Universitat Autònoma de Barcelona

Efficient Data Management Strategies for Sequence Alignment on Heterogeneous Clusters

Chen Shaolong

Efficient Data Management Strategies for Sequence Alignment on Heterogeneous Clusters

Chen Shaolong

Supervisor: Miquel Angel Senar Rosell

Department of Computer Architecture and Operating System

Universidad Autònoma de Barcelona

This dissertation is submitted for the degree of

Doctor of Philosophy

April 2019

Dedicate this thesis to my loving grandmother

. . .

Declaration

This dissertation is submitted by Chen Shaolong for the degree of Doctor of Philosophy at the Universidad Autònoma de Barcelona, under the supervision of Prof. Miquel Angel Senar Rosell, Computer Architecture and Operating System Department, PhD. in Computer Science. I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgements.

Chen Shaolong

April 2019

Acknowledgements

After an intensive period of four years in UAB, now is the day to write section acknowledgements to everyone is the finishing touch on my dissertation. This four years in Barcelona has been an unforgettable memory during my whole life, not only in the academic arena, but also on the personal level. I would like to reflect on the people who have supported and helped me so much throughout this period.

I first would like to acknowledge my advisor Miquel Angel Senar who helps me a lot for the last four years in UAB. He taught me much about how to analysis and solve problems, how to present your slides and do your research. His kindnesses and generosities help me not only shape my own perspective profoundly on research activities, but also on my personal life. He is a great person to cooperate with during the last four years.

I also would like to thank all the colleagues in CAOS of UAB. They are Josefina, Pilar, Liu Zhengchun, Cesar, Joe, Cecilia and so on. They show kindly heart and make me feel like living in my mother country. Special thanks to Pilar and Cecilia, their kindness helps me a lot.

I also would like to thank all the friends in Cerdanyola Del Valles. I lived in this small town for four years and never forget this impressive memory in life. Special thanks to my roommates, Lu Xiaoxue, Lv Jike, Liu Bin and Li Liang, they help me a lot and make me feel like living at home. Special thanks to my best friend in Barcelona and my cat, Pan Yu and Jaja, without them, I would feel lonely myself.

Finally, I would like to thank my parents and girlfriend Fang, their patience, love and support are indispensable part for me. Without them, I can not complete this thesis . Special to my grandmother, you left me forever. I always miss you.

This research would not be possibly completed without the financial support of China Scholarship Council(CSC) under reference number 201406890007 and Spanish Project under contract TIN2014- 53234-C2-1-R. I am especially indebted to Na Yan and Prof.Li Hong, who work in Shanghai University and have been supportive of my grant goals during last four years. I am also especially appreciated to Gemma and Daniel in UAB, thanks for their great support and assistant in my academic time in UAB.

Thank you very much, everyone!

Chen, Shaolong

Barcelona, April, 2019.

Abstract

Among the high performance computing systems, the Intel Xeon Phi is an accelerator that turns out to be a very attractive alternative to improve the performance of applications with intense computing needs that are traditionally executed in systems based on multicore servers. These applications can be migrated from a multicore server to an accelerator with a low coding effort because both systems are based on nuclei with the same basic architecture.

In our study, we focused our attention on BWA, one of the most popular sequence aligners, and we have analyzed different modes of execution of BWA in various heterogeneous computing systems that incorporate an accelerator.

The alignment of sequences is a fundamental phase in the analysis of genomic variants and has a high computational cost. Although its coding to run in a multicore system can be simple, achieving good performance is not easy in this type of systems, as our results show. We have developed and evaluated different strategies that have been applied on BWA and, of all of them, we conclude that the MDPR variant, which combines data parallelization and data replication, is the one that provides the best results in all systems evaluated. MDPR has a generic design that allows it to be used in different heterogeneous systems. On the one hand, we have applied it in a system consisting of a server with Intel Xeon multicore processors and a Xeon Phi accelerator. And, on the other hand, we have also evaluated it in other heterogeneous systems based on multicore servers equipped with AMD and Intel processors.

In all these hardware configurations, we have tested two dynamic modes and one static mode of data distribution in MDPR. Our experimental results show that the best results for MDPR are obtained when the static mode of data distribution is applied. The dynamic strategy based on round robin achieves a similar performance without the off-line overhead incurred by the static mode. Although our proposal was applied to BWA using human genome data samples, this strategy can be easily applied to other sequence data and other alignment tools that have operating principles similar to those of the BWA aligner.

Resumen

Entre los sistemas de computación de alto rendimiento, el Intel Xeon Phi es un acelerador que resulta ser una alternativa muy atractiva para mejorar el rendimiento de aplicaciones con necesidades de cómputo intensas que tradicionalmente se ejecutan en sistemas basados en servidores multinúcleo. Esas aplicaciones se pueden migrar de un servidor multinúcleo a un acelerador con un bajo esfuerzo de codificación porque ambos sistemas se basan en núcleos con una misma arquitectura básica.

En nuestro estudio, centramos nuestra atención en BWA, uno de los alineadores de secuencia más populares, y hemos analizado diferentes modos de ejecución de BWA en varios sistemas informáticos heterogéneos que incorporan un acelerador.

La alineación de secuencias es una fase fundamental en el análisis de variantes genómicas y tiene un alto coste computacional. Aunque su codificación para ejecutarse en un sistema de múltiples núcleos puede ser simple, lograr un buen rendimiento no es fácil en este tipo de sistemas, como muestran nuestros resultados. Hemos desarrollado y evaluado diferentes estrategias que se han aplicado en BWA y, de todas ellas, llegamos a la conclusión de que la variante MDPR, que combina la paralelización de datos y la replicación de datos, es la que proporciona los mejores resultados en todos los sistemas evaluados. MDPR tiene un diseño genérico que permite su uso en diferentes sistemas heterogéneos. Por un lado, lo hemos aplicado en un sistema que consta de un servidor con procesadores multinúcleo Intel Xeon y un acelerador Xeon Phi. Y, por otro lado, también lo hemos evaluado en otros sistemas heterogéneos basados en servidores multinúcleo equipados con procesadores AMD e Intel.

En todas estas configuraciones de hardware, hemos probado dos modos dinámicos y un modo estático de distribución de datos en MDPR. Nuestros resultados experimentales muestran que los mejores resultados para MDPR se obtienen cuando se aplica el modo estático de distribución de datos. La estrategia dinámica basada en round robin logra un rendimiento similar sin el sobrecoste inicial que requiere el modo estático. Aunque nuestra propuesta se aplicó a BWA utilizando muestras de datos del genoma humano, esta estrategia

se puede aplicar fácilmente a otros datos de secuencia y a otras herramientas de alineación que tienen principios operativos similares a los del alineador BWA.

Resum

Entre els sistemes de computació d'alt rendiment, l'Intel Xeon Phi és un acelerador que resulta ser una alternativa molt atractiva per millorar el rendiment d'aplicacions amb necessitats de còmput intensiu que s'executen tradicionalment en sistemes basats en servidors multinucli. Aquestes aplicacions es poden migrar d'un servidor multinucli a un accelerador amb un esforç de codificació baix perquè tots dos sistemes es basen en una mateixa arquitectura bàsica.

En el nostre estudi, centrem la nostra atenció en BWA, un dels alineadors de seqüència més populars, i hem analitzat diferents modes d'execució de BWA en diversos sistemes de computació heterogenis que incorporen un accelerador.

L'alineació de seqüències és una fase fonamental en l'anàlisi de les variants genòmiques i té un cost computacional alt. Encara que la seva codificació per executar-se en un sistema de múltiples nuclis pot ser senzilla, aconseguir un bin rendiment en aquest tipus de sistemes no és fàcil, com mostren els nostres resultats. Hem desenvolupat i avaluat diferents estratègies que s'han aplicat a BWA i, de totes, arribem a la conclusió que la variant MDPR, que combina la paral·lelització de dades i la replicació de dades, ésla que aconsegueix els millors resultats en tots sistemes avaluats. MDPR té un disseny genèric que permet el seu ús en diferents sistemes heterogenis. Per un costat, l'hem aplicat en un sistema que consta d'un servidor amb processadors multinucli Intel Xeon i un accelerador Xeon Phi. I, per un altre costat, també l'hem avaluada en altres sistemes heterogènics basats en servidors multinucli equipats amb processadors AMD i Intel.

En totes aquestes configuracions de maquinari, hem provat dos modes dinàmics de distribució de dades en MDPR i un d'estàtic. Els nostres resultats experimentals mostren que els millors resultats per a MDPR s'obtennen quan s'aplica el mode de distribució de dades estàtic. L'estratègia dinàmica obté resultats similars sense el sobrecost inicial del model estàtic. Encara que la nostra proposta s'hagi aplicat a BWA utilitzant mostres de dades del

genoma humà, aquesta estratègia es pot aplicar fàcilment amb altres dades de seqüències i altres eines d'avanç que tinguin principis operatius semblants als de l'algorisme BWA.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Background

1.1.1 Genome Data Booming

Nowadays, the amount of sequence data has explosively increased from the GB level to the TB level as the computation cost decreases from 10 million in 2007 to 1000 in 2017 [1]. Figure 1.1 shows the growth of DNA sequence data in Genbank and its user amount in last 24 years(1989-2013) [2]. GenBank is the world's largest annotated collection of publicly available DNA sequences. Genome sequence data and biological users are increasing rapidly with the emerging the next generation sequence technology.

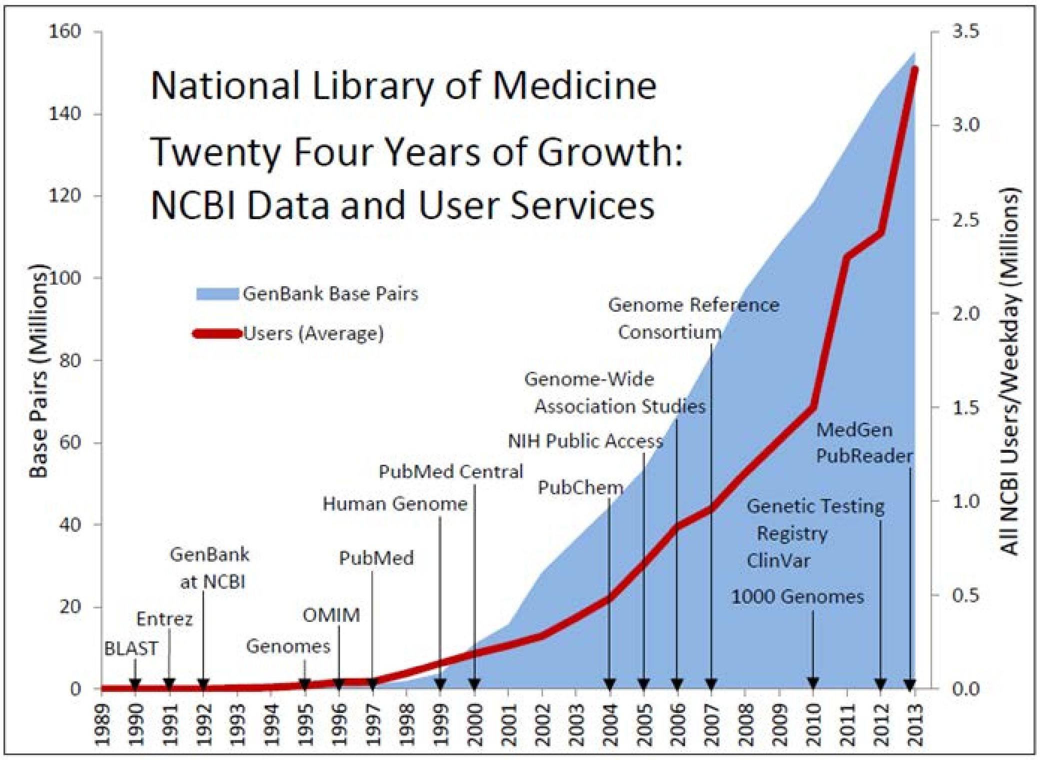


Fig. 1.1 Growth of NCBI DNA sequence data and user amount in 24 years, from 1989 to 2013

The decreasing costs of sequencing drives DNA research developing rapidly. As shown in Figure 1.2 that illustrates the costs per genome based the U.S [3]. National Institute of Health, with the NGS technology burgeoning, sequencing a human genome has decreased in cost from $100 million in 2001 to $1000 in 2017, which leads to an explosive increasing of sequence data from the GB level to the TB level. While the amount of genome data is doubling every 12 months, computing power of a single processor is only doubling every 18 months. Which means that biology's big data sets are being generated faster than improvements in storage and processing of computing technology. Therefore, new advances in software tools are needed to handle this challenge by efficiently exploiting technological features presented in modern parallel architectures.

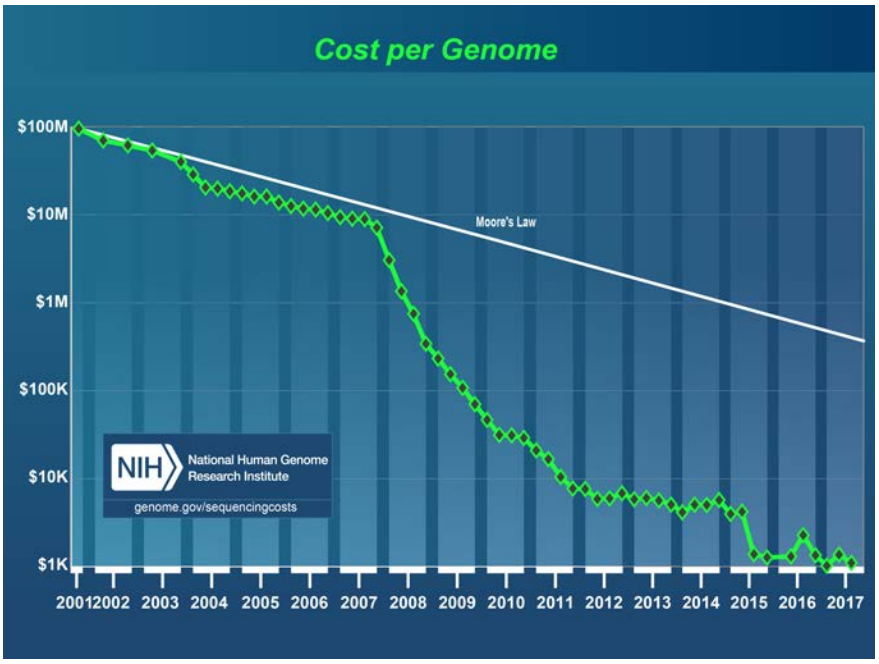


Fig. 1.2 The cost of sequencing a human genome vs Moore's law in HPC hardware

1.1.2 HPC Hardware Limitation

In recent years, heterogeneous systems consisting of different architectures have become a significant trend in the high performance computing(HPC) arena [4]. GPUs have obtained a significant popularity despite the difficulty associated with the usage of specific programming languages such as OpenCL or CUDA [5]. Intel Xeon Phi, which is based on a common architecture x86 and provides 57 to 61 cores on one board, is an attractive solution that can easily transform programs written in the same language used on multicore CPU architectures [6], without requiring the complexity of porting processes.

Figure 1.3 displays evolution of the number of cores for per processor in TOP 500 supercomputer list [7]. Obviously, the processing units integrated in such systems tend to provide more parallelism in processors year by year. Moreover, manycore integrate in one coprocessor such as Intel Xeon Phi and GPUs, has been paid growing up attention nowadays. Manycore architectures, such as GPUs, have obtained considerable popularity, despite the fact that the specific programming languages such as OpenCL and CUDA requiring significant coding skills [5]. Intel Xeon Phi [6] is another manycore coprocessor, which has became an attractive solution that existing programs on multicore systems can be easily ported because Xeon Phi is also based on the same x86 architecture.

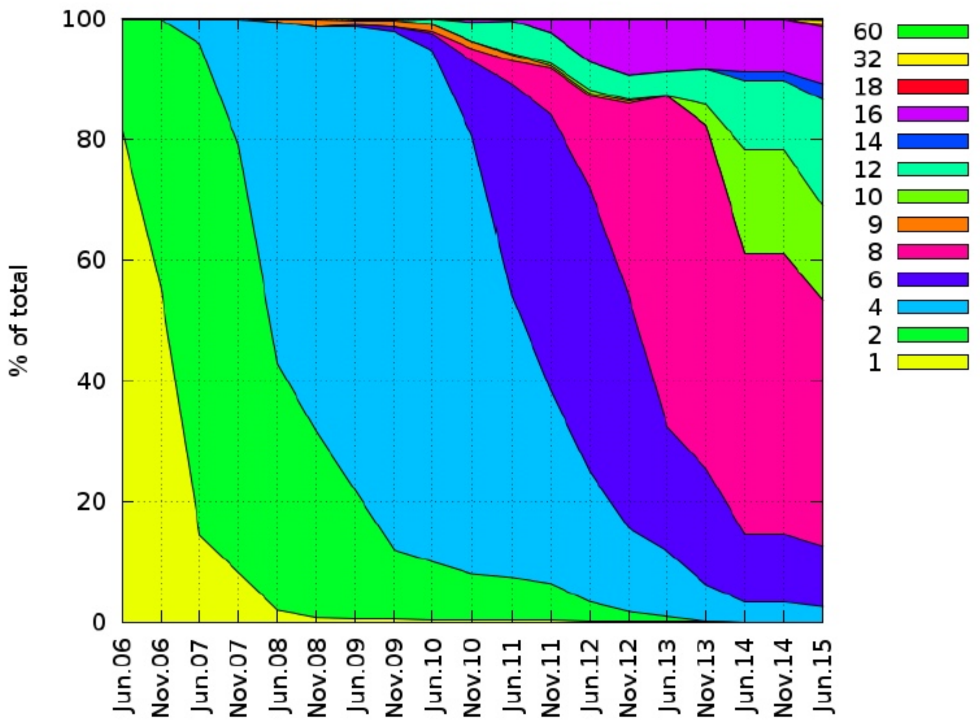


Fig. 1.3 Evolution of the number of cores per processor from TOP500 supercomputer list, from June 2006 to 2015

For the sake of better utilization of heterogeneous cluster, it is common now that application programs apply hybrid MPI and OpenMP programming on the HPC systems [8], as shown in Figure 1.4. Three significant motivations for this hybrid programming model, one is that the reduction of memory consumption, one is that performance improvement at high core counts, another is that we could utilize the heterogeneous architecture under the shared and distributed memory level.

