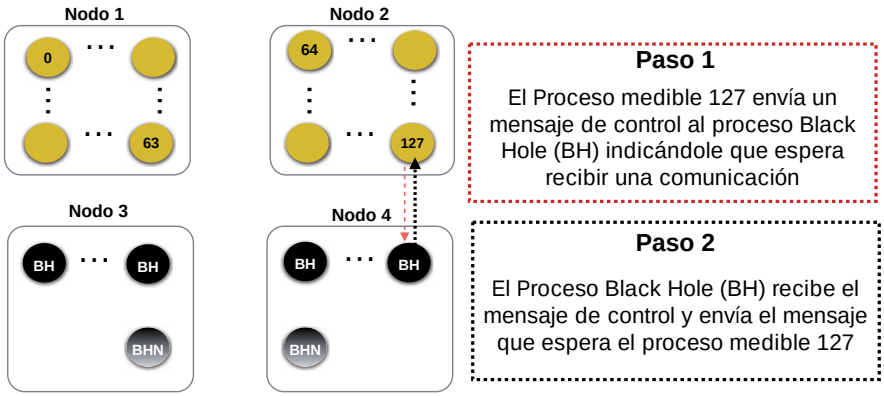
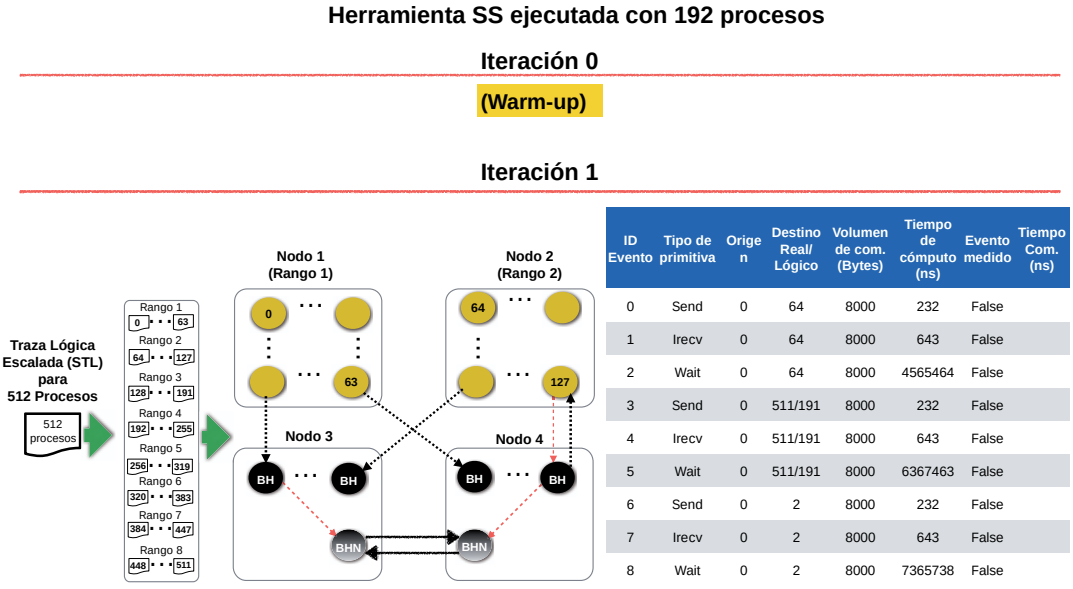


Figura 7.16: Mensaje de control informando de una recepción externa



ID Evento	Tipo de primitiva	Origen	Destino Real/ Lógico	Volumen de com. (Bytes)	Tiempo de cómputo (ns)	Evento medido	Tiempo Com. (ns)
0	Send	0	64	8000	232	False	
1	Irecv	0	64	8000	643	False	
2	Wait	0	64	8000	4565464	False	
3	Send	0	511/191	8000	232	False	
4	Irecv	0	511/191	8000	643	False	
5	Wait	0	511/191	8000	6367463	False	
6	Send	0	2	8000	232	False	
7	Irecv	0	2	8000	643	False	
8	Wait	0	2	8000	7365738	False	

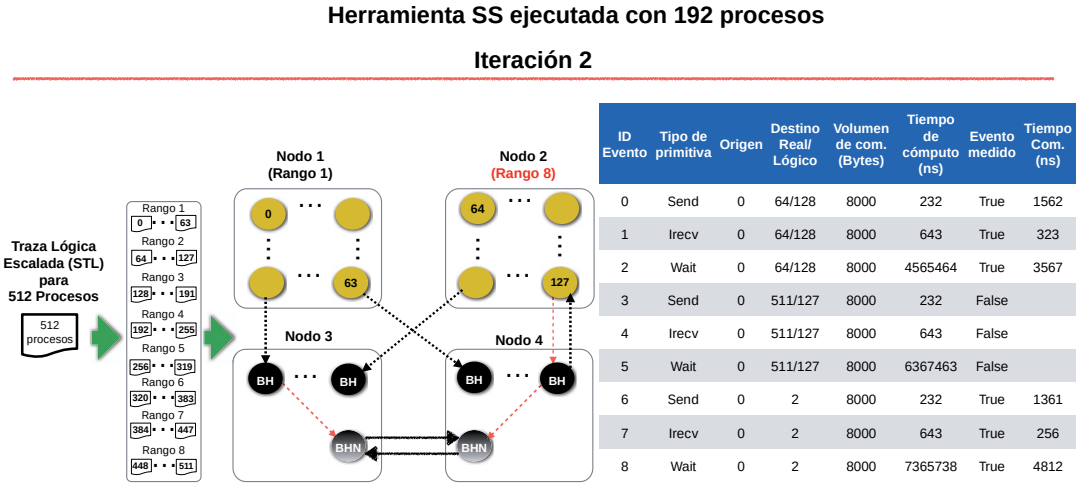
Figura 7.17: Ejemplo de ejecución de la herramienta SS - Iteración 1

proceso 64 al 127, por lo tanto, realizan comunicaciones Internodo, las cuales pueden ser medidas, por lo tanto, se mide su tiempo de comunicación, se guarda, y se marca el atributo *medido* del evento como *true*.

Los siguientes tres eventos de comunicación (del ID 3 al ID 5) intercambian información con el proceso 511, puesto que ese proceso no se encuentra en ninguno de los RangeST seleccionados en la iteración, sus comunicaciones no pueden ser medidas, realizando comunicaciones externas (EComms), intercambiando información con el proceso BH con indetificador 191.

El evento con ID 3, puesto que es un evento de envío *(MPI_Send)*, envía un mensaje al proceso BH con indetificador 191, una vez recibido, puesto que es el primer mensaje que ha llegado al nodo *Back Hole*, el proceso BH le envía un mensaje de control al proceso Black Hole Noise (BHN), con el objetivo de activar las comunicaciones ruido y simular el mismo número de comunicaciones en tránsito que tendría la aplicación si la ejecutáramos con todos los recursos. Puesto que es una comunicación externa, se ejecuta pero no se mide hasta que tanto el origen como el destino del evento se encuentren entre los rangos de medición.

Una vez que se ha enviado el mensaje y lo ha recibido el proceso BH, se ejecuta el siguiente evento de comunicación (ID 4), puesto que es una recepción asíncrona



ID Evento	Tipo de primitiva	Origen	Destino Real/ Lógico	Volumen de com. (Bytes)	Tiempo de cómputo (ns)	Evento medido	Tiempo Com. (ns)
0	Send	0	64/128	8000	232	True	1562
1	Irecv	0	64/128	8000	643	True	323
2	Wait	0	64/128	8000	4565464	True	3567
3	Send	0	511/127	8000	232	False	
4	Irecv	0	511/127	8000	643	False	
5	Wait	0	511/127	8000	6367463	False	
6	Send	0	2	8000	232	True	1361
7	Irecv	0	2	8000	643	True	256
8	Wait	0	2	8000	7365738	True	4812

Figura 7.18: Ejemplo de ejecución de la herramienta SS - Iteración 2

(MPI_ Irecv), abre el buffer donde recibir el mensaje y pasa al siguiente evento de comunicación (ID 5), donde recibe el mensaje. Puesto que espera recibir una comunicación externa de un proceso BH, el proceso origen (0), envía un mensaje de control al proceso BH, indicándole que espera una comunicación con unos determinados parámetros (destino, tipo de mensaje, tamaño del buffer, tipo de dato y TAG del mensaje). Una vez el proceso BH ha recibido el mensaje de control, envía la comunicación al proceso medible (0), con el objetivo de completar el evento de comunicación. Puesto que es una recepción externa, al igual que con el envío externo del evento anterior, no se mide la comunicación hasta que tanto el origen como el destino del evento se encuentren entre el rango de procesos medibles, únicamente se ejecuta con el objetivo de representar el comportamiento real de la aplicación.

Finalmente, los últimos 3 eventos comunican con el proceso 2, puesto que se encuentra en el RangeST 1, y tanto el origen como destino se encuentran en el mismo nodo de medición, realizando comunicaciones intranodo, pueden ser medidas. Por lo tanto, se mide su tiempo de comunicación, se guarda, y se marca el atributo *medido* del evento como *true*.

Una vez que todos los procesos han ejecutado todos los eventos de comunicación de la fase, se acaba la primera iteración. Puesto que quedan eventos por medir del proceso 0, se realiza una nueva iteración del algoritmo para un nuevo rango de procesos de medición.

Con el objetivo de medir la fase para el proceso 0, el RangeST 1 (0-63) se deja igual que en la primera iteración, y en el segundo nodo de medición se coloca un nuevo RangeST 2, el cual tiene el rango de procesos del 447 al 511 (Rango 8), tal y como se muestra en la figura 7.18. En esta segunda iteración, se medirán los eventos de comunicación que no fueron medidos en la iteración anterior.

A diferencia de la iteración anterior, en este caso, las 3 primeras comunicaciones de la fase se han convertido en comunicaciones externas, ya que su proceso destino (64) no pertenece a ningún RangeST seleccionado para esta nueva iteración, interactuando con los procesos BH. Puesto que se midieron en la primera iteración, ya conocemos su tiempo de comunicación, y únicamente son ejecutadas para mantener el comportamiento real de la fase.

Los siguientes 3 eventos de comunicación, los cuales comunican con el proceso 511, ahora pueden ser medidos, puesto que su destino se encuentra en el RangeST 2, siendo sus comunicaciones Internodo, por lo tanto su tiempo de comunicación es medido, guardado, y se marca su atributo *medido* como true.

Finalmente, los últimos 3 eventos comunican con el proceso 2, puesto que se encuentran entre el rango de procesos medibles, y tanto el origen como destino se encuentran en el mismo RangeST (nodo 1), realizando comunicaciones Intranodo, pueden ser medidas. Puesto que ya fueron ejecutadas y medidas en la primera iteración, no se vuelven a medir, únicamente son ejecutadas.

Finalmente, todos los eventos de comunicación de la fase para el proceso 0 han sido medidos, por lo tanto la fase está completa para este proceso y es marcada como fase medida. En caso de quedar procesos para los cuales la fase aún no estuviera medida, se ejecutaría una nueva iteración con nuevos rangos de procesos, con el objetivo de medir sus comunicaciones, y se seguiría iterando hasta que todos los procesos hayan medido la fase.

Una vez se han medido todos los eventos de comunicación para todas las fases de la traza ST4NP, tal y como veremos en el siguiente capítulo, ya podemos predecir el tiempo de cada una de las fases relevantes de la aplicación, y por ende, obtener el tiempo predicho de la aplicación.

8

Predicción del rendimiento de la aplicación

8.1. Introducción

Una vez que se ha medido el tiempo de comunicación de todas las fases que componen la traza física escalada (ST4NP), ya podemos predecir el tiempo de ejecución de cada fase relevante de la aplicación, mediante los cuales obtener el tiempo de ejecución predicho de la aplicación.

A lo largo de este capítulo se presenta como obtener el tiempo predicho de ejecución de la aplicación, así como el rendimiento de la aplicación en escalabilidad fuerte.

8.2. Predicción del tiempo de ejecución de la aplicación

Una vez que se ha realizado la última etapa de la metodología y todos los eventos de comunicación de las fases que componen la traza física escalada (ST4NP) han sido medidos, se obtiene el tiempo de ejecución predicho de cada fase.

Como paso final para obtener el rendimiento de la aplicación para el número de procesos para el cual fue construida la ST4NP, es necesario proyectar el tiempo de cada fase por su peso predicho, el cual está contenido en la ST4NP, puesto que fue modelizado durante la etapa de modelización lógica de la aplicación y concretado para el número de procesos específico, durante la creación de la traza lógica Escalada (LT4NC), tal y como se muestra en la figura 8.1, donde se presenta una visión global de la metodología P3S a modo de recordatorio de la misma.

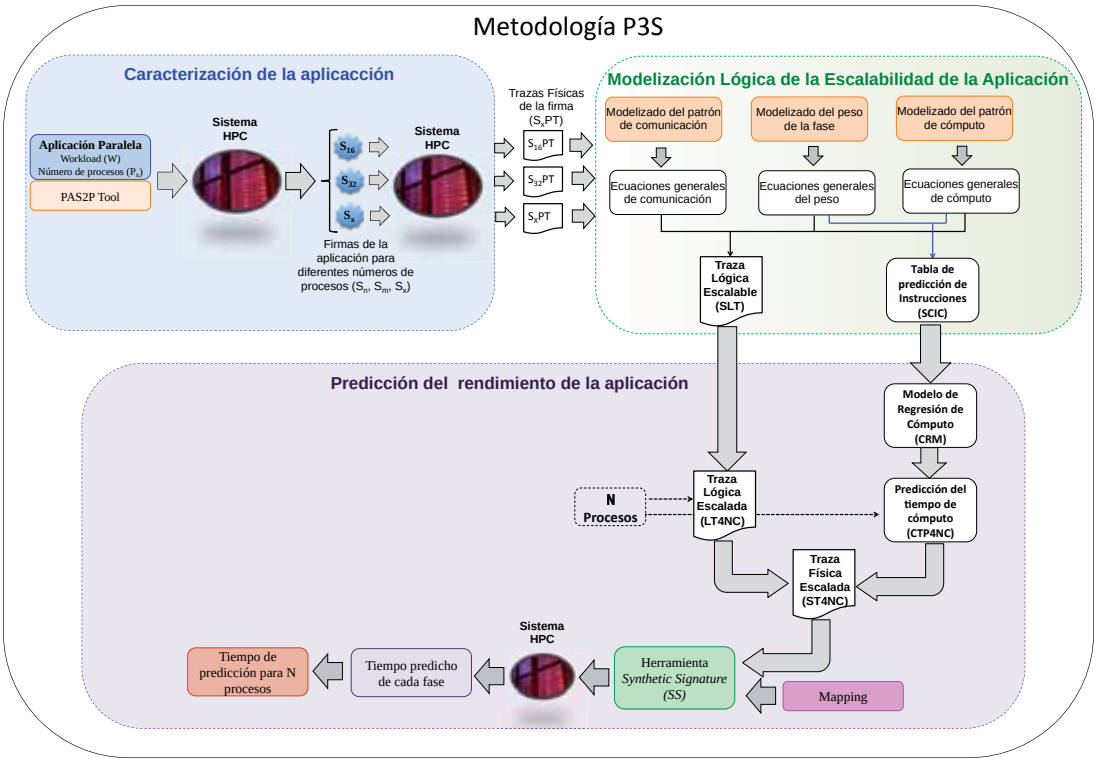


Figura 8.1: Visión global de la metodología P3S

Con el objetivo de obtener el tiempo de ejecución predicho de la aplicación, se utiliza la ecuación 8.1 , donde:

- PET: es el tiempo total de ejecución predicho de la aplicación.
- m: es el número total de fases de la traza física escadala (ST4NP).
- $PhaseETi$: es el tiempo predicho de ejecución de la fase i.
- Wi: es el peso predicho de la fase i.

$$PET = \sum_{i=1}^{m} (PhaseET_i)(W_i) \tag{8.1}$$

Aplicada la ecuación, obtendremos el tiempo predicho de ejecución de la aplicación, para el número de procesos *(N)* para el cual se desea predecir el rendimiento de la aplicación.

8.3. Predicción de la escalabilidad de la aplicación

Puesto que el objetivo de la metodología P3S es obtener el comportamiento de la aplicación en escalabilidad fuerte en un determinado sistema, es necesario predecir un conjunto de puntos mediante los cuales generar la curva de rendimiento de la aplicación.

Con el objetivo de obtener este conjunto de puntos, a partir de la traza lógica escalable (SLT) y de la tabla de predicción de instrucciones (SCIN), generadas en la etapa de modelización lógica de la aplicación, se genera la traza física escalable(ST4NC) para un número de procesos diferente.

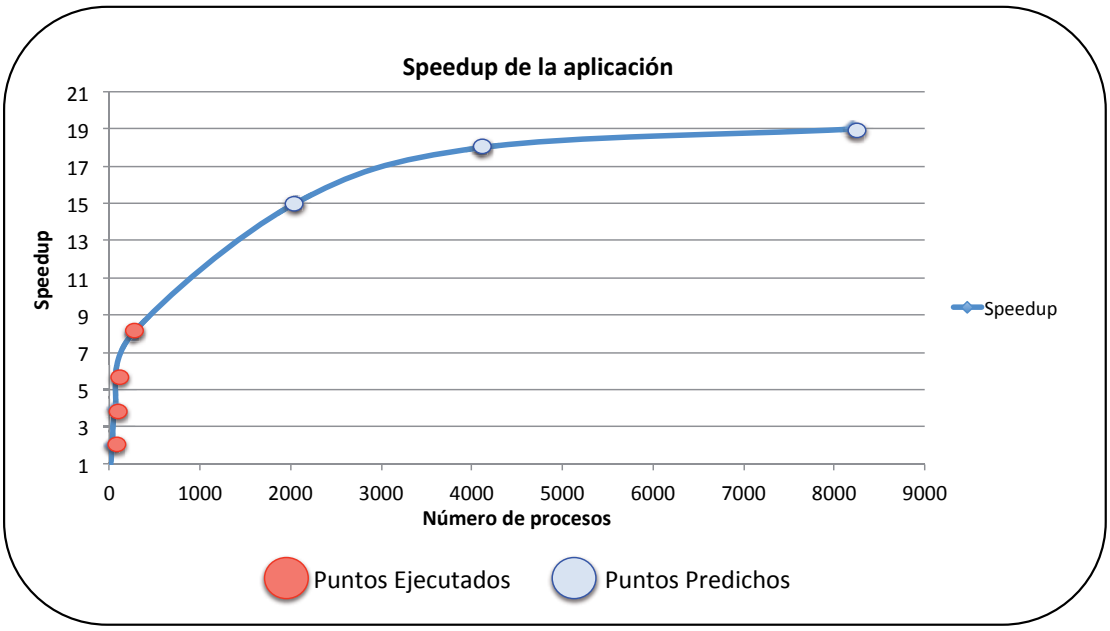


Figura 8.2: Curva de rendimiento de la aplicación

A fin de predecir el tiempo de la aplicación para este nuevo número de procesos, se deberá realizar nuevamente la última etapa de la metodología (predicción del rendimiento de la aplicación), consistente en predecir el tiempo de cómputo y el tiempo de comunicación de la aplicación, ya que dependen del número de procesos de la aplicación, no siendo necesario volver a realizar la etapa de caracterización de la aplicación y modelizado lógico de la aplicación, ya que no dependen del número de procesos a predecir, únicamente de las propiedades intrínsecas de la aplicación.

Obtenidos un conjunto de puntos predichos a medida que la aplicación escala, serán añadidos a los puntos reales de la aplicación, obtenidos de la ejecución del conjunto

inicial de firmas, con el fin de generar la curva de rendimiento de la aplicación, tal y como se muestra en la figura 8.2.

Esta curva de rendimiento podrá ser utilizada tanto por usuarios como administradores de sistemas, con el objetivo de seleccionar los recursos más apropiados a la hora de ejecutar la aplicación en el sistema destino.

9

Resultados experimentales

9.1. Introducción

A lo largo de este capítulo se mostrarán los resultados de predicción obtenidos utilizando la metodología P3S *(Prediction of Parallel Program Scalability)*. Con el objetivo de validar la calidad de predicción de la metodología, se compara el tiempo de predicción obtenido a partir de su utilización, con el tiempo real de ejecución de la aplicación y, el tiempo de predicción obtenido con la herramienta PAS2P.

Para mostrar los resultados experimentales, se ha seleccionado un amplio conjunto de aplicaciones científicas de paso de mensajes, con diferentes patrones tanto de comunicación como de cómputo. Además, a la hora de seleccionar las aplicaciones científicas, se ha tenido en cuenta que puedan ser ejecutadas para un elevado número de procesos y que posean algoritmos representativos en el mundo de las aplicaciones científicas de paso de mensajes.

A continuación, se realiza una descripción general de las aplicaciones seleccionadas, las cuales fueron utilizadas para realizar la validación experimental de los capítulos 5 y 6:

- *BT (Block Tridimensional)* y *SP (Scalar Pentadiagonal)* [59]: Resuelven sistemas sintéticos no linéales de PDEs utilizando tres algoritmos diferentes que incluyen Block tridiagonal, Scalar Pentadiagonal y symmetric succesive over-relaxation (SSOR).
- *CG (Conjugate Gradient)* [59]: Estima el valor propio más pequeño de una gran matriz escasamente simétrica utilizando la iteración inversa con el método del

Tabla 9.1: Características del Cluster CAPITA

Cluster	Hardware	Software
CAPITA	8 nodos x 64 cores AMD Opteron 6262. 512 Cores en total 256GB de Memoria RAM por nodo. 2TB de Memoria Total Interconexión Infiniband QDR	Red Hat Linux OpenMPI 1.6.5 PAPI 5.1

gradiente conjugado como una subrutina para resolver el sistemas de ecuaciones lineales.

- *Sweep3D* [60]: Aplicación de simulación avanzada y cómputo real. Es capaz de resolver ordenadas discretas de 1-grupo independientes del tiempo en coordenadas 3D cartesianas (XYZ). Esta geometría XYZ se representa por un grid rectangular. El cómputo y la memoria se incrementan substancialmente dependiendo de la carga de trabajo. El incremento de la carga de trabajo predomina en el número de iteraciones internas.

- *N-Body:* Aplicación que simula la interacción de un sistema dinámico de partículas bajo la influencia de diferentes fuerzas físicas, tales como la gravedad.

Como entorno experimental, se ha utilizado el Cluster CAPITA, las características del cual son mostradas en la tabla 9.1. La topología de la red de interconexión se muestra en la figura 9.1, como se puede apreciar, todos los nodos del cluster se encuentran a 1 *Hop* de distancia entre ellos.