In term of reduction of memory footprint, the reason is that designing and implementing MPI applications generally requires that some same data be replicated among MPI processes, while OpenMP is based on the shared data access. Additional, modern HPC system always owns a limited hardware resource, especially a small size in memory bank. Typically only 1

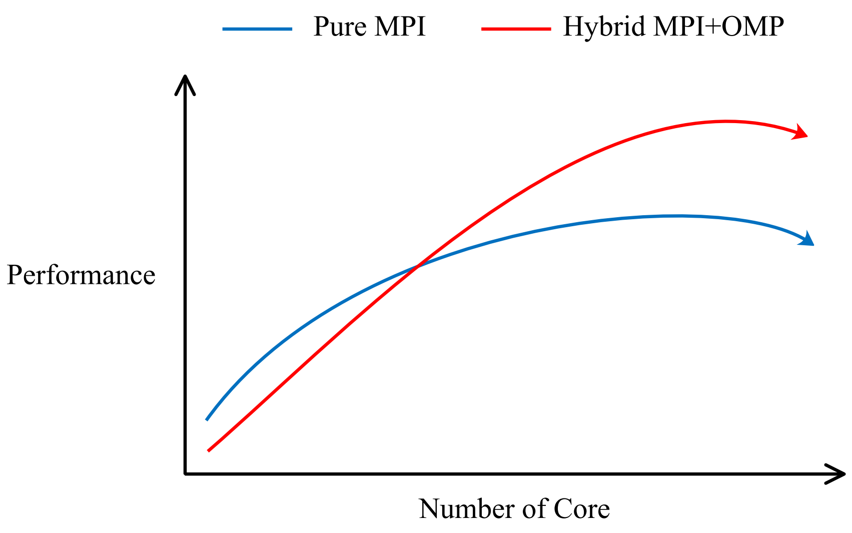


Fig. 1.4 Typical performance curves of pure MPI and hybrid MPI+OpenMP

or 2 GB memory per core. Thus, comparing to pure MPI execution, hybrid MPI and OpenMP programming consumes less memory footprint, comparing to pure OpenMP execution, hybrid MPI and OpenMP programming improves the efficiency of utilization of memory.

In term of performance improvement, when low core counts are applied, overheads between MPI and OpenMP make the performance of the pure MPI achieve better than that of hybrid MPI and OpenMP. On the contrary, when at high core counts, hybrid MPI and OpenMP would outperform pure MPI and pure OpenMP, due to the exploiting additional parallelism in OpenMP or MPI. Ultimately, in term of memory architecture, hybrid MPI and OpenMP programming mode give us a better utilization of the shared and distributed memory level, to reduce communication overheads, load imbalance and memory access costs, especially in our NUMA architecture, which is a local shared memory among local cores.

1.1.3 Sequence Alignment

Compared to computing evolution (doubling the number of transistors every 18 months), the amount of sequence genome data only doubles every 12 months [9]. This spectacular growth of genome data constitutes an important challenge that requires efficient applications to process them.

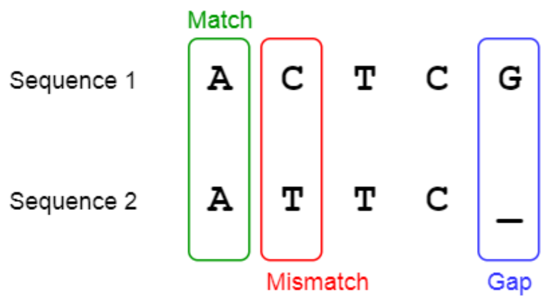


Fig. 1.5 A simple example of sequence alignment

Sequence alignment, variant calling and variant annotation comprises three fundamental operations in genome data studies [10]. Sequence alignment is a crucial step that can provide primary consequences for the other two remaining procedures. Sequence alignment involves mapping short reads to a genome sequence reference as accurately as possible [11].

Figure 1.5 displays a simple example of sequence alignment process which allows mismatch and gap in the algorithm of alignment. Each aligner owns its algorithm to calculate distance between short read and genome reference. In this case, we mapped two sequences according to best score in algorithm and posit ACTCG and ATTC-. Mismatch is allowed in the second positon and gap in the last position. This result could be useful for next workflows in sequence variant analysis.

Although many aligners have been developed in recent years, such as MAQ, SOAP and BLAST, they all exhibit a significant execution time and a large memory footprint. BWA aligner is one of the most prevalent and widespread sequence alignment tool available. As many existing mappers, BWA takes advantage of parallelization techniques in order to reduce the computational demands involved in the alignment of millions of reads. Although, its execution time is similar to other existing aligners, BWA (Burrows-Wheeler Aligner) [12] takes advantage of the BWT indexing technique that decreases the memory requirements of the alignment process. This reduction in memory needs combined with its mapping accuracy have made BWA one of the most popular aligners in the scientific community.

However, BWA exhibits significant scalability problems when it is run on systems with large number of cores. Similar scalability problems are observed on manycore architectures, such as the Intel Xeon Phi, or in hybrid architectures combining both multicore CPUs and manycore accelerators. These behaviors were also observed in our experiment in the following section.

1.2 Motivation

Genome data is booming with the development of next generation sequence technology, high performance computer hardware can not achieve the speed of genome data, which limits the performance improvement of sequence aligners. All factors indicts that more efficient software is needed in variant analysis in the high performance computing arena.

As shown in Figure 1.6, we choose the widespread and popular sequence alignment application, BWA aligner, to evaluate its performance. We set up a base point which implements BWA aligner on Intel Xeon under one thread. Speed-up is measured on that base point. Original BWA aligner only can be executed on Xeon platform(Sandman) without revision. With applying 24 threads, as Figure 1.6 illustrates, it achieves the highest performance, 11.5-fold speed-up under the execution of Intel Xeon. The tendency of speed-up varies from linear to very smooth and approaches its performance pole under 24 threads, but its thread efficiency decreases to 48%. It's finite to enhance the performance of BWA aligner by increasing the number of working threads, we are demanded to further research in a more efficient way.

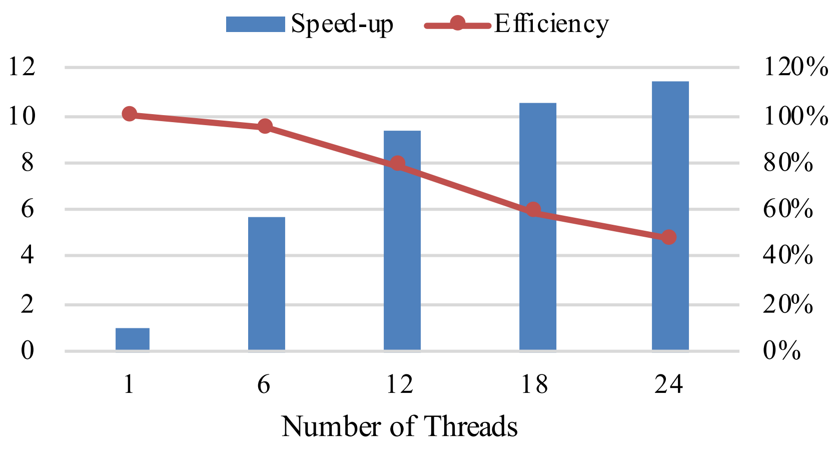


Fig. 1.6 BWA scalability on 24-thread system of Intel Xeon. Speed-up and efficiency are measured according to the value of 1-thread BWA aligner. Speed-up is illustrated at the left y-axis and efficiency at the right y-axis.

In order to demonstrate our motivation clearly, we illustrate a result of our experiments in advance. The addition of multi-threading to an aligner does not guarantee it will utilize threads efficiently. In fact, it is quite common to observe that efficiency decreases when the thread count grows large enough.

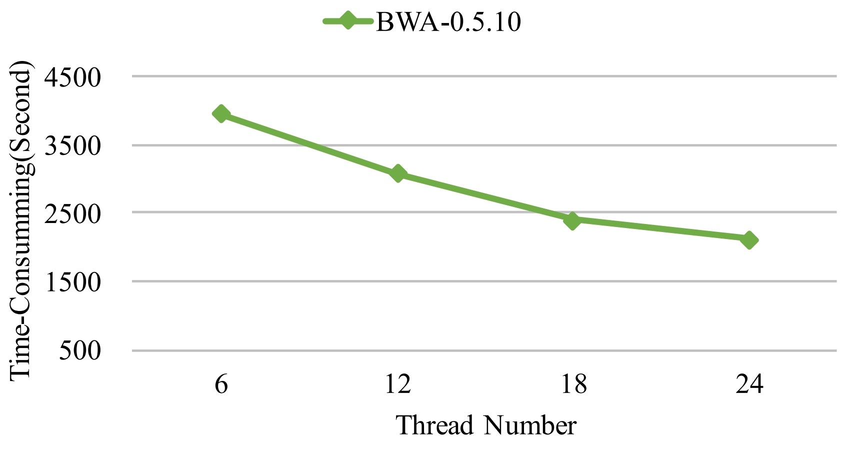


Fig. 1.7 BWA scalability on 2-socket 12-core 24-thread system of Intel Xeon

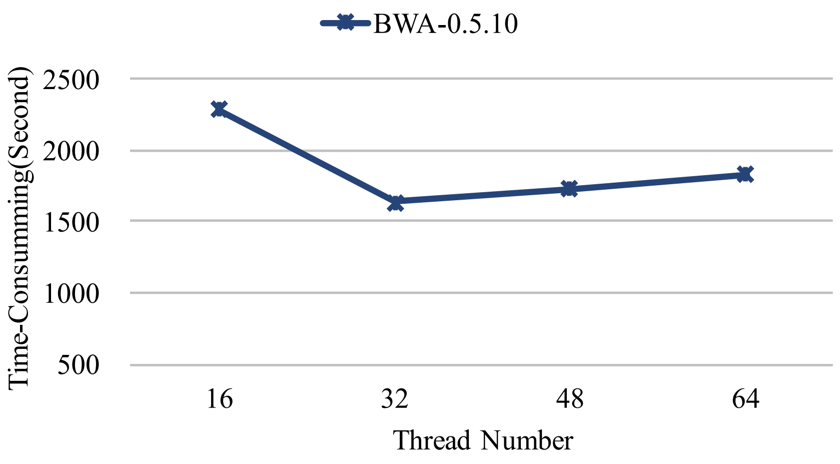


Fig. 1.8 BWA scalability on 4-socket 32-core 64-thread system of Intel Xeon

This phenomenon is observed and shown in Figures 1.7 and 1.8. These figures illustrate the results obtained by BWA aligner when mapping a large short-read genome example of 17.4GB against human genome reference. Each figure illustrates the results obtained when BWA aligner was executed on two different servers. Both systems are based on NUMA architectures but they differ in the total number of processors and cores available. The first system (Fig.1.7) owns two-socket Intel Xeon CPU with 12 cores. Twenty-four threads can be executed simultaneously using Hyper Thread technology. The second system (Fig.1.8) is a 4-socket Intel Xeon CPU. Each socket contains 8-core processor, so the total number of cores is 32 and 64 threads can be executed simultaneously. In both cases, each socket is connected with a memory bank, which constitutes a NUMA node. NUMA nodes are connected by QuickPath Interconnect links.

Interesting results occur in the experiments. When BWA runs in these NUMA systems, threads are spread throughout the system and memory is allocated using a first-touch data allocation policy, which means that, when a program is started on a CPU, data requested by that program will be stored on a memory bank corresponding to its local CPU. In particular, this behavior affects the allocation of the reference index of genome in the implementation of BWA.

Threads that run on the same NUMA node where the reference index is allocated have a lower latency when they access it. Therefore, when the number of threads increases, all the speed up gained due to multi-thread is mitigated by the latency of remote accesses and traffic saturation of interconnection links. This degradation of performance can be observed in Figure 1.7 which shows a continuous reduction of execution times, but the gain in performance reduces as more threads are added in the execution.

Being a system with only two NUMA nodes, performance degradation starts to be significant when more than 12 threads are used and threads are allocated on both NUMA nodes. Especially, when the application goes from 18 to 24 threads, potential gains due to the extra threads. However, the extra threads have a small impact in execution time due to the memory latency issues on both NUMA nodes, which is the crucial factor that affects performance.

When the thread count grows large enough, problems with memory accesses are getting worse. As Figure 1.8 illustrates, BWA runs in a large NUMA system, and when more than 32 threads are used, execution times are degraded and adding more threads has a negative impact in overall performance. This loss in performance can be explained, on the one hand, by the increased latency of remote memory accesses and, on the other hand, by the memory bank in which the reference genome is allocated becoming a bottleneck that generates memory contention.

More discussions and details we would talk about in the following sections. Multiple threads ability can not enable the threads efficiency, specially when facing a large amount of genome sequence dataset. In general, our HPC system always has a limited hardware, how can we make maximal usage of computing hardware with less coding to improve performance in sequence alignment? It is worth something to do some further research in this arena.

1.3 Objective

In this thesis, the ultimate goal we focus on is that exploring efficient data management strategies of data parallelization and data replication for variant analysis workflows, specially sequence alignment, on the heterogeneous cluster which consists of multicore and manycore system.

The following objectives we explore in the thesis with the purpose of achieving our goals:

- Definition of a data management strategy of MDPR to efficiently utilize the heterogeneous cluster on BWA aligner application by evaluating its performance on various values of thread groups and three data distribution schedule on the heterogeneous cluster. This MDPR strategy could be summarized in the following steps:
  - Data replication on the genome reference
  - Data parallelization on the short read
  - Data distribution of static mode on the heterogeneous cluster
  - Data distribution of dynamic mode that comprises even and round-robin fashion on the heterogeneous cluster
- Through the evolution of data management strategies to recognize efficiently evaluate of data parallelization and data replication for variant analysis workflows. This development could be concluded in the following contents:
  - A strategy of DP based on data replication of genome reference on the homogeneous cluster
  - A strategy of DR based on data parallelization of short read on the homogeneous cluster
  - A strategy of DPR which combines DP and DR strategies on the homogeneous cluster

  - A strategy of MDPR generated from DPR strategy that utilizes the heterogeneous cluster

- Evaluation of some relevant performance factors that demonstrates the impact of system architecture and sequence aligner, which comes to perform a significant result on the overall performance.
  - NUMA effect
  - Three basic execution mode on the multicore-manycore heterogeneous architecture
  - Three thread libraries: Pthread, OpenMP and Intel Cilk

1.4 Contribution

In this thesis, we explore different strategies that allow BWA running on the heterogeneous systems, such as manycore and multicore systems. We analyze some proposed strategies in the literature and propose new strategies and crucial elements to improve global performance. This paper concentrates on which we proposed the strategies of data parallelization and data replication and we made an evaluation of BWA-based aligners in the native mode, offload mode and symmetric mode in manycore-based systems. From our experiment works, best results were obtained with the strategy that combines data parallelism and data replication. We denoted this combined strategy as MDPR(Multiple level Data Parallelization and Replication).

The strategy (MDPR) utilizes multiple instances in order to reduce memory bottlenecks by decreasing memory congestion and improving memory locality. Our results in the experiments show that MDPR achieved a considerable speed up compared with other implementations of BWA.

From these results, we present a more extensive study where we analyze the performance of different BWA variants on multicore and manycore systems. We have also adjusted the design of MDPR so that it can be executed indistinctly in multicore or manycore systems, and we have augmented it with different mechanisms of dynamic data distribution.

Main contributions of this thesis can be summarized as follows:

- We provide an OpenMP version of BWA aligner. Although BWA has taken advantage of multi-threading ability with Pthread library, it illustrates a poor performance in scalability when a large threads were applied. Moreover, we also made a comparison among Pthread library, OpenMP library and Intel Cilk library, and found OpenMP-based BWA we proposed outperforms others in terms of overall time and load balance.
- We compare three basic execution modes, native mode, offload mode and symmetric mode, in which we could utilize the heterogeneous architecture of Intel Xeon and Xeon Phi. Moreover, thread scalability performance of three execution modes was evaluated in the experiments.
- We propose four strategies (DP, DR, DPR and MDPR) that cover different combinations of data parallelization and data replication when we applied them to BWA, one of the most popular genome read mapper, in order to explore efficient data parallelism in the homogeneous and heterogeneous systems.
- We analyze three data distribution schedule in MDPR strategy, one static mode and two dynamic modes. Static mode focuses on the data load among nodes based on the native mode running performance. Dynamic modes distribute dataset among nodes in a fashion of even and round-robin data management. Although static mode achieve the best time-consuming of all, it needs a complicated configurations beforehand. In term of complexity and convenience, dynamic mode could be a better choice in performance improvement, specially round-robin mode could reach an approximate time-cost with static mode.
- We carry out an experimental study comparing different versions of BWA on multi-core and manycore systems, including the strategies proposed in this work and other

alternatives proposed in the literature, such as mBWA, pBWA, BWA-ALN, etc. This study has shown us the best alternatives to run BWA on manycore systems based on Intel Xeon Phi and other multicore systems based on Intel and AMD processors.

1.5 Outline

The works of the thesis is structured as follows.

- **Chapter 1: Introduction**

  In this chapter, some crucial elements in this research are proposed, like motivations, objectives of works. We also give our contributions and outline for better reviewing.

- **Chapter 2: Literature Review on Related Works**

  In this chapter, we make a literature reviews on related works in performance improvement of sequence alignment on the heterogeneous multicore-manycore architecture and sequence aligners, especially the most popular and widespread aligner, BWA aligner.

- **Chapter 3: Variant Analysis**

  In this chapter, relevant subjects concerning variant analysis in DNA sequence are discussed. We discuss more information about alignment since we concentrate on sequence aligner.

- **Chapter 4: Heterogeneous Multicore-Manycore Architectures**

  In this chapter, we provide a brief description of the basic architectures that we have used in our study, namely, multicore systems with NUMA nodes and manycore accelerators, Intel Xeon, Intel Xeon Phi, and AMD Opteron.

- **Chapter 5: Data Management Strategies Based on Data Parallelization and Replication**

In this chapter, we discuss the strategies of DP, DR, DPR and MDPR which are based on data parallelization and replication. Detailed architecture is illustrated in this chapter for viewers to readily understand our works. We also make a comparison with these strategies with our previous research.

- **Chapter 6: Experimental Implementation and Environment**

  In this chapter, sample datasets we utilize, implementation systems we apply, related software we use and experiment environment we set, are illustrated in details here.

- **Chapter 7: Experimental Results and Evaluation**

  In this chapter, we not only make an evaluation of the three basic execution modes in the heterogeneous system made of Intel Xeon and Xeon Phi, but also propose a OpenMP-based version of BWA aligner. The data management strategies, including DP, DR, DPR and MDPR, are also evaluated in this chapter.

- **Chapter 8: Discussion and Conclusion**

  In this chapter we discuss about the results from the experiments and conclude the achievements we achieve in the research. We also illustrate some promising perspectives on the future research directions.

Chapter 2

Literature Review on Related Works

Our work explores effective data parallelism solutions on the sequence aligner that are suitable for the heterogeneous architectures.

Majority of works on the alignment provide significant performance improvements. However, they are only designed to run on multicore architecture or on GPUs, and underestimated in another manycore architecture like the Intel Xeon Phi. Although some existing studies ([11] and [13]-[14]) that were designed to execute aligner on multicore and manycore architecture and finally achieve a considerable performance improvement, they exhibits some advantages and bottlenecks we would discuss below.

First we concentrate literature review on the performance improvement of sequence aligners in multicore-manycore heterogeneous system. Afterwards, research about BWA aligner is evaluated in many arenas to improve the performance of alignment application.