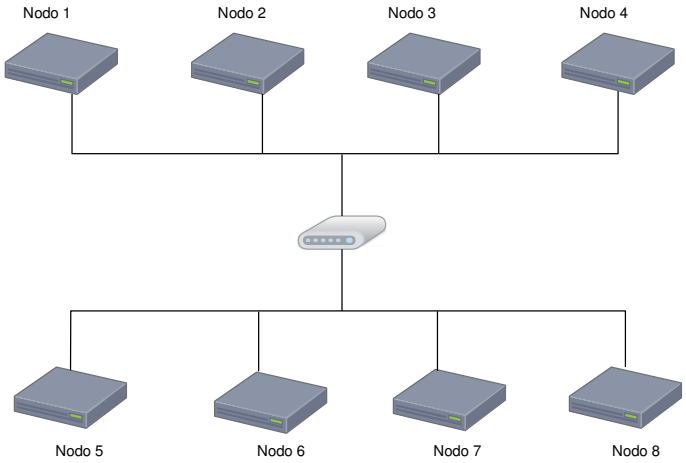


Figura 9.1: Topología de la red de interconexión del cluster CAPITA

9.2. Predicción del rendimiento de la aplicación

En esta sección se muestra la calidad de predicción de la metodología P3S para las aplicaciones: N-Body, Sweep3D, CG, SP y BT. Por limitación de recursos del sistema, las aplicaciones CG, Sweep3D y N-Body fueron ejecutadas y predichas para 512 procesos, mientras que las aplicaciones BT y SP fueron predichas para 484 procesos, ya que únicamente pueden ser ejecutadas utilizando un número de procesos cuya raíz cuadrada sea un número exacto, debido a esta limitación en el modelo de ejecución de la aplicación, no pueden ser ejecutadas para 512 procesos. Todas las aplicaciones fueron mapeadas con afinidad 1:1, es decir, 1 proceso por core.

A la hora de llevar a cabo la validación experimental para el conjunto de aplicaciones seleccionadas, se ha seguido el siguiente *workflow*:

a) Se ha caracterizado la aplicación mediante un conjunto de 4 firmas para un reducido número de procesos. Para CG, N-Body y Sweep3D se han ejecutado las firmas para 16, 32, 64 y 128 procesos, mientras que para BT y SP se han ejecutado las firmas para 16, 32, 64 y 81 procesos.

b) Se realizado la etapa de modelizado lógico de la aplicación con el fin de obtener la traza lógica escalable (STL) y la tabla de predicción de instrucciones (SCIC).

c) Se ha predicho el tiempo de cómputo para cada fase considerando el número de procesos para el cual se desea predecir la aplicación.

d) Se ha generado la traza física escalada (ST4NP), la cual se ha utilizado como entrada de la herramienta *Synthetic Signature (SS)*, con el objetivo de obtener el tiempo de comunicación.

e) Una vez predicho el tiempo de cada fase de la aplicación, se ha obtenido el tiempo total de ejecución de la aplicación.

f) Se ha analizado el error de predicción comparando el tiempo predicho de la aplicación, con el tiempo obtenido mediante la ejecución de la firma PAS2P, y el tiempo de ejecución de la aplicación.

Tabla 9.2: Modelo de la fase 1 de la aplicación N-Body (Ecuaciones utilizadas para generar la traza lógica escalable (SLT) para n procesos)

Fase	Ecuación General de Comunicación (Dest.)	Ecuación General del Volumen de comunicación	Ecuación General del Patrón de cómputo (Num. Insts.)
Fase 0	$1) f(n) = 1$	$y(n) = 1E + 08n^{(-1)}$	i(n)=421
	$2) f(n) = n - 1$	$y(n) = 1E + 08n^{(-1)}$	i(n)=((10,654,234,406,436,900/n)/w(n))
	$3) f(n) = n - 1$	$y(n) = 1E + 08n^{(-1)}$	i(n)=0
	$4) f(n) = 0$	$y(n) = 1E + 08n^{(-1)}$	i(n)= 43,448
Ecuación General del peso para la fase 1 w(n)= 9.375*n			

Puesto que las etapas de modelizado de la aplicación y predicción del tiempo de cómputo fueron presentadas en capítulos anteriores, únicamente se muestra el procedimiento completo de la metodología para la aplicación N-Body. Para el resto de aplicaciones se muestra el tiempo de predicho de cada fase y el tiempo total predicho de la aplicación utilizando la metodología P3S.

9.2.1. Predicción del rendimiento para la aplicación N-Body

A continuación, se muestra el procedimiento completo de la metodología P3S para obtener el rendimiento de la aplicación N-Body para 512 procesos. Dicha aplicación fue ejecutada con un *workload* de entrada de 4.000.000 de partículas.

Para obtener el rendimiento de la aplicación N-Body para 512 procesos, se ejecutó la firma de la aplicación para 16, 32, 64 y 128 procesos con el objetivo de generar el modelo de la aplicación. A partir de la ejecución del conjunto de firmas, se obtuvo 1 fase relevante, la cual fue analizada a fin de modelizar sus ecuaciones generales de comportamiento, mediante las cuales generar la traza lógica escalable (SLT), la cual es mostrada en la tabla 9.2, y la tabla de predicción de instrucciones (SCIC), la cual es mostrada en la tabla 9.3.

Una vez generada la SLT, se generó la Traza Lógica Escalada (LT4NC) de la aplicación N-Body para 512 procesos, la cual se muestra en la tabla 9.4. Dicha tabla se compara con la obtenida mediante la herramienta PAS2P. Como se puede apreciar en la tabla, se ha predicho correctamente el patrón de comunicación (origen-destino), el volumen de comunicación fue predicho con un error del 0 % para todas las primitivas MPI de la fase, y el número de instrucciones se predijo con un error máximo del 1,46 %.

Generada la Traza Lógica Escalada (LT4NC), se predijo el tiempo de cómputo de la fase. Puesto que tal y como se muestra en la tabla 9.3, el número de instrucciones de

Tabla 9.3: Tabla de predicción de instrucciones (SCIC) de la fase 1 de la aplicación N-Body

Número de Instrucciones	Número de Procesos	Peso	Desplazamiento del Peso	Número total de Instrucciones
Valores medidos (Conjunto inicial de Firmas)				
4191198656832	16	159	1	10654209382980600
1047799783673	32	318	1	10654220598654600
261949999372	64	636	1	10654212774452900
Valores predichos (Generados con el modelo lógico de la fase)				
4187985222656	16	159	1	10654234406346129
1046996305664	32	318	1	10654234406494345
261749076416	64	636	1	10654234406501233
65437269104	128	1272	1	10654234406436900
16359008832	256	2544	1	10654234406411368
4089829319	512	5088	1	10654234406499643
1022457330	1024	10176	1	10654234406500004
255614332	2048	20352	1	10654234406503753
63903583	4096	40704	1	10654234406500432
15975896	8192	81408	1	10654234406532397
.....	.....	.....	.....	.....

Tabla 9.4: Paramétros de la fase predichos (Traza lógica Escalada (LT4NC)) aplicando las ecuaciones de la tabla 9.2 para 512 procesos

Fase Real					**Fase Predicha**					**Error de predicción**	
Fase - Peso	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Insts.**	**Fase - Peso**	**Prim. MPI**	**Origen-Des.t**	**Volumen Com.**	**Num. Insts.**	**Volumen Com.**	**Num. Insts.**
Fase 0-5088	ISend	0-1	250,000	421	Fase 0 - 5088	ISend	0-1	250,000	421	0.0 %	0 %
	IRecv	0-511	250,000	4,089,829,319		IRecv	0-511	250,000	4,149,949,339	0.0 %	1.46 %
	WaitAll	0-511	250,000	0		WaitAll	0-511	250,000	0	0.0 %	0 %
	WaitAll	0-0	250,000	43,448		WaitAll	0-0	250,000	43,448	0.0 %	0 %

Tabla 9.5: Error de predicción del tiempo de cómputo para las fase 1 de la aplicación N-Body para 512 procesos

Número de fase	Tiempo real de cómputo obtenido con PAS2P (ns)	Tiempo predicho de cómputo (ns)	Error de Predicción (%)	Ecuación de regresión generada
Fase 0	3,761,982,365	3,825,096,457	1.65 %	$y(n) = 8 * 10^{16} * n^{-2,702}$

la fase se mantiene prácticamente constante, a medida que aumentamos el número de procesos de la aplicación, se procedió a buscar un punto lejano para ajustar el modelo de regresión de cómputo inicial *(CRMi)*, mediante el cual predecir el tiempo de cómputo de la fase. Para ello, se ejecutó la aplicación con un *workload* de 500.000 partículas, con el objetivo de obtener el punto lejano con el cual generar el modelo de regresión de cómputo ajustado *(CRMc)*.

Se ejecutó la firma de la aplicación con 128 procesos utilizando este nuevo *workload*, obteniendo un tiempo de cómputo de 6,222,894 nanosegundos y un número de instrucciones de 16,179,992.

Según la tabla de predicción de instrucciones (SCIC), mostrada en la tabla 9.3, el número de instrucciones más cercano a 16,179,992, corresponde a la ejecución de la aplicación N-Body ejecutada para el *workload* original para 8,192 procesos.

Una vez tenemos el tiempo de cómputo (6,222,894 ns) y el número de procesos (8912) para el *workload* original, generamos el modelo de regresión de cómputo ajustado *(CRMc)*, obteniendo una función predictora ($y(n) = 8 * 10^{16} * n^{-2,702}$), donde la variable y es el tiempo de cómputo y la variable x es el número de procesos a predecir.

Obtenida esta nueva función predictora, será utilizada para predecir el tiempo de cómputo para 512 procesos. Como podemos ver en la tabla 9.5, donde se muestra el tiempo de cómputo obtenido mediante PAS2P, y el obtenido mediante la metodología P3S, utilizando la función de regresión del modelo *CRMc*, obtenemos un error de predicción de 1.65 %.

Una vez generada la traza lógica escalada para 512 procesos (LT4NC) y predicho tiempo de cómputo de la fase, se genera la traza física escalada (ST4NP) para 512 procesos, la cual se muestra en la tabla 9.6

La ST4NP, será utilizada como *input* de la herramienta *SS*, con el objetivo de medir el tiempo de comunicación de cada evento de la fase, con el fin de predecir el tiempo de ejecución de la aplicación para 512 procesos.

Tabla 9.6: Traza Física Escalada (ST4NC) para 512 procesos

Fase - Peso	Prim. MPI	Origen-Destino	Volumen de Comunicación	Número de Instrucciones	Tiempo de Cómputo (ns)
Fase 0-5088	ISend	0-1	250,000	421	1,420
	IRecv	0-511	250,000	4,089,829,319	3,825,096,457
	WaitAll	0-511	250,000	0	0
	WaitAll	0-0	250,000	43,448	67,802

La herramienta *SS*, tal y como se muestra en la tabla 9.8, fue ejecutada con un total 130 procesos, 128 procesos medibles y 2 procesos Black Holes (BH). Los procesos BH fueron mapeados en un único nodo, utilizando un total de 3 nodos para predecir el rendimiento de la aplicación, 2 para los procesos medibles y 1 nodo para los procesos BH. Se generaron únicamente 2 procesos BH, ya que solamente dos procesos medibles del rango de procesos ubicados en el nodo 1 tenían comunicaciones externas *(Ecomm)*.

Utilizando la metodología P3S, conseguimos ahorrar prácticamente el 75 % de recursos (382 cores) en comparación con la herramienta PAS2P, la cual necesita los 512 cores para predecir el rendimiento de la aplicación.

El resultado obtenido mediante la utilización de la metodología P3S se muestra en la tabla 9.7. La tabla muestra el tiempo predicho de ejecución, utilizando tanto la herramienta PAS2P como la metodología P3S, para cada fase de la aplicación, y el tiempo total predicho de la aplicación. Además, en la última columna (AET) de la tabla, se muestra el tiempo de ejecución de la aplicación en el sistema, utilizando afinidad 1:1, es decir, 1 proceso por core. Finalmente, se muestra tanto el error de predicción obtenido con la herramienta PAS2P y la metodología P3S.

Puesto que la aplicación N-Body únicamente contiene una fase relevante, el tiempo

Tabla 9.7: Predicción de la aplicación N-Body para 512 procesos usando la metodología P3S y la firma de PAS2P.

		Metodología P3S			Firma PAS2P			AET (Seg.)
Fase ID	Peso (W)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	
0	5088	3.8924	19,804.53		3.907435	19,881.02		
			PET: 19,804.53	1.04		PET: 19,881.02	0.65	20,012.94
PhaseET: Tiempo de ejecución de la fase PETE: Error de predicción de la aplicación					PET: Tiempo de ejecución predicho AET: Tiempo de ejecución de la aplicación			

Tabla 9.8: Recursos utilizados por la metodología P3S y por la firma de PAS2P

Aplicación	Metodología P3S				Firma PAS2P	
	Procesos medibles	Procesos Black Hole	Procesos Totales	**Recursos utilizados**	Procesos totales	**Recursos utilizados**
SP	128	91	219	**219**	484	**484**
BT	128	91	219	**219**	484	**484**
N-Body	128	2	130	**130**	512	**512**
Sweep3D	128	32	160	**160**	512	**512**
CG	128	32	160	**160**	512	**512**

predicho de la fase coincide con el tiempo predicho de ejecución de la aplicación. Como se puede observar en la tabla, el error de predicción obtenido es 1.04 %, con respecto al tiempo de ejecución de la aplicación, siendo ligeramente superior al obtenido con la herramienta PAS2P, la cual predice el comportamiento de la aplicación con un error aproximado al 0.65 %.

Si nos fijamos en el tiempo de ejecución de la herramienta *SS* (SYET) para obtener la predicción del rendimiento de la aplicación, el cual es mostrado en la tabla 9.9, vemos que es de 1,982.65 segundos, siendo aproximadamente un 10 % del tiempo total de ejecución de la aplicación. Comparando el tiempo de ejecución de la herramienta (SYET) con el tiempo de ejecución de la firma PAS2P (SET), vemos que prácticamente es el mismo tiempo de ejecución, ya que los dos métodos ejecutan el mismo número de fases de la aplicación. La principal diferencia radica en el número de recursos utilizado para realizar la predicción.

Tabla 9.9: Tiempos de predicción utilizando la herramienta *SS* y la firma PAS2P

Aplicación	Metodología P3S				Firma PAS2P				AET
	SYET (Seg.)	PET (Seg.)	PETE (%)	Cores usados	SET (Seg.)	PET (Seg.)	PETE (%)	Cores usados	(Seg.)
SP	133.98	6,982.48	0.39	219	134.98	6,967.84	0.18	484	6,995.66
BT	221.84	11,703.24	2	219	229.24	11,992,43	0.5	484	11,937.58
N-Body	1,982.65	19,808.58	1.02	130	2,012.94	19,884.93	0.63	512	20,012.94
Sweep3D	308.02	4,764.91	4.16	160	307.12	4,580.37	0.13	512	4,574.32
CG	578.34	11,108.52	7.91	160	514.47	11,216.79	9.6	512	10,229.2
SYET: Tiempo de ejecución de la herramienta SS SET: Tiempo ejecución firma									

9.2.2. Predicción del rendimiento para la aplicación Sweep3D

La aplicación Sweep3D se ejecutó con un *workload* input.250 con 1000 iteraciones, y se predijo su rendimiento para 512 procesos. Se ejecutaron un total de 4 firmas para 16, 32, 64 y 128 procesos, con el objetivo de generar la traza física escalada (ST4NP) para 512 procesos, utilizada como entrada de la herramienta SS para predecir el rendimiento de la aplicación.

A partir de la ejecución del conjunto de firmas de la aplicación se obtuvo un total de 5 fases relevantes, las cuales fueron modelizadas con el objetivo de obtener sus reglas generales de comportamiento con las cuales generar la ST4NP, utilizada como *input* por la herramienta SS.

Para predecir el tiempo de ejecución de la aplicación Sweep3D para 512 procesos, como se muestra en la tabla 9.8, se ejecutó la herramienta SS con un total de 160 procesos, 128 procesos medibles y 32 procesos BH. Debido a que utilizamos nodos de 64 cores, de los 128 procesos medibles, 64 procesos fueron mapeados en un nodo de medición (nodo 1) y los 64 restantes en otro nodo de medición (nodo 2). Puesto que todos los nodos del sistema presentan una distancia de 1 *Hop* entre ellos, pueden seleccionarse 2 nodos cualesquiera para mapear los procesos. Con respecto a los procesos BH, se crearon 32 procesos BH, ya que 32 procesos medibles presentaban comunicaciones externas *(EComm)*. Estos procesos BH fueron mapeados en 2 nodos *Black Hole*, con 16 procesos BH en cada nodo, debido a que de los 64 procesos medibles de cada nodo, 16 de ellos presentaban comunicaciones EComm, siendo necesario mapear los procesos BH de cada rango de procesos medibles, en nodos diferentes.

A diferencia de PAS2P, donde se requieren todos los recursos (512 cores) para predecir la aplicación, la metodología P3S ahorró 352 cores del sistema para predecir el rendimiento de la aplicación, ahorrando aproximadamente el 68 % de los recursos.

La tabla 9.10 muestra el tiempo de ejecución predicho para cada fase de la aplicación mediante la utilización de la herramienta PAS2P y la metodología P3S, así como el tiempo total de ejecución de la aplicación. Además, se muestra el error de predicción obtenido con los dos métodos utilizados.

Si comparamos el tiempo de predicción de la metodología P3S con el tiempo de la aplicación, obtenemos un error del 4.16 %. Si observamos el error de predicción obtenido

Tabla 9.10: Predicción de la aplicación Sweep3D para 512 procesos usando la metodología P3S y la firma de PAS2P.

		Metodología P3S			Firma PAS2P			AET (Seg.)
Fase ID	Peso (W)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	
0	100	10.9437	1,094.37		10.0477	1,004.77		
1	100	0.1209	12.09		0.1158	11.58		
2	99	14.9401	1,479.13		14.8701	1,472.13		
3	100	10.9942	1,099.42		10.1899	1,018.99		
4	100	10.799	1,079.90		10.729	1,072.90		
		PET:	4,764.91	4.16	PET:	4,580.37	0.13	4,574.32
PhaseET: Tiempo de ejecución de la fase PETE: Error de predicción de la aplicación					PET: Tiempo de ejecución predicho AET: Tiempo de ejecución de la aplicación			

mediante la utilización de PAS2P (0.13 %), vemos que el error de predicción de la metodología P3S con respecto a PAS2P aumenta un 4.03 %, esto es debido a la desviación del tiempo de ejecución de la fase 0 y 3, ya que si comparamos el tiempo de ejecución predicho obtenido para esas fases mediante el uso de la herramienta PAS2P, con el tiempo ejecución de las fases obtenido mediante la utilización de la metodología P3S, vemos que tienen una desviación aproximada del 8 %. Esto es debido a la predicción del tiempo de cómputo de las fases.

En cuanto al tiempo de ejecución de la herramienta *SS* (SYET), el cual es mostrado en la tabla 9.9, es de 308.02 segundos, siendo aproximadamente el 6.5 % del tiempo de ejecución de la aplicación.

9.2.3. Predicción del rendimiento para la aplicación SP

La aplicación SP se ejecutó con un *workload* de entrada *E* con 1000 iteraciones, y se predijo su rendimiento para 484 procesos. Se ejecutaron un total de 5 firmas para 16, 36, 49, 64 y 100 procesos, con el objetivo de generar la traza física escalada (ST4NP) para 484 procesos, utilizada como entrada de la herramienta SS para predecir el rendimiento de la aplicación. Se predijo la aplicación para 484 procesos en lugar de para 512 procesos, ya que el modelo de ejecución de la aplicación, obliga al usuario a introducir un número de procesos cuya raíz cuadrada sea un número exacto.

A partir de la ejecución del conjunto de firmas de la aplicación se obtuvo un total de

19 fases relevantes, las cuales fueron modelizadas con el objetivo de obtener sus reglas generales de comportamiento con las cuales generar la ST4NP, utilizada como *input* por la herramienta SS.

La tabla 9.8 muestra los recursos utilizados por la herramienta SS para predecir el rendimiento de la aplicación con 484 procesos. A diferencia de la herramienta PAS2P, la cual necesita ejecutarse con 484 procesos para obtener el tiempo predicho de ejecución de la aplicación, tal y como se muestra en tabla 9.8, la herramienta SS se ejecutó utilizando un total de 219 procesos, ahorrando aproximadamente el 54 % de los recursos (265 cores).

Del total de recursos utilizados por la herramienta SS (219 cores) para obtener la predicción del tiempo de ejecución de la aplicación, 128 cores se dedicaron a procesos medibles, de los cuales 64 procesos se mapearon en el nodo de medición 1 y, los 64 restantes en el nodo de medición 2. Los 91 procesos restantes, se utilizaron como procesos BH, de los cuales, 45 procesos fueron mapeados en un nodo *Black Hole* (nodo 3) y los 46 procesos restantes en el otro nodo *Black Hole* (nodo 4). La distribución de los procesos BH en los nodos *Black Hole* (nodo 3 y nodo 4) se ha realizado en función del número total de procesos medibles de cada nodo de medición que presentaban comunicaciones externas.

La tabla 9.11 muestra el tiempo de ejecución predicho obtenido mediante el uso de la metodología P3S para cada fase de la aplicación, y el tiempo total de ejecución predicho de la aplicación para 484 procesos. Además, también se muestra el tiempo de predicción obtenido mediante el uso de la herramienta PAS2P y el tiempo de ejecución de la aplicación, utilizando todos los recursos con afinidad 1:1. Si comparamos el tiempo de predicción de la metodología P3S con el tiempo de la aplicación, obtenemos un error de predicción del 0.39 %. Si observamos el error de predicción obtenido mediante la utilización de PAS2P (0.18 %), vemos que tanto el error de predicción de la metodología P3S, como el error de predicción de PAS2P, están por debajo del 1 %, obteniendo un error de predicción muy cercano entre ellos, utilizando un conjunto reducido de recursos.

En cuanto al tiempo de ejecución de la herramienta SS (SYET), el cual es mostrado en la tabla 9.9, es de 133.98 segundos, siendo aproximadamente el 2 % del tiempo de ejecución de la aplicación.

Tabla 9.11: Predicción de la aplicación SP para 484 procesos usando la metodología P3S y la firma de PAS2P.

		Metodología P3S			Firma PAS2P			AET (Seg.)
Fase ID	Peso (W)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	
0	19019	0.0165559	314.876		0.016907	321.55		
1	1001	0.0170513	17.06		0.017306	17.32		
2	1001	0.0635411	63.60		0.064669	64.73		
3	1001	0.44701	44.80		0.052503	52.55		
4	19019	0.0479076	911.15		0.049842	947.95		
5	1001	0.06422253	64.28		0.06224	62.30		
6	1001	0.0199191	19.93		0.020216	20.23		
7	19019	0.0203635	387.29		0.020728	394.21		
8	998	0.360381	359.60		0.36999	369.25		
9	999	2.12375	2,121.62		2.15466	2,152.50		
10	999	0.0620155	61.95		0.043173	43.13		
11	19019	0.05277	1,003.65		0.04980	947.1512		
12	1001	0.0632947	63.35		0.062685	62.74		
13	1001	0.0199074	19.92		0.018871	18.89		
14	19019	0.0192093	366.883		0.019147	364.15		
15	1001	0.0716831	71.85		0.072686	72.75		
16	1001	0.050116	50.1667		0.051867	51.9188		
17	19019	0.050582	962.036		0.050238	955.48		
18	1001	0.0637313	63.79		0.063556	63.6192		
		PET:	6,982.48	0.39	PET:	6,967.84	0.18	6,995.66
PhaseET: Tiempo de ejecución de la fase PETE: Error de predicción de la aplicación					PET: Tiempo de ejecución predicho AET: Tiempo de ejecución de la aplicación			

9.2.4. Predicción del rendimiento para la aplicación BT

La aplicación BT se ejecutó con un *workload* de entrada *E* con 1000 iteraciones, y se predijo su rendimiento para 484 procesos. Se ejecutaron un total de 5 firmas para 16, 36, 49, 64 y 100 procesos, con el objetivo de generar la traza física escalada (ST4NP) para 484 procesos, utilizada como entrada de la herramienta SS para predecir el rendimiento de la aplicación. Al igual que para la aplicación anterior (SP), se predijo la aplicación para 484 procesos.