2.1 Performance Improvement of Sequence Aligner in the Heterogeneous System

For the purpose of better utilizing modern system architectures with alignment application, we need to make a compromise between these two crucial factors. Some previous works from our research group [15] [16] [17] that investigated memory interleaving and data parallelization strategy we proposed on NUMA and Bi-ring architecture under four aligner cases which came to the result that memory interleaving and data parallelization strategy could provide a considerable performance improvement on such two architectures.

Blast [18], Bowtie [19], SOAP2 [20] and BWA [21], many sequence aligners were put forward to solve this booming problem. They are suitable for multicore system while complex coding job is needed for transmitting to manycore part. Many works keep a view of alignment in the architecture of multicore.

Zhang et al. [22] illustrated that sequence alignment is memory-sensitive and tried to optimize BWT index on multicore system. This optimization can not be readily transited to manycore system since it is special designed for multicore system. Although it could compile on the manycore system of Intel Xeon Phi since an identical architecture they used, it can not reach an improvement of performance since they owns different bits wide.

Olivier et al. [23] compared different task parallel schedule of OpenMP and Qthreads on multi-socket multicore architecture and concluded that hierarchical work stealing scheduler is competitive in all benchmarks. One crucial factor that this paper indicates is task parallel schedule plays a significant role in overall performance. Thus we tried different schedule, such as static and dynamic schedule in data distribution, to explore in the experiments.

Macedo et al. [24] introduced an using multi-threading and multi-instance in master/slave parallel strategy to facilitate multiple sequence alignment on heterogeneous multicore clusters. Their results display allocation policy and the master node has great impact on the overall performance of the system. Although they are something similar with our research, they do not explore our ideas of data replication in DR strategy and data parallelization in DP strategy. Our experiments also display the evolution of data parallelization step by step. Jing Nagarajan et al. [25] presented multiple parallelization skills could be useful on multicore system. But they have not tried on the manycore architecture and BWA-backtrack algorithm as we did in the experiments.

Chen et al. [26] raised an offloading acceleration of the seed extension of BWA aligner in FPGA architecture. Houtgast et al. [27][28] made an effort to adopt alignment application on GPU-based and FPGA-based architectures. Tens of research about alignment performance improvement on FPGA are discussed in details in paper [29]. Otherwise, same phenomena occurs on GPU architecture. Many papers [30] about alignment improvement on GPUs are proposed in HPC field. However, it is observed that a few reviews of alignment on Intel Xeon Phi manycore architecture.

2.2 Performance Improvement on BWA Aligner

Many research describe different approaches to improve the performance of alignment application, specially BWA aligner on one special architecture system, multicore or GPUs architectures.

Zhang et al. [22] showed that BWA is a memory-sensitive aligner and improved its performance by optimizing cache mechanisms on the multicore architectures. But two disadvantages they have. On the one hand, the skill of cache optimization requires a comprehensive coding ability, we can not easily transit for other platforms. On the other hand, they do not try experiments on memory-sensitive system, manycore architecture like Intel Xeon Phi and GPUs.

Jing Nagarajan et al. [25] found that BWA aligner owns their performance bottlenecks in thread scalability, scattering and gathering of data and memory bandwidth limitations, respectively. These restrictions are consistent with we observed in the experiments of the scalability evaluation of BWA aligner. Afterwards they introduced an approach of multiple parallelization to accelerate BWA aligner in multicore system, which consists of thread parallelization, data parallelization, and concurrent parallelization. The differences between this work and ours is that we focus on OpenMP thread library instead of Pthread they concentrate, and we also illustrate the evolution and the evaluation of data parallelization strategy.

Houtgast et al. [31] accelerated BWA by offloading some highly parallel computing which is applied on the GPU system. Houtgast et al. [32] utilized a simple adaptive load balance algorithm on GPUs for BWA aligner and achieved a considerable improvement in performance. However, this two research proposed above focus on BWA-MEM algorithm in GPU system, instead of the most widespread algorithm, BWA-backtrack. We did experiments for dynamic data distribution among heterogeneous systems which quite differs with the simple adaptive load balance algorithm, and we also compared one offload version of BWA, mBWA, which works in manycore architecture, Intel Xeon Phi.

BarraCUDA [33] is an implementation of BWA aligner based on GPU which achieves about 3X speed-ups when comparing to the multi-threading original BWA version. Moreover, CUSHAW [34] and SOAP3 [35], they are GPU-based implementation of BWA aligner as well as BarraCUDA and already reach a considerable improvement of performance compared to the original BWA aligner. However, these two versions of BWA do not permit gapped alignments in procedure, which could lead to some mapping results undiscovered. Tens of research about alignment performance improvement in GPU platform as well as another HPC architecture called FPGA are discussed in details in paper [29]. But there are few works of BWA aligner existing in FPGA.

For the distributed computing among multiple clusters, by utilizing Hadoop technology, BigBWA [36] achieved a significant improvement of performance on multiple nodes. While SparkBWA [37] tried to use another big data technology, Spark, to boost BWA aligner. This is some versions that might be convenient for distributed computing implementation.

There are also many other proposals focusing on the utilization of BWA under the heterogeneous architecture, especially one multicore system with MIC architecture, Intel Xeon + Intel Xeon Phi.

pBWA [14] is an efficient parallel version of BWA based on the Open MPI library. It not only preserves the multi-thread capability provided by BWA but also adds efficient parallelization for core alignment functions on the heterogeneous cluster. Unfortunately, on the one hand, it cannot be evaluated at the case of large number of instances of pBWA on the Xeon Phi architecture as a limited memory is used. On the other hand, even data distribution among multiple instances which leads to a crucial bottleneck in the heterogeneous system cause of the huge difference might be on the performance of dissimilar architectures, according to our experiments in MDPR.

Xinmin Tian et al. [13] presented several practical SIMD vectorization techniques on MIC architecture to achieve improvements in performance of BWA. However, such SIMD techniques require sophisticated coding skills in order to incorporate with the existing genome aligners. Suejb Memeti et al. [38] and Lipeng Wang et al. [39] provided parallel solutions for large scale DNA analysis which exploits thread-level and SIMD techniques in the Xeon Phi, but their solution does not deal with data-level parallelism as we explore in the research.

New aligner MICA [40] optimized MIC limitation and explored the extra parallelism inside each MIC core. But MICA still an initial version which needs many configurations before implementation. Yingbo Cui et al. [11] proposed mBWA which works according to offload mode with Intel Xeon Phi. mBWA utilizes a multi-level parallelization strategy that includes data IO parallelization and a parallel pipeline in reads alignment because of data-dependent. However, mBWA is not an open source project and can not be executed readily under multiple Intel Xeon Phi nodes. Charlotte Herzeel et al. [41] evaluated an Intel Cilk implementation of BWA on the heterogeneous systems that consists of Intel Xeon Phi, and at some extent corrected data load imbalance that produced by differences in data reads. It is worth something that could provide a third-part view of thread library when we compared with Pthread and OpenMP libraries.

Chapter 3

Variant Analysis

In this chapter, relevant subjects concerning variant analysis in DNA sequence are discussed. The properties of a DNA molecule and genome sequence are shown in Section 3.1. Then the details of sequence alignment are discussed in Section 3.2. Following by Section 3.3 we analyze the performance of BWA, a widespread sequence alignment application, to demonstrate some interesting behaviors in BWA aligner. Section 3.4 and 3.5 is a short overview of variant calling and variant annotation, which consist of the two remaining main functions in the variant analysis workflows.

3.1 Overview

Biological information is stored in deoxyribonucleic acid (DNA). DNA is a kind of nucleic acid composed of double chains, as shown in Figure 3.1, that coil around each other to form a double helix carrying the genetic instructions used in the growth, development, functioning and reproduction of all known living organisms and many viruses [42]. This double-stranded structure could store the same biological information in both chains.

This double-stranded structure is also known as polynucleotides since they are composed of simpler monomeric units called nucleotides [43]. There are four types of nitrogen-

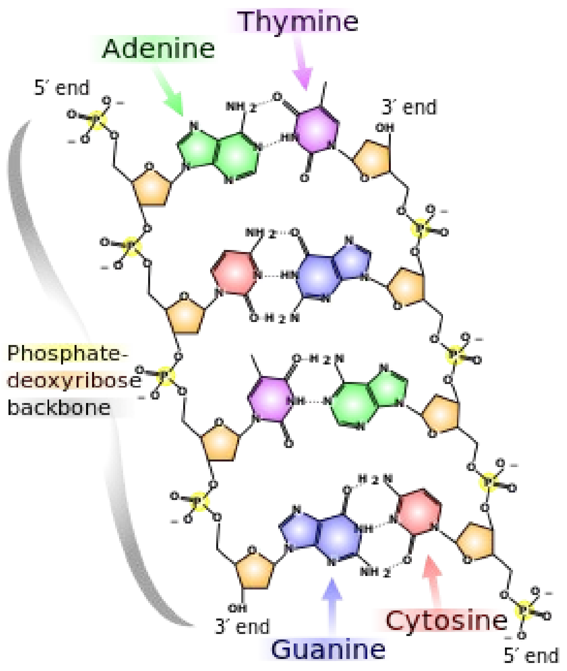


Fig. 3.1 The molecular structure of DNA, four types of nitrogen-containing nucleobases: Adenine[A], Thymine[T], Guanine[G] and Cytosine[C], two base pairings(A-T, C-G)

containing nucleobases called Adenine(A), Thymine(T), Guanine(G) and Cytosine(C). According to the rule of base pairing (A with T, C with G), two separate strands are bound together into a double-stranded structure. But most parts of DNA do not serve for protein sequences which is denoted as non-coding parts. The non-coding genome consists in the majority part of fraction of the total sequence genome in many species. For human DNA sequence, only 1.5% of the human genome can code protein, while 50%+ of the human genome consisting of the non-coding part in genome sequence [44]. However, these non-coding section in genome sequence may still encode some functional RNA molecules that is crucial elements involved in the gene expression [45].

For next generation sequence (NGS) in human gene, variant analysis is necessary to distinguish Mendelian disorders, complex diseases and somatic mutations in DNA sequence. In fact, three common scenarios are considered for human gene in NGS data [10].

- Identification of causative genes in Mendelian disorders (germline mutations),

- Identification of candidate genes in complex diseases for further functional studies and
- Identification of constitutional mutations as well as driver and passenger genes in cancer (somatic mutations).

As shown in Figure 3.2, it is the basic workflow for variant analysis in genome sequencing [10]. There are four significant workflows consist of the main pipelines in variant analysis, quality assessment, read alignment, variant identification and variant annotation. Quality assessment is to evaluate the quality of raw reads from NGS platform, trim and correct them when necessary in order to reach defined standards. Nowadays, many NGS dataset projects provide standard NGS datasets. It is convenient for the tools of variant analysis of NGS to use without any efforts. Thus, read alignment or sequence alignment, variant identification or variant calling, and variant annotation are consist of the most important steps in variant analysis.

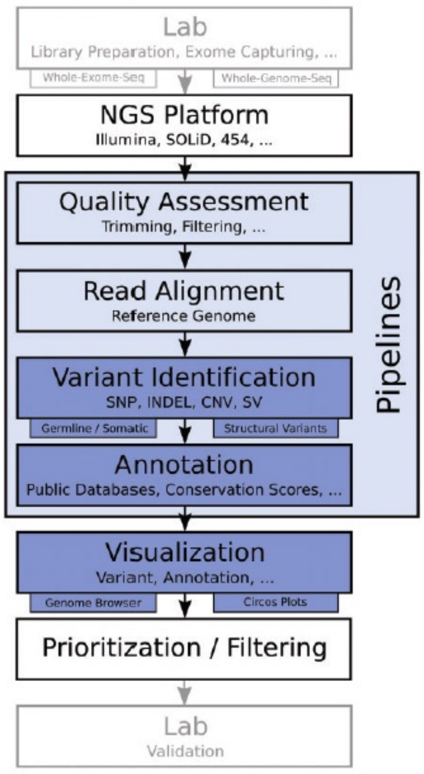


Fig. 3.2 Basic workflow of variant analysis for genome sequencing. Quality assessment, read alignment, variant identification and variant annotation are the four crucial steps of all.

This thesis concentrates on sequence alignment that mapping short read to a reference genome which is the central and fundamental workflow of variant analysis. For these common scenarios above in variant analysis, lots of applications and methods are proposed in next generation sequence arena over recent decades years [46][47]. Many applications have been launched to efficiently process millions of short reads in different variant analysis workflows. mrFAST, SSAHA2, MAQ, Bowtie, BWA, YOABS, Novoalign, SOAP and Stampy. Among these aligners, application BWA is one of the most prevalent and widespread sequence alignment tool which permits mismatches and gaps with low memory. The most significant, variant identification and variant annotation can be more convenient approached with the output of BWA. Figure 3.4 gives a view of three basic workflows in variant analysis in details.

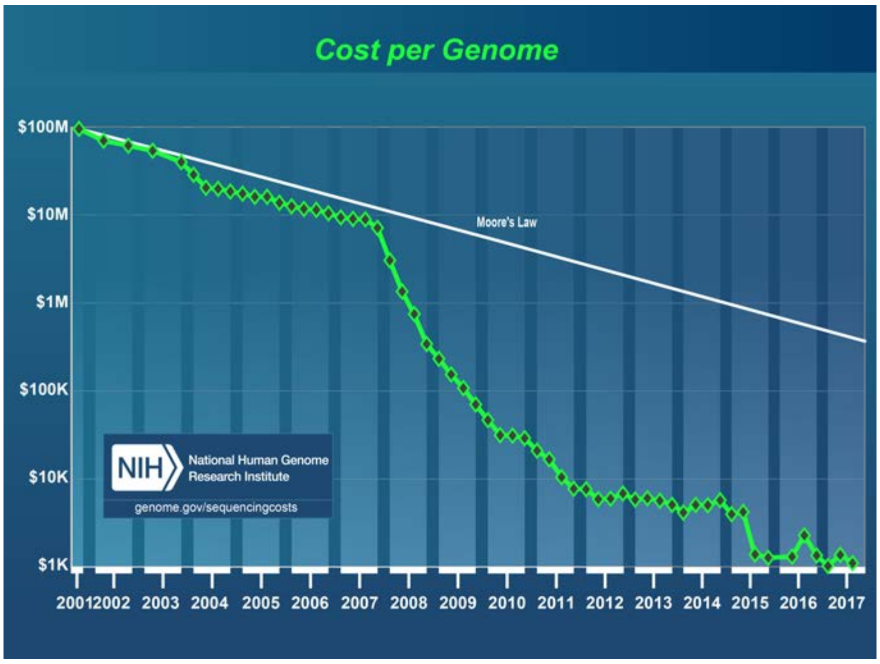


Fig. 3.3 The cost of sequencing a human genome over years

However, applications in variant analysis field still are facing an crucial challenge in high performance computing. The decreasing costs of sequencing drives DNA research developing rapidly. As shown in Figure 3.3 that illustrates the costs per genome based the U.S. National Institute of Health [3], with the NGS technology burgeoning, sequencing a human genome has decreased in cost from $100 million in 2001 to $1000 in 2017, which leads to an explosive increasing of sequence data from the GB level to the TB level. While the amount of genome data is doubling every 12 months, computing power of a single processor is only doubling every 18 months. Which means that biology's big data sets are being generated faster than improvements in storage and processing of computing technology. Therefore, new advances in software tools are needed to handle this challenge by efficiently exploiting technological features presented in modern parallel architectures.

3.2 Sequence Alignment

3.2.1 Overview

For the purpose of better illustration of three key workflows in variant analysis, we display Figure 3.4 to display the differences among them [48]. We first need to known where is short read located in reference genome, especially disease analysis in human genome.

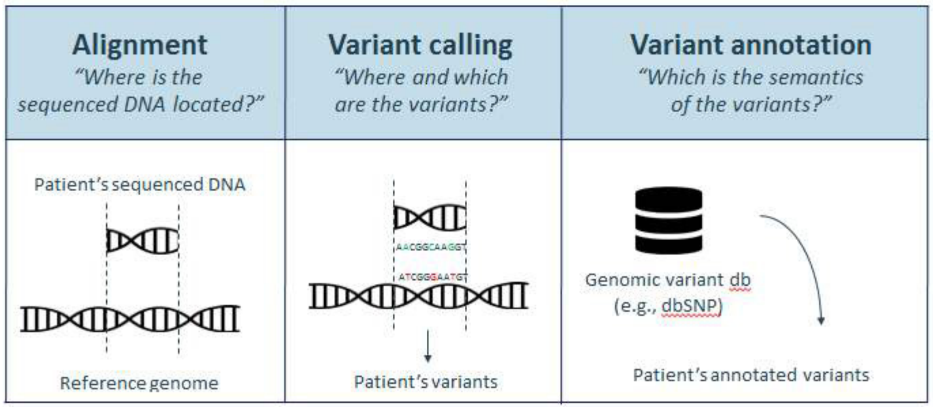

Alignment
"Where is the sequenced DNA located?"
Variant calling
"Where and which are the variants?"
Variant annotation
"Which is the semantics of the variants?"
Patient's sequenced DNA
Reference genome
AACGGCAAGG
ATCGGGAATG
Patient's variants
Genomic variant db (e.g., dbSNP)
Patient's annotated variants

Fig. 3.4 Three crucial workflows in the variant analysis for genome sequencing

Sequence alignment is the process of mapping short read to reference genome, which indicates where the sequenced DNA is located. Variant Calling or variant identification, is the workflow which identifies variants from DNA sequence data, which indicates where and which the variants are. Variations in the DNA sequence are responsible for genetic disorder and human diseases. Variant annotation is the process to predict the functional impact of variants by using the annotational applications, which indicates which the semantics of the variants is. It now becomes increasingly important with the enormous amount of data produced by NGS platform are needed to handle.

3.2.2 Short Read Alignment

This thesis concentrates on sequence alignment. Sequence alignment can be divided into two groups, one is multiple sequence alignment and the other is pairwise sequence alignment. Multiple sequence alignment is generalized to the multiple sequences by seeking an alignment that maximizes the sum of similarities for all pairs of sequences [49]. It is useful for protein structure and function prediction, phylogeny inference and other common tasks in sequence analysis. Pairwise sequence alignment concentrates on comparing only two sequences, one sequence denoted as short read and one reference genome.