A partir de la ejecución del conjunto de firmas de la aplicación, se obtuvo un total de 16 fases relevantes, las cuales fueron modelizadas con el objetivo de obtener sus reglas generales de comportamiento, con las cuales generar la ST4NP, utilizada como *input*

por la herramienta SS, para medir el tiempo de comunicación de los eventos de cada fase y obtener su tiempo de ejecución predicho.

La tabla 9.8 muestra los recursos utilizados por la herramienta SS para predecir el rendimiento de la aplicación con 484 procesos. A diferencia de la herramienta PAS2P, la cual necesita ejecutarse con 484 procesos para obtener el tiempo predicho de ejecución de la aplicación, tal y como se muestra en tabla 9.8, la herramienta SS se ejecutó utilizando un total de 219 procesos, ahorrando aproximadamente el 54 % de los recursos (265 cores).

Del total de recursos utilizados por la herramienta SS (219 cores) para obtener la predicción del tiempo de la aplicación, 128 cores se dedicaron a procesos medibles, de los cuales 64 procesos se mapearon en el nodo de medición 1 y, los 64 restantes en el nodo de medición 2. Los 91 cores restantes, se utilizaron para procesos BH, de los cuales, 45 procesos fueron mapeados en un nodo *Black Hole* (nodo 3) y los 46 procesos restantes en el otro nodo *Black Hole* (nodo 4). La distribución de los procesos BH en los nodos *Black Hole* (nodo 3 y nodo 4), se ha realizado en función del número total de procesos medibles de cada nodo de medición que presentaban comunicaciones externas.

Se utilizó la misma configuración de ejecución que en la aplicación anterior (SP) para ejecutar la herramienta SS, debido a que las fases relevantes de la aplicación presentan el mismo patrón de comunicación.

La tabla 9.12 muestra el tiempo de ejecución predicho obtenido mediante el uso de la metodología P3S para cada fase de la aplicación, y el tiempo total predicho de la aplicación para 484 procesos. Además, también se muestra el tiempo de predicción obtenido mediante el uso de la herramienta PAS2P y el tiempo de ejecución de la aplicación, utilizando todos los recursos con afinidad 1:1. Si comparamos el tiempo de predicción de la metodología P3S con el tiempo de la aplicación, obtenemos un error de predicción (PETE) del 2 %, siendo ligeramente superior al error de predicción obtenido mediante la herramienta PAS2P, donde obtenemos un error de predicción del 0.5 %.

En cuanto al tiempo de ejecución de la herramienta *SS* (SYET), el cual es mostrado en la tabla 9.9, es de 221.84 segundos, siendo aproximadamente el 1.85 % del tiempo de ejecución de la aplicación.

Tabla 9.12: Predicción de la aplicación BT para 484 procesos usando la metodología P3S y la firma de PAS2P.

		Metodología P3S			Firma PAS2P			AET (Seg.)
Fase ID	Peso (W)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	
0	19,019	0.028032	533.14		0.029617	563.28		
1	1,001	0.03002	30.05		0.029468	29.49		
2	1,001	0.13722	137.35		0.138483	138.62		
3	1,001	0.14293	143.07		0.150507	150.65		
4	20,020	0.12922	2,586.98		0.129773	2,598.05		
5	1,001	0.021312	21.33		0.021309	21.39085		
6	19,019	0.026293	499.82		0.026745	508.65		
7	1	0.27412	0.27		0.274861	0.27		
8	998	0.3024	301.79		0.311660	311.0367		
9	999	1.8432	1,841.35		1.907566	1,905.65		
10	999	0.121478	121.35		0.122684	122.56		
11	20,020	0.10457	2,093.49		0.111621	2,234.64		
12	1,001	0.02822	28.24		0.028743	28.7		
13	19,019	0.025991	494.32		0.026759	508.92		
14	1,001	0.14294	143.08		0.145930	146.07		
15	1,001	0.12964	129.76		0.121122	121.24		
15	20,020	0.12358	2,597.77		0.130050	2,603.60		
		PET:	11,703.24	2.0	PET:	11,992.94	0.5	11,937.58
PhaseET: Tiempo de ejecución de la fase PETE: Error de predicción de la aplicación					PET: Tiempo de ejecución predicho AET: Tiempo de ejecución de la aplicación			

9.2.5. Predicción del rendimiento para la aplicación CG

La aplicación CG se ejecutó con un *workload* de entrada *E* con 1000 iteraciones, y se predijo su rendimiento para 512 procesos. Se ejecutaron un total de 4 firmas para 16, 32, 64 y 128 procesos, con el objetivo de generar la traza física escalada (ST4NP) para 512 procesos, utilizada como entrada de la herramienta SS para predecir el rendimiento de la aplicación para 512 procesos.

A partir de la ejecución del conjunto de firmas de la aplicación se obtuvo un total de 11 fases relevantes, las cuales fueron modelizadas con el objetivo de obtener sus reglas generales de comportamiento con las cuales generar la ST4NP, utilizada como *input* por la herramienta SS.

La tabla 9.8 muestra los recursos utilizados por la herramienta SS para predecir el

rendimiento de la aplicación con 512 procesos. A diferencia de la herramienta PAS2P, la cual necesita ejecutarse con 512 procesos para obtener el tiempo predicho de ejecución de la aplicación, tal y como se muestra en tabla 9.8, la herramienta SS se ejecutó utilizando un total de 160 procesos, ahorrando aproximadamente el 68 % de los recursos (352 cores).

Del total de recursos utilizados por la herramienta SS (160 cores) para obtener la predicción del tiempo de la aplicación, 128 cores se dedicaron para procesos medibles, de los cuales 64 procesos se mapearon en el nodo de medición 1 y, los 64 restantes en el nodo de medición 2. Los 32 cores restantes, se utilizaron como procesos BH, de los cuales, 16 procesos fueron mapeados en un nodo *Black Hole* (nodo 3) y los 16 procesos restantes en el otro nodo *Black Hole* (nodo 4). La distribución de los procesos BH en los nodos *Black Hole* (nodo 3 y nodo 4) se ha realizado en función del número total de procesos medibles de cada nodo de medición que presentaban comunicaciones externas.

La tabla 9.13 muestra el tiempo de ejecución predicho obtenido mediante el uso de la metodología P3S para cada fase de la aplicación, y el tiempo total predicho de la aplicación para 512 procesos. Además, también se muestra el tiempo de predicción obtenido mediante el uso de la herramienta PAS2P y el tiempo de ejecución de la aplicación, utilizando todos los recursos con afinidad 1:1. Si comparamos el tiempo de predicción de la metodología P3S con el tiempo de la aplicación, obtenemos un error de predicción (PETE) del 7.91 %, siendo inferior al error de predicción obtenido mediante la herramienta PAS2P, donde obtenemos un error de predicción del 9.6 %.

En cuanto al tiempo de ejecución de la herramienta *SS* (SYET), el cual es mostrado en la tabla 9.9, es de 578.34 segundos, siendo aproximadamente el 5.2 % del tiempo de ejecución de la aplicación.

Como ha sido mostrado a lo largo de la experimentación presentada en esta sección, la metodología P3S permite predecir el rendimiento de las aplicaciones paralelas, utilizando un conjunto reducido de recursos del sistema, ofreciendo una calidad de predicción muy cercana a la que ofrece la herramienta PAS2P, la cual utiliza todos los recursos que requiere la aplicación para ser ejecutada. Para todas las aplicaciones utilizadas, la metodología P3S predijo el tiempo de ejecución de la aplicación con una calidad de predicción superior al 90 %. Por otro lado, el tiempo utilizado por la herramienta SS para medir las comunicaciones y obtener la predicción del tiempo de ejecución de la aplicación, se mantiene similar al obtenido utilizando la herramienta PAS2P, presentando una

Tabla 9.13: Predicción de la aplicación CG para 512 procesos usando la metodología P3S y la firma de PAS2P.

		Metodología P3S			Firma PAS2P			AET (Seg.)
Fase ID	Peso (W)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	PhaseET (Seg.)	PhaseET * W (Seg.)	PETE (%)	
0	12,525	0.344828	4,318.98		0.478946	5,998.32		
1	1	0.37532	0.37		0.38987	0.38987		
2	499	0.362585	180.93		0.1011449	143.39		
3	12,524	0.457247	5,726.56		0.344352	4,312.66		
4	12,525	0.0496373	621.707		0.040246	504.04		
5	501	0.43439	217.62		0.474756	237.85		
6	501	0.0412984	20.6905		0.002860	1.43280		
7	1	0.0134754	0.01		0.008464	0.008		
8	1	0.0335572	0.03		0.053516	0.05		
9	1	0.043369	0.04		0.037647	0.037		
10	498	0.0432984	21.5626		0.037451	0.037		
		PET:	11,108.52	7.91	PET:	11,216.79	9.6	10,229.2
PhaseET: Tiempo de ejecución de la fase PETE: Error de predicción de la aplicación					PET: Tiempo de ejecución predicho AET: Tiempo de ejecución de la aplicación			

desviación entre 1 % y 5 % entre los tiempos de ejecución de las 2 herramientas, siendo aproximadamente entre el 1 % y el 10 % del tiempo total de ejecución de la aplicación.

9.3. Predicción de la curva de escalabilidad de la aplicación

En esta sección, se muestran las curvas de escalabilidad predichas para las aplicaciones seleccionadas. Con el objetivo de generar las curvas de escalabilidad, se utilizó la metodología P3S para predecir diversos puntos de la curva, los cuales se añadieron a los puntos ejecutados para el conjunto de firmas iniciales, utilizadas como entrada de la metodología P3S.

Las aplicaciones CG, BT, y SP fueron ejecutadas utilizando el *workload* C; la aplicación Sweep3D fue ejecutada utilizando un *workload* input.80 con 11 iteraciones; y la aplicación N-Body fue ejecutada con un *workload* de 100,000 partículas.

Tabla 9.14: Predicción del tiempo de ejecución de la aplicación N-Body con 100,000 partículas de 16 a 512 procesos

Número de procesos	Recursos utilizados (Cores)	Tiempo predicho (Seg.)	Tiempo de ejecución de la aplicación (Seg.)	Error de predicción (%)
Tiempos obtenidos mediante la ejecución de la firma con PAS2P				
16	16	399.26	401.36	0.52
32	32	198.20	200.28	1.04
64	64	99.09	101.04	1.96
Tiempos obtenidos mediante la metodología P3S				
128	128	49.22	51.18	3.98
256	130	23.38	26.43	4.13
512	130	21.58	22.10	2.40

9.3.1. Predicción de la escalabilidad para la aplicación N-Body

Se ejecutaron las firmas de la aplicación para 16, 32 y 64 procesos, con un *workload* de entrada de 100,000 partículas, con el objetivo de generar el modelo de la aplicación, mediante el cual construir la traza física escalada (ST4NP) para 128, 256 y 512 procesos, la cual será utilizada como entrada de la herramienta SS. En la tabla 9.14, se muestran los tiempos predichos de la aplicación para 128, 256 y 512 procesos, y los recursos utilizados para obtener la predicción, utilizando la metodología P3S. Además, en la misma tabla, se muestran los tiempos predichos obtenidos de la ejecución del conjunto inicial de firmas con la herramienta PAS2P, así como los tiempos de ejecución de la aplicación en el sistema, para las diferentes ejecuciones. Como podemos observar en la tabla para las predicciones realizadas con la metodología P3S, se utilizó un número de recursos máximo de 130 cores para las predicciones de 256 y 512 procesos, obteniendo un error máximo de predicción de un 4.13 % para el tiempo predicho de 256 procesos. En cuanto a las predicciones para el conjunto inicial de firmas de la aplicación, se ha obtenido un error de predicción inferior al 2 % en todos los casos.

A partir del tiempo de ejecución predicho con la metodología P3S para 128, 256 y 512 procesos, y el tiempo de ejecución predicho mediante la ejecución de las firmas de la aplicación para 16, 32 y 64 procesos, se generó la curva de escalabilidad de la aplicación para el *workload* de entrada de 100,000 partículas, en el sistema destino. La figura 9.2 muestra el *speedup* predicho, el cual es comparado con el *speedup* real de la aplicación, obtenido ejecutando la aplicación en el sistema. Para calcular el *speedup* se utilizó como punto de referencia la ejecución para 16 procesos.

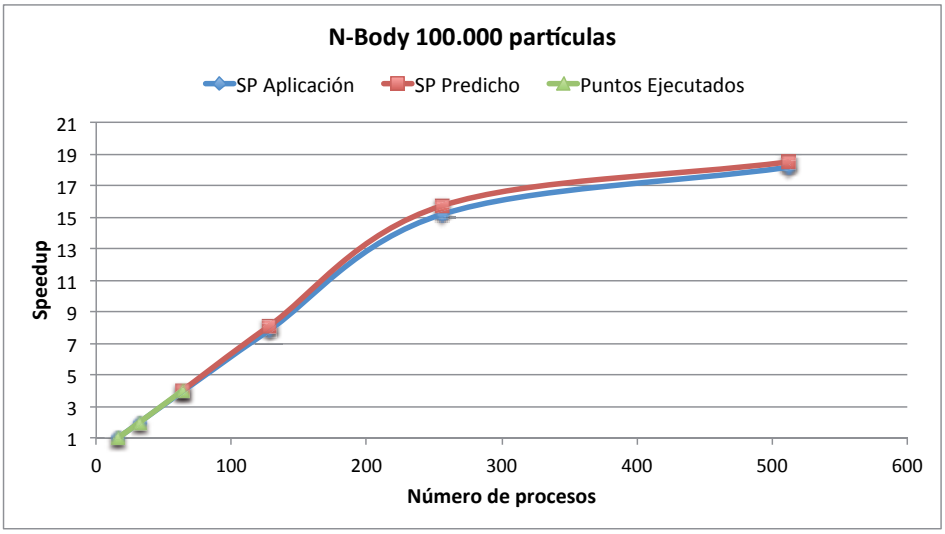


Figura 9.2: Speedup de la aplicación N-Body para 100,000 partículas

Tabla 9.15: Predicción del tiempo de ejecución de la aplicación Sweep3D con input.80 11 iteraciones de 16 a 512 procesos

Número de procesos	Recursos utilizados (Cores)	Tiempo predicho (Seg.)	Tiempo de ejecución de la aplicación (Seg.)	Error de predicción (%)
Tiempos obtenidos mediante la ejecución de la firma con PAS2P				
16	16	1138.21	1140.89	0.23
32	32	469.19	473.39	0.88
64	64	239.62	243.22	1.48
128	128	133.28	137.52	3.08
Tiempos obtenidos mediante la metodología P3S				
256	160	90.46	92.73	2.44
512	160	78.04	80.53	3.09

Como se puede apreciar en la gráfica, la aplicación presenta una escalabilidad prácticamente lineal hasta 256 procesos, momento a partir del cual el rendimiento de la aplicación comienza a decaer, introduciendo una mayor ineficiencia en el sistema a medida que aumentamos el número de procesos de la aplicación, por lo que podemos concluir, que el número más apropiado de procesos para ejecutar la aplicación N-Body con 100,000 partículas sería 256 procesos, ya que es el punto donde la eficiencia de la aplicación en el sistema destino es máxima.

9.3.2. Predicción de la escalabilidad para la aplicación Sweep3D

Se ejecutaron las firmas de la aplicación para 16, 32, 64 y 128 procesos, con el *workload* de entrada input.80 con 11 iteraciones, con el objetivo de generar el modelo de la aplicación, mediante el cual construir la traza física escalada (ST4NP) para 256 y 512 procesos, la cual será utilizada como entrada de la herramienta SS. En la tabla 9.15, se muestran los tiempos predichos de la aplicación para 256 y 512 procesos, y los recursos utilizados para obtener la predicción, utilizando la metodología P3S. Como podemos observar, se utilizó un número de recursos máximo de 160 cores para las predicciones de 256 y 512 procesos, obteniendo un error máximo de predicción de un 3.09 % para el tiempo predicho de 512 procesos. Además, en la misma tabla, también se muestran los tiempos predichos obtenidos de la ejecución del conjunto inicial de firmas con la herramienta PAS2P. Para los tiempos predichos con la herramienta PAS2P, se ha obtenido un error de predicción máximo de 3.08 %.

A partir de los puntos predichos con la metodología P3S para 256 y 512 procesos, y los puntos predichos obtenidos mediante la ejecución de las firmas de la aplicación para 16, 32, 64 y 128 procesos, se generó la curva de escalabilidad de la aplicación para el *workload* de entrada input.80 con 11 iteraciones, en el sistema destino. La figura 9.3 muestra el *speedup* predicho, el cual es comparado con el *speedup* real de la aplicación, obtenido ejecutando la aplicación en el sistema. Para calcular el *speedup* se utilizó como punto de referencia la ejecución para 16 procesos.

Como se aprecia en la gráfica, la aplicación escala hasta 256 procesos, momento a partir del cual el rendimiento de la aplicación comienza a decaer, introduciendo una mayor ineficiencia en el sistema a medida que aumentamos el número de procesos de la aplicación, por lo tanto, podemos concluir que el número más apropiado de recursos para ejecutar la aplicación con el *workload* utilizado, si se desea hacer un uso eficiente del sistema, es ejecutar la aplicación utilizando 256 procesos.

9.3.3. Predicción de la escalabilidad para la aplicación SP

Se ejecutaron las firmas de la aplicación para 16, 36, 49 y 64 procesos, con el *workload* de entrada C, con el objetivo de generar el modelo de la aplicación, mediante el cual construir la traza física escalada (ST4NP) para 121, 256 y 484 procesos, la cual será utilizada como entrada de la herramienta SS. En la tabla 9.16 se muestran los tiempos

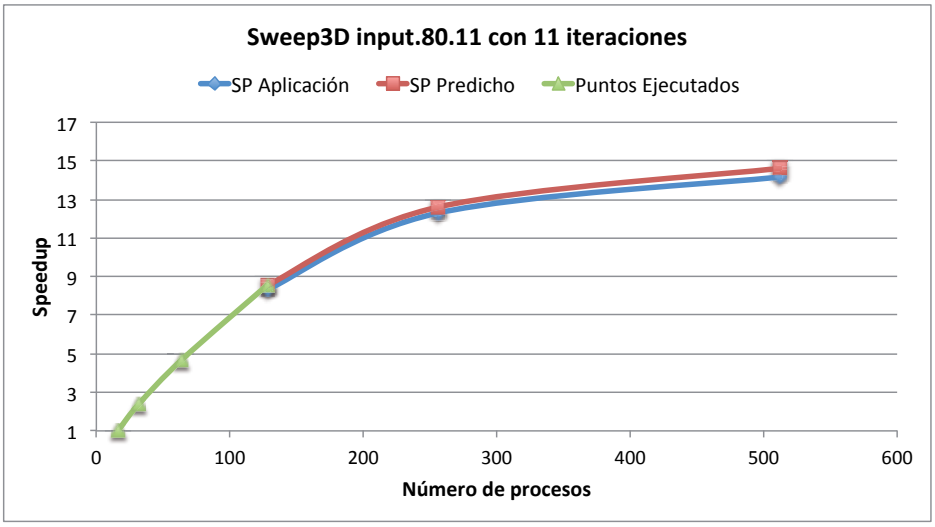


Figura 9.3: Speedup de la aplicación Sweep3D para el workload input.80 con 11 iteraciones

Tabla 9.16: Predicción del tiempo de ejecución de la aplicación SP Clase C de 16 a 484 procesos

Número de procesos	Recursos utilizados (Cores)	Tiempo predicho (Seg.)	Tiempo de ejecución de la aplicación (Seg.)	Error de predicción (%)
Tiempos obtenidos mediante la ejecución de la firma con PAS2P				
16	16	321.03	324.00	0.92
36	36	146.37	148.35	1.33
49	49	107.89	109.99	1.90
64	64	684.65	87.76	3.54
Tiempos obtenidos mediante la metodología P3S				
121	121	42.38	45.13	6.09
256	219	25.28	26.6	4.96
484	219	23.5	21.9	6.80

predichos de la aplicación para 121, 256 y 484 procesos, y los recursos utilizados para obtener la predicción, utilizando la metodología P3S. Como se puede observar en la tabla, se utilizó un número de recursos máximo de 219 cores para las predicciones de 256 y 484 procesos, obteniendo un error máximo de predicción de un 6.80 % para el tiempo predicho de 484 procesos. Además, en la misma tabla, se muestran los tiempos predichos obtenidos de la ejecución del conjunto inicial de firmas con la herramienta PAS2P, como se muestra en la tabla, se obtuvo un error de predicción máximo de 3.05 % para la ejecución de la firma con 64 procesos.

A partir de los puntos predichos con la metodología P3S para 121, 256 y 484 procesos,

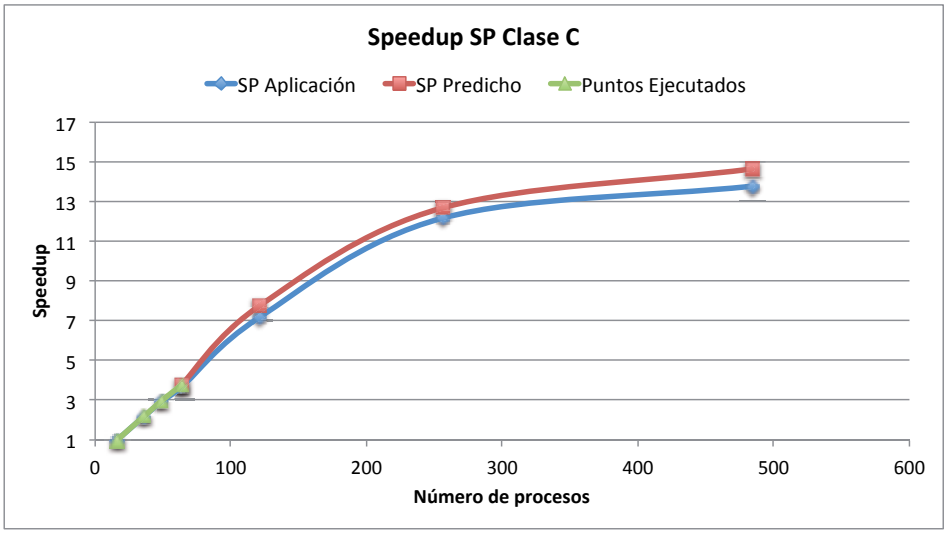


Figura 9.4: Speedup de la aplicación SP Clases C

y los puntos predichos obtenidos mediante la ejecución de las firmas de la aplicación para 16, 36, 49 y 64 procesos, se generó la curva de escalabilidad de la aplicación para el *workload* de entrada C, en el sistema destino. La figura 9.4 muestra el *speedup* predicho, el cual es comparado con el *speedup* real de la aplicación, obtenido ejecutando la aplicación en el sistema. Para calcular el *speedup* se utilizó como punto de referencia la ejecución para 16 procesos.

Como podemos ver en la gráfica, la aplicación SP presenta una escalabilidad prácticamente lineal hasta 256 procesos, momento a partir del cual el rendimiento de la aplicación comienza a decaer, introduciendo una mayor ineficiencia en el sistema a medida que aumentamos el número de procesos de la aplicación, por lo tanto, podemos concluir que el número más apropiado de recursos para ejecutar la aplicación con el *workload* C, si se desea hacer un uso eficiente del sistema, es ejecutar la aplicación utilizando 256 procesos.