This thesis focuses on pairwise sequence alignment which involves finding the optimal alignment between two sequences. Making a score based on the similarity or difference, then to determine the optimal positions. One of the most important things in pairwise sequence alignment algorithm is the scoring process, which produces a matrix to track the scores assigned to each positions. Normally, a positive score and a penalty will be assigned for a match and a mismatch, respectively. Three basic aspects are considered when assigning scores [50], as shown in Figure 3.5:

- Match value - Value assigned for matching characters
- Mismatch value - Value assigned for mismatching or different characters
- Gap penalty - Value assigned for spaces

Each pairwise sequence alignment algorithm has its own standard of evaluation of these three basic aspects. Needleman-Wunsch algorithm [51] is the first aligner in 1970 which utilizes dynamic programming to obtain global alignment between two sequences. In 1981, Smith-Waterman algorithm [52] tries to achieve local alignment between two sequences

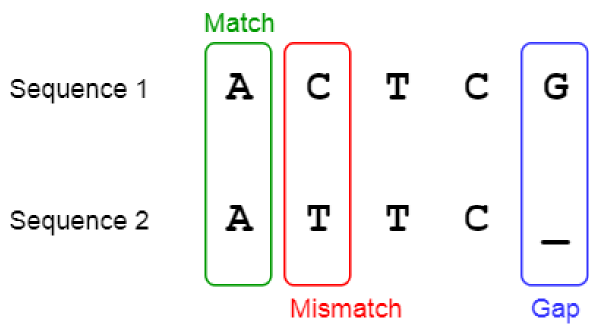


Fig. 3.5 Example of pairwise sequence alignment

under the effects of Needleman-Wunsch algorithm. This two algorithms are very similar but only slight differences in the process of scoring. This two algorithms represent two types in the pairwise sequence alignment, global alignment and local alignment [53].

- Global alignment: obtain the best optimal alignment over the whole length of two given sequences.
- Local alignment: obtain the best optimal alignment in sub-sequences between two given sequences.

As shown in figure 3.6, aligning genome ’ATGAGCTGAATTGCTG’ with short read ’TAGCTATGCGC’ in global and local type in alignment. The results illustrate they focus on different length of sequence according to their definition in algorithm.

3.2.3 Existing Aligner

Over decade years in the development of NGS, many applications have been developed on sequence alignment stage, such as BLAST, Bowtie, BWA, MAQ, SOAP and YOABS. Read aligners need to construct an index for short reads or for genome reference or for both. Depending on the index construction, read aligners can be classified into three groups [30]: hash table-based, FM-index based and merge sorting based. The former two groups

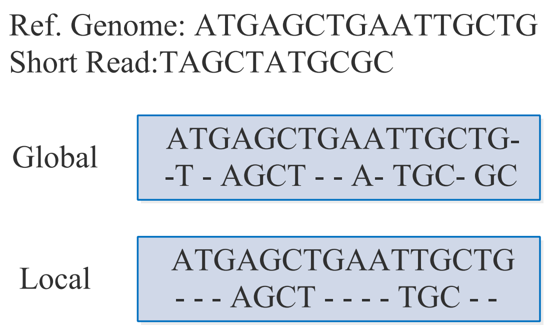


Fig. 3.6 Two types of pairwise sequence alignment

contain the majority of aligners used by the scientific community. BLAST, MAQ, SOAP and ZOOM are examples of hash table-based aligners. Their main drawback is the large memory consumption required for the index. On the contrary, the memory requirements of aligners based on the FM-index mechanism are significantly lower, which explains their increasing popularity.

Sequence aligners that are based on hash table group and FM-index group are as follows:

Hash table-based aligners

- MAQ [54] is an aligner that builds index dataset against short read through using quality scores for alignment.
- SOAP [55] is an application that permits gapped and ungapped alignment by utilizing the new generation Illumina-Solexa sequencing technology.
- BLAST [56] support a parallel version that permits Popular approaches to parallelize BLAST include query distribution, hash table segmentation, computation parallelization, and database segmentation.

- mrsFAST [57] is Micro-read substitution-only Fast Alignment Search Tool. mrsFAST is a cache-oblivious short read mapper that optimizes cache usage to get higher performance.
- ZOOM [58] enables insertion and deletion type errors and uses confidence score information and pair-end sequencing data to enhance the mapping accuracy. This software exploits extended spaced seeds technology for the mapping efficiency and accuracy.
- SHRiMP [59] is a software package for mapping reads from a donor genome against a reference genome, which allows q-gram filtering technique and the standard dynamic programming (Smith-Waterman) string matching algorithm.
- ABySS [60][61] supports parallelized sequence assembler to overcome inability to mapping large amounts of datasets from large-scale next generation sequence platform.

FM-index based aligners

- BWA [62] is a widespread sequence alignment application over the world in terms of accuracy and efficiency. BWA performs gapped alignment based on inexact matching with FM index for mapping low-divergent sequences against a large reference genome, such as the human genome. BWA consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM.
- Bowtie [63][64] are proposed towards ungapped and shorter short read(no more than 50bp) alignment in version 1(Bowtie) and gapped and longer short read alignment in version 2(Bowtie2).
- SOAP2 [65] is developed from SOAP, but in version 2 it uses a combination of the FM-index and hashing to increase the speed. According to the results from authors, significantly more memory is consumed in SOAP2 than BWA and Bowtie.

- Segemehl [66] uses a dynamic programming method called the Meyers bit-vector algorithm to find gaps and mismatches which leads to significantly more memory than BWA and Bowtie.

Application BWA, implemented by Heng Li [12], is one of the most prevalent and widespread sequence alignment tool and it has a fast processing speed with low memory footprint because BWA utilizes FM-index instead of hash table and suffix index which acquires for huge memory on constructing index for a genome. In order to efficiently align short sequencing reads against a large reference sequence, it permits mismatches and gaps. Li's experiments indicate that BWA is approximately vary from 10 to 20 times faster than MAQ. BWA consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. This paper focus on the first algorithm which is designed for Illumina sequence reads up to 100bp. Our study focuses on BWA-backtrack that has already been used by other works in the literature, such as mBWA [11], pBWA [14] and BMIC [67]. Thus, we will try to provide a view as complete as possible of the performance that each of the different strategies can obtain when applied to the same basic application.

3.3 Burrows-Wheeler Aligner: BWA

3.3.1 Introduction

Many applications have been developed for solving the problem of sequence alignment (such as BLAST, Bowtie, BWA, MAQ, SOAP and YOABS). Read aligners need to construct an index for short reads or for genome reference or for both. Depending on the index construction, read aligners can be classified into three groups [30]: hash table-based, FM-index based and merge sorting based. The former two groups contain the majority of aligners used by the scientific community. Most of them are based either on hash table or suffix index technologies. These technologies exhibit a large consumption of computer memory while BWT-based algorithms (FM-index technology) exhibit a small memory footprint. BWA aligner provides a better solution for genome sequence analysis. Although such as Bowtie, SOAP and Blast they could also support BWT-based technique by using different algorithm for mismatches and gaps, they are evaluated to produce faster albeit less accurate results than BWA [19].

BWA, implemented by Heng Li [8], is one of the most prevalent and widespread sequence alignment tool belonging to the BWT family. It has a fast processing speed with low memory footprint, because BWA is based on BWT technique to store and perform searches on the reference genome. The index of reference genome is created via Burrows-Wheeler Transform (BWT) and processed in chunks of a fixed size using a round-robin pattern. In order to efficiently align short sequencing reads against a large reference sequence, it permits mismatches and gaps. BWA consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. The first algorithm is designed for Illumina sequence reads up to 100bp, while the rest two for longer sequences ranged from 70bp to 1Mbp. BWA-MEM and BWA-SW share similar features such as long-read support and split alignment, but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads [21].

This thesis focuses on the first algorithm, BWA-backtrack, which is designed for Illumina sequence reads up to 100bp. In this thesis, we have used a basic implementation of BWA, BWA 0.5.10 ALN module, and supports execution in native mode and offload mode, as well as symmetric mode on the architectures of Intel Xeon processors and Xeon Phi coprocessors. Although backtrack algorithm in BWA 0.5.10 is not the newest version of BWA which Li proposed, but many applications like pBWA, mBWA, BMIC in genome sequence still support this version and this algorithm because they lack of update. Thus, this can make us to provide a view as complete as possible of the performance that each of the different strategies can obtain when applied to the same basic application.

Figure 3.7 shows two fundamental flows of BWA-backtrack module. At the first step we need to building the BWT index against the reference genome by using command "bwa

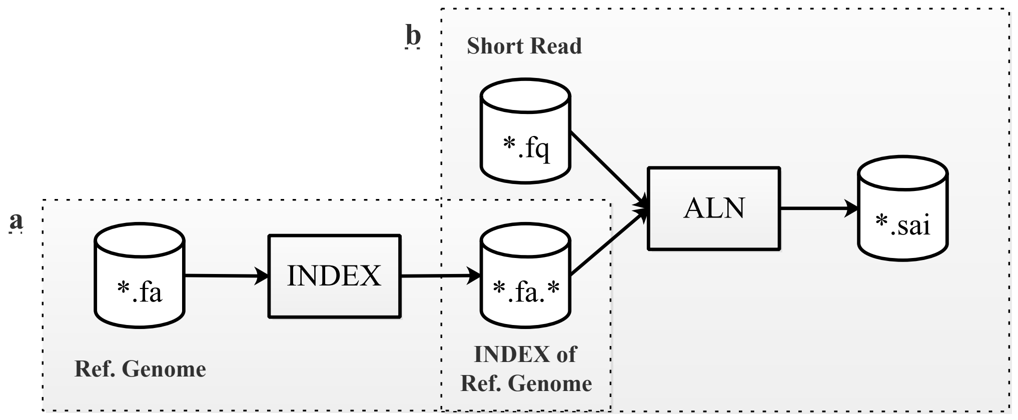


Fig. 3.7 Two flows of BWA-backtrack. a. Building the BWT index against the reference genome. b. Alignment of short read against the reference genome using BWA-backtrack module

index". Afterwards, mapping short read to the reference genome using command "bwa aln" in BWA aligner. More details are shown in the following [68].

1. Building the index for the genome reference.

   The user first needs to build the BWT transformation of the reference genome using the BWA index tool. BWA index takes as input a *.fa file and produces several *.fa.* files to store the BWT transformation.

2. Mapping short read to genome reference.

   The BWA program uses the reference genome index, BWT transformation(*.fa.* file), from step 1 for the actual alignment process against short read file *.fq. Final results of alignment would merge to *.sai file in the end.

3.3.2 Performance Analysis

Nowadays, a large amount of sequence aligners enable multi-threading ability when processing the biological datasets. However, the addition of multi-threading to an aligner does not guarantee that processing resources are used efficiently. In fact, it is quite common to observe that efficiency decreases when the thread count grows large enough [16].

This phenomenon is shown in Figure 3.8, 3.9 and 3.10. These figures illustrate the results obtained by three aligners (BWA, SOAP2 and BOWTIE2) when mapping two large short read genome examples of 17.6GB and 4.3GB against human genome reference. Each figure illustrates the results obtained when aligners are executed on two different servers. Both servers are based on NUMA architecture but they differ in the total number of processors, cores and short read data size available. The first system (Fig.3.8) owns 2-socket 2-NUMA Intel Xeon CPU with 12 cores (Sandman). Twenty-four threads can be executed simultaneously using Hyper Thread technology. The second system (Fig.3.9 and 3.10) is a 4-socket 4-NUMA Intel Xeon CPU (Penguin). Each socket contains 8-core processor, so the total number of cores is 32 and 64 threads can be executed simultaneously. In both cases, each socket is connected to a memory bank, which constitutes a NUMA node. NUMA nodes are connected by QuickPath Interconnect link.

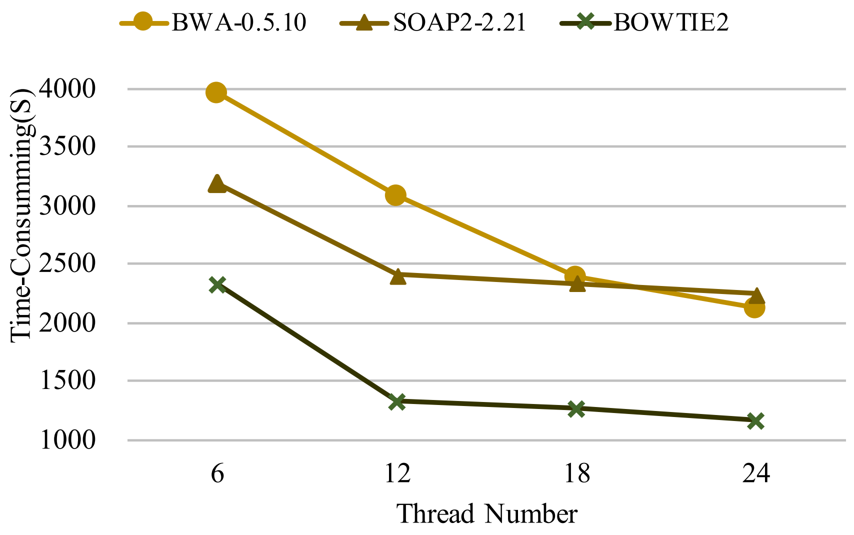


Fig. 3.8 Scalability on 2-socket 2-NUMA 24-thread system, genome Data 17.6GB

This behavior of performance degradation could be easily observed in Figure 3.8. A continuous decreasing of execution time is illustrated while the gain in performance reduces with more threads are utilized in the experiments. When 12 and more threads are implemented

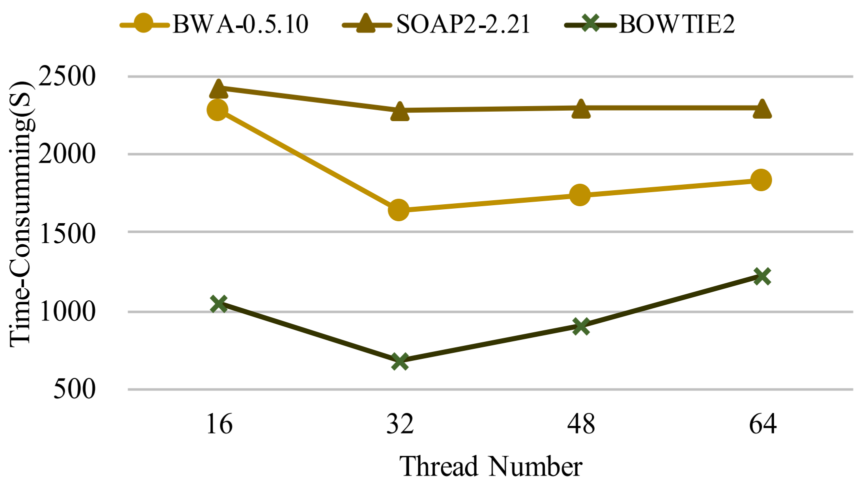


Fig. 3.9 Scalability on 4-socket 4-NUMA 64-thread system, genome Data 17.6GB

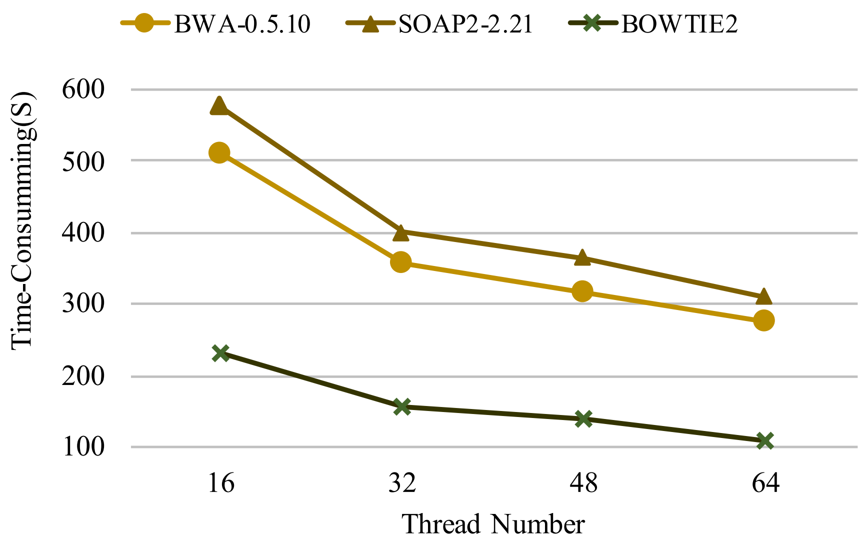


Fig. 3.10 Scalability on 4-socket 4-NUMA 64-thread system, genome Data 4.3GB

in this two NUMA nodes system, a considerable of performance commences to lose since the threads would be allocated on both NUMA nodes. The memory bottleneck becomes the crucial issue on the performance although a potential increasing could be achieved due to the extra threads applied. As shown in Figure 3.9, when we tried a large NUMA system, 4-NUMA nodes, the time-consuming of the aligners is degraded and even adding more threads has a negative impact when 32 and more threads are utilized. This abnormal phenomena in performance could be explained as follows,

1. the growing of remote access in memory,
2. the memory congestion where the genome reference is allocated.

When BWA aligner runs in these systems, threads are spread throughout the system and memory is allocated using a first-touch data allocation policy, which means that, when a program is started on a CPU, data requested by that program will be stored on a memory bank corresponding to its local CPU. In particular, this behavior affects the allocation of the reference index genome in the case of BWA. Threads that run on the same NUMA node where the reference index is allocated have a lower latency when they access it.

Therefore, when the number of threads increases, all the speed up gained due to multi-threading is mitigated by the latency of remote accesses and traffic saturation of interconnection links. This degradation of performance can be observed in Figures 3.8 and 3.10 which shows a continuous reduction of execution times, but the gain in performance reduces as more threads are added in the execution. Being a system with only two NUMA nodes in Figure 3.8, performance degradation starts to be significant when more than 12 threads are used and threads are allocated on both NUMA nodes. Especially, when the application goes from 18 to 24 threads, potential gains due to the extra threads. However, the extra threads have a small impact in execution time due to the memory latency issues on both NUMA nodes, which is the crucial factor that affects performance.

When the thread count grows large enough, problems with memory accesses are getting worse. As Figure 3.8 and 3.9 illustrate, BWA runs in a large NUMA system, and when more than 32 threads are used, execution times are degraded and adding more threads has a negative impact in overall performance. This loss in performance can be explained, on the one hand, by the increased latency of remote memory accesses and, on the other hand, by the memory bank in which the reference genome is allocated becoming a bottleneck that generates memory contention.

Figure 3.8 and 3.9 illustrate the scalability of threads when a large short read dataset (17.6GB) maps to human genome sequence while Figure 3.10 displays a smaller short read (4.3GB segment) is executed in 4-NUMA 64-thread system. In this smaller dataset case, it performs an identical trend that more threads applied take less time-cost. The same behavior occurs in Intel Xeon Phi architecture. Exploring more efficient data parallelism is necessary, special under a limited system.

Additional, BWA creates the index of genome reference beforehand. Later, short read data is processed using fixed size batches, in a round-robin pattern among threads. Each thread in BWA processes one batch of data, 256K by default, that is independently mapped to the genome reference. 256K is our default setting in the experiments according to the results we observed that 256K could reach a balanced performance.