9.3.4. Predicción de la escalabilidad para la aplicación para CG

Se ejecutaron las firmas de la aplicación para 16, 32, 64 y 128 procesos, con un *workload* de entrada C, con el objetivo de generar el modelo de la aplicación, mediante el cual construir la traza física escalada (ST4NP) para 256 y 512 procesos, la cual será utilizada como entrada de la herramienta SS. En la tabla 9.17, se muestran los tiempos

Tabla 9.17: Predicción del tiempo de ejecución de la aplicación CG Clase C de 16 a 512 procesos

Número de procesos	Recursos utilizados (Cores)	Tiempo predicho (Seg.)	Tiempo de ejecución de la aplicación (Seg.)	Error de predicción (%)
Tiempos obtenidos mediante la ejecución de la firma con PAS2P				
16	16	2041.52	2049.26	0.42
32	32	1169.42	1178.37	0.75
64	64	751.12	755.13	0.53
128	128	429.25	448.22	4.23
Tiempos obtenidos mediante la metodología P3S				
256	160	242.25	253.22	4.33
512	160	223.72	233.42	4.14

predichos de la aplicación para 256 y 512 procesos, y los recursos utilizados para obtener la predicción, utilizando la metodología P3S. Como podemos observar, se utilizó un número de recursos máximo de 160 cores para las predicciones de 256 y 512 procesos, obteniendo un error máximo de predicción de un 4.33 % para el tiempo predicho de 256 procesos. Además, en la misma tabla, se muestran los tiempos predichos obtenidos de la ejecución del conjunto inicial de firmas con la herramienta PAS2P, obteniendo un error máximo de predicción del 4.23 % para la predicción de la aplicación para 128 procesos.

A partir de los puntos predichos con la metodología P3S para 256 y 512 procesos, y de los puntos predichos obtenidos mediante la ejecución de las firmas de la aplicación para 16, 32, 64 y 128 procesos, se generó la curva de escalabilidad de la aplicación para el *workload* de entrada C, en el sistema destino. La figura 9.5 muestra el *speedup* predicho, el cual es comparado con el *speedup* real de la aplicación, obtenido ejecutando la aplicación en el sistema. Para calcular el *speedup* se utilizó como punto de referencia la ejecución para 16 procesos.

Como podemos ver, la aplicación CG presenta una escalabilidad prácticamente lineal hasta 256 procesos, momento a partir del cual el rendimiento de la aplicación comienza a decaer, introduciendo una mayor ineficiencia en el sistema a medida que aumentamos el número de procesos de la aplicación, por lo tanto, podemos concluir que el número más apropiado de recursos para ejecutar la aplicación, si se desea hacer un uso eficiente del sistema, es ejecutar la aplicación utilizando 256 procesos.

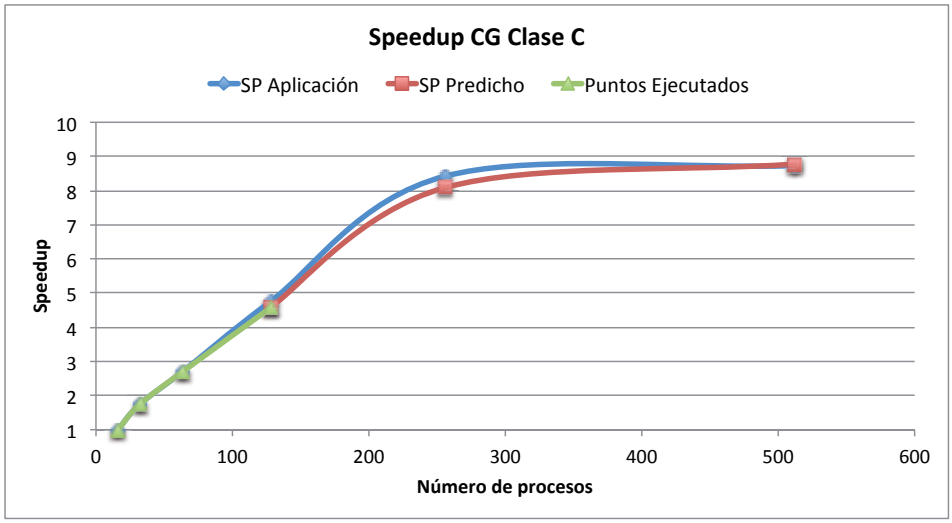


Figura 9.5: Speedup de la aplicación CG Clases C

Tabla 9.18: Predicción del tiempo de ejecución de la aplicación BT Clase C de 16 a 484 procesos

Número de procesos	Recursos utilizados (Cores)	Tiempo predicho (Seg.)	Tiempo de ejecución de la aplicación (Seg.)	Error de predicción (%)
Tiempos obtenidos mediante la ejecución de la firma con PAS2P				
16	16	268.67	272.67	1.48
36	36	124.57	128.07	2.80
49	49	93.13	97.33	4.50
64	64	77.12	80.58	4.48
Tiempos obtenidos mediante la metodología P3S				
121	121	60.27	63.7	5.69
256	219	52.58	54.58	5.69
484	219	46.17	49.37	6.93

9.3.5. Predicción de la escalabilidad para la aplicación para BT

Se ejecutaron las firmas de la aplicación para 16, 36, 49 y 64 procesos, con el *workload* de entrada C, con el objetivo de generar el modelo de la aplicación, mediante el cual construir la traza física escalada (ST4NP) para 121, 256 y 484 procesos, la cual será utilizada como entrada de la herramienta SS. En la tabla 9.18, se muestran los tiempos predichos de la aplicación para 121, 256 y 484 procesos, y los recursos utilizados para obtener la predicción, utilizando la metodología P3S. Como podemos observar en la tabla, se utilizó un número de recursos máximo de 219 cores para las predicciones de 256 y 512 procesos, obteniendo un error máximo de predicción de un 6.93 % para

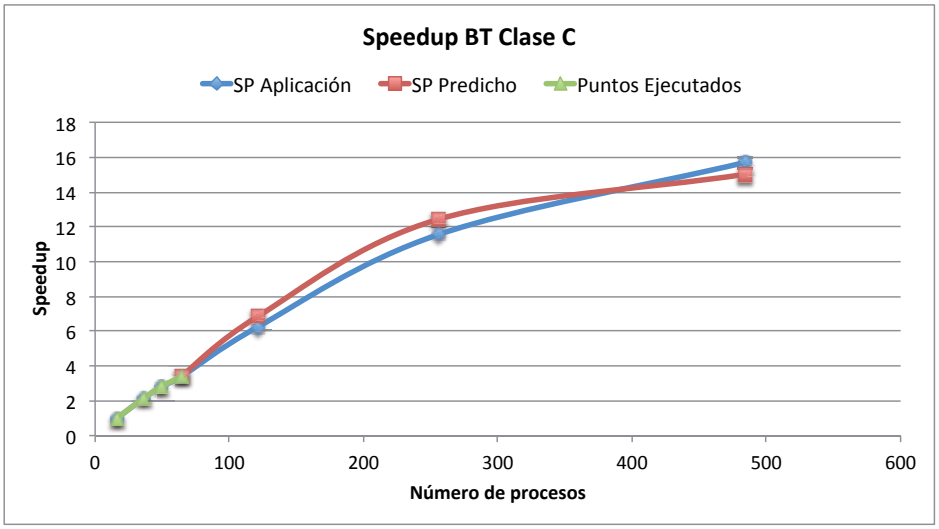


Figura 9.6: Speedup de la aplicación BT Clases C

predicción de la aplicación para 484 procesos. Además, en la misma tabla, se muestran los tiempos predichos obtenidos de la ejecución del conjunto inicial de firmas con la herramienta PAS2P, obteniendo un error de predicción máximo de un 4.50 % para la predicción de la aplicación para 64 procesos.

A partir de los puntos predichos con la metodología P3S para 121, 256 y 484 procesos, y de los puntos predichos obtenidos mediante la ejecución de las firmas de la aplicación para 16, 36, 49 y 64 procesos, se generó la curva de escalabilidad de la aplicación para el *workload* de entrada C, en el sistema destino. La figura 9.6 muestra el *speedup* predicho, el cual es comparado con el *speedup* real de la aplicación, obtenido ejecutando la aplicación en el sistema. Para calcular el *speedup* se utilizó como punto de referencia la ejecución para 16 procesos.

Como podemos apreciar en la gráfica, la aplicación escala hasta 121 procesos, momento a partir del cual el rendimiento de la aplicación comienza a decaer, introduciendo una mayor ineficiencia en el sistema a medida que aumentamos el número de procesos de la aplicación, por lo tanto, podemos concluir que el número más apropiado de recursos para ejecutar la aplicación con el *workload* C en el sistema utilizado, serían 121 cores, ya que es el punto donde la eficiencia de la aplicación en el sistema destino es máxima.

Como ha sido mostrado a lo largo de la experimentación presentada en esta sección, además de predecir el rendimiento de la aplicación paralela para un número específico de

procesos, la metodología puede ser utilizada para predecir la curva de escalabilidad de la aplicación paralela con una alta calidad de predicción, por encima del 90 %, utilizando un conjunto reducido de recursos, proporcionando al usuario información para ejecutar su aplicación de forma eficiente en el sistema.

10

Conclusiones

10.1. Introducción

En este capítulo se presentan las conclusiones finales derivadas de este trabajo de investigación. Además, se describen las líneas de trabajo futuras surgidas a partir de la elaboración de esta tesis doctoral, las cuales pueden ser consideradas con el objetivo de continuar trabajando en el área de predicción del rendimiento de la escalabilidad de aplicaciones paralelas. Como último punto, se presentan las contribuciones realizadas surgidas de este trabajo.

10.2. Conclusiones finales

Durante este trabajo se ha presentado la metodología P3S *(Prediction of Parallel Program Scalability)*, la cual permite predecir el comportamiento de la escalabilidad fuerte de aplicaciones paralelas de paso de mensajes, en un determinado sistema, de una manera rápida y eficiente, utilizando un conjunto limitado de recursos.

La metodología P3S, está basada en analizar el comportamiento repetitivo de las aplicaciones paralelas de paso de mensajes. Este tipo de aplicaciones están compuestas por un conjunto de fases identificables, las cuales se van repitiendo a lo largo de toda la aplicación, independientemente del número de procesos de la aplicación. La metodología consta de tres etapas. Una primera etapa de caracterización, donde se caracterizan las fases relevantes de la aplicación paralela, a partir de la información obtenida al ejecutar un conjunto de firmas de la aplicación, para un número reducido de procesos. Una segunda etapa de modelización de la aplicación, donde se genera el modelo lógico

que describe el comportamiento de escalabilidad de cada fase relevante de la aplicación paralela, a partir del cual construir la traza lógica escalable de la aplicación. Esta traza permitirá predecir el comportamiento lógico de la aplicación, a medida que se aumenta su número de procesos. Finalmente, la última etapa de la metodología consiste en predecir el rendimiento de la aplicación para un número específico de procesos, en un computador paralelo concreto. Para ello, la traza lógica escalable será concretada para el número de procesos para el cual se desea predecir el rendimiento de la aplicación, con el objetivo de estimar el tiempo de cómputo y comunicación de cada fase relevante de la aplicación, para ese número de procesos, con la finalidad de obtener el tiempo predicho de la aplicación.

Actualmente, la metodología está enfocada a aplicaciones paralelas de paso de mensajes, las cuales han sido desarrolladas utilizando patrones de comunicación punto a punto, siguiendo reglas de comportamiento a medida que la aplicación escala.

Con el objetivo de estudiar y analizar la robustez de cada una de las tres etapas de la metodología, se validó independientemente cada etapa, utilizando un conjunto representativo de aplicaciones científicas de paso de mensajes, siendo satisfactorios todos los resultados obtenidos para cada etapa de la metodología.

La etapa de modelizado lógico de la aplicación ha sido validada para seis aplicaciones científicas de paso de mensajes, las cuales poseen algoritmos de comunicación representativos en el mundo de las aplicaciones científicas de paso de mensajes. El patrón de comunicación de cada fase de las aplicaciones paralelas utilizadas fue predicho correctamente en todos los casos. El volumen de comunicación fue predicho con un error medio aproximado del 3 %, el número de instrucciones de cómputo fue predicho con un error medio aproximado del 2 % y, finalmente, el peso predicho de la fase fue obtenido con un error del 0 % para todas las fases de las aplicaciones utilizadas.

Para mostrar el funcionamiento del modelo de predicción del tiempo cómputo, se utilizaron siete aplicaciones científicas de paso de mensajes, las cuales permiten ser escaladas para un elevado número de procesos. El error medio de predicción del tiempo de cómputo, para todas las fases de las aplicaciones utilizadas, fue de aproximadamente el 4.5 %.

Finalmente, los experimentos realizados para la validación de la predicción del tiempo de ejecución de la aplicación, y la generación de la curva de escalabilidad de la

aplicación, se llevaron a cabo utilizando cinco aplicaciones científicas de paso de mensajes. Los resultados obtenidos presentan un error de predicción medio aproximado del 3 %.

Además de permitir predecir el rendimiento de la escalabilidad de las aplicaciones paralelas de paso de mensajes, la metodología proporciona al usuario información del comportamiento de cada fase relevante de la aplicación, a medida que se incrementa su número de procesos, la cual puede ser analizada, con el objetivo de detectar posibles fases ineficientes, las cuales se conviertan en cuellos de botella potenciales a medida que la aplicación escala.

Por otro lado, la metodología también brinda la posibilidad a los administradores de sistemas de planificar políticas de *scheduling*, con el objetivo de hacer un uso eficiente del sistema, ya que nos permite predecir de forma rápida un conjunto de aplicaciones a ser ejecutadas en el sistema, con el objetivo de diseñar la mejor política de planificación en las colas de entrada al sistema.

Con el objetivo de obtener los resultados de una forma automática y rápida, se ha desarrollado un prototipo que implementa la metodología P3S.

10.3. Trabajo futuro y líneas abiertas

A continuación, se presentan las líneas de trabajo futuro surgidas a partir de la elaboración de este trabajo, las cuales pueden ser consideradas con el objetivo de continuar trabajando en el área de predicción del rendimiento de la escalabilidad de aplicaciones paralelas, y más concretamente en la extensión de la metodología P3S.

Actualmente, la metodología está desarrollada para predecir el rendimiento de la escalabilidad fuerte de aplicaciones paralelas de paso de mensajes en un determinado sistema. Debido al gran número de aplicaciones científicas de paso de mensajes que trabajan con escalabilidad débil (diferentes tamaños de *workloads*), las cuales además tienen que proporcionar el resultado en un tiempo determinado, sería interesante extender la metodología para este tipo de escalabilidad. Con el objetivo de obtener la predicción de rendimiento para la escalabilidad débil, las etapas de caracterización y modelizado lógico de la aplicación, pueden continuar utilizándose de la misma forma, ya que el modelizado de la aplicación es independiente del tipo de escalabilidad utilizado, centrando los esfuerzos en la etapa de predicción del rendimiento, para la cual se

tendrían que proponer nuevos modelos con el objetivo de predecir el rendimiento de la aplicación.

La metodología permite predecir el rendimiento de la aplicación sobre un sistema determinado. Un punto importante sería poder extrapolar o proyectar el tiempo de cómputo de cada fase de la traza física escalada para otros sistemas, con el fin de poder utilizar la traza física escalada generada en un sistema base, para predecir el rendimiento de la aplicación en diferentes sistemas destino.

Aunque la metodología ha sido ampliamente validada para un amplio conjunto de aplicaciones científicas, debido a la limitación del sistema del que se disponía para realizar la validación experimental, no ha podido ser validada para un elevado número de procesos (del orden de 8k o 16k). Resultaría interesante validar la metodología sobre un elevado número de procesos.

Durante la etapa de modelizado lógico de la aplicación, el modelo propuesto para predecir el patrón de comunicación de cada fase, está diseñado para patrones de comunicación punto a punto, no considerando actualmente las primitivas colectivas que proporciona MPI. Resultaría interesante extender el modelo de comunicación para considerar aplicaciones que posean primitivas colectivas para intercambiar datos. Se ha empezado a diseñar una extensión del modelo de comunicación, el cual considera las primitivas colectivas. Para ello, se ha tenido en cuenta que las primitivas colectivas, internamente realizan comunicaciones punto a punto para intercambiar la información entre todos los procesos que componen la aplicación. El objetivo es analizar el comportamiento interno de las comunicaciones punto a punto que realizan las primitivas colectivas, con el fin de modelizar su evolución, a medida que se incrementa el número de procesos de la aplicación.

En la etapa de predicción del rendimiento de la aplicación, el método propuesto con el cual ajustar el modelo de regresión de cómputo mediante la obtención del punto lejano, está limitado a que el número de instrucciones de la fase se mantenga constante, a medida que la aplicación escala. Una posible mejora, consistiría en proponer métodos para seguir obtenido el punto lejano para fases irregulares, las cuales no mantengan constante el número de instrucciones a medida que la aplicación escala, ya que nos permitiría mejorar la calidad de predicción de la fase, y por ende, la calidad de predicción del rendimiento de la aplicación.

A la hora de predecir el tiempo de comunicación de los eventos que componen las fases, la herramienta SS actualmente no permite ejecutar primitivas colectivas MPI en un conjunto reducido de recursos. Puesto que internamente las primitivas colectivas se transforman en comunicaciones punto a punto, una posible alternativa sería transformar este tipo de primitivas por comunicaciones punto a punto para poder ser ejecutadas por la herramienta SS.

Actualmente, la metodología no considera las aplicaciones paralelas de paso de mensajes con Entrada/Salida. Una línea futura podría ser modelizar el comportamiento de la Entrada/Salida a medida que la aplicación escala, con el fin de generar sus reglas generales de comportamiento. Esto nos permitiría predecir las fases de la aplicación que tuvieran Entrada/Salida para un número mayor de procesos.

10.4. Lista de publicaciones

A continuación, se muestran las diferentes publicaciones surgidas de este trabajo:

1. **Sandra Mendez, Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. A new approach for Analyzing I/O in parallel scientific applications. CACIC 2012 - XVIII CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACION. Páginas 337-346. 2012**

   BREVE RESUMEN: En este trabajo se presenta una metodología para analizar las aplicaciones paralelas de paso de mensajes considerando: cómputo, comunicación y operaciones de E/S mediante el uso de la herramienta PAS2P como analizador. En este trabajo colaboré en el análisis y evaluación de la parte de cómputo y comunicación de la aplicación parela, con el objetivo de extender PAS2P para considerar aplicaciones científicas con E/S.

2. **Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. A tool for selecting the right target machine for Parallel Scientific Applications. Proceedings of the International Conference on Computational Science, (ICCS). Páginas: 1824-1833. 2013**

   BREVE RESUMEN: En este trabajo se presenta la herramienta PAS2P y sus utilidades. Se ha dedicado especial atención a las nuevas funcionalidades de la herramienta, donde además de predecir el rendimiento de la aplicación en máquinas

destino, se ha dotado a la herramienta de la capacidad de analizar las aplicaciones paralelas de paso de mensajes dinámicamente. Esta nueva capacidad ha sido utilizada para identificar las fases de la aplicación a medida que la aplicación incrementa su número de procesos.

3. **Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. Metodología para predecir el rendimiento de aplicaciones paralelas. XXIV Jornadas Sarteco. Páginas: 163-168. 2013**

   BREVE RESUMEN: En este trabajo se presenta una visión general de la metodología P3S, explicando cada etapa de la cual está compuesta.

4. **Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. Predicting the communication pattern evolution for scalability analysis. CACIC 2013 - XIX CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACION. Páginas: 316 - 325. 2013**

   BREVE RESUMEN: Este trabajo se centra en la etapa de modelizado de las comunicaciones de la aplicación. Se presenta el conjunto de algoritmos mediante los cuales modelizar el patrón de comunicaciones de la aplicación a medida que la aplicación escala su número de procesos.

5. **Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. Analysis of scalability: A parallel application model approach. IEEE CLUSTER 2014. Páginas: 294-295. 2014**

   BREVE RESUMEN: En este trabajo se presenta una visión general metodología P3S, centrándose en la etapa de modelizado de la aplicación.

6. **Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. Modeling Parallel Applications for Scalability Analysis: An approach to predict the communication pattern. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA. 2015. Aceptado**

   BREVE RESUMEN: En este trabajo se presenta la etapa de la metodología P3S correspondiente a la modelización lógica del patrón de comunicación de la aplicación.

7. **Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. Scalability of Parallel Applications: An approach to predict the computational behavior. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA. 2015. Aceptado**

   BREVE RESUMEN: En este trabajo se presenta la etapa de la metodología P3S correspondiente al modelo de predicción del tiempo de cómputo.

8. **Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. Synthetic Signature Program for Performance Scalability. Proceedings of the International Conference on Parallel Processing and Applied Mathematics, PPAM. 2015. Aceptado**

   BREVE RESUMEN: En este trabajo se presenta la herramienta *Synthetic Signature (SS)*, utilizada para predecir el tiempo de comunicación y obtener el rendimiento de la aplicación para N procesos.

Apéndice A

Herramienta PAS2P

A.1. Introducción

Este apéndice presenta una breve descripción sobre la herramienta PAS2P [8], la cual permite analizar y predecir el rendimiento de las aplicaciones paralelas de paso de mensajes en sistemas destino. La herramienta PAS2P se ha desarrollado a partir de la metodología PAS2P [6] [64], con el objetivo de resultar transparente y automática al usuario.

A la hora de analizar el rendimiento de las aplicaciones paralelas, la complejidad recae en poder entender el comportamiento de la aplicación durante su ejecución en un determinado sistema, obtener las fases significativas que tienen impacto en el rendimiento, y analizarlas con el fin de detectar y aislar problemas de rendimiento. Por otro lado, predecir el rendimiento de aplicaciones paralelas requiere un elevado período de tiempo, debido a que el mejor predictor es usar la propia aplicación, pero ejecutar la aplicación completa requiere un tiempo elevado, especialmente si queremos predecir el rendimiento en varios sistemas.

La metodología PAS2P intenta resolver este tipo de problemas mediante la extracción de la firma de la aplicación. Esta firma, representa el comportamiento significativo de la aplicación, ya que está construida seleccionando únicamente las partes relevantes (fases) del código de la aplicación, es decir, las que tienen impacto en el rendimiento, y su frecuencia de repetición (pesos). Una vez la firma de la aplicación ha sido obtenida, puede ser utilizada para analizar el comportamiento cómputo-comunicación de la aplicación de forma eficiente, ya que se centra sólo en analizar las fases relevantes de la

aplicación. Por otro lado, también puede utilizarse para predecir el rendimiento de la aplicación en sistemas destino con un alto grado de predicción (por encima del 90 %) en un período reducido de tiempo.

Debido a la complejidad de la metodología, tanto generar como ejecutar la firma de la aplicación no resulta un proceso trivial para los usuarios. Con el objetivo de facilitar su usabilidad, se ha desarrollado la herramienta PAS2P [8], la cual permite generar y ejecutar la firma de la aplicación de forma transparente y automática.

A.2. Descripción de la herramienta PAS2P

La figura A.1 ilustra los módulos de los cuales se compone la herramienta. Como se puede apreciar, la herramienta está compuesta por 4 módulos: el *módulo de instrumentación*, encargado de instrumentar la aplicación dinámicamente para generar el log de trazas de cada proceso de la aplicación; el *módulo de análisis,* dedicado a obtener las fases relevantes de la aplicación y su frecuencia de repetición (peso); el *módulo de generación de la firma*, encargado de generar la firma de la aplicación a partir de las fases relevantes y sus pesos, y finalmente, el *módulo de predicción de rendimiento*, dedicado a ejecutar la firma en sistemas destino con el objetivo de predecir el rendimiento de la aplicación.

La herramienta está compuesta de dos etapas claramente diferenciadas, una primera etapa de generación de la firma, y una segunda etapa de ejecución de la firma. Durante la primera etapa, la herramienta se ejecuta en la máquina base, con el objetivo de instrumentar la aplicación dinámicamente y extraer sus fases significativas y pesos, con las cuales generar la firma de la aplicación. Una vez se ha obtenido al firma de la aplicación, se lleva a cabo la segunda etapa de la herramienta, la cual está dedicada a ejecutar la firma de la aplicación en sistemas destino con la finalidad de analizar y predecir su comportamiento.

A continuación, se presentan en detalle cada una de las etapas de las cuales se compone la herramienta.

A.2.1. Generación de la firma

Con el objetivo de encontrar el comportamiento de la aplicación tanto en cómputo como en comunicación, el módulo de instrumentación instrumenta la aplicación dinámi-

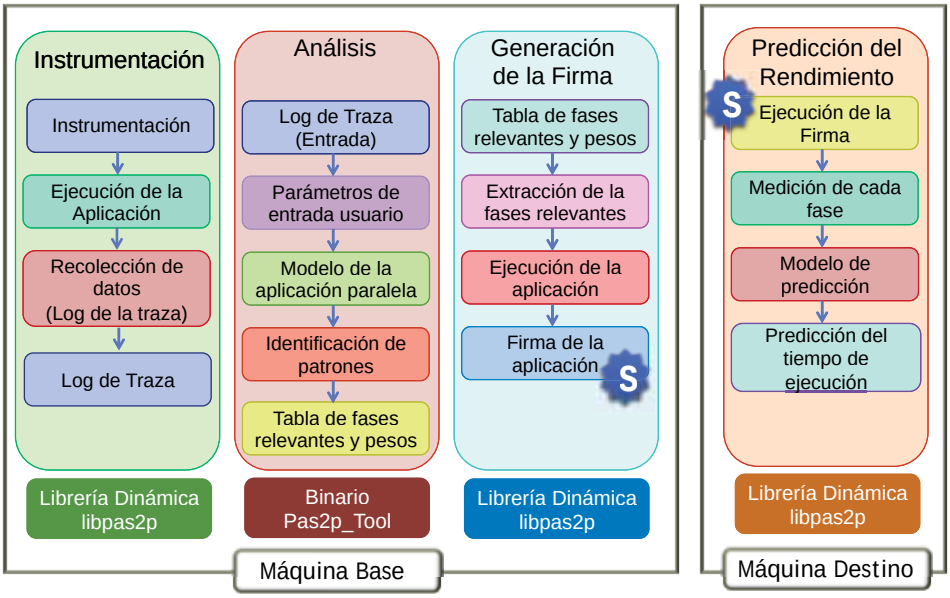


Figura A.1: Etapas de la metodología PAS2P

camente en el sistema base con el objetivo de obtener un log de trazas de cada proceso.

Para ello, utiliza la interposición estándar de funciones *(LD_PRELOAD)* para interceptar y recolectar los eventos de comunicación de la aplicación paralela, de forma dinámica, durante la ejecución de la aplicación. Con el fin de interceptar los eventos de comunicación dinamicamente, utiliza la librería dinámica *PAS2PLib*, la cual se carga en el entorno de ejecución de la aplicación, tal y como se muestra en la figura A.2. Una vez que todos los eventos de comunicación han sido recolectados, se genera un log con la traza de la aplicación.

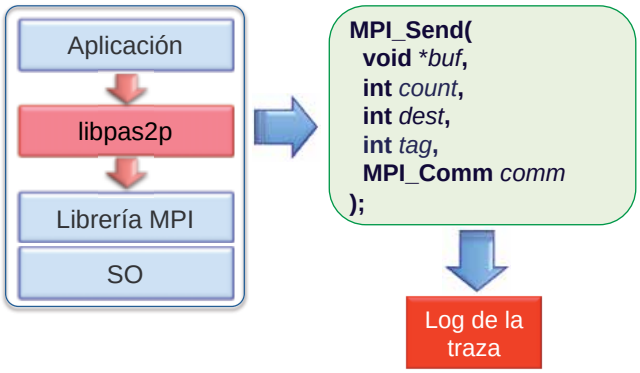


Figura A.2: Interposición Librería PAS2PLib

Una vez se ha generado el log de la aplicación, el módulo de análisis ordenará la traza mediante un reloj lógico global, en función de la ordenación por precedencias *(happened-before relation)* de los eventos. Ordenado el log de trazas, se obtiene el modelo lógico de la aplicación. El algoritmo para obtener este modelo lógico está basado en el algoritmo de Lamport [65]. Una vez se ha obtenido el modelo de la aplicación, se procederá a identificar los patrones de la aplicación con el objetivo de encontrar el comportamiento representativo de la aplicación. Este comportamiento es encontrado a través de un procedimiento que identifica por similitud las secuencias más relevantes de la aplicación (fases) y les asigna un peso en función del número de veces que ocurrió en la aplicación (peso), tal y como se muestra en la figura A.3. Las fases están delimitadas por dos eventos de comunicación, y estarán compuestas por patrones de comunicación y cómputo.

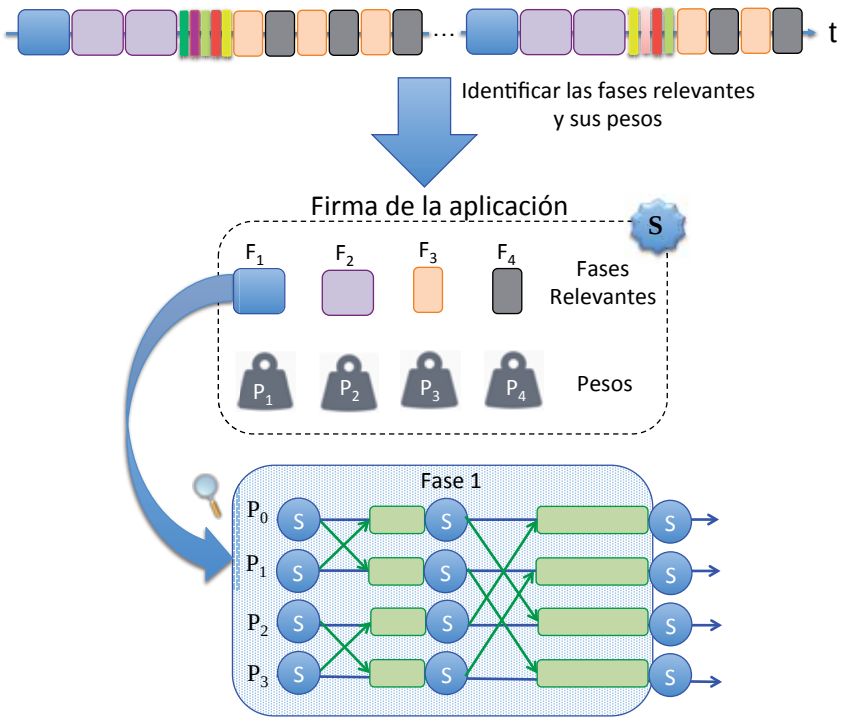


Figura A.3: Generación de la firma de la aplicación

Identificadas las fases relevantes de la aplicación, el último paso es generar la firma de la aplicación, para ello, se utiliza el módulo de generación de la firma, el cual re-ejecutará la aplicación con el objetivo de extraer las partes del código que constituirán la

firma. Una vez todas las fases relevantes han sido seleccionadas y extraídas, se generará la firma de la aplicación, que contendrá el conjunto de fases relevantes y su frecuencia de repetición (pesos).

A.2.2. Ejecución de la firma

Una vez la firma de la aplicación ha sido generada, puede ser utilizada para analizar y predecir el tiempo de ejecución de la aplicación en sistemas destino. Para ello, se utiliza el módulo de predicción de rendimiento, el cual ejecutará la firma en sistemas destino, con el objetivo de predecir el tiempo de ejecución de la aplicación. A fin de predecir el tiempo total de la aplicación, el módulo utiliza la ecuación A.1, donde PET es el tiempo de ejecución predicho, n es el número de fases, TEFasei es el tiempo de ejecución de la fase y finalmente Pi es el peso de la fase i.

$$PET = \sum_{i=1}^{n} (TEFasei)(Pi) \tag{A.1}$$

Además de predecir el tiempo de ejecución de la aplicación, la firma extrae información del comportamiento de cada una de las fases, la cual podrá ser utilizada para caracterizar y analizar las fases relevantes de la aplicación en el sistema destino. Esta información es guardada en una traza por proceso, tal y como se muestra en la figura A.4. Como se puede observar en la figura, la traza proporciona información del ID de la fase, el tipo de primitiva MPI, el origen y destino de cada mensaje, el volumen de comunicación en bytes, el tiempo de comunicación en nanosegundos y, el tiempo de cómputo entre eventos de comunicación en nanosegundos.

Fase ID	Tipo de primitiva	Origen	Destino	Volumen de comunicación (Bytes)	Tiempo de comunicación (ns)	Tiempo de cómputo (ns)
1	MPI_Irecv	0	1	4000	9	4000
1	MPI_Send	0	1	4000	12	2345
1	MPI_Wait	0	1	4000	989	83593535
1	MPI_Irecv	0	2	2000	9	7533
1	MPI_Send	0	2	2000	14	5366
1	MPI_Wait	0	2	2000	983	45326854

Figura A.4: Ejemplo de traza de la firma para el proceso 1

A.3. Integración del módulo de instrumentación con la librería de contadores hardware PAPI

Puesto que originalmente la herramienta PAS2P no recolectaba información de bajo nivel, como por ejemplo el número de instrucciones, necesaria en la metodología P3S para analizar y modelizar las fases de la aplicación paralela, se hace necesario dotar al módulo de instrumentación de nuevas funcionalidades con el objetivo de recolectar este tipo de información.

A fin de obtener información de bajo nivel para cada fase de la aplicación, tal y como se muestra en la figura A.5, se ha integrado el módulo de instrumentación con la librería de contadores hardware PAPI [66] *(Performance Application Programming Interface)*, la cual permite obtener información de los contadores hardware del procesador, con el objetivo de analizar el rendimiento de las aplicaciones.

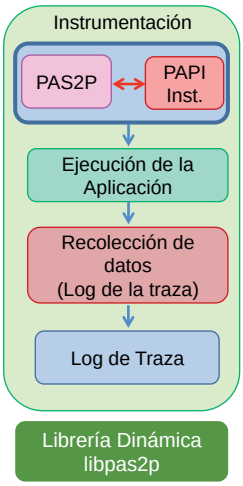


Figura A.5: Modulo de instrumentación integrado con librería PAPI

Puesto que uno de los objetivos iniciales de la herramienta PAS2P era generar logs de trazas lo más compactados posible, los cuales posean únicamente la información imprescindible para analizar y predecir las fases de la aplicación, se ha añadido a la traza únicamente la información necesaria utilizada en la diferentes etapas de la metodología P3S. La tabla A.6 muestra la información del log obtenida después de integrar el módulo con la librería PAPI. Como se puede apreciar en la tabla, se han añadido 3 campos nuevos, el número de instrucciones, los ciclos y el número de fallos del último nivel de la jerarquía de memoria cache (LLC).

Fase ID	Tipo de primitiva	Origen	Destino	Volumen de comunicación (Bytes)	Tiempo de cómputo (ns)	Tiempo de comunicación (ns)	Número de instrucciones	Número de ciclos	Fallos de cache L2
1	MPI_Irecv	0	1	4000	4000	9	756	256	32
1	MPI_Send	0	1	4000	2345	12	456	156	13
1	MPI_Wait	0	1	4000	83593535	989	456746733	876546	7654
1	MPI_Irecv	0	2	2000	7533	9	975	345	45
1	MPI_Send	0	2	2000	5366	14	875	256	65
1	MPI_Wait	0	2	2000	45326854	983	357876543	9876567	3466

Figura A.6: Ejemplo de traza de la firma con información de bajo nivel para el proceso 0

A.4. Paralelización del módulo de Análisis

Originalmente, el módulo de análisis estaba compuesto por un binario serie llamado *pas2p_tool*, el cual, a partir de las trazas de cada proceso y, basándose en el algoritmo de Lamport, creaba una traza lógica con un reloj lógico global, con el objetivo de mantener las dependencias entre los eventos de comunicación, con el fin de generar el modelo lógico abstracto de la aplicación, el cual es independiente de la máquina y es utilizado para obtener y extraer las fases relevantes de la aplicación.

Dependiendo de la memoria del nodo del clúster y del tamaño de la traza lógica, el cual depende tanto del número de procesos como del tiempo de ejecución de la aplicación, puede resultar que el tamaño la traza lógica sea mayor que el tamaño de la memoria del nodo, forzando al binario a tener que hacer swapping al disco, incrementando considerablemente el tiempo de análisis.

Puesto que una de las premisas de la metodología P3S, es obtener la predicción del rendimiento de la escalabilidad de aplicaciones paralelas de paso de mensajes en un período acotado de tiempo, se ha paralelizado el módulo de análisis de la herramienta PAS2P, con la finalidad de obtener el conjunto inicial de firmas utilizado por metodología P3S, en un período limitado de tiempo.

Con el objetivo de evitar los accesos a la memoria virtual y reducir el tiempo de generación del modelo paralelo de la aplicación, se ha paralelizado el módulo de análisis mediante el paradigma de memoria distribuida, utilizando la librería de paso de mensajes MPI, con la finalidad de aprovechar los recursos utilizados por el módulo de instrumentación para ejecutar la aplicación.

El modulo paralelo, tal y como se muestra en la figura A.7, se compone de 3 etapas: carga de datos, modelo abstracto de la aplicación e identificación del patrón, las cuales,

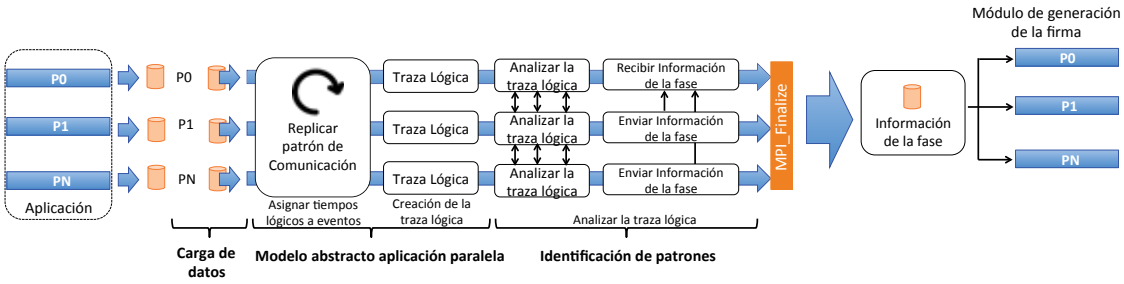


Figura A.7: Visión general de la paralización del módulo analizado

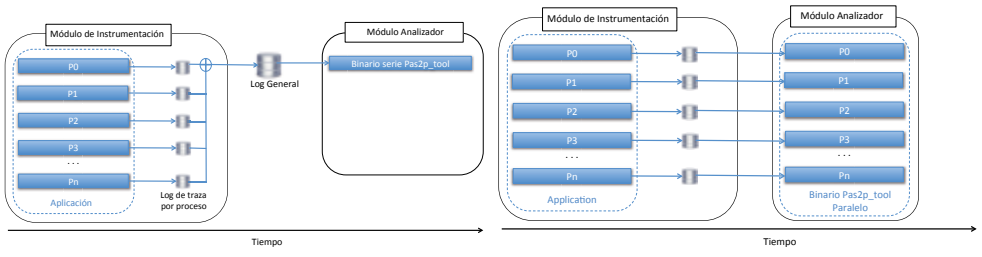
como se puede ver en la imagen, suceden en paralelo hasta obtener las fases relevantes de la aplicación.

A continuación, se explican las 3 etapas comparándolas brevemente con las etapas serie del modulo de análisis inicial.

A.4.1. Carga de datos

Con el objetivo de generar el modelo abstracto de la aplicación, tal y como se muestra en la figura A.8.a, el módulo serie concatena los Logs de trazas de cada proceso en un log global, el cual será cargado en memoria con el objetivo de ser analizado en un proceso serie.

A diferencia del módulo serie, el módulo paralelo aprovecha los recursos utilizados durante la ejecución de la aplicación, creando el mismo número de procesos que fueron utilizados para ejecutar la aplicación, tal y como se muestra en la figura A.8.b. Una vez que los procesos han sido creados, cada proceso carga su propio log de traza, de esta forma se consigue reducir la memoria total por nodo.



(a) Módulo Analizador Serie (b) Módulo Analizador Paralelo

Figura A.8: Carga del Log de traza

Puesto que ahora el mismo evento de comunicación (origen-destino) se encuentra en dos procesos diferentes, los cuales no comparten el mismo espacio de memoria, es necesario etiquetar de forma única los eventos de comunicación a fin de poder relacionar los mensajes que pertenezcan al mismo evento de comunicación, entre los diferentes procesos. Para ello, tal y como se muestra en la figura A.9, se crea un ID relación único para cada evento de comunicación. Con el fin de crear un ID único se utilizan *Bit fields*, los cuales se crean a partir de los siguientes parámetros:

- *Contador:* Se utilizan dos vectores, uno de envío y otro de recepción donde cada proceso conoce el número de mensajes enviados y recibidos de cada proceso.
- *Proceso:* Proceso donde el evento ocurre (origen del mensajes)
- *Destino:* Destino de la comunicación

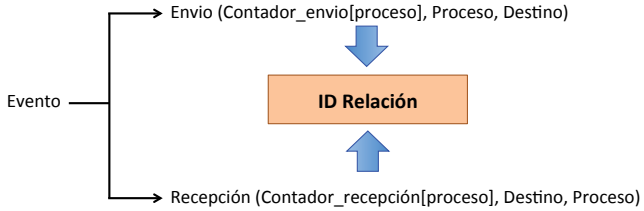


Figura A.9: Generación del ID Relación

Mediante este conjunto de parámetros se crea un ID único para cada evento de comunicación, de esta forma es posible relacionar los eventos entre los diferentes procesos, con el objetivo de generar el modelo abstracto de la aplicación, tal y como veremos en la siguiente etapa.

A.4.2. Modelo abstracto de la aplicación

Una vez se ha cargado el log de traza, el siguiente paso es crear el modelo abstracto de la aplicación, para ello, es necesario dotar a cada evento de comunicación de un Tiempo Lógico (TL), mediante el cual ordenar los eventos de comunicación. Con el objetivo de ordenar estos eventos, se utiliza un algoritmo basado en Lamport [65].

A continuación, se muestra el algoritmo paralelo propuesto mediante el cual generar el modelo abstracto de la aplicación.

Puesto que cada proceso tiene su tog de traza, es necesario enviar el Tiempo Lógico (TL) entre los diferentes procesos, con el objetivo de asignar a los eventos de recepción su TL. Tomando ventaja de la distribución de los datos, a la hora de diseñar el algoritmo, se optó por replicar el patrón de comunicación de la aplicación para enviar el TL entre los diferentes procesos. Para ello, si el algoritmo encuentra una primitiva de envío la convierte en un MPI_Send, y envía su TL como contenido del mensaje, mientras que si es un mensaje de recepción se convierte en una primitiva MPI_Recv. Cuando una colectiva es detectada, se convierte en una primitiva MPI_Allreduce, y se selecciona el máximo TL, como tiempo lógico.

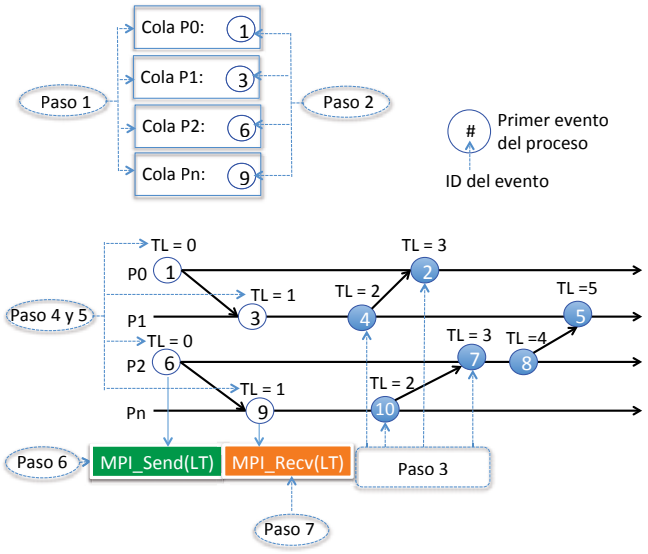


Figura A.10: Insertar tiempos lógicos utilizando el algoritmo de ordenación paralelo

A continuación, mediante el ejemplo de la figura A.10, se explican los diferentes pasos del algoritmo paralelo:

1. Una cola para cada proceso es creada y el primer evento de cada proceso es insertado en cada cola.

2. El primer evento (CurrentEvent) es sacado de la cola de cada proceso.

3. El siguiente evento a CurrentEvent (ForwardEvent) es insertado en la cola de cada proceso.

4. Se busca el proceso anterior (BackEvent) al currentEvent con el TL mayor y un tiempo físico mejor que CurrentEvent. Si hay varios eventos con el mismo TL y uno de ellos es de emisión, se toma este. Si no, se toma cualquier evento de recepción.

5. Si el BackEvent es una emisión, el Tiempo Lógico (TL) del CurrentEvent se incrementa un Tiempo Lógico con respecto al BackEvent (BackEvent + 1), y si BackEvent es una recepción, el Tiempo Lógico de currentEvent será igual al del BackEvent. En caso de ser el primer evento de envío del proceso se asigna al CurrentEvent un TL = 0.

6. Si el evento CurrentEvent es un envío, se crea un mensaje MPI_Send el cual tiene como dato el TL de CurrentEvent, y se usa el ID Relación obtenido en la etapa de carga del log, como tag del mensaje. Si el CurrentEvent es un evento de recepción (RecvEvent), se crea un mensaje MPI_Recv el cual tiene como tag, igual que el evento de envío, el ID Relación del menaje, y recibe el TL como contenido del mensaje.

7. Una vez el evento de recepción (RecvEvent) ha recibido el TL, se asigna RecvEvent = CurrentEvent +1.

8. Si la cola no esta vacía, el algoritmo vuelve al paso 3 para continuar asignando tiempos lógicos a los eventos.

Una vez que todos los eventos tienen asignado un tiempo lógico, se crea una tabla con el modelo abstracto de la aplicación. A diferencia de la tabla serie, tal como se muestra en la figura A.11, la tabla generada por el módulo paralelo está distribuida entre los diferentes procesos de la aplicación, teniendo únicamente 1 sola dimensión, la longitud máxima de la cual es el número máximo de eventos (ticks) de la aplicación. De esta manera conseguimos reducir el espacio de memoria del nodo de cómputo.