We illustrate an experiment results about an increasing batch size as displayed in figures 3.11 and 3.12. The former was implemented in Sandman (Xeon) with 24 threads, the latter in Sandman-Xeon Phi with 240 threads. The batch size - the number of sequence reads each thread loads into memory to be processed - we change batch size in different value from 64K, 128K, 256K, 512K, 1M and 2M, and implement in different architectures. The default batch size, 256K, achieves a balanced performance in terms of time-consuming as displayed in figures. And other cases in both systems achieve very close time-cost or more.

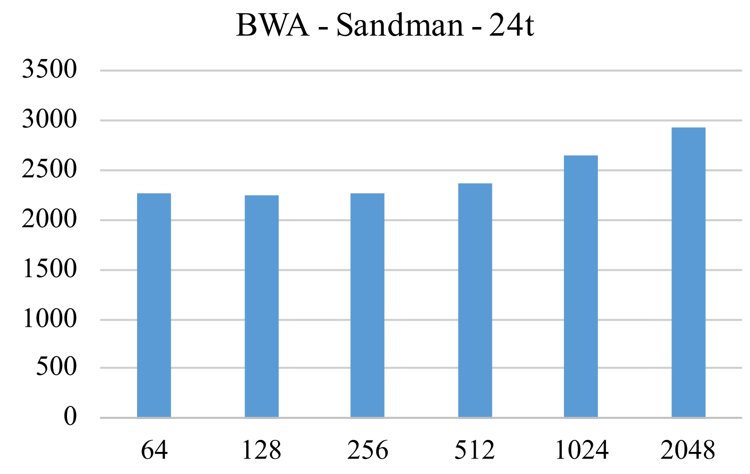


Fig. 3.11 The increasing batch size in Sandman with 24 threads

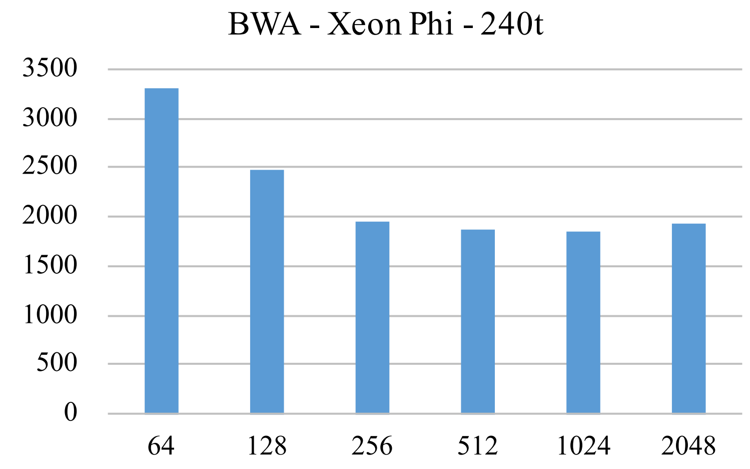


Fig. 3.12 The increasing batch size in Sandman-Xeon Phi with 240 threads

3.3.3 Conclusion

Genome read aligners provide the relative position of short reads within a reference genome. Despite the particular differences of each aligner, they all share a similar mode of operation that can be summarized as follows:

1. There is a set of reads that can be mapped to the reference genome independently.
2. There is a reference genome data structure (genome index) that is read-only data and is used to map each individual read.
3. The results consist in populating a shared data structure that would be written on an output file at the end of the alignment.

We focus our study on BWA, written by Heng Li [12], which is one of the most popular sequence alignment tools from the FM-index based family. BWA consists of three algorithms, BWA-backtrack, BWA-SW and BWA-MEM. Our study focuses on BWA-backtrack that has already been used by other works in the literature, such as mBWA [11], pBWA [14] and BMIC [67]. Thus, we will try to provide a view as complete as possible of the performance that each of the different strategies can obtain when applied to the same basic application.

BWA creates the index of genome reference beforehand. Later, short read data is processed using fixed size batches, in a round-robin pattern among threads. Each thread in BWA processes one batch of data, 256K by default, that is independently mapped to the genome reference. 256K is our default setting in the experiments according to the results we observed that 256K could reach a balanced performance.

Like many other aligners, BWA has multi-threading capability (using the Pthread library) to solve read mapping operations in parallel since there is no data dependency in this action. The addition of multi-threading to an aligner does not guarantee that processing resources are used efficiently. In fact, it is quite common to observe that efficiency decreases when the thread count grows large enough [16].

The library and the organization used to handle application threads can cause certain variations in the execution times obtained. In paper [17], it was observed that the parallelization using Pthread, OpenMP and Cilk presented noticeable differences in performance, with OpenMP and Cilk being the best alternatives to Pthread, which was the library originally used in BWA. Besides the thread library, the main performance problems in genome read aligners arise from the use of memory in NUMA systems. Memory allocation is done according to a first-touch policy and this means that the genome reference index is allocated in a single bank, that will become a bottleneck when multiple threads try to access it [15].

3.4 Variant Calling

Variations in the DNA sequence are responsible for genetic disorder and human diseases. The changes in genes is the change in the order of nucleotides in the DNA sequence of a person. Variant Calling or variant identification, is the workflow which identifies variants from DNA sequence data [69].

As shown in Figure 3.13, a CRAM file aligned to a reference genomical region as visualized in Ensembl [70]. Differences are highlighted in red in the reads, and will be called as variants.

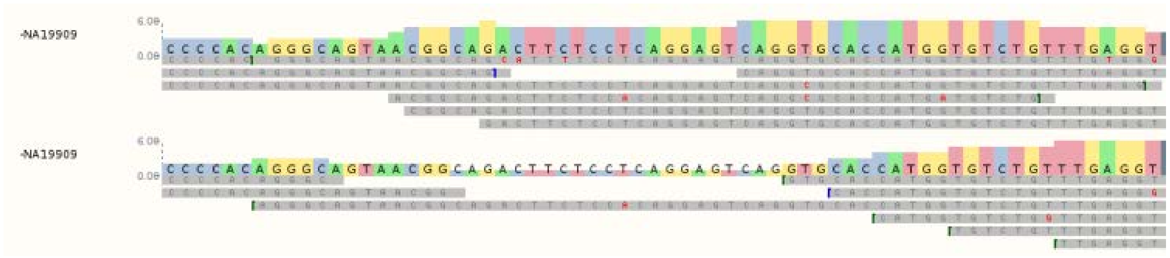

NA19909
6.00
0.00
CCCCACAGGGCAGTAACGGCAGACTTCTCCTCAGGAGTCAGGTGCACCATGGTGTCTGTTTGAGGT
NA19909
6.00
0.00
CCCCACAGGGCAGTAACGGCAGACTTCTCCTCAGGAGTCAGGTGCACCATGGTGTCTGTTTGAGGT

Fig. 3.13 The example of variant calling or variant identification

There are many callers developed to solve this problem now. GATK [71], SAMtools [72], CRISP [73], SV identification- BreakDancer [74] and so on. Although these callers have provided many efficient methods or algorithms when facing this problem, all they follow a three step process:

- Creating FASTQ files after carrying out whole genome sequencing - this is quality assessment
- Aligning the sequence results of FASTQ files to a reference genome - this is sequence alignment
- Identify where the aligned reads differ from the reference genome and write to a VCF file

These three steps above indicts a fact that sequence alignment owns a crucial role in the variant analysis workflows.

3.5 Variant Annotation

Variant annotation is the process to predict the functional impact of variants by using the annotation applications [75]. It now becomes increasingly important with the enormous amount of data produced by NGS platform are needed to handle. For variant annotation in genome sequence, based on different feature used by the annotation tool, the variant annotation can be classified into three crucial category [76].

- Gene based annotation
- Knowledge based annotation
- Functional annotation

Many variant annotation tools are developed for solving this challenge. Some of the available SNPs annotation tools are as follows SNPeff [77], ANNOVAR [78], FATHMM [79], PhD-SNP [80], AnnTools [81], MutationTaster [82], SNPdat [83] and FAST-SNP [84].

Chapter 4

Heterogeneous Multicore-Manycore Architecture

Nowadays, systems based on NUMA (Non Uniform Memory Access) architectures are the most used in the field of high performance computing. These NUMA systems are often used in conjunction with manycore accelerators that increase the number of threads available in the system at a reasonable cost. In this section we provide a brief description of the basic architectures that we have used in our study, namely, multicore systems with NUMA nodes and manycore accelerators. Afterwards, we also tried some experiments about thread effect that has a crucial behavior in overall performance.

4.1 Multicore System: AMD Opteron and Intel Xeon

4.1.1 Architecture

Figure 4.1 shows the main architectures of the systems of two multicore systems. Two multicore systems comprise one AMD Opteron based server and one Intel Xeon based server. Our AMD Opteron and Intel Xeon systems have NUMA nodes with different link level across processors (net level vs ring level). Additional, Intel Xeon and Xeon Phi comprise another heterogeneous system in the experiments.

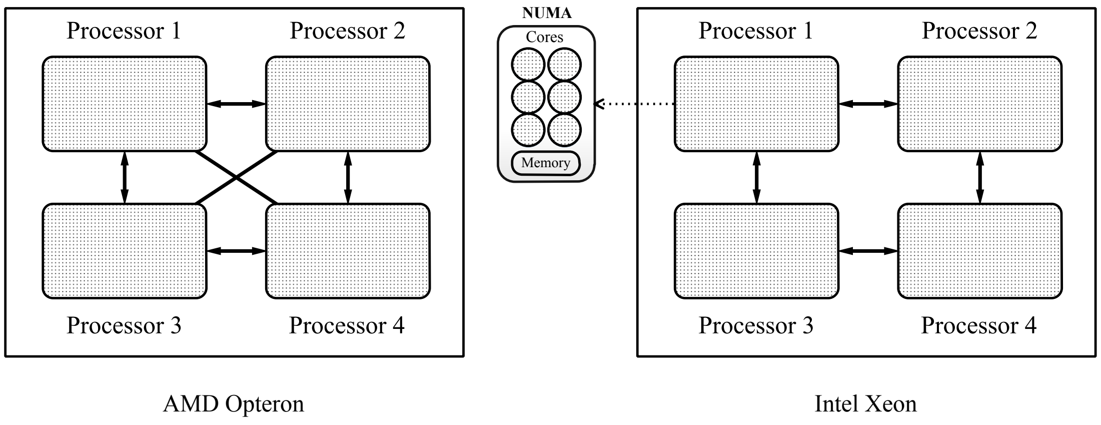


Fig. 4.1 Multicore architecture: AMD Opteron and Intel Xeon

Table 4.1 AMD Opteron processor distance

Processor	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

Tables 4.1 and 4.2 demonstrate distance matrix of 4 sockets system of AMD Opteron and Intel Xeon as shown in Figure 4.1. AMD Opteron has 4 nodes fully connected with each other which leads to an longest diameter of 1 hop, while Intel Xeon owns 4 nodes square topology which implies one more hop in some nodes, like 2 hops in nodes 2 and 3, nodes 1 and 4. More hops means extra latency in network connection and less slow in bandwidth. This phenomenon indicates that memory policy plays an important element in overall performance when multi-thread or multi-instance implementation.

Table 4.2 Intel Xeon processor distance

Processor	1	2	3	4
1	0	1	1	2
2	1	0	2	1
3	1	2	0	1
4	2	1	1	0

4.1.2 NUMA effect

NUMA(Non-Uniform Memory Access) is more and more common because performance improvement is achieved when memory controllers get closer to execution units on processors. Since 2000 and now, Linux OS provides NUMA support and optimizes typical HPC applications in a decent performance without much complex modification. In multi-socket system, NUMA nodes are commonly found in memory level. For example, a typical server owns two sockets and two NUMA nodes is widespread nowadays, see Figure 4.2. Latency for a memory access (random access) is about 100ns while access to memory on a remote node adds another 50 percent to that number [85].

As shown in Figure 4.2, two processors and two NUMA nodes are interconnect directly to the socket that they are on. A memory access from one core to local memory bank (local access) is less latency overhead than a memory access from one core to another memory bank(remote access). If latency for local access costs about 100ns so that it will takes about 150ns in remote access [85]. Another issue happens when the traversal of the memory interconnect in remote access, it causes memory contention on the interconnect and memory controllers, specially when a large number of data operations occurs. Thus avoiding remote accesses could decrease the memory latency and increase the overall performance of application.

Memory policy determines the allocation of memory in NUMA. There are many kinds of memory policies that are designed for different situations, but only two policies, local and interleave memory policies, are the most common in operating system in general.

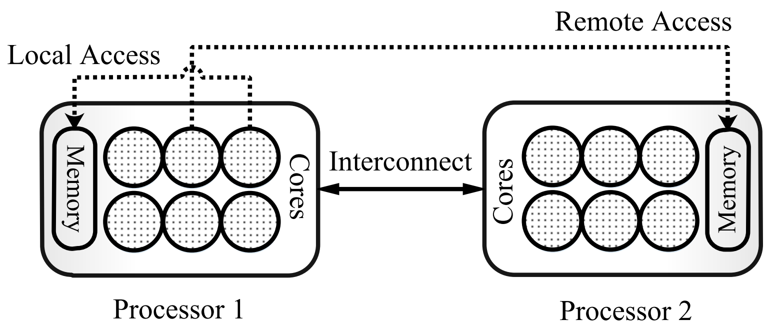


Fig. 4.2 Memory access in NUMA nodes

Two crucial memory policies are list in the following:

- LOCAL

  The allocation of memory only occurs in the memory bank of local node where the program is currently initial. Local memory policy is the default allocation policy when the system is running.

- INTERLEAVE

  The allocation of memory is used to distribute memory in a round-robin pattern among multiple processors in the system for the purpose of an even access loading in the interconnect and the memory banks. Round-robin pattern means a page will be firstly allocated in processor 1, then in processor 2, then in processor 1 again, etc.

Figure 4.3 displays a parallel summation of an array on a 2-socket 2-NUMA node with different memory policies [86].

As shown in figure 4.3.a, the array is allocated on a single socket (local policy) but owns accesses coming from the threads on both sockets. This behavior is commonly observed in the experiments, because the first-touch policy in default is applied. The large local access leads to the sockets' memory bandwidth becomes a crucial bottleneck.

If the array is allocated on both sockets (interleave policy) across the machine's sockets, as shown in figure 4.3.b, the remote access between inter-sockets consumes the main bandwidth

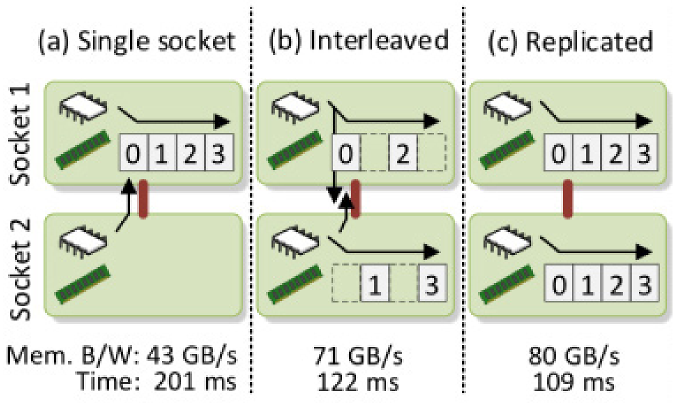


Fig. 4.3 Parallel summation of an array on a 2-socket 2-NUMA node with different memory policies

in the interconnect. In this case, both sockets' memory bandwidth are utilized to decrease the execution time and improve the overall performance.

If the array is replicated on both sockets, see figure 4.3.c, although more memory space is consumed, we remove the congestion in the interconnect as a bottleneck that has a large impact on performance, and localize memory access within local socket.

There is one characteristic of memory policy that should refer in the paper. Memory policy only in effect when the first allocation occurs. This is so-called the first touch. First touch indicates where a page is distributed according to one memory policy when one process first gets a page in the system. This behavior leads to a factor that memory could be allocated on NUMA nodes where is not allowed by the memory policy in the system because an earlier process has allocated the data into memory.

In this thesis, AMD-Opteron and Intel Xeon systems have both a NUMA architecture with four independent sockets, and each socket has a one or two multicore processors and each processor is attached to a local memory bank. Each processor and its corresponding memory bank is referred as a NUMA node. One of the most significant of characteristics of NUMA systems is that accesses from threads running in local cores (located in the same NUMA node) have lower latency than the accesses that go to physical memory located in different NUMA nodes (as shown in Figure 4.2). This asymmetry in memory accesses involves additional complexity when executing parallel applications because access to remote banks increases the execution time and, in the case of genome alignment, also introduces congestion problems.

4.2 Manycore System: Intel Xeon Phi

4.2.1 Architecture

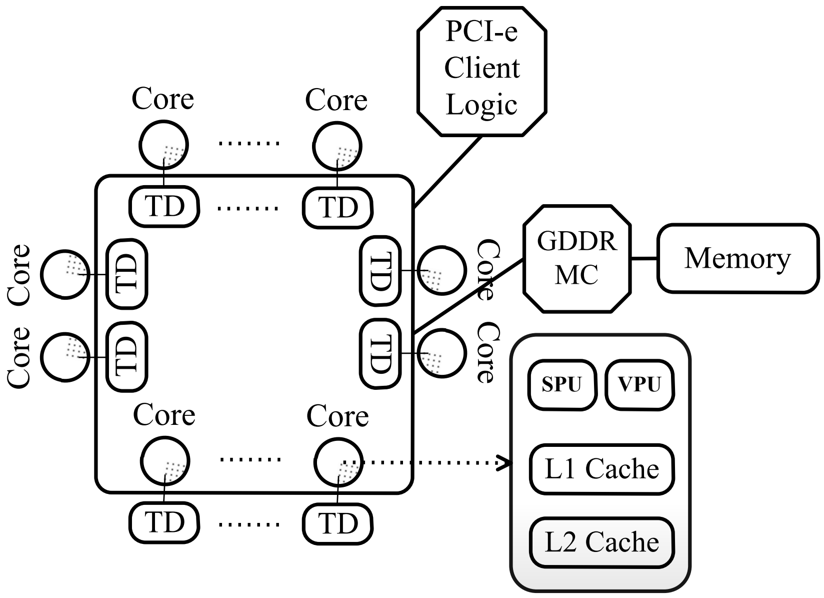


Fig. 4.4 Intel Xeon Phi coprocessor chip architecture

As shown in Figure 4.4, a simple diagram of the logical layout of some of the critical chip components of the Intel Xeon Phi coprocessor architecture. Intel Xeon Phi accelerator has a bi-ring architecture with 57 to 61 cores inside. Each core in the Xeon Phi has a SPU, a VPU with 512-bit SIMD capability, and a private 512KB L2 cache that is maintained fully coherent by a global-distributed tag directory (TD) [87], which forms a unified shared L2 cache of 30.5MB. Xeon Phi can achieve 1 TFlops in double precision or 2 TFlops in single precision. In addition, memory bandwidth can reach 352GB per second theoretically thanks to 16 memory channels. The memory controllers (GDDR MC) and the PCIe Client Logic provide a direct interface to the GDDR5 memory and the PCI express bus, respectively.

Main components of the Intel Xeon Phi coprocessor architecture is listed in the following:

- CORE (Coprocessor Cores)

  based on the identical core architecture of x86 with Intel Xeon under some crucial modifications. Each core can support 4 hardware threads and run at the speed of 1GHz, so that Xeon Phi can keep 1 TFlops in double precision or 2 TFlops in single precision.

- TD (Tag Directories)

  maintain all L2 caches and provide a global distributed and shared cache of 30.5MB among the cores.

- RING (Ring Interconnect)

  the interconnect among all components of the Intel Xeon Phi which could provide a bi-direction ring access support.

- GDDR MC (GDDR Memory Controller)

  Interface between the ring and GDDR memory which supports 8 memory controllers with 2 GDDR5 channels running at 5.5 GT/s, that can compute the theoretical memory bandwidth as 352 GB/s.

- PCIe interface (Peripheral Component Interconnect Express)

  support a standard input/output (I/O) protocol such as PCIe to communicate with the host Intel Xeon.

Within the Intel Xeon Phi's core:

- VPU (Vector Processing Unit)

  support 512-bit vector operations on the floating-point(16 single-precision or 8 double-precision) arithmetic operations as well as integer operations.

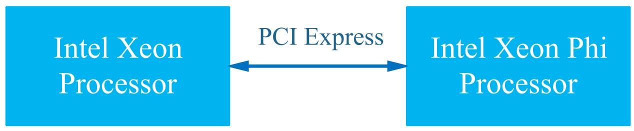


Fig. 4.5 Intel Xeon processor and Intel Xeon Phi coprocessor system

- SPU (Scalar Processing Unit)

  an in-order architecture based on the Intel Pentium processor family.

- L1 Cache

  32 KB L1 instruction cache and 32 KB L1 data cache.

- L2 Cache

  provide 512KB size which are maintained by TD.

Intel Xeon Phi is an convenient coprocessor designed in x86 architecture which enables many applications could be easily transform to Intel Xeon Phi platform without much complicated coding. However, in order to pursuit higher improvement of performance that fits the hardware architecture, the applications still should be highly parallel offload, efficiently vectorizable and overlapping the operations of I/O [88]. Thus, Intel Xeon Phi is commonly used as a coprocessor with Intel Xeon or other x86-based architecture system, as shown in figure 4.5, they are connected by PCI express in high band. More details about this combined architecture we discuss in Chapter 6.2.

4.2.2 Basic Execution Mode

The Intel Xeon Phi has a x86 architecture that allows for three execution modes, as shown in figure 4.6. An application is implemented on a server that has a Xeon Phi accelerator can be executed in three different ways [89]:

- Native mode

  the application is executed on each component (processor or accelerator) independently.

- Symmetric mode

  the accelerator is seen as a regular node in the system and the application is executed on all components (main processor and an accelerator).

- Offload mode

  the application is executed on the main processor but selected highly parallel sections pass to the accelerator.

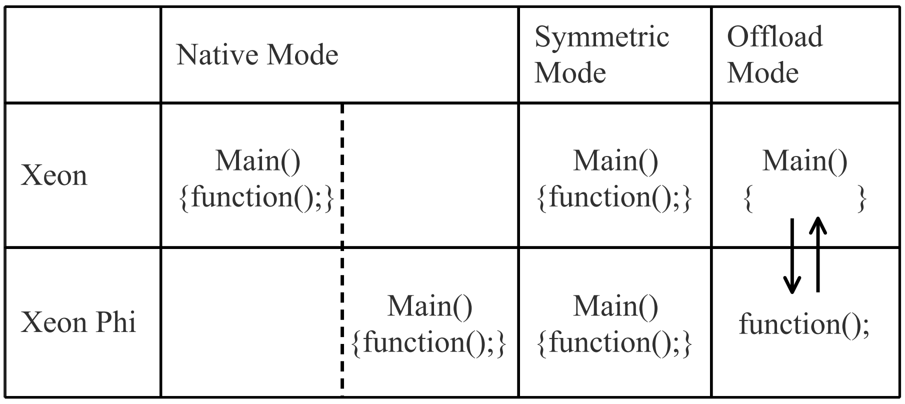


Fig. 4.6 Three basic execution modes on the heterogeneous system of Intel Xeon and Intel Xeon Phi

Load balance between Xeon and Xeon Phi on the symmetric mode is rather significant and their time-consuming should be identical in order to approach their best performance. In this thesis, we would use different modes to distribute data load between two systems. One common fashion is static mode which means we settle up load balance by optimizing the workload based on the running time of their native mode execution. For example, if the Intel Xeon Phi coprocessor's performance on native mode is 0.7-fold that of the same problem running on the Intel Xeon processor on native mode, then the ratio of workload on symmetric mode is 10 for the Intel Xeon processor, and 7 for the Intel Xeon Phi coprocessor. We also tried other two dynamic distribution - even distribution and round-robin distribution - to illustrate a comprehensive comparison in the experiments.

4.3 Thread Parallelism Library

Parallel programming techniques such as MPI and OpenMP are utilized to boost the performance of applications in a distributed or share computing environment while threads are limited to a single computer system. All threads in a process need to share the identical memory address in the system. The purpose of using the thread parallelism library is to accelerate program faster than ever.

Many works [90][91] has reported that thread parallelism plays a significant part in high parallel computing applications. Many programs enable multi-threading ability to enhance parallelism in solving read mapping operations since there is no data dependency in variant analysis workflows.

Pthread and OpenMP thread libraries are these two kinds of the most popular and widespread in HPC scientific arena. However, Pthread and OpenMP perform some disadvantages and bottlenecks in some special hardware architecture. Over years pass years, there are many other kinds of thread libraries to provide options for researchers. MassiveThreads [92], Qthreads [93], Hood [94], Intel Cilk [95] and so on, they are designed form various perspectives and destinations. In this thesis, we would demonstrate not only Pthread and OpenMP thread libraries, but also the Cilk library proposed by Intel Corporation.

4.3.1 Pthread

POSIX thread (Pthread) supports multi-threading capability for many mainstream systems, such as Linux, Windows, Mac OS, Android, FreeBSD, Solaris and etc. Pthread is a parallel execution model that is independent from a language, which allows one program to manage multiple concurrent jobs at the same time. Each concurrent job is regarded as a pthread, its creation and release is achieved by making calls to pthread. Thus master pthread can spawn a concurrent flow of works that contains multiple pthreads. This so we called fork-join parallel flow [96], as shown in figure 4.7, this is typical flow in fork-join parallelization. A single process can contain multiple tasks, a single task can execute multiple threads, all of which are executing the identical program. These threads share the same global memory, so that Pthread is designed for multi-processor or multi-core that shared memory machines. This underlying architecture can be shared memory architecture of UMA and NUMA.

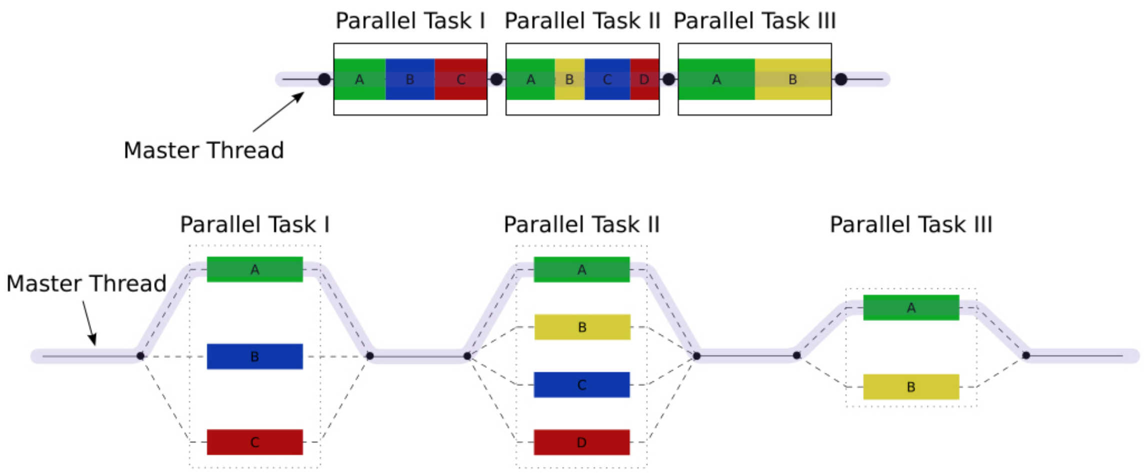


Fig. 4.7 The fork-join parallel flow

Pthread has disadvantages we already observed in the experiments, many resource and time-consuming were taken in the management of thread group. BWA has multi-threading capability (using the Pthread library) to solve read mapping operations in parallel since there is no data dependency in these actions. The addition of multi-threading to an aligner does not guarantee that processing resources are used efficiently. In fact, it is quite common to observe that efficiency decreases when the thread count grows large enough. In the BWA aligner experiments, we observed each pthread would be released after processing one batch size from short read. This creation and release of pthread occupy large time and resource.

4.3.2 OpenMP

The nonprofit technology consortium OpenMP Architecture Review Board (OpenMP ARB) maintains OpenMP project which has attracted a large attention over the world. OpenMP ARB is a joint group of major group of major computer hardware and software corporations, including AMD, IBM, Intel, Cray, HP, Fujitsu, Nvidia, NEC, Red Hat, Texas Instruments, Oracle Corporation, and more.

OpenMP is an abbreviation of Open Multi-Processing which provides an application programming interface that supports multi-processor multi-core shared memory programming on most platforms and operating systems, such as Solaris, AIX, HP-UX, Linux, macOS, and Windows. OpenMP permits a simpler and more flexible interface for programmers than Pthread in developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer.

OpenMP supports fork-join parallel flow as shown in figure 4.7 that makes parallel execution most effective on multi-processor or multi-core systems where the process flow can be scheduled to run on another processor, thus gaining speed through parallel or distributed processing.

OpenMP thread does not release itself when finishing one batch size of short read, but continue to process next batch size data and another, according to the results of our experiments in BWA algner. Less time-waiting and time-processing were observed in the OpenMP thread. We would apply this OpenMP technique in our strategy of data management in the experiments.

4.3.3 Intel Cilk

Originally developed in the 1990s at the Massachusetts Institute of Technology (MIT) in the group of Charles E. Leiserson [95], Cilk was later commercialized as Cilk++ by a spin off company, Cilk Arts. That company was subsequently acquired by Intel, which increased compatibility with existing C and C++ code, calling the result Cilk Plus [97].

Intel Cilk is a simple extension based on C and C++ language that supports constructs for parallel control and synchronization. Intel Cilk enables software in parallelism that could explore both the thread and the vector capability commonly available in HPC hardware.

Observing the results from the typical cost of spawning a parallel thread only takes between 2 and 6 times comparing to the cost of of a C or C++ function call on a variety of contemporary machines. Cilk owns a low overhead among works because the work-first principle is implemented.

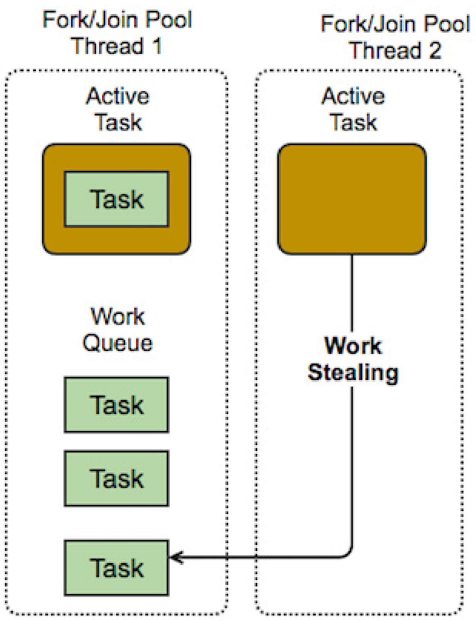


Fig. 4.8 An example of work-stealing mechanism

The work-first principle is one important part of the runtime load-balancing scheduler inspired by Dijkstra-like [98] that is designed for shared-memory system. The work-first principle is to minimize the scheduling overhead borne by the work of a computation. Specifically, move overheads out of the work and onto the critical path [95]. The critical-path is the maximum number of instructions on any directed path in the DAG, and the computation of the critical-path corresponds to the amount of time required by an infinite-processor execution. The work-first principle concentrates on the significance of minimizing the work overhead in scheduling and plays a rather crucial position in Cilk parallel system.

Another factor Intel Cilk uses for reaching a better load-balancing in scheduler is a work-stealing mechanism. This mechanism or algorithm, namely that the idle processors would steal threads' jobs from busy processors [99], as shown in figure 4.8. In order to guarantee that the cost of stealing contributes only to critical-path overhead, and not to work overhead in Cilk's scheduler, Cilk exploits a Dijkstra-like protocol - THE protocol, to manage the runtime of ready threads in the work-stealing algorithm. The THE protocol permits no additional work overhead in scheduler when sending an exception signal to a working processor.

Chapter 5

Data Management Strategies Based on Data Parallelization and Data Replication

The scalability problems of BWA are mainly due to the contention generated by the need for multiple threads to access the same memory bank where the reference genome is located. To mitigate this problem we propose a series of alternatives based on a basic common scheme. In this scheme, there is a first level of parallelism based on groups of threads that handle groups of reads. On the other hand, the reference genome will be replicated in different instances, each of which will be accessible by different thread groups.

The number of thread groups and the number of replicas of the reference genome can vary and can be organized in different ways. The final objective is to analyze which variant obtains the best results in multicore or manycore architectures. In the manycore systems we are interested in finding strategies that are efficient using all the resources of the system (that is, executing in symmetric mode). We are also interested in solutions for manycore systems that can be easily transferred to multicore systems composed of servers with different characteristics in terms of performance. The variants that have been implemented and tested are the ones described below.

This chapter is divided into three sections. First with an overview about the strategies we proposed and its evolutionary step. Followed by the strategies of DP, DR and DPR that center on the utilization of data management strategies in the homogeneous architecture. Last, we combine data parallelization and data replication to a strategy called MDPR (Multi-level Data Parallelization and Replication) that exploits the usage of the heterogeneous architecture.

5.1 Overview

In our previous work [16] we presented a strategy based on data parallelization and data replication which could provide significant improvements in efficiency when applied to BWA. We denote this strategy as DPR (Data Parallelization and Replication) and its BWA implementation as DPR-BWA. The main idea consists of two steps. Creating a multi-process application where different instances of BWA have a replica of the genome index. The set of short read to be aligned is divided among the different instances that perform the alignment using multiple threads. The replication of the genome index improves locality of accesses and decreases the congestion in the usage of this shared structure by different threads. This approach proved to be effective when was applied in the native mode, both to a Xeon-based system with NUMA architecture and to a Xeon Phi accelerator with MIC architecture.

In our another previous work [17], we explore a generalization of DPR and we propose a multi-level strategy that can be applied in the symmetric mode on Xeon-Xeon Phi servers. The new strategy is denoted as MDPR (Multi-level Data Parallelization and Replication). Its basic structure is shown in Figure 5, which illustrates three basic parallelization levels.

According to our experimental results, MDPR shows significant improvements in execution time thanks to this multi-level parallelization strategy. The MDPR strategy also benefits from the usage of OpenMP threads instead of the original Pthread library provided by BWA.

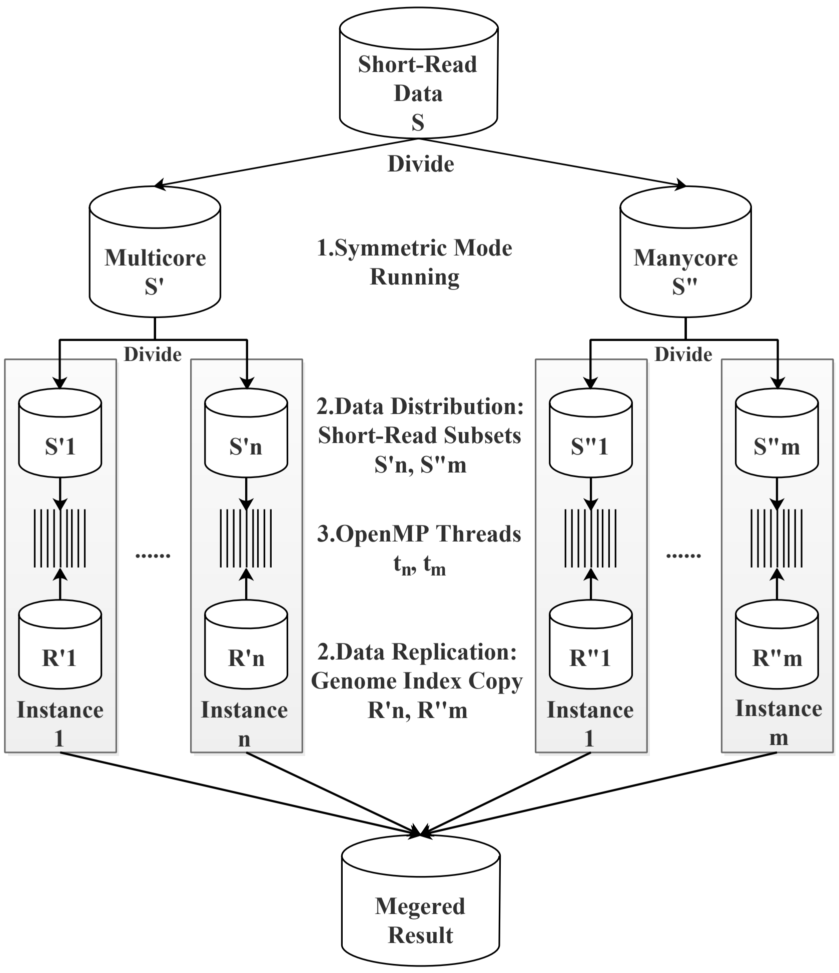


Fig. 5.1 MDPR-BWA implementation procedure in the case of n and m instances are applied in multicore and manycore parts (Intel Xeon + Xeon Phi), respectively.

Improvements in execution time of 3X and 1.5X were obtained by MDPR compared to an offload version of BWA (mBWA) and another symmetric implementation of BWA (sBWA), respectively. We also showed that a native version of BWA (called DPR-BWA), which outperforms other native versions of BWA, both in Xeon and in Xeon Phi systems.

Although a promising result has been achieved in our previous works, we still need to revise it that avoiding bottlenecks and make a forward in performance.

In latest version [100], we present a more extensive study where we analyze the performance of different BWA variants on multicore and manycore systems. We also adjusted the design of MDPR so that it can be executed indistinctly in multicore or manycore systems, and we have augmented it with different mechanisms of dynamic data distribution. Additional, we applied more experiments with two datasets and three heterogeneous architectures to display a convincing consequence we got.