A.4.3. Identificación de patrones

Con el objetivo de identificar las fases de la aplicación paralela, PAS2P analiza la tabla creada en la etapa anterior, la cual contiene el modelo abstracto de la aplicación. En la versión serie del analizador, la tabla tiene un visión global, estando cargada en

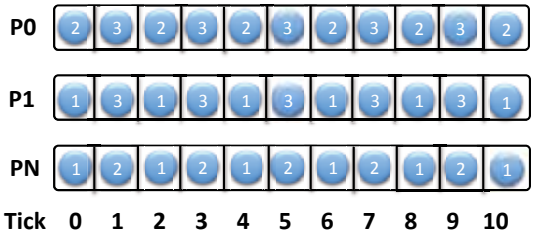


Figura A.11: Traza lógica de la aplicación utilizando el módulo paralelo

la memoria de un nodo del clúster. Esta visión global permite buscar la repetitividad de los eventos, a medida que se avanza en el tiempo lógico (tick) de la tabla, de una manera sencilla, recorriendo el vector que almacena los datos y comparándolos.

Puesto que en la versión paralela la tabla está distribuida entre todos los procesos del analizador, es necesario realizar un proceso de sincronización con el objetivo de poder comparar los eventos de un mismo Tiempo Lógico (tick). Para ello, tal y como se muestra en la figura A.12 (paso 1), cada vez que se avanza un tick en la tabla de cada proceso, el analizador realiza una primitiva MPI_Allreduce, de esta manera se consigue sincronizar todos los procesos a medida que se avanza en la tabla de cada proceso.

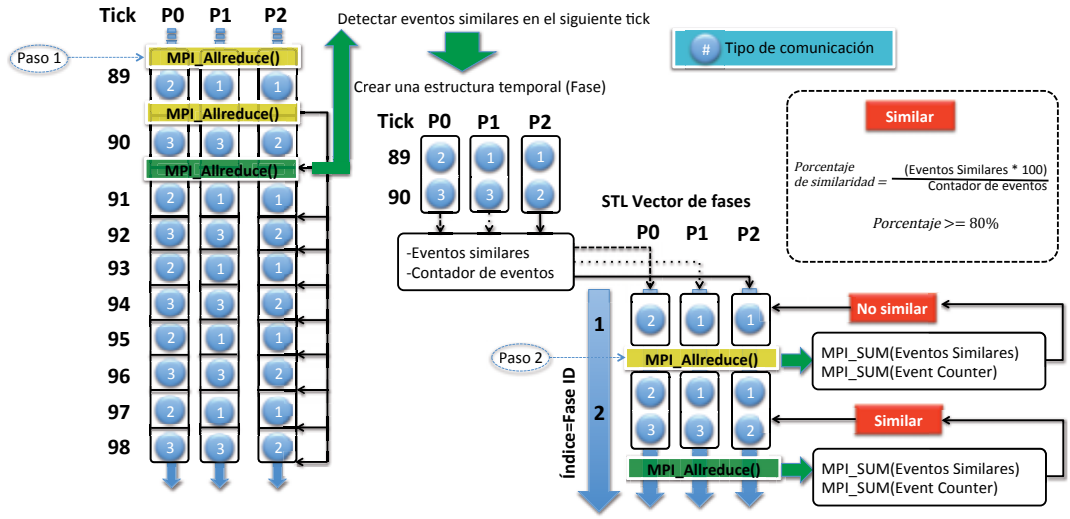


Figura A.12: Paralización del algoritmo de identificación de fases

Tal y como se muestra en la misma figura, con el objetivo de comparar si una fase es similar o no, cada proceso compara localmente la similitud de sus eventos con los eventos de las fases existentes, siguiendo los mismos parámetros de comparación que el

algoritmo serie:

- El tamaño de la fase (número de ticks) tiene que ser el mismo
- Se compara si dos eventos tienen el mismo tipo de comunicación y similar volumen de comunicación (5 % de diferencia).
- El tiempo de cómputo entre dos eventos tiene que ser similar (90 % de similitud o más)

Si para algún proceso existe similitud (similitud local), se realizan dos primitivas colectivas MPI_Allreduce (paso 2) a fin de obtener la similitud global y decidir si la fase ya existe o no. Para ello, el primer MPI_Allreduce envía el número de eventos similares de cada proceso entre la fase temporal y la fase existente, con el objetivo de obtener el número máximo de eventos similares entre las 2 fases. Seguidamente, se realiza otra primitiva colectiva MPI_Allreduce, con el número total de eventos de la fase existente de cada proceso, con el objetivo de obtener los eventos totales de la fase. Una vez tenemos el número total de eventos de la fase y el número total de eventos similares entre la fase temporal y la fase existente, si el porcentaje de similitud es mayor al 80 %, la las dos fases serán iguales y se incrementará el peso de la fase existente, en caso contrario, se definirá como una fase nueva.

Una vez se ha procesado la tabla distribuida, cada proceso del analizador tiene información local de las fases. Con el objetivo de generar la firma de la aplicación, es necesario unificar la información de las fases en un único proceso. Para ello, se realiza un procedimiento de dos etapas. Como se muestra en la figura A.13, en la primera etapa, cada nodo define un proceso máster, el cual es el proceso con el rango menor obtenido con la primitiva MPI_RANK, dicho proceso recibirá la información de las fases del resto de procesos del nodo.

Una vez se ha completado esta primera etapa, y el proceso máster de cada nodo tiene toda la información de las fases del resto de procesos del nodo, el algoritmo crea grupos de nodos siguiendo la función $f(n) = log_n$, donde n es el número total de nodos del clúster utilizados por el módulo analizador. El proceso 0 de cada nodo, el cual contiene la información de las fases de todos los procesos del nodo, envía su información al proceso 0 del nodo menor del grupo. Como se muestra en la figura A.14, este proceso se repite hasta que el proceso 0 del nodo 1 tiene la información de las fases de todos

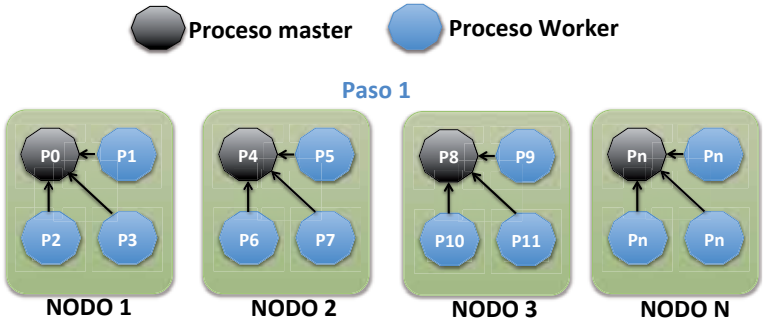


Figura A.13: Paso1: Los procesos másters del nodo reciben los datos de los procesos workers del nodo

los procesos. Mediante este sistema conseguimos que el módulo analizador permita ser escalado a miles de procesos, ya que realizamos comunicaciones entre grupos de nodos, evitando realizar comunicaciones colectivas entre todos procesos del analizador, lo que saturaría la red de interconexión presentando un cuello de botella.

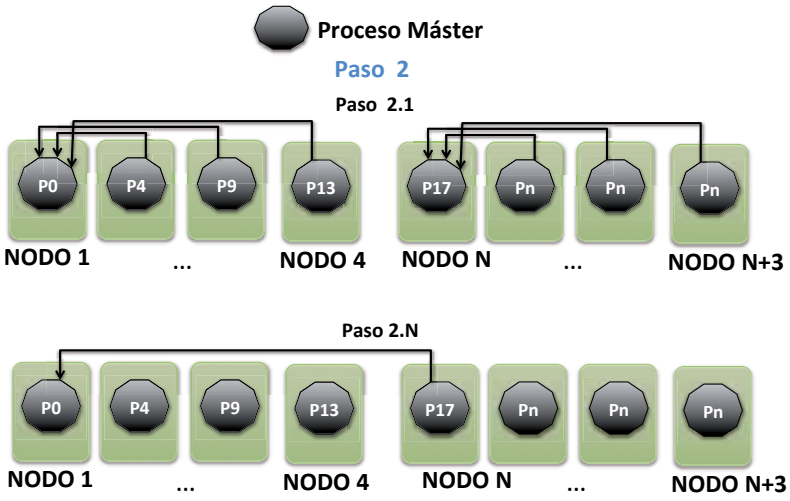


Figura A.14: Paso2: Procesos Masters reciben datos de los procesos master del grupo de nodos

A.4.4. Evaluación Experimental del módulo paralelo

Con el objetivo de validar el módulo paralelo y comprobar las mejoras obtenidas frente al módulo serie, se ejecutaron un conjunto de aplicaciones científicas de paso de mensajes para un Workload constante, y se fue aumentando el número de procesos a

Tabla A.1: Características de los clústers utilizados

Cluster	Hardware	Software
DELL	8 nodos x 64 cores AMD Opteron 6262. 512 Cores en total 64GB de Memoria RAM por nodo. 512 GB de Memoria Total Interconexión Infiniband QDR	Red Hat Linux OpenMPI 1.6.5
NOVA	8 nodes x 16 cores AMD Opteron 6262. 128 Cores en total 48GB de Memoria RAM por nodo. 384 GB de Memoria Total Interconexión Infiniband QDR	Red Hat Linux OpenMPI 1.6.5

fin de escalar las aplicaciones y analizar el rendimiento de la escalabilidad del módulo paralelo.

A continuación, se realiza una descripción general de las aplicaciones seleccionadas:

- *SP (Scalar Pentadiagonal)*: Resuelven sistemas sintéticos no linéales de PDEs utilizando tres algoritmos diferentes que incluyen Block tridiagonal, Scalar Pentadiagonal y symmetric succesive over-relaxation (SSOR).
- *CG (Conjugate Gradient)*: Estima el valor propio más pequeño de una gran matriz escasamente simétrica utilizando la iteración inversa con el método del gradiente conjugado como una subrutina para resolver el sistemas de ecuaciones lineales.
- *LU (Lower-Upper)*: Resuelve un sistema de ecuaciones dinámico de fluidos (CFD), mediante la utilización del método Symmetric Gauss-Seide.
- *Sweep3D*: Aplicación de simulación avanzada y cómputo real. Es capaz de resolver ordenadas discretas de 1-grupo independientes del tiempo en coordenadas 3D cartesianas (XYZ). Esta geometría XYZ se representa por un grid rectangular. El cómputo y la memoria se incrementan substancialmente dependiendo de la carga de trabajo. El incremento de la carga de trabajo predomina en el número de iteraciones internas.

Como entorno experimental, se han utilizado dos clústers con diferente tamaño de memoria principal por nodo, las características del cual son mostradas en la tabla A.1. Las aplicaciones LU y SP fueron ejecutadas en el clúster DELL con 64 GB de memoria principal, mientras que las aplicaciones Sweep3D y CG fueron ejecutadas en el clúster Nova con un tamaño de memoria principal por nodo de 48 GB.

Como se puede comprobar en la figuras A.17, A.16, A.18 A.15, donde se muestran los tiempos de análisis del módulo paralelo frente al módulo serie de las aplicaciones

seleccionadas en la experimentación, se obtiene mejor rendimiento para todas las aplicaciones utilizando el módulo paralelo frente al módulo serie. Esta mejora se debe a dos motivos principales, el primer motivo es debido a que la traza lógica está distribuida entre los diferentes nodos del sistema utilizados por el módulo analizador, evitando así los accesos a memoria virtual (disco físico). El segundo motivo, es la propiedad natural de los algoritmos paralelos, los cuales presentan un rendimiento mucho mayor a los algoritmos serie debido a la paralización de tareas.

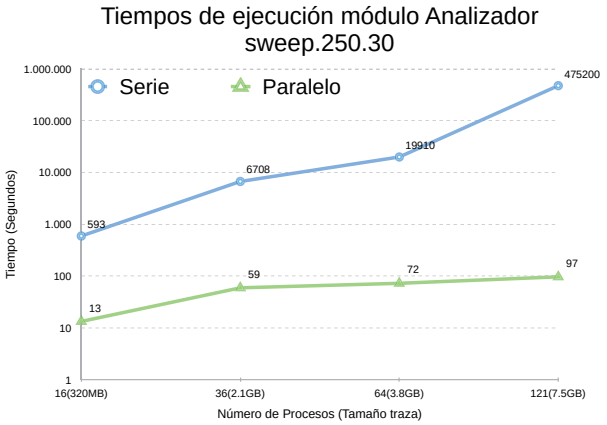
Como se aprecia en la figura A.15.a, donde se muestra el tiempo de análisis tanto de la versión paralela del módulo analizador, como de la versión serie, para la traza lógica de la aplicación Sweep3D, el tiempo de ejecución del módulo paralelo aumenta considerablemente al pasar de 16 procesos a 36 procesos, esto se debe a que para 16 procesos la traza es analizada entre los procesos de un mismo nodo, mientras que a partir de 36 procesos, se hace uso de la red de interconexión GigaEthernet para intercambiar información y obtener las fases de la aplicación. Por ese mismo motivo, a partir de 36 procesos, a medida que se incrementa el número de procesos del analizador, cuando se empieza a hacer uso de la red de interconexión los tiempos se estabilizan aumentando proporcionalmente al incrementar el tamaño de la traza (número de ticks y procesos).

Lo mismo sucede para la ejecución de la figura A.16.a, donde se presenta el rendimiento del módulo para la traza lógica de la aplicación LU. Con el fin de analizar la traza lógica, se ha utilizado el clúster DELL con 64 GB de memoria por nodo. Al igual que sucedía con la aplicación anterior, al pasar de 64 procesos, donde todas las comunicaciones son internas al nodo, a 128 procesos, el tiempo de ejecución del módulo paralelo incrementa considerablemente, debido al uso de la red de interconexión. A partir de 128 procesos, los tiempos de análisis se estabilizan, incrementando proporcionalmente al tamaño de la traza.

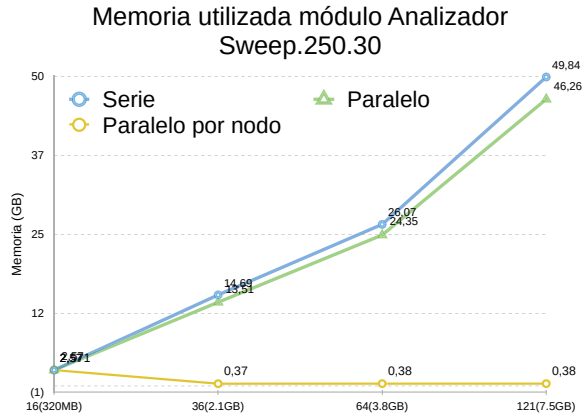
Por otro lado, como se muestra en la figura A.15.b, la versión paralela mantiene la traza lógica de la aplicación Sweep3D distribuida en la memoria de los nodos utilizados por los procesos del módulo analizador, a medida que se incrementa el número de procesos, mientras que la traza serie, a partir de 121 procesos empieza a utilizar la memoria virtual debido a que el tamaño de la traza (49,8 GB), es mayor al tamaño físico de la memoria principal por nodo (48GB). Puesto que la ejecución serie para 121 procesos no cabe en memoria, parte de la traza tiene que ser cargada en memoria virtual,

por lo que el tiempo de análisis, tal y como se muestra en la figura A.15.a, incrementa considerablemente con respecto a la ejecución anterior (64 procesos).

Otro punto interesante a comentar, como se puede apreciar tanto en las figuras A.16.b como A.15.b, es que el tamaño de la memoria por nodo es prácticamente constante al aumentar el número de procesos, con lo cual, nos permitirá escalar el módulo de análisis a miles de procesos, manteniendo el tiempo de análisis acotado.



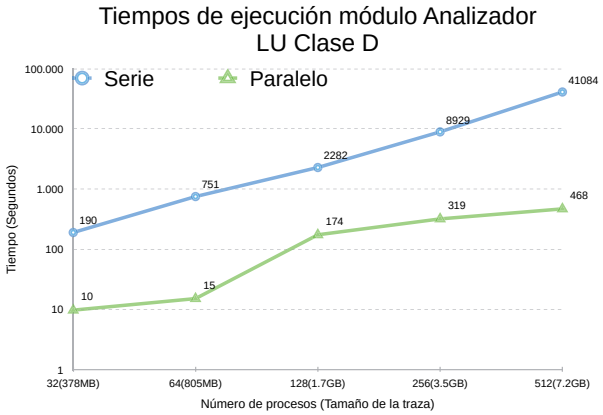
(a) Tiempo de análisis



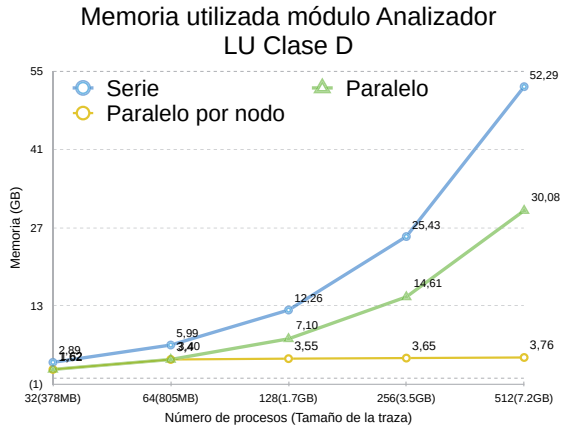
(b) Uso de la memoria

Figura A.15: Análisis de la traza usando Sweep.250.30

Con el objetivo de comparar el speedup de los módulos en las mismas condiciones, sin tener que acceder a memoria virtual, la figura A.17.a muestra el tiempo de ejecución para analizar la traza lógica obtenida de la aplicación SP, el tamaño de la cual cabe en



(a) Tiempo de análisis



(b) Uso de la memoria

Figura A.16: Análisis de la traza usando LU Clase C

memoria para todas las ejecuciones realizadas tanto para la versión serie como para la versión paralela, tal y como se muestra A.17.b, por lo que no realiza accesos a memoria virtual. Como se puede apreciar en la figura A.17.a, a medida que se incrementa el número de procesos de la aplicación, el módulo paralelo obtiene un mejor Speedup, llegando a obtener un Speedup de 6,3 para la ejecución de 121 procesos. Esto es debido a la paralelización del algoritmo de asignación de tiempos lógicos y a la paralelización del algoritmo de identificación de fases relevantes.

Lo mismo sucede para la traza lógica de la aplicación CG, la cual fue analizada utilizando el cluster NOVA de 48GB de memoria por nodo. Como se puede apreciar en

la figura A.18.b, la traza puede alocarse completamente en memoria principal tanto para el módulo serie como para el módulo paralelo, evitando accesos a la memoria virtual.

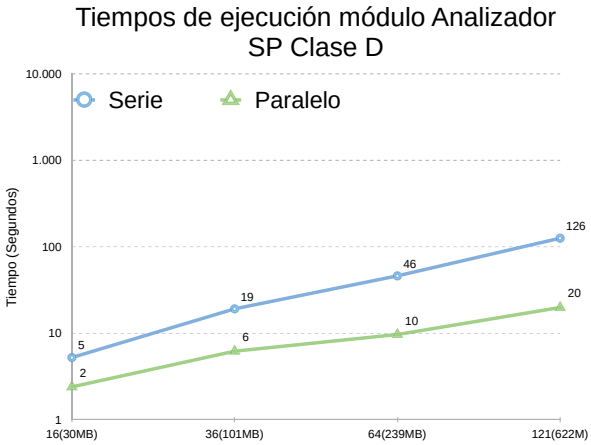
A diferencia de la traza lógica de la aplicación para SP, la traza lógica para la aplicación CG presenta un tamaño 8 veces mayor, llegando a un tamaño de 5,2 GB para la ejecución de la aplicación para 128 procesos. Como se puede comprobar en la figura A.18.a, a medida que mayor es la traza lógica de la aplicación, mayor Speedup presenta el módulo paralelo frente al módulo serie, llegando a obtener un Speedup de 87 para la ejecución con 128 procesos. Esto es debido tanto a la complejidad del método serie para relacionar los eventos a la hora de asignarles un tiempo lógico, como a la complejidad del algoritmo de búsqueda para identificar las fases de la aplicación, las cuales han sido reducidas en la implementación paralela, aprovechando mejor la paralización cuanto mayor es la traza lógica de la aplicación.

A.5. Identificación de patrones de la aplicación utilizando el número de instrucciones

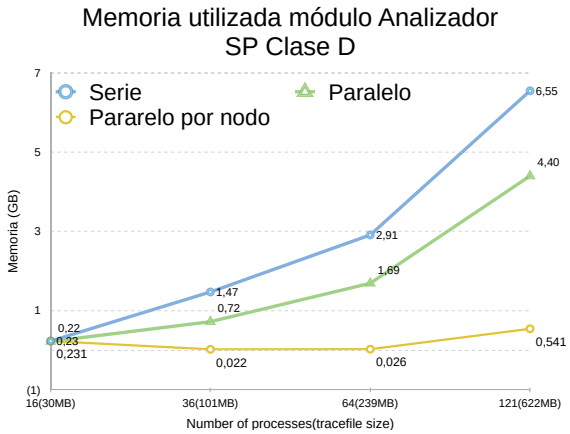
Como se comentó anteriormente, con el objetivo de identificar las fases de la aplicación, PAS2P utiliza un algoritmo de similitud.

Tanto de la versión serie como de la versión paralela del algoritmo, tal y como se muestra en la figura A.19, tienen impuesto como condición que una fase tiene que estar delimitada entre 2 eventos de comunicación.

Dependiendo de la aplicación, puede suceder que alguna fase comience realizando cómputo, en lugar de realizar un evento de comunicación, identificando erróneamente el algoritmo de PAS2P las fases de la aplicación. La figura A.20 muestra un ejemplo de esta situación. A partir de la secuencia de fases $AAABBB$, puesto que la fase B comienza con una secuencia de cómputo en lugar de un evento de comunicación, el algoritmo de PAS2P identifica las fases *A', B' y C'*, siendo la fase *B'* la unión de la fase A, más el segmento de cómputo de la fase B, hasta llegar a su evento de comunicación, mientras que la fase *C'* estaría comprendida por el evento de comunicación de la B, más sus segmentos de cómputo. Como se puede apreciar, debido a la forma de identificar las fases, el algoritmo de PAS2P identifica una nueva fase obtenida a partir de la unión de otras fases.



(a) Tiempo de análisis

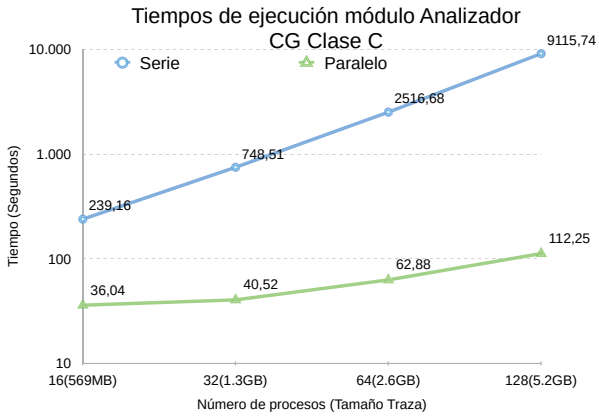


(b) Uso de la memoria

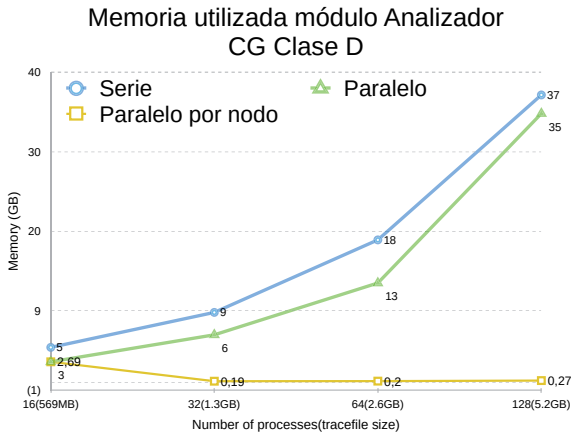
Figura A.17: Análisis de la traza usando SP Clase C

A la hora de predecir el rendimiento de la aplicación, puesto que las fases son ejecutadas en el sistema, no se hace necesario tener un corte tan preciso de las fases, pero sí a la hora de analizar la escalabilidad de la fase, ya que como se muestra en la figura A.21, el número total de instrucciones de la fase está comprendido por la suma de todos sus segmentos de cómputo.