5.2 Data Management on Homogeneous Architecture: DP, DR and DPR Strategies

5.2.1 Analysis of Sequence Alignment Procedure

We are exploring data management that involves both parallelism and replication. As shown in Figure 5.2, two datasets are involved in the alignment process. N corresponds the set of all short reads, and n is the number of read subsets that totally divided. M represents the reference genome, and m is the total number of replicas made of such genome. We set that Ni maps to Mj and $Ni+k$ maps to $Mj+k$ in the alignment. Hence, $N1 = ... = Ni = ... = Nn = N/n, M = M1 = ... = Mj = ... = Mm, 1 <= i <= n, 1 <= j <= m$.

Although genome reference also can be partitioned as we did for short read dataset that could dramatically conserve memory usage, three bottlenecks in this schema of genome splitting can be a significant effect that degrades performance. The index of each genome reference should be rebuilt and it costs a considerable time in this procedure. Besides, final merged results from each subsets(or blocks) need to adjusted according to the new position of every nucleobase in blocks. Last, in terms of quality of results, it would lose some accuracy in results cause of cut offs between successive blocks of reference genome. Thus as discussion above we ignore partition in genome reference dataset.

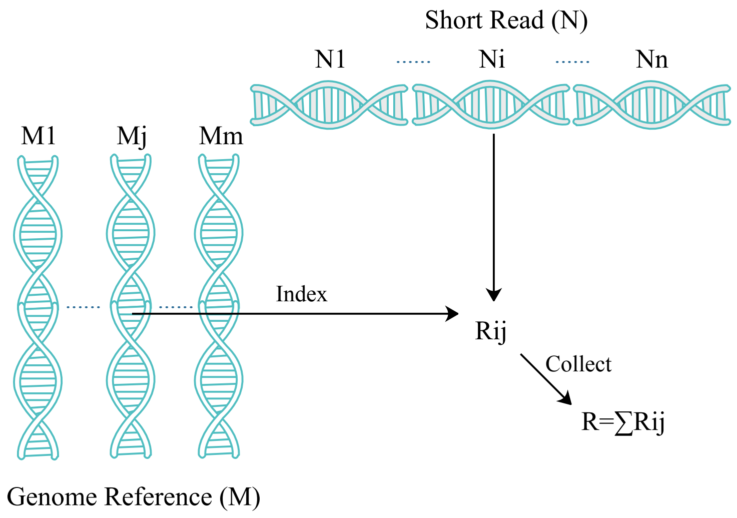


Fig. 5.2 Sequence alignment procedure

In our approach, BWA is organized as a set of thread groups (Ng) that are responsible form mapping some reads. In practice, a thread group represents a group of threads in OpenMP or a process in MPI, as shown in Figure 5.4. Each thread group is executed using a different number of threads depending on the hardware resource behind. One thread group in alignment is logically organized as a collection of threads that work on independent datasets of short read and share the genome index among the threads. For a total number of potential threads Nt and thread groups Ng, each thread group consists of Nt/Ng threads. For example, Nt is 24 and 240 in the cases of Intel Xeon and Xeon Phi, respectively. When applying two thread groups ($Ng = 2$), 12(24/2) and 120(240/2) threads are included in each thread group in the cases of Intel Xeon and Xeon Phi, respectively. When applying three groups ($Ng = 3$), 8(24/3) and 80(240/3) threads are included in one thread group in the cases of Intel Xeon and Xeon Phi, respectively.

The process of alignment has three main phases, as displayed in figure 5.3:

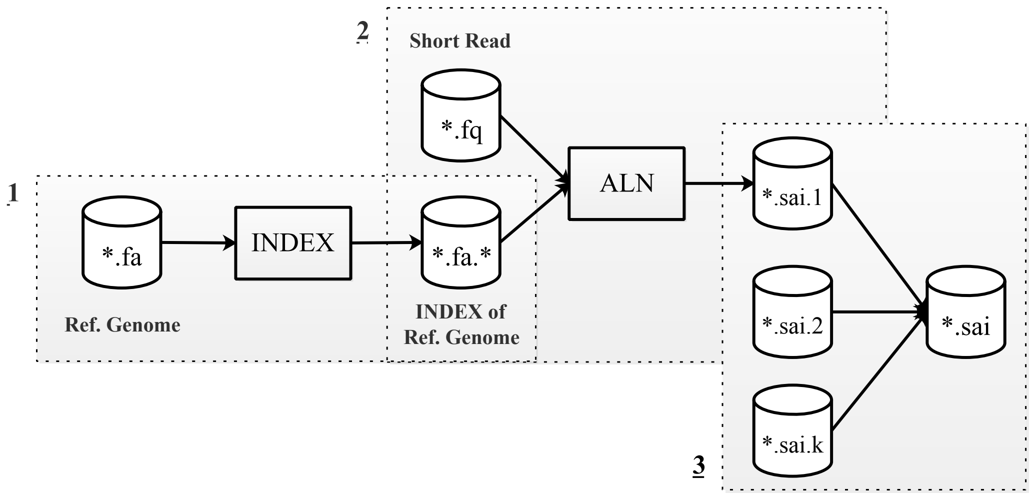


Fig. 5.3 Three main phases in the process of sequence alignment

1. Building the index for the genome reference Mj; the time of this phase is $T(Mj) = T(M)$;
2. Mapping short read subset Ni to genome reference Mj, which achieves mapping sub-result $Rij = Map(Ni, Mj)$; the time of this step is $T(Rij)$;
3. Collecting all alignment sub-results and merging them into final output $R = \sum Rij$; the time of this step time is $T(R)$;

Total execution time is basically $T = T(M) + T(R) + \sum_{i=1,j=1}^{i=n,j=m} T(Rij)$, being the first two steps the ones that can benefit from parallelization. Actually, the mapping consumes most of the execution time. Hence, we can simplify the expression to $T = \sum_{i=1,j=1}^{i=n,j=m} T(Rij)$.

5.2.2 DP, DR and DPR Strategies

There are different variations that can be applied to the values for n and m.

1. Original execution: $n = 1, m = 1$.

This case corresponds to the original structure of BWA. As shown in Figure 5.4.a, multiple OpenMP threads are processing all short reads using single reference genome.

2. DP strategy: $n > 1, m = 1$.

   In this case (see Figure 5.4.b), short reads are evenly divided into to n parts based on the number of thread groups that are used (Ng), and a single copy of the reference genome is used by all thread groups. We refer this strategy as data parallelization (DP). Alignment process runs in every thread group with the shared genome index and its relative subset of short read. Subset results are finally gathered and merged from all thread groups.

3. DR strategy: $n = 1, m > 1$.

   In this case, the reference genome is replicated m times, while short reads are shared among all thread groups. We refer to this strategy as data replication(DR). As shown in Figure 5.4.c, the reference genome is replicated for every thread group in memory. The alignment process in each thread group accesses to the shared set of short reads. Finally, the results are merged as in the previous case.

4. DPR strategy: $n > 1, m > 1$.

   In this case, both short reads and reference genome have several parts or copies (set of reads is divided in n parts and reference genome is replicated m times, being n and m the same value). As shown in Figure 5.4.d, each group of threads has one reference genome replica and short read dataset is evenly into as many parts as thread groups. This strategy is referred as DPR as it combines the previous two strategies.

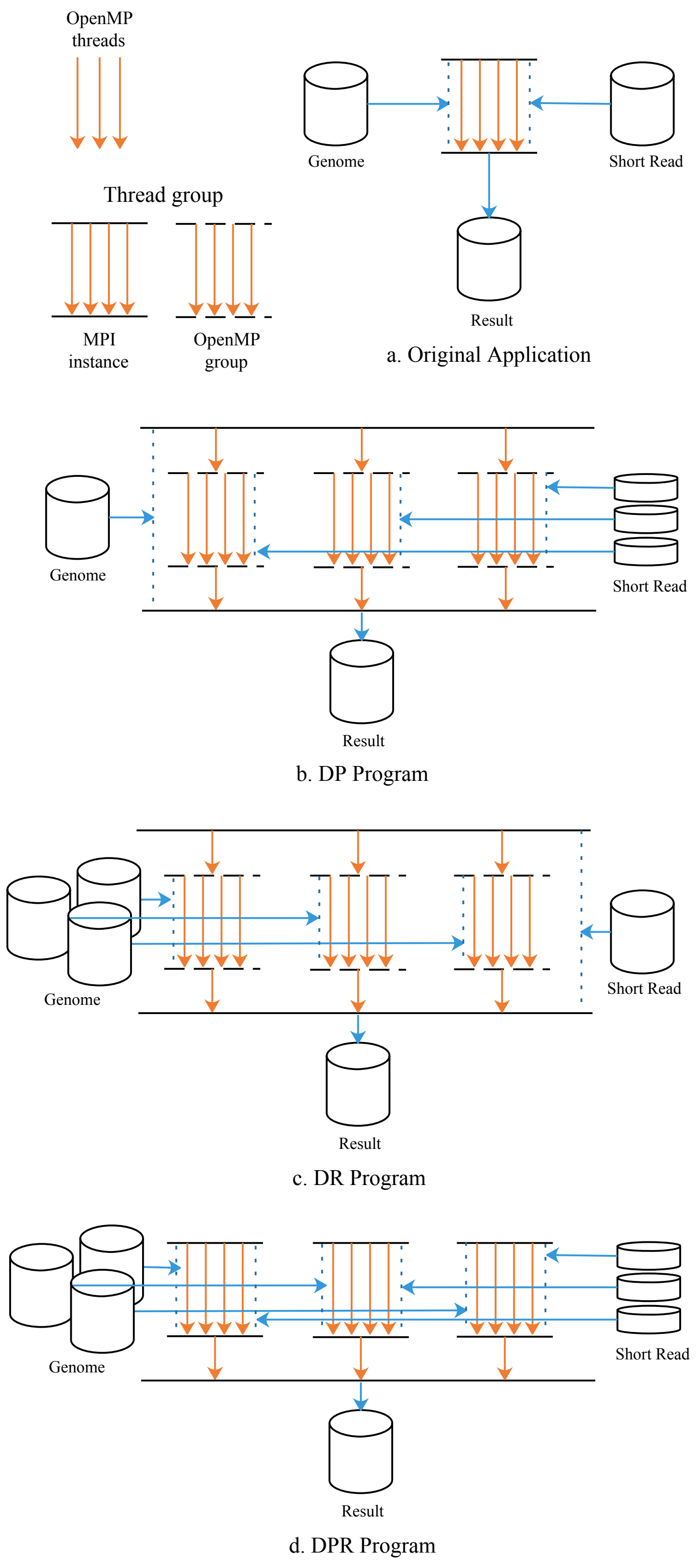


Fig. 5.4 Implementation of DP, DR and DPR

5.3 Data Management on Heterogeneous Architecture: MDPR Strategy

5.3.1 MDPR Strategy

The strategies described above can be applied in homogeneous multicore systems with shared memory. In the case of systems based on accelerators (executing in symmetric mode) or in the case of independent multicore systems, we propose a new strategy that generalizes DPR. It is referred as MDPR (Multi-level Data Parallelization and Replication) and is illustrated in Figure 5.5.

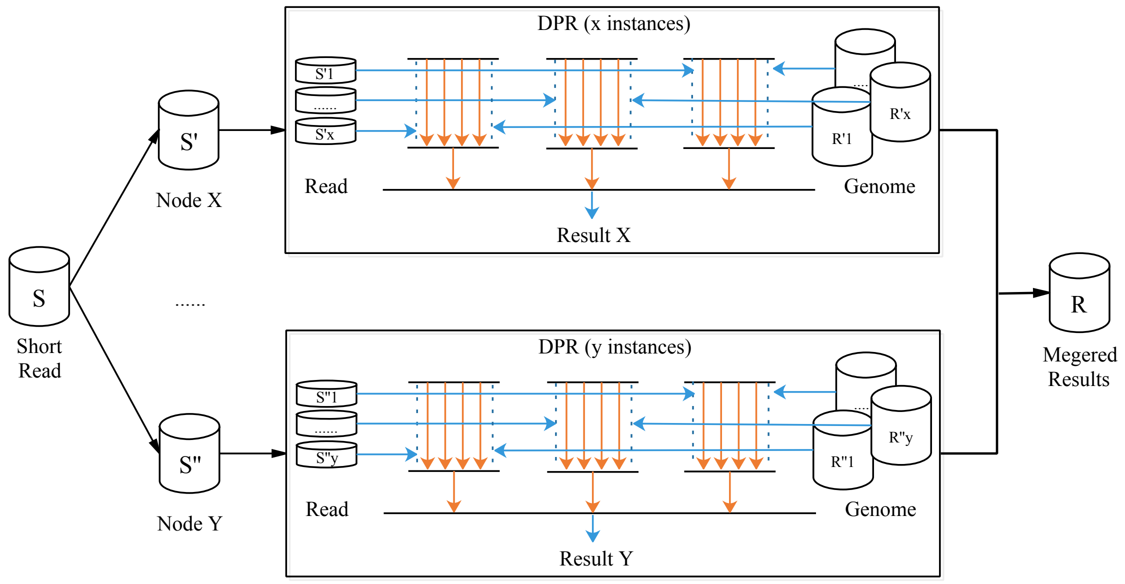


Fig. 5.5 MDPR strategy with the case of x and y instances are implemented in node X and Y, respectively

The short read dataset (S) is partitioned into several subsets: one subset for each node in the system (Figure 5.5 shows the case of two nodes: node X(S') and node Y(S'')). We have several thread groups or instances in node X (denoted as $S'1, ..., S'i, ..., S'x$, $1 <= i <= x$) and several thread groups in node Y (denoted as $S''1, ..., S''j, ..., S''y, 1 <= j <= y$). Thread groups are generated by taking into account the memory requirements of the aligner (in particular, the size of the reference genome) and the amount of available space on the memory banks in systems. Reference genome replicas are denoted as $R'1,...,R'x$, and $R''1,...,R''y$. Each BWA instance executes several threads to align short reads assigned to it. It is worth mentioning that that the number of instances on a heterogeneous architecture does not require to be identical. In fact, in the experimental part we'll show results obtained with various combinations that we tested.

According to this schema, the best performance should obtained if all thread groups consume a similar time. However, the heterogeneity of the whole system may leads to different execution times if short reads are distributed evenly. This load imbalance situation can be ameliorated by partitioning the short read dataset in an uneven way. This partitioning can be done in different ways and we have evaluated three possibilities: a static distribution that divides the dataset off-line and two dynamic distribution mechanisms that divide the dataset at run-time.

To cover data latency in the data distribution of MDPR, only data start position and data ending position are communicated with each thread groups instead of transferring dataset. Although we show a two-node case in the illustration of MDPR, it is also suitable for multiple nodes.

5.3.2 Static Distribution

In this mode, size of datasets S' and S'' are computed beforehand and the initial dataset of short read is divided statically beforehand. The ratio between S' and S'' is computed according to the relative performance achieved by the best result achieved by DPR in each architecture, as shown in figure 5.6.

For instance, in the case of two nodes (X and Y), for a certain number of instances on the node X and on the node Y (x and y instances, respectively), the time-cost Tx and Ty are measured. Then the Ty/Tx ratio between node X and node Y is used to divide the input data S. Namely, the ratio between S' and S'' is computed according to the relative performance

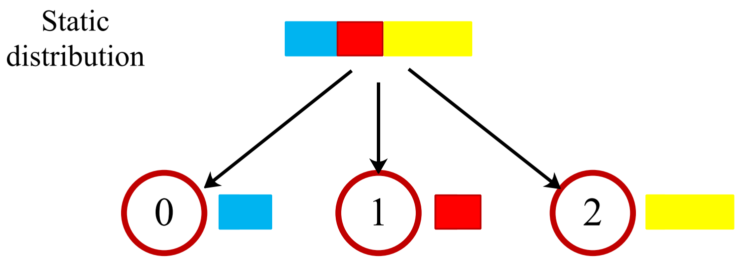


Fig. 5.6 An example of static distribution in MDPR strategy

achieved in each node. Hence, for the node X we have $size(S') = size(S) * Ty/(Tx + Ty)$ and for the node Y we have $size(S'') = size(S) * Tx/(Tx + Ty)$.

Obviously, this static distributions does not introduce any overhead at run-time but it requires the execution of several configurations that might be time-consuming in large systems.

5.3.3 Even Distribution

Each instance in MDPR is dynamically distributed with an identical size $(size(S)/(x + y))$ of short read dataset(S) based on the total number of instances $(x + y)$ we utilize, as shown in figure 5.7.

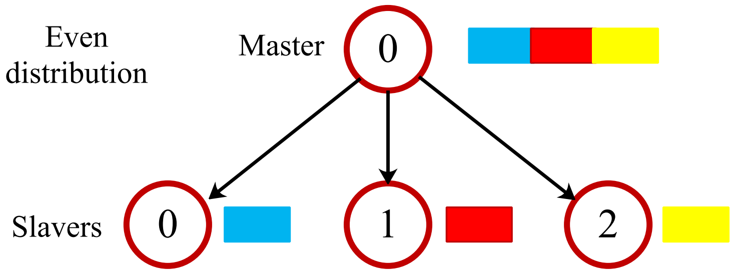


Fig. 5.7 An example of even distribution in MDPR strategy

In fact, the ratio between node X and node Y could set up as x/y. In node X we have $size(S') = size(S) * x/(x+y)$, and in node Y we have $size(S'') = size(S) * y/(x+y)$. As discussed in the static distribution section, the original dataset is divided at the beginning of the execution. But in even distribution, data distribution and execution are handled automatically.

This even mechanism does not introduce any significant overhead although it does not guarantee in general a good load balance if relative performance of nodes is significantly different.

5.3.4 Round-robin Distribution

This distribution implies that a master instance distributes chunks of reads to slave instances when they have finished previous chunks, as shown in figure 5.8.

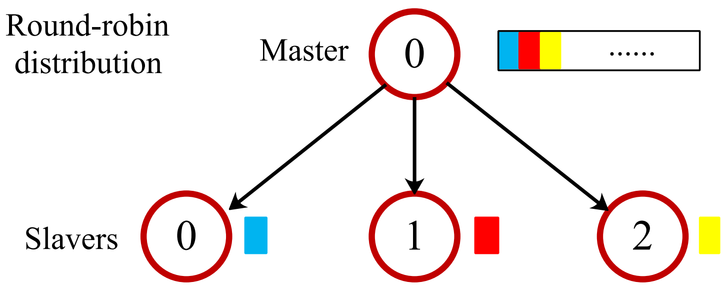


Fig. 5.8 An example of round-robin distribution in MDPR strategy

This mechanism should guarantee a better load balance than the previous one at the expense of a larger overhead at run time. As we known that each thread in BWA aligner once time processes a fixed size(256K in default) and each instance only reads a chunk from short read dataset for alignment. Thus chunk size should be at least 256K*(available thread number in one instance).

For example, if we execute BWA aligner in a system with supporting 64 available threads, the chunk size should be at least 16M(256K*64) when one instance is applied, or 8M(256K*32) when two instances are applied. Namely, more instances in one system would lead a more minimal chunk size.