A medida que la fase escala, manteniendo constante su carga de trabajo, el número total de instrucciones de la fase se mantiene constante, puesto que la fase realiza el mismo trabajo distribuido entre diferente número de procesos. Si el algoritmo no identifica las fases correctamente, el número de instrucciones de las fases no puede ser modelizado a



(a) Tiempo de análisis



(b) Uso de la memoria

Figura A.18: Análisis de la traza usando CG Clase C

medida que el número de procesos de la aplicación aumenta, debido a que la fase está compuesta por segmentos de cómputo de fases distintas, los cuales pueden tener un comportamiento distinto a medida que escala el número de procesos de la aplicación.

Con el objetivo de solucionar este problema, se propone segunda etapa en el algoritmo de identificación de fases, mediante la cual ajustar el número de instrucciones de la fases. Esta segunda etapa se ejecuta una vez se han obtenido las fases relevantes iniciales y se basa en detectar cuando una fase está compuesta por segmentos de código de dos fases distintas.

A continuación, se explica el algoritmo propuesto siguiendo el ejemplo de la figura A.20. A partir de la secuencia de fases $AAABBB$, el algoritmo de identificación de fases

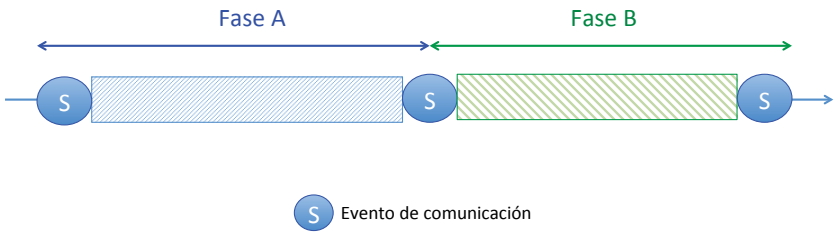


Figura A.19: Fases de la aplicación delimitadas entre dos eventos de comunicación

de PAS2P ha obtenido las fases relevantes *A', B' y C'*, siendo *B'* la unión de las fases *A* y el primer segmento de cómputo de *B*.

Obtenidas las fases relevantes, se ejecuta la segunda etapa del algoritmo, la cual consiste en recorrer la secuencia de fases obtenidas en la etapa anterior con el objetivo de ajustar el número de instrucciones de las fases. Esta segunda etapa consta de los siguiente pasos:

1. Se empieza en la primera fase relevante obtenida (fase actual) y se compara su estructura con la fase que le precede (fase anterior). El algoritmo compara el patrón de comunicación y el volumen de comunicación. Si estos dos parámetros son diferentes, las fases son diferentes y se salta al paso 5. En caso contrario se continua al siguiente paso del algoritmo.

   Para ilustrar este ejemplo tomaremos como fase actual B' (unión de fases A y B). Si comparamos el patrón de comunicación y el volumen de comunicación de la fase actual (B') con la fase anterior (A'), vemos que son iguales, puesto que el evento de comunicación a comparar es el de la fase A para las fases A' y B', por lo tanto pasamos al paso 2 del algoritmo.

2. Se compara el número de instrucciones de la fase actual (B') y la fase anterior (A'). Si el número de instrucciones de la fase actual (B') es mayor al número de instrucciones de la fase anterior (A'), se identifica la fase como una unión de dos fases. En caso contrario son dos fases diferentes.

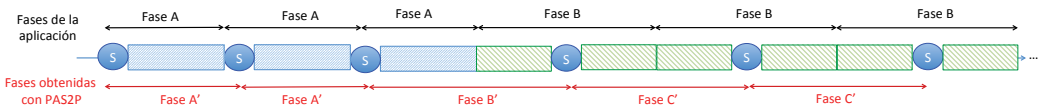


Figura A.20: Fases obtenidas mediante PAS2P

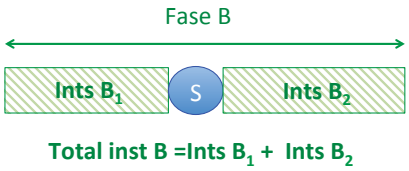


Figura A.21: Instrucciones totales de la fase

3. Con el objetivo de corregir el número de instrucciones de la fase actual (B'), se aplica la siguiente ecuación:

$$Inst_FaseAct = Inst_FaseAct - Inst_FaseAnt \tag{A.2}$$

   Quedando la fase actual únicamente con él número de instrucciones del primer segmento de B'.

4. Puesto que la fase actual (B') contiene la fase anterior (A'), la cual no ha sido identificada correctamente durante la primera etapa del algoritmo, el peso de la fase anterior (A') tiene que ser compensando añadiéndole el número de veces que no se identificó A', para ello el peso de la fase actual (B') se suma al peso de la fase anterior (A').

5. Una vez que la fase ha sido compensada se pasa a la siguiente fase, si no hay más fases se finaliza el algoritmo.

Una vez se han corregido las fases, seguimos teniendo las fases iniciales detectadas por PAS2P, para el ejemplo utilizando A', B' y C', pero con el número de instrucciones y peso corregido, permitiendo que las fases puedan ser analizadas y modeladas.

Apéndice B

Validación experimental de la modelización lógica de la aplicación

B.1. Introducción

Este apéndice presenta en detalle la evaluación experimental de la modelización lógica de la aplicación, para el conjunto de aplicaciones paralelas de paso de mensajes mostradas en la tabla resumen 5.6, en el capítulo 5: *Modelo lógico de aplicaciones paralelas para el análisis y predicción de la escalabilidad*

B.2. Modelo lógico de la aplicación BT

La tabla B.1 muestra el modelo lógico de las fases 3, 4, 5 y 6 de la aplicación BT. A partir del modelo lógico de las fases, se ha predicho su estructura para 1024 procesos.

La tabla B.2 muestra la estructura predicha de las fases para 1024 procesos y, la real obtenida mediante la ejecución de la firma de la aplicación para 1024 procesos. Como se muestra en la tabla, se ha predicho el patrón de comunicación correctamente para todas las fases. El volumen de comunicación se ha predicho con un error máximo aproximado del 7,1 % para la fase 3, mientras que el número de instrucciones se ha predicho con un error máximo aproximado del 4 % para la fase 4. El peso se ha predicho con un error del 0 % para todas las fases.

Tabla B.1: Modelo de la aplicación para BT (SLT)

Fase	Ecuación General de Comunicación (Dest.)	Ecuación General Volumen de comunicación	Ecuación General Patron de Cómputo (Num. Insts.)
Fase 3	1)$f(n) = 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	2)$f(n) = \sqrt(n) - 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	3)$f(n) = n - \sqrt(n)$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	4)$f(n) = n - \sqrt(n)$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	5)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	6)$f(n) = n - \sqrt(n) + 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	7)$f(n) = 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	8)$f(n) = \sqrt(n) - 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	9)$f(n) = n - \sqrt(n)$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	10)$f(n) = n - \sqrt(n)$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	11)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=662
	12)$f(n) = n - \sqrt(n) + 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=1386
	13)$f(n) = 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	14)$f(n) = \sqrt(n) - 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	15)$f(n) = n - \sqrt(n)$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	16)$f(n) = n - \sqrt(n)$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	17)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	18)$f(n) = n - \sqrt(n) + 1$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	19)$f(n) = 0$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	20)$f(n) = 0$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	21)$f(n) = 0$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	22)$f(n) = 0$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	23)$f(n) = 0$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=0
	24)$f(n) = 0$	$y(n) = 8E + 06n^{(-0,421)}$	i(n)=((687.226.272.251.904/n)/w(n))
Ecuación General del peso para la fase 3 w(n)= (251*(sqrt(n)-1))+502			
Fase 4	1)$f(n) = \sqrt(n)$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=699
	2)$f(n) = n - \sqrt(n)$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=577
	3)$f(n) = n - \sqrt(n)$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=((30.875.907.250.176/n)/w(n))
Ecuación General del peso para la fase 4 w(n)=(251*(sqrt(n)-1))+502			
Fase 5	1)$f(n) = n - \sqrt(n)$	$y(n) = 7E + 06n^{(-0,997)}$	i(n)=699
	2)$f(n) = \sqrt(n)$	$y(n) = 7E + 06n^{(-0,997)}$	i(n)=577
	3)$f(n) = \sqrt(n)$	$y(n) = 7E + 06n^{(-0,997)}$	i(n)=((5.396.299.375.924/n)/w(n))
Ecuación General del peso para la fase 5 w(n)=(251*(sqrt(n)-1))+502			
Fase 6	1)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=699
	2)$f(n) = n - \sqrt(n) + 1$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=577
	3)$f(n) = n - \sqrt(n) + 1$	$y(n) = 4E + 07n^{(-0,997)}$	i(n)=((34.826.618.059.384/n)/w(n))
Ecuación General del peso para la fase 6 w(n)=(251*(sqrt(n)-1))+502			

Tabla B.2: Parámetros predichos de las fases de la aplicación BT (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento para n = 1024

Fase Real					Fase Predicha					Error de Predicción	
Fase - Peso	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Fase - Peso**	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Volumen Com.**	**Num. Ints.**
	1)Irecv	0-1	403520	662		Irecv	0-1	432268	662	7,1 %	0 %
	2)Irecv	0-31	403520	662		Irecv	0-31	432268	662	7,1 %	0 %
	3)Irecv	0-32	403520	662		Irecv	0-32	432268	662	7,1 %	0 %
	4)Irecv	0-992	403520	662		Irecv	0-992	432268	662	7,1 %	0 %
	5)Irecv	0-63	403520	662		Irecv	0-63	432268	662	7,1 %	0 %
	6)Irecv	0-993	403520	662		Irecv	0-993	432268	662	7,1 %	0 %
	7)Isend	0-1	403520	662		Isend	0-1	432268	662	7,1 %	0 %
	8)Isend	0-31	403520	662		Isend	0-31	432268	662	7,1 %	0 %
	9)Isend	0-32	403520	662		Isend	0-32	432268	662	7,1 %	0 %
	10)Isend	0-992	403520	662		Isend	0-992	432268	662	7,1 %	0 %
	11)Isend	0-63	403520	662		Isend	0-63	432268	662	7,1 %	0 %
Fase 3-	12)Isend	0-993	403520	1386	Fase 3-	Isend	0-993	432268	1386	7,1 %	0 %
8282	13)Waitall	0-1	403520	0	8283	Isend	0-1	432268	0	7,1 %	0 %
	14)Waitall	0-31	403520	0		Waitall	0-31	432268	0	7,1 %	0 %
	15)Waitall	0-32	403520	0		Waitall	0-32	432268	0	7,1 %	0 %
	16)Waitall	0-992	403520	0		Waitall	0-992	432268	0	7,1 %	0 %
	17)Waitall	0-63	403520	0		Waitall	0-63	432268	0	7,1 %	0 %
	18)Waitall	0-993	403520	0		Waitall	0-993	432268	0	7,1 %	0 %
	19)Waitall	0-0	403520	0		Waitall	0-0	432268	0	7,1 %	0 %
	20)Waitall	0-0	403520	0		Waitall	0-0	432268	0	7,1 %	0 %
	21)Waitall	0-0	403520	0		Waitall	0-0	432268	0	7,1 %	0 %
	22)Waitall	0-0	403520	0		Waitall	0-0	432268	0	7,1 %	0 %
	23)Waitall	0-0	403520	0		Waitall	0-0	432268	0	7,1 %	0 %
	24)Waitall	0-0	403520	80072815		Waitall	0-0	432268	81023712	7,1 %	1,18 %
	1)Isend	0-32	40560	699		Isend	0-32	39883	699	1,69 %	0 %
Fase 4-	2)Irecv	0 -992	40560	577	Fase 4-	Irecv	0-992	39883	577	1,69 %	0 %
8283	3)Wait	0-992	40560	3498778	8283	Wait	0-992	39883	3640257	1,69 %	4,04 %
	1)Isend	0-992	6760	699		Isend	0-993	6979	699	3,13 %	0 %
Fase 5-	2)Irecv	0-32	6760	577	Fase 5 -	Irecv	0-63	6979	577	3,13 %	0 %
8283	3)Wait	0-32	6760	630208	8283	Wait	0-63	6979	636221	3,13 %	0,95 %
	1)Isend	0-63	40560	699		Isend	0-63	39883	699	1,69 %	0 %
Fase 6-	2)Irecv	0-993	40560	577	Fase 6 -	Irecv	0-993	39883	577	1,69 %	0 %
8283	3)Wait	0-993	40560	4103557	8283	Wait	0-993	39883	4106044	1,69 %	0,1 %

B.3. Modelo lógico de la aplicación CG

La tabla B.3 muestra el modelo lógico de las fases 2 y 3 de la aplicación CG. A partir del modelo lógico de las fases, se ha predicho su estructura para 4096 procesos.

La tabla B.4 muestra la estructura predicha de las fases para 4096 procesos y, la real obtenida mediante la ejecución de la firma de la aplicación para 4096 procesos. Como se muestra en la tabla, se ha predicho el patrón de comunicación correctamente para todas las fases. El volumen de comunicación se ha predicho con un error máximo aproximado del 4 % para la fase 3, mientras que el número de instrucciones se ha predicho con un error máximo aproximado del 5.2 % para la fase 3. El peso se ha predicho con un error del 0 % para todas las fases.

B. VALIDACIÓN EXPERIMENTAL DE LA MODELIZACIÓN LÓGICA DE LA APLICACIÓN

Tabla B.3: Modelo de la aplicación para CG (SLT)

Fase	Ecuación General de Comunicación (Dest.)	Ecuación General Volumen de comunicación	Ecuación General Patron de Cómputo (Num. Insts.)
Fase 2	1)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=654
	2)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=483
	3)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=663
	4)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=654
	5)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=483
	6)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=663
	7)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=654
	8)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=483
	9)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=663
	10)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=654
	11)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=483
	12)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=663
	13)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=654
	14)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=483
	15)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=663
	16)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=654
	17)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=483
	18)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8$	i(n)=((28.460.861.440.000/n)/w(n))
Ecuación General del peso para la fase 2 w(n)= 2500			
Fase 3	1)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=660
	2)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=483
	3)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=((2.162.913.280.000/n)/w(n))
	4)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=660
	5)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=483
	6)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=((2.162.913.280.000/n)/w(n))
	7)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=660
	8)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=483
	9)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=((2.162.913.280.000/n)/w(n))
	10)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=660
	11)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=483
	12)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=((2.162.913.280.000/n)/w(n))
	13)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=660
	14)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=483
	15)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=((2.162.913.280.000/n)/w(n))
	16)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=660
	17)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=483
	18)$[\#Comm.c(n) = log_2(n)/2, f(y)_{y=\#Com..,1c(n)} = 2^{(y-1)}]$	$y(n) = 8E + 08n^{(-1)}$	i(n)=((2.162.913.280.000/n)/w(n))
	19)$f(n) = 0$	$y(n) = 8E + 08n^{(-1)}$	i(n)=660
	20)$f(n) = 0$	$y(n) = 8E + 08n^{(-1)}$	i(n)=483
	21)$f(n) = 0$	$y(n) = 8E + 08n^{(-1)}$	i(n)=((3.290.490.880.000/n)/w(n))
Ecuación General del peso para la fase 3 w(n)= 2500			

Tabla B.4: Parámetros predichos de las fases de la aplicación CG (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento n = 4096.

Fase Real					Fase Predicha					Error de Predicción	
Fase - Peso	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Fase - Peso**	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Volumen Com.**	**Num. Ints.**
	1)Isend	0-32	8	654		Isend	0-32	8	654	0 %	0 %
	2)Irecv	0-32	8	483		Irecv	0-32	8	483	0 %	0 %
	3)Wait	0-32	8	663		Wait	0-32	8	663	0 %	0 %
	4)Isend	0-16	8	654		Isend	0-16	8	654	0 %	0 %
	5)Irecv	0-16	8	483		Irecv	0-16	8	483	0 %	0 %
	6)Wait	0-16	8	663		Wait	0-16	8	663	0 %	0 %
	7)Isend	0-8	8	654		Isend	0-8	8	654	0 %	0 %
Fase 2	8)Irecv	0-8	8	483		Irecv	0-8	8	483	0 %	0 %
	9)Wait	0-8	8	663		Wait	0-8	8	663	0 %	0 %
	10)Isend	0-4	8	654		Isend	0-4	8	654	0 %	0 %
	11)Irecv	0-4	8	483		Irecv	0-4	8	483	0 %	0 %
	12)Wait	0-4	8	663		Wait	0-4	8	663	0 %	0 %
	13)Isend	0-2	8	654		Isend	0-2	8	654	0 %	0 %
	14)Irecv	0-2	8	483		Irecv	0-2	8	483	0 %	0 %
	15)Wait	0-2	8	663		Wait	0-2	8	663	0 %	0 %
	16)Isend	0-1	8	654		Isend	0-1	8	654	0 %	0 %
	17)Irecv	0-1	8	483		Irecv	0-1	8	483	0 %	0 %
	18)Wait	0-1	8	2698623		Wait	0-1	8	2779381	0 %	3 %
	1)Isend	0-1	187504	660		Isend	0-1	195312	660	4 %	0 %
	2)Irecv	0-1	187504	483		Irecv	0-1	195312	483	4 %	0 %
	3)Wait	0-1	187504	211613		Wait	0-1	195312	211222	4 %	0,18 %
	4)Isend	0-2	187504	660		Isend	0-2	195312	660	4 %	0 %
	5)Irecv	0-2	187504	483		Irecv	0-2	195312	483	4 %	0 %
	6)Wait	0-2	187504	211613		Wait	0-2	195312	211222	4 %	0,18 %
	7)Isend	0-4	187504	660		Isend	0-4	195312	660	4 %	0 %
Fase 3	8)Irecv	0-4	187504	483		Irecv	0-4	195312	483	4 %	0 %
	9)Wait	0-4	187504	211613		Wait	0-4	195312	211222	4 %	0,18 %
	10)Isend	0-8	187504	660		Isend	0-8	195312	660	4 %	0 %
	11)Irecv	0-8	187504	483		Irecv	0-8	195312	483	4 %	0 %
	12)Wait	0-8	187504	211613		Wait	0-8	195312	211222	4 %	0,18 %
	13)Isend	0-16	187504	660		Isend	0-16	195312	660	4 %	0 %
	14)Irecv	0-16	187504	483		Irecv	0-16	195312	483	4 %	0 %
	15)Wait	0-16	187504	211613		Wait	0-16	195312	211222	4 %	0,18 %
	16)Isend	0-32	187504	660		Isend	0-32	195312	660	4 %	0 %
	17)Irecv	0-32	187504	483		Irecv	0-32	195312	483	4 %	0 %
	18)Wait	0-32	187504	211613		Wait	0-32	195312	211222	4 %	0,18 %
	19)Isend	0-0	187504	660		Isend	0-0	195312	660	4 %	0 %
	20)Irecv	0-0	187504	483		Irecv	0-0	195312	483	4 %	0 %
	21)Wait	0-0	187504	305384		Wait	0-0	195312	321337	4 %	5,22 %

B.4. Modelo lógico de la aplicación Sweep3D

La tabla B.5 muestra el modelo lógico de las fases 2 y 3 de la aplicación Sweep3D. A partir del modelo lógico de las fases, se ha predicho su estructura para 4096 procesos.

La tabla B.6 muestra la estructura predicha de las fases para 4096 procesos y, la real obtenida mediante la ejecución de la firma de la aplicación para 4096 procesos. Como se muestra en la tabla, se ha predicho el patrón de comunicación correctamente para todas las fases. El volumen de comunicación se ha predicho con un error máximo aproximado del 3 % para ambas fases, mientras que el número de instrucciones se ha predicho con un error máximo aproximado del 8.5 % para la fase 3. El peso se ha predicho con un error del 0 % para todas las fases.

Tabla B.5: Modelo de la aplicación para Sweep3D (SLT)

Fase	**Ecuación General de Comunicación (Dest.)**	**Ecuación General Volumen de comunicación**	**Ecuación General Patron de Cómputo (Num. Insts.)**
Fase 3	1)$f(n) = 1$ 2)$f(n) = \sqrt{(}n) - 1$	$y(n) = 6E + 07n^{(-1)}$ $y(n) = 6E + 07n^{(-1)}$	i(n)=582 i(n)=67264233472
Ecuación General del peso para la fase 3 w(n)=1			
Fase 4	1)$f(n) = \sqrt{(}n) - 1$ 2)$f(n) = 1$	$y(n) = 6E + 07n^{(-1)}$ $y(n) = 6E + 07n^{(-1)}$	i(n)=65506254848 i(n)=582
Ecuación General del peso para la fase 4 w(n)=1			

Tabla B.6: Parámetros predichos de las fases de la aplicación Sweep3D (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento para n = 4096.

Fase Real					**Fase Predicha**					**Error de Predicción**	
Fase - Peso	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Fase - Peso**	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Volumen Com.**	**Num. Ints.**
Fase 3-1	1)Recv 2)Recv	0-1 0-64	15120 15120	582 15121745	Fase 3 - 1	Send Recv	0-1 0-64	14648 14648	582 16421932	3.2 % 3.2 %	0 % 8.5 %
Fase 4-1	1)Send 2)Send	0-64 0-1	15120 15120	15118041 582	Fase 4-1	Send Recv	0-64 0 -1	14648 14648	15992738 582	3.2 % 3.2 %	5,4 % 0 %

B.5. Modelo lógico de la aplicación SP

La tabla B.7 muestra el modelo lógico de las fases 1, 2, 3, 4, 5 y 6 de la aplicación SP. A partir del modelo lógico de las fases, se ha predicho su estructura para 1024 procesos.

La tabla B.8 muestra la estructura predicha de las fases para 1024 procesos y, la real obtenida mediante la ejecución de la firma de la aplicación para 1024 procesos. Como se muestra en la tabla, se ha predicho el patrón de comunicación correctamente para todas las fases. El volumen de comunicación se ha predicho con un error máximo aproximado del 3.9 % para las fases 2 y 5, mientras que el número de instrucciones se ha predicho con un error máximo aproximado del 3.1 % para las fases 2 y 5. El peso se ha predicho con un error del 0 % para todas las fases.

Tabla B.7: Modelo de la aplicación para SP (SLT)

Fase	Ecuación General de Comunicación (Dest.)	Ecuación General Volumen de comunicación	Ecuación General Patron de Cómputo (Num. Insts.)
Fase 1	1)$f(n) = 1)$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=605
	2)$f(n) = \sqrt(n) - 1$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=((4.452.921.090.048/n)/w(n))
	3)$f(n) = \sqrt(n) - 1$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=((9.224.744.054.784/n)/w(n))
Ecuación General del peso para la fase 1 w(n)=(500*(sqrt(n)-1))			
Fase 2	1)$f(n) = n - \sqrt(n) + 1$	$y(n) = 6E + 07n^{(-1,206)}$	i(n)=605
	2)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 6E + 07n^{(-1,206)}$	i(n)= ((5.114.119.246.848 /n)/w(n))
	3)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 6E + 07n^{(-1,206)}$	i(n)=((7.667.703.902.208/n)/w(n))
Ecuación General del peso para la fase 2 w(n)=(500*(sqrt(n)-1))			
Fase 3	1)$f(n) = 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	2)$f(n) = \sqrt(n) - 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	3)$f(n) = n - \sqrt(n)$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	4)$f(n) = n - \sqrt(n)$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	5)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	6)$f(n) = n - \sqrt(n) + 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	7)$f(n) = 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	8)$f(n) = \sqrt(n) - 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	9)$f(n) = n - \sqrt(n)$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	10)$f(n) = n - \sqrt(n)$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	11)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=572
	12)$f(n) = n - \sqrt(n) + 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=5382
	13)$f(n) = 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	14)$f(n) = \sqrt(n) - 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	15)$f(n) = n - \sqrt(n)$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	16)$f(n) = n - \sqrt(n)$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	17)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	18)$f(n) = n - \sqrt(n) + 1$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	19)$f(n) = 0$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	20)$f(n) = 0$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	21)$f(n) = 0$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	22)$f(n) = 0$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	23)$f(n) = 0$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=0
	24)$f(n) = 0$	$y(n) = 9E + 06n^{(-0,438)}$	i(n)=((1.354.879.010.955.264/n)/w(n))
Ecuación General del peso para la fase 3 w(n)=(500*(sqrt(n)-1))			
Fase 4	1)$f(n) = \sqrt(n)$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=605
	2)$f(n) = n - \sqrt(n)$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=((4.452.921.090.048/n)/w(n))
	3)$f(n) = n - \sqrt(n)$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=((9.224.744.054.784/n)/w(n))
Ecuación General del peso para la fase 4 w(n)=(500*(sqrt(n)-1))			
Fase 5	1)$f(n) = n - \sqrt(n)$	$y(n) = 6E + 07n^{(-1,206)}$	i(n)=605
	2)$f(n) = \sqrt(n)$	$y(n) = 6E + 07n^{(-1,206)}$	i(n)= ((5.114.119.246.848 /n)/w(n))
	3)$f(n) = \sqrt(n)$	$y(n) = 6E + 07n^{(-1,206)}$	i(n)=((7.667.703.902.208/n)/w(n))
Ecuación General del peso para la fase 5 w(n)=(500*(sqrt(n)-1))			
Fase 6	1)$f(n) = 2 * \sqrt(n) - 1$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=605
	2)$f(n) = n - \sqrt(n) + 1$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=((4.452.921.090.048/n)/w(n))
	3)$f(n) = n - \sqrt(n) + 1$	$y(n) = 6E + 06n^{(-0,788)}$	i(n)=((34.826.618.059.384/n)/w(n))
Ecuación General del peso para la fase 6 w(n)=(500*(sqrt(n)-1))			

Tabla B.8: Parámetros predichos de las fases de la aplicación SP (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento para n = 1024.

Fase Real					Fase Predicha					Error de Predicción	
Fase - Peso	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Fase - Peso**	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Volumen Com.**	**Num. Ints.**
	1)Isend	0-1	25344	605		Isend	0-32	25470	605	0,5 %	0 %
Fase 1-	2)Irecv	0 -31	25344	313780	Fase 1-	Irecv	0-992	25470	321567	0,5 %	2,42 %
15531	3)Wait	0-31	25344	571916	15531	Wait	0-992	25470	580036	0,5 %	1,41 %
	1)Isend	0-993	13520	605		Isend	0-993	14051	605	3,92 %	0 %
Fase 2-	2)Irecv	0-63	13520	270485	Fase 2 -	Irecv	0-63	14051	279992	3,92 %	3,14 %
15531	3)Wait	0-63	13520	474382	15531	Wait	0-63	14051	482132	3,92 %	1,63 %
	1)Irecv	0-1	402480	572		Irecv	0-1	39883	572	1.69 %	0 %
	2)Irecv	0-31	402480	572		Irecv	0-31	39883	572	1.69 %	0 %
	3)Irecv	0-32	402480	572		Irecv	0-32	39883	572	1.69 %	0 %
	4)Irecv	0-992	402480	572		Irecv	0-992	39883	572	1.69 %	0 %
	5)Irecv	0-63	402480	572		Irecv	0-63	39883	572	1.69 %	0 %
	6)Irecv	0-993	402480	572		Irecv	0-993	39883	572	1.69 %	0 %
	7)Isend	0-1	402480	572		Isend	0-1	39883	572	1.69 %	0 %
	8)Isend	0-31	402480	572		Isend	0-31	39883	572	1.69 %	0 %
	9)Isend	0-32	402480	572		Isend	0-32	39883	572	1.69 %	0 %
	10)Isend	0-992	402480	572		Isend	0-992	39883	572	1.69 %	0 %
	11)Isend	0-63	402480	572		Isend	0-63	39883	572	1.69 %	0 %
Fase 3-	12)Isend	0-993	402480	5382	Fase 3-	Isend	0-993	39883	5382	1.69 %	0 %
15531	13)Waitall	0-1	402480	0	15531	Isend	0-1	39883	0	1.69 %	0 %
	14)Waitall	0-31	402480	0		Waitall	0-31	39883	0	1.69 %	0 %
	15)Waitall	0-32	402480	0		Waitall	0-32	39883	0	1.69 %	0 %
	16)Waitall	0-992	402480	0		Waitall	0-992	39883	0	1.69 %	0 %
	17)Waitall	0-63	402480	0		Waitall	0-63	39883	0	1.69 %	0 %
	18)Waitall	0-993	402480	0		Waitall	0-993	39883	0	1.69 %	0 %
	19)Waitall	0-0	402480	0		Waitall	0-0	39883	0	1.69 %	0 %
	20)Waitall	0-0	402480	0		Waitall	0-0	39883	0	1.69 %	0 %
	21)Waitall	0-0	402480	0		Waitall	0-0	39883	0	1.69 %	0 %
	22)Waitall	0-0	402480	0		Waitall	0-0	39883	0	1.69 %	0 %
	23)Waitall	0-0	402480	0		Waitall	0-0	39883	0	1.69 %	0 %
	24)Waitall	0-0	402480	84685163		Waitall	0-0	85192456	0	1.69 %	0,5 %
	1)Isend	0-32	25344	605		Isend	0-32	25470	605	0,5 %	0 %
Fase 4-	2)Irecv	0 -992	25344	313780	Fase 4-	Irecv	0-992	25470	321567	0,5 %	2,42 %
15531	3)Wait	0-992	25344	571916	15531	Wait	0-992	25470	580036	0,5 %	1,41 %
	1)Isend	0-992	13520	605		Isend	0-993	14051	605	3,92 %	0 %
Fase 5-	2)Irecv	0-32	13520	270485	Fase 5 -	Irecv	0-63	14051	279992	3,92 %	3,14 %
15531	3)Wait	0-32	13520	474382	15531	Wait	0-63	14051	482132	3,92 %	1,63 %
	1)Isend	0-63	25344	605		Isend	0-63	25470	605	0,5 %	0 %
Fase 6-	2)Irecv	0-993	25344	313780	Fase 6 -	Irecv	0-993	25470	321567	0,5 %	2,42 %
15531	3)Wait	0-993	25344	571916	15531	Wait	0-993	25470	580036	0,5 %	1,41 %

B.6. Modelo lógico de la aplicación LU

La tabla B.5 muestra el modelo lógico de las fases 1 y 2 de la aplicación LU. A partir del modelo lógico de las fases, se ha predicho su estructura para 4096 procesos.

La tabla B.6 muestra la estructura predicha de las fases para 4096 procesos y, la real obtenida mediante la ejecución de la firma de la aplicación para 4096 procesos. Como se muestra en la tabla, se ha predicho el patrón de comunicación correctamente para todas las fases. El volumen de comunicación se ha predicho con un error del 0 % para todas las fases, mientras que el número de instrucciones se ha predicho con un error máximo aproximado del 7.2 % para la fase 2. El peso se ha predicho con un error del 0 % para todas las fases.

Tabla B.9: Modelo de la aplicación para LU (SLT)

Fase	**Ecuación General de Comunicación (Dest.)**	**Ecuación General Volumen de comunicación**	**Ecuación General Patron de Cómputo (Num. Insts.)**
Fase 1	1)$f(n) = 1$ 2)$f(n) = 2 * \sqrt{(n)}$	$y(n) = 2000$ $y(n) = 4040$	i(n)=((754.989.465.600/n)/w(n)) i(n)=760
Ecuación General del peso para la fase 1 $w(n) = (300 * (log_2(n) - 3))$			
Fase 2	1)$f(n) = 2 * \sqrt{(n)}$ 2)$f(n) = 1$	$y(n) = 2000$ $y(n) = 4040$	i(n)=760 i(n)=((22.111.521.177.600/n)/w(n))
Ecuación General del peso para la fase 2 $w(n) = (300 * (log_2(n) - 3))$			

Tabla B.10: Parámetros predichos de las fases de la aplicación LU (Traza Lógica Escalada (LT4NC)), aplicando las ecuaciones generales de comportamiento para n = 4096 procesos.

Fase Real					**Fase Predicha**					**Error de Predicción**	
Fase - Peso	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Fase - Peso**	**Prim. MPI**	**Origen-Dest.**	**Volumen Com.**	**Num. Inst.**	**Volumen Com.**	**Num. Ints.**
Fase 1-2700	1)Send 2)Send	0-1 0-64	2000 4040	67688 760	Fase 1-2700	Send Recv	0-1 0-64	2000 4040	68268 760	0 % 0 %	0,8 % 0 %
Fase 2-2700	1)Recv 2)Recv	0-64 0-1	2000 4040	760 1853908	Fase 2-2700	Send Recv	0-64 0 -1	2000 4040	760 1999378	0 % 0 %	0 % 7,2 %

Bibliografía

[1] N. Attig, P. Gibbon, and Th., "Trends in supercomputing: The european path to exascale," vol. 182, pp. 2041 – 2046, 2011. 1

[2] R. Nishtala, P. Hargrove, D. Bonachea, and K. Yelick, "Scaling communication-intensive applications on bluegene/p using one-sided communication and overlap," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–12, May 2009. 1

[3] S. Marras, J. Kelly, M. Moragues, A. Mueller, M. Kopera, M. Vazquez, F. Giraldo, G. Houzeaux, and O. Jorba, "A review of element-based galerkin methods for numerical weather prediction. finite elements, spectral elements, and discontinuous galerkin," in *Archives of Computational Methods in Engineering*, 2015 (Accepted). 2

[4] M. Peng, J. R. A. Schmalz, A. Zhang, *et al.*, "Towards the development of the national ocean service san francisco bay operational forecast system," *Journal of Marine Science and Engineering*, vol. 2, no. 1, pp. 247–286, 2014. 2

[5] B. Barney *et al.*, "Introduction to parallel computing," *Lawrence Livermore National Laboratory*, vol. 6, no. 13, pp. 1–10, 2010. 2

[6] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature for performance analysis and prediction," in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, pp. 2009–2019, 2015. 4, 39, 165

[7] J. C. Phillips, R. Braun, W. Wang, J. C. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. V. Kalé, and K. Schulten, "Scalable molecular dynamics with

NAMD," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005. 4

[8] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "A tool for selecting the right target machine for parallel scientific applications," in *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013*, pp. 1824–1833, 2013. 5, 40, 51, 165, 166

[9] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "Analysis of scalability: A parallel application model approach," in *2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22-26, 2014*, pp. 294–295, 2014. 7

[10] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "Modeling parallel applications for scalability analysis: An approach to predict the communication pattern," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2015*, 2015 (Pending of publication). 8

[11] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "Scalability of parallel applications: An approach to predict the computational behavior," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2015*, 2015 (Pending of publication). 8

[12] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "Synthetic signature program for performance scalability," in *Proceedings of the International Conference on Parallel Processing and Applied Mathematics, PPAM 2015*, 2015 (Pending of publication). 8

[13] S. Paki, *The design and analysis of parallel algorithm.s Prentice Hall Inc.* January 1989. 11

[14] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, *et al.*, "Openmpi: Goals, concept, and design of a next generation mpi implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 97–104, 2004. 14, 16

[15] G. Godza and V. Cristea, "Comparative study of cow and smp computer configurations," in *Parallel Computing in Electrical Engineering, 2002. PARELEC '02. Proceedings. International Conference on*, pp. 205–210, 2002. 14

[16] M. Ben-Ari, *Principles of Concurrent Programming. Prentice-Hall.* 1990. 16

[17] F. Wolf and B. Mohr, "Automatic performance analysis of hybrid mpi/openmp applications," *J. Syst. Archit.*, vol. 49, pp. 421–439, Nov. 2003. 16

[18] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov, "Parallel computing experiences with cuda," *Micro, IEEE*, vol. 28, pp. 13–27, July 2008. 16

[19] S.-L. Chu and C.-C. Hsiao, "Opencl: Make ubiquitous supercomputing possible," in *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*, pp. 556–561, Sept 2010. 16

[20] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to parallel computing: design and analysis of algorithms.* Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1994. 16

[21] V. Plagianakos, N. Nousis, and M. Vrahatis, "Locating and computing in parallel all the simple roots of special functions using {PVM}," *Journal of Computational and Applied Mathematics*, vol. 133, no. 12, pp. 545 – 554, 2001. 16

[22] R. Silva, L. Buyya, *Parallel Programming Paradigms, High Performance Cluster Computing: Programming and Applications, Vol. 2.* NJ, USA: Prentice Hall, 1999. 17

[23] Green500, "http://www.green500.org/," 22

[24] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pp. 483–485, 1967. 23

[25] J. Gustafson and Q. Snell, "Hint: A new way to measure computer performance," *Hawaii International Conference on System Sciences*, vol. 0, p. 392, 1995. 23

[26] M. D. Hill, "What is scalability?," *SIGARCH Comput. Archit. News*, vol. 18, no. 4, pp. 18–21, 1990. 24

[27] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications," *Int. J. High Perform. Comput. Appl.*, vol. 14, pp. 330–346, Nov. 2000. 24

[28] F. da Silva and H. Senger, "Scalability analysis of embarassingly parallel applications on large clusters," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1 –8, april 2010. 24

[29] B. J. Barnes, J. Garren, D. K. Lowenthal, J. Reeves, B. R. de Supinski, M. Schulz, and B. Rountree, "Using focused regression for accurate time-constrained scaling of scientific applications," in *IPDPS*, pp. 1–12, 2010. 29

[30] B. C. Lee and D. M. Brooks, "Methods of inference and learning for performance modeling of parallel applications," in *in Proc. of the International Symposium on Principles and Practices of Parallel Programming*, pp. 249–258, 2007. 29

[31] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *Proceedings of the 11th International Euro-Par Conference on Parallel Processing*, Euro-Par'05, pp. 196–205, 2005. 29

[32] D. R. Martínez, J. C. Cabaleiro, T. F. Pena, F. F. Rivera, and V. B. Pérez, "Accurate analytical performance model of communications in MPI applications," in *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, pp. 1–8, 2009. 29

[33] D. R. Martínez, J. C. Cabaleiro, T. F. Pena, F. F. Rivera, and V. B. Pérez, "Performance modeling of MPI applications using model selection techniques," in *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP 2010, Pisa, Italy, February 17-19, 2010*, pp. 95–102, 2010. 29

[34] G. Zheng, G. Kakulapati, and L. Kale, "Bigsim: a parallel simulator for performance prediction of extremely large parallel machines," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pp. 78–88, April 2004. 30

[35] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2899–2917, 2014. 31

[36] T. Hoefler, T. Schneider, and A. Lumsdaine, "Loggopsim: Simulating large-scale applications in the loggops model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pp. 597–604, 2010. 31

[37] F. Ino, N. Fujimoto, and K. Hagihara, "Loggps: A parallel computational model for synchronization analysis," in *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, PPoPP '01, pp. 133–142, 2001. 32

[38] E. Grobelny, D. Bueno, I. Troxel, A. D. George, and J. S. Vetter, "Fase: A framework for scalable performance prediction of hpc systems and applications," *Simulation*, vol. 83, no. 10, pp. 721–745, 2007. 32

[39] P. De and V. Mann, "jitsim: A simulator for predicting scalability of parallel applications in presence of os jitter," in *Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part I*, EuroPar'10, pp. 117–130, 2010. 32

[40] E. Vicente and R. Matias, "Exploratory study on the linux os jitter," in *Computing System Engineering (SBESC), 2012 Brazilian Symposium on*, pp. 19–24, Nov 2012. 32

[41] J. Dujmović, "Automatic generation of benchmark and test workloads," in *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, WOSP/SIPEW '10, pp. 263–274, 2010. 33

[42] A. M. Joshi, L. Eeckhout, and L. K. John, "The return of synthetic benchmarks," 2010. 33

[43] J. Logan, S. Klasky, H. Abbasi, Q. Liu, G. Ostrouchov, M. Parashar, N. Podhorszki, Y. Tian, and M. Wolf, "Understanding I/O performance using I/O skeletal applications," in *Euro-Par 2012 Parallel Processing - 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings*, pp. 77–88, 2012. 33

[44] M. Geimer, F. Wolf, B. J. N. Wylie, D. Becker, D. Böhme, W. Frings, M.-A. Hermanns, B. Mohr, and Z. Szebenyi, "Recent developments in the Scalasca toolset," in *Tools for High Performance Computing 2009, Proc. of the 3rd Parallel Tools Workshop, Dresden, Germany, September 2009* (M. S. Müller, M. M. Resch, W. E. Nagel, and A. Schulz, eds.), ch. 4, pp. 39–51, 2010. 34

[45] W. Zhang, A. M. Cheng, and J. Subhlok, "Dwarfcode: A performance prediction tool for parallel applications," in *IEEE Transactions on Computers. (TC)*, 2015 (Acepted). 34

[46] P. Ratn, F. Mueller, B. R. de Supinski, and M. Schulz, "Preserving time in large-scale communication traces," in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ICS '08, pp. 46–55, 2008. 34

[47] X. Wu, K. Vijayakumar, F. Mueller, X. Ma, and P. Roth, "Probabilistic communication and i/o tracing with deterministic replay at scale," in *Parallel Processing (ICPP), 2011 International Conference on*, pp. 196–205, 2011. 34

[48] X. Wu, F. Mueller, and S. Pakin, "Automatic generation of executable communication specifications from parallel applications," in *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May 2011 - Workshop Proceedings*, pp. 2089–2092, 2011. 34

[49] X. Wu, V. Deshpande, and F. Mueller, "Scalabenchgen: Auto-generation of communication benchmarks traces," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 1250–1260, 2012. 35

[50] J. Zhai, T. Sheng, J. He, W. Chen, and W. Zheng, "Fact: fast communication trace collection for parallel applications through program slicing," in *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pp. 1–12, 2009. 35

[51] Q. Xu and J. Subhlok, "Construction and elevation of coordinated performance skeleton," in *International Conference on High Performance Computing*, pp. 73–86, 2008. 35

[52] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pp. 45:1–45:12, 2013. 36

[53] S. Sharkawi, D. DeSota, R. Panda, S. Stevens, V. E. Taylor, and X. Wu, "SWAPP: A framework for performance projections of HPC applications using benchmarks," in *IPDPS*, pp. 1723–1731, 2012. 36

[54] Z. Jidong, C. Wenguang, and W. Zheng, "Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node," pp. 305–314, 2010. 37

[55] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Supercomputing '02, pp. 1–17, 2002. 37

[56] D. A. Weikle, S. A. McKee, and W. A. Wulf, "Caches as filters: A new approach to cache analysis," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1998. Proceedings. Sixth International Symposium on*, pp. 2–12, IEEE, 1998. 37

[57] L. D. Rose, L. D. Rose, Y. Zhang, Y. Zhang, D. A. Reed, and D. A. Reed, "Svpablo: A multi-language performance analysis system," in *In 10th International Conference on Performance Tools*, pp. 352–355, 1998. 37

[58] S. Girona, J. Labarta, and R. M. Badia, "Validation of dimemas communication model for mpi collective operations," in *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 39–46, 2000. 37

[59] D. Bailey, E. Barszcz, J. Barton, and D. Browning, "The NAS Parallel Benchmarks," *International Journal of Supercomputer Applications*, vol. 5, pp. 66–73, Jan 1991. 40, 67, 70, 78, 131

[60] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional," *Journal of High Performance Computing Applications*, vol. 14, pp. 330–346, Jan 2000. 79, 132

[61] S. Hioki, "QCDMPI—pure QCD monte carlo simulation code with mpi," *Nuclear Physics B-Proceedings Supplements*, vol. 63, pp. 1000–1002, Apr. 1998. 97

[62] E. Learmann, "Recent results from lattice qcd simulations," *Nuclear Physics A*, vol. 610, pp. 1–12, 1996. 98

[63] B. Goglin, J. Hursey, and J. M. Squyres, "netloc: Towards a comprehensive view of the hpc system topology," in *Fifth International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI 2014)*, IEEE, 2014. 111

[64] A. Wong, D. Rexachs, and E. Luque, "Extraction of parallel application signatures for performance prediction," *HPCC, 10th IEEE International Conference*, pp. 223–230, 2010. 165

[65] L. Lamport and C. Time, "The Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978. 168, 173

[66] J. Dongarra, A. D. Malony, S. Moore, P. Mucci, and S. Shende, "Performance instrumentation and measurement for terascale systems," in *European Center for Parallelism of Barcelona*, pp. 53–62, 2003. 170

Lista de acrónimos

BH: Black Hole Process

BHN: Black Hole Noise process

SCIC: Scalable Computational Instruction Count.

CRM: Computational Regression Model.

CT4NC: Computational Time for N Cores.

EComm: External communications

IG: Instruction Group

LT4NC: Logical Trace for N Cores.

MeasurableP: Measurable Process

PAS2P: Parallel Application Signature for Performance Prediction.

P3S: Prediction Parallel Program Scalability.

RangeST: Range of Scaled Traces processes

S_xPT: Signature Physical Trace for x processes.

SLT: Scalable Logical Trace.

ST4NP: Scaled Trace for N Processes.

SS: Synthetic Signature

ST: Signature Tree.

Lista de abreviaturas de variables utilizadas

F_i: Fase i de la aplicación

$NPROCS$: Número total de procesos

NI_i: Número de instrucciones del proceso i

$NumeroDeProcesosBH$: Número total de procesos Black Hole (BH)

$PhaseET_i$: Tiempo predicho para la fase i de la aplicación

PET: Tiempo de ejecución predicho de la aplicación

P_i: Proceso i de la aplicación

S_x: Firma de la aplicación para x procesos

$TotalIG_i$: Número de instrucciones del proceso i

$TotComEnTransitoFase_i$: Número de comunicaciones en tránsito de la fase i

$TotEcommsFase_i$: Número total de instrucciones externas de la fase i

$TotCommRuidoFase_i$: Número total de comunicaciones ruido de la fase i

W_i: Peso de la fase i