As shown in Figure 5.8, chunks of short read are delivered under a round-robin fashion. It would receive one another chunk job when completing previous one. In order to cover the possible time-waiting for delivery chunks by the master in slaver instances, the round-robin experiments is conducted on the minimal size of chunk with one instance running with BWA aligner, and only data start position and data ending position is shared among neighbour instances.

Chapter 6

Experimental Implementation and Environment

6.1 Sample Dataset

Table 6.1 shows the sample datasets (SRR766060 and YH110112) we used in our experiments. The first one is obtained from the 1000 Genomes Project [101] and the second one corresponds to a genome sequence of Han Chinese individual project [102].

1KGP is an abbreviation of The 1000 Genomes Project that launched in January 2008 to set up the detailed catalogue of human genome under an internatinal research effort, including China, the United States, the United Kingdom, Japan, Kenya, Italy, Nigeria, Peru and etc [103]. Han Chinese individual project is the first Asia human genome sequenced that providing the first large-scale whole genome sequencing resource representative of the largest ethnic group in the world [104]. These two projects provide us a convincing genome dataset in the experiments.

As shown in Table 6.1, HS37D5 corresponds to the human reference genome, about 3GB size, and its BWA index, up to 4.4GB. Our dataset of short read, one is SRR766060

(SRR), paired-end, with total size equal to 17.4GB(8.7GB*2). The other is YH110112 (YH), single-end, we required 4.4GB segment from original source data.

These datasets are originated from widely confident projects that are the community resource research to establish by far the most detailed catalog of human genetic variation which aims to release data rapidly for the benefit of the scientific community. Although the experiments were conducted on the datasets of human genome, the methodology could be available for all species as well. A minimum of 6 executions of each independent cases with the same environment and configurations are carried out for the purpose of reaching mean values with high confidence intervals.

Table 6.1 Experiment dataset

Name	Data Set	Data Size	Read Length	Read Number	-
Short Read	SRR766060(SRR)	17.4GB	100	34.2M	Paired-end
-	YH110112(YH)	4.4GB	100	14.8M	Segment
Genome Reference	HS37D5	3GB	-	-	Human Genome

6.2 Implementation System

Two different heterogeneous systems which comprises distinct architectures inside are conducted in the experiments. One is the heterogeneous system of multicore and manycore architectures, which is denoted as S1, see Figure 6.1. The other is the heterogeneous system of Intel NUMA node and AMD NUMA node, we called this S2, see Figure 6.2. The details of two architectures we had discussed in the previous chapter.

Table 6.2 summarizes the main characteristics of the systems used in our experiments. Sandman and Sandman-Xeon Phi constitutes the heterogeneous system(S1) made of 2 Intel Xeon E5-2620 processors and one Intel Xeon Phi 7120 accelerator. Sandman has 2 processors, 2-NUMA nodes (with 32 GB of memory in each node), 12 cores and 24 available threads with hyper threading enabled. Sandman-Xeon Phi consists of 60 cores, 240 available

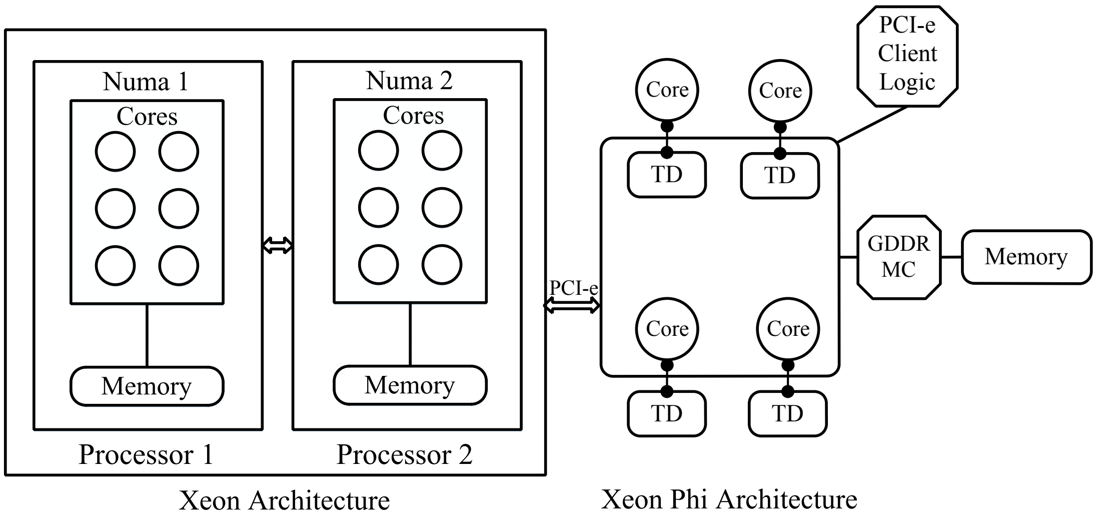


Fig. 6.1 The system of S1, Intel Xeon + Intel Xeon Phi

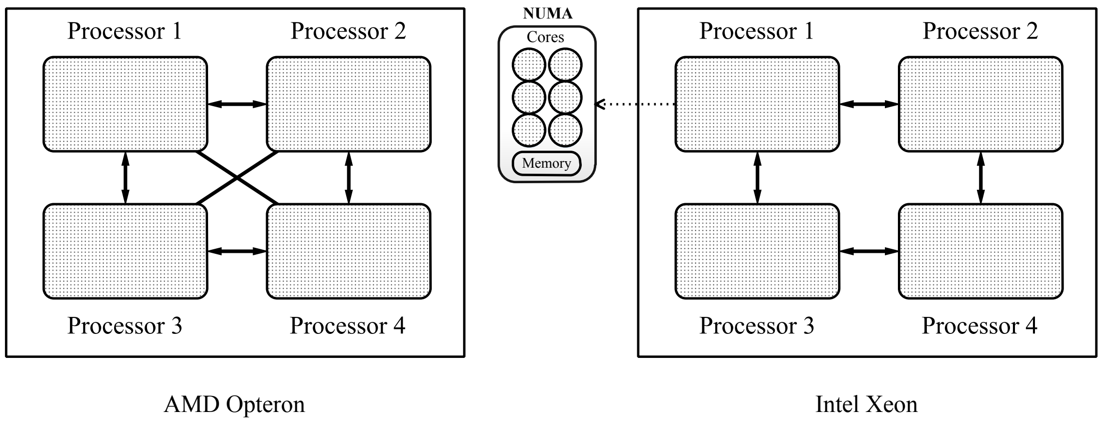


Fig. 6.2 The system of S2, AMD Opteron + Intel Xeon

threads and 16GB memory on card. Each core runs at 1333Mhz and possesses 4 hardware threads.

Table 6.2 Experiment system

System	Hostname	Socket	Processor	NUMA	Core	Thread	Memory	Data Width
S1	Sandman	Intel Xeon E5-2620	2	2	12	24	64GB	64bits
	Sandman -Xeon Phi	Intel Xeon Phi 7120	1	1	60	240	16GB	512bits
S2	Penguin	Intel Xeon E5-4620	4	4	32	64	128GB	64bits
	Batman	AMD Opteron Processor 6376	4	8	32	64	128GB	64bits
S3	AMD+Intel	AMD Opteron+Intel Xeon	17	25	126	252	576GB	64bits

The second system(S2) that we used in our experiments is made of two multicore servers: Penguin (equipped with Intel processors) and Batman (equipped with AMD processors). Penguin has 4 Xeon processors, 4-NUMA nodes, 32 cores, 64 threads and 32GB memory size per processor, and Batman has 4 sockets (with 2 processors per socket which implies a total of 8-NUMA nodes), 64 threads and 128GB memory bank. The architectures of both systems are illustrated in Figures 6.1 and 6.2.

Furthermore, a large-scale system that supports 252 threads 126 cores with AMD Opteron and Intel Xeon were implemented in the experiments of MDPR strategy.

6.3 Related Software

6.3.1 BWA-ALN-Xeon-Phi

BWA aligner is an efficient software program for sequence alignment against a very large reference genome which consumes less memory print than others. BWA consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. This BWA-ALN-Xeon-Phi [105] program focuses on the first algorithm which is designed for Illumina sequence reads up to 100bp.

In order to utilize the heterogeneous cluster in cooperation for genome mapping, we take advantages of one project named BWA-ALN-Xeon-Phi-0.5.10 which is based on original application BWA-0.5.10, and it supports two execution modes, native mode and symmetric mode. The source code of BWA-ALN-Xeon-Phi-0.5.10 can be downloaded from https://github.com/intel-mic/bwa-aln-xeon-phi-0.5.10. All implementation is executed under applying Intel Parallel Studio XE 2013 environment. All MPI execution runs on the MPI 4.1.3 version.

Some significant optimizations have been applied in this project based on the performance of BWA-backtrack algorithm of BWA-0.5.10 version. It enables a symmetric execution mode in cooperation for genome mapping that could use both Intel Xeon processors and Intel Xeon Phi coprocessors at the same time under some configurations beforehand.

Crucial advantages and disadvantages in BWA-ALN-Xeon-Phi project we list below which we utilize in the experiments of this thesis:

1. This BWA-ALN-Xeon-Phi replaces Pthread with OpenMP in core loops, we extend this function to index-building step and other significant loops.
2. This BWA-ALN-Xeon-Phi applies task parallelism in addition to OpenMP that is utilized in the DP and DR strategies but we call this thread group in order to distinguish the new function of task parallelism of OpenMP.
3. Symmetric execution mode is implemented with BWA-ALN-Xeon-Phi in our previous work. However, a large number of configurations before implementation is necessary and complex. We move symmetric execution mode from BWA-ALN-Xeon-Phi to original BWA-0.5.10 with less configurations and achieve a faster execution.

6.3.2 mBWA

There are three basic execution modes that could be used in the heterogeneous multicore and manycore architecture of Xeon-Xeon Phi-based system. Offload mode, the application executes on the Intel Xeon, and in order to reach a better performance it offloads selected highly parallel and computationally intensive work to the Intel Xeon Phi. This is the most widely used with Xeon Phi as an accelerator. On the contrary, there is another opposite execution mode. The application runs on the Xeon Phi and offloads selective work into Xeon. But this is a very rare working way.

In this thesis, we demonstrate an offload version of BWA, mBWA, to evaluate the three basic execution modes. mBWA is an application developed by Cui [11] which starts BWA on the Xeon processor and offloads core loops to the Xeon Phi. According to the experiments of authors, mBWA outperforms BWA in a significant performance with a combination of offload mode and three-stage parallel pipeline strategy. As shown in Figure 6.3, the performance of mBWA increases linearly with the number of threads and coprocessors increasing.

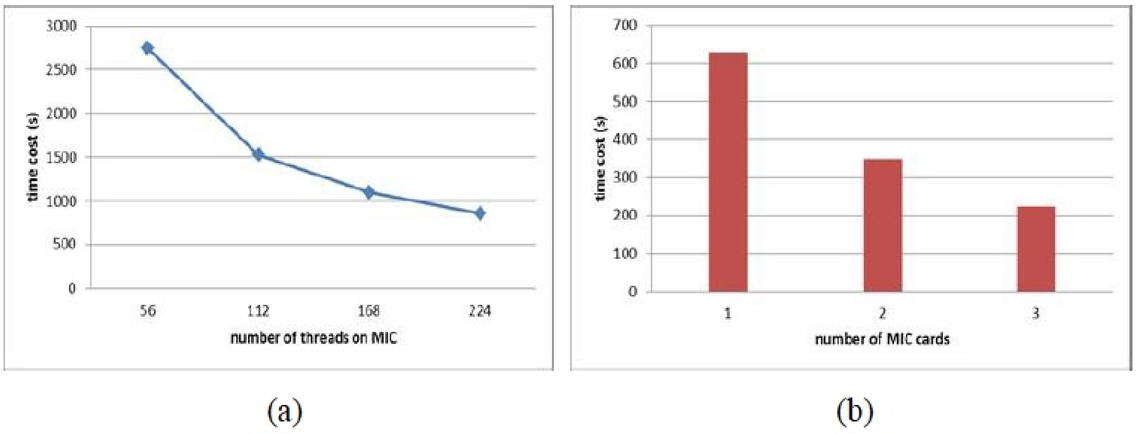


Fig. 6.3 Effectiveness of MIC. (a) Scalability of mBWA on Intel Xeon Phi. (b) Performance of mBWA under different number of Intel Xeon Phi

We list key characteristics that made optimizations in mBWA as follows:

1. A three-stage pipeline is designed for the sequence alignment process. The three stages include data input, reads aligning and data output. The pipeline increases efficiency by overlapping data IO with the actual alignment process.

2. The alignment kernel of BWA is ported into the Intel MIC coprocessor by re-organizing data transformation in the kernel. In addition, CPU and MIC perform the alignment corporately in the offload mode.

6.3.3 pBWA

pBWA [14] is a parallel mapping version of BWA which is based on the OpenMPI library [106]. It enables multi-threading capability (Pthread) for achieving an efficient parallelization for core alignment mapping. According to our experiments we reach, we modify Pthread library to OpenMP library in pBWA, in order to enable efficient multi-threading ability. This ability enables maximal usage of a hybrid MPI and OpenMP programming so that pBWA could be executed in both parallel and multi-threading simultaneously.

According to the results of the experiments done in the research, pBWA could utilize the advantage of HPC cluster and approach a considerable performance improvement by cutting down the overall time from weeks to hours for rather large DNA sequence datasets, as displayed in Table 6.3. The 350 million reads dataset was run with 240 processors in pure parallelization (i.e. 1 threading per processor) and using 48processors, each running 5 threads, therefore both for a total of 240 threads (the 240P and 48P/5T columns in Table 6.3) [14].

Moreover, parallelization supports slightly better speedup and efficiency at rather large amounts of sequence data for aln model than multiple threads in the identical number of threads in sampe model, which is due to lack of multi-threading in sampe (240P vs. 48P@5T in Table 6.3). This is also one reason that we choose to focus in BWA-ALN model.

In this thesis, pBWA-based versions are proposed in the experiment according to the revision we need. Although pBWA that based on the version of BWA-0.5.9 has made some favorable optimizations in aln and samse/sampe process of alignment, as shown in Table 6.3, we only focus on the aln process, BWA-backtrack algorithm. We list the revision necessary when implement our strategies in the experiments as follows:

Table 6.3 pBWA executed with 350 million paired 50 bp reads. T, the number of threads; P, the number of processors; Time-used for running aln for each of the pair read file. Efficiency calculated based on the combined time in minutes (m) or seconds (s) of aln and sampe commands, and is calculated as Speedup divided by the number of threads or processors or the combined total threads.

	1 T	24 P/speedup	48 P/speedup	96 P/speedup	240 P/speedup	48 P @ 5 T (240 T)/speedup	240 P @ 12 T (2880 T)/speedup
aln 1	7611 m	606 m, 12.6	294 m, 25.9	140 m, 54.4	62 m, 122.8	66 m; 115.3	13 m, 585.4
aln 2	6950 m	495 m, 14.0	253 m, 27.5	124 m, 56.0	55 m, 126.4	59 m, 117.8	12 m, 579.2
sampe	520 m	67 m, 7.7	34 m, 15.3	24 m, 21.7	16 m, 32.5	34 m, 15.3	16 m, 32.5
Totals	15081 m	1168 m/12.9	581m/26.0	288 m/52.4	132 m/114.3	159m/94.8	41 m/367.8
Efficiency	1	0.53	0.54	0.55	0.47	0.40	0.13

- Original pBWA version utilize Pthread library to enhance multi-threading ability, but in our experiments Pthread library is removed and OpenMP is applied in the core loops instead.
- In the performance evaluation of MDPR, we utilize pBWA version to revise the necessary code in program in order to implement our strategies of dynamic schedule of data distribution, such as even distribution and round-robin distribution.

6.3.4 Intel VTune Amplifier

Performance analysis is one of our crucial works in the thesis. Intel VTune amplifier is the basic software of performance analysis we utilize in the evaluation of the experiments. Intel VTune amplifier interacts with users by gathering crucial profiling data which provides a simplified analysis and interpretation for necessary optimizations. This software is a commercial application provided by Intel corporation that could support both a GUI and command interaction for Linux and Microsoft Windows OS. Many features can be work on both operation systems as illustrated in the following part [107]:

- Software sampling
- JIT profiling support
- Locks and waits analysis

- Threading timeline
- Source view
- Hardware event sampling
- Memory Access Analysis
- Storage Analysis

We used VTune software to provide performance counter numbers to support our explanations of performance results from the experiments. In the performance comparison of different thread libraries, VTune provides us information details about thread time-cost in each threads.

Chapter 7

Experimental Results and Evaluation

In this section we show several experiments comparing different versions of BWA in various proposed strategies in order to find the one that provides better performance.

The first set of experiment focuses on Xeon Phi-based system that shows a comparison among the basic mode of BWA (native mode) and strategies from the literature that run in offload mode and in symmetric mode on Xeon Phi systems.

Next, a set of experiments evaluates the impact of three thread libraries to enable multi-threading ability for BWA aligner, including Pthread, Intel Cilk and OpenMP we proposed. This experiments illustrate a brief of overheads and work distribution among the threads.

Finally, The next experiments analyze the behavior of the strategies based on data parallelization or data replication described in the section of data management strategies. Experiments carried out with the MDPR strategy are shown to evaluate the performance of the three mechanisms of data distribution both on our manycore system and on our multicore system.

7.1 Evaluation of Parallelism on Basic Execution Modes

7.1.1 Implementation

These experiments evaluate the scalability of three basic execution modes of BWA aligner according to the description provided in Section 4(i.e., native, offload and symmetric modes). We ran the applications on our multicore-manycore system (S1: Sandman and Sandman-Xeon Phi) and we used two datasets, SRR and YH. We tested the BWA versions shown in Table 7.1.

Table 7.1 BWA aligner versions on basic execution mode

Architecture	Native Mode	Symmetric Mode	Offload Mode
Xeon	BWA-Xeon	sBWA	
Xeon Phi	BWA-Xeon Phi		mBWA

BWA-Xeon and BWA-Xeon Phi correspond to the original version of BWA running in native mode either in the Xeon part of the system or in the Xeon Phi part of the system. sBWA is a straightforward implementation of BWA running in symmetric mode with two instances of BWA aligner (one in Xeon and one in Xeon Phi) connected by MPI services. Finally, mBWA [11], runs BWA in an offload mode. It launches in the Xeon part and the main time consuming parts are derived to the Xeon Phi part. These four BWA-based versions corresponds to three basic execution modes, as shown in Table 7.1.

Load balance between Xeon and Xeon Phi on the symmetric mode is rather significant and their time-consuming should be close in order to approach their best performance. In this experiments, we would use one common fashion - static mode - which means we settle up load balance by optimizing the workload based on the runtime of their native mode execution. For example, if the Intel Xeon Phi coprocessor's performance on native mode is 0.7-fold that of the same problem running on the Intel Xeon processor on native mode, then the ratio of workload on symmetric mode is 10 for the Intel Xeon processor, and 7 for the Intel Xeon Phi coprocessor.

7.1.2 Performance Analysis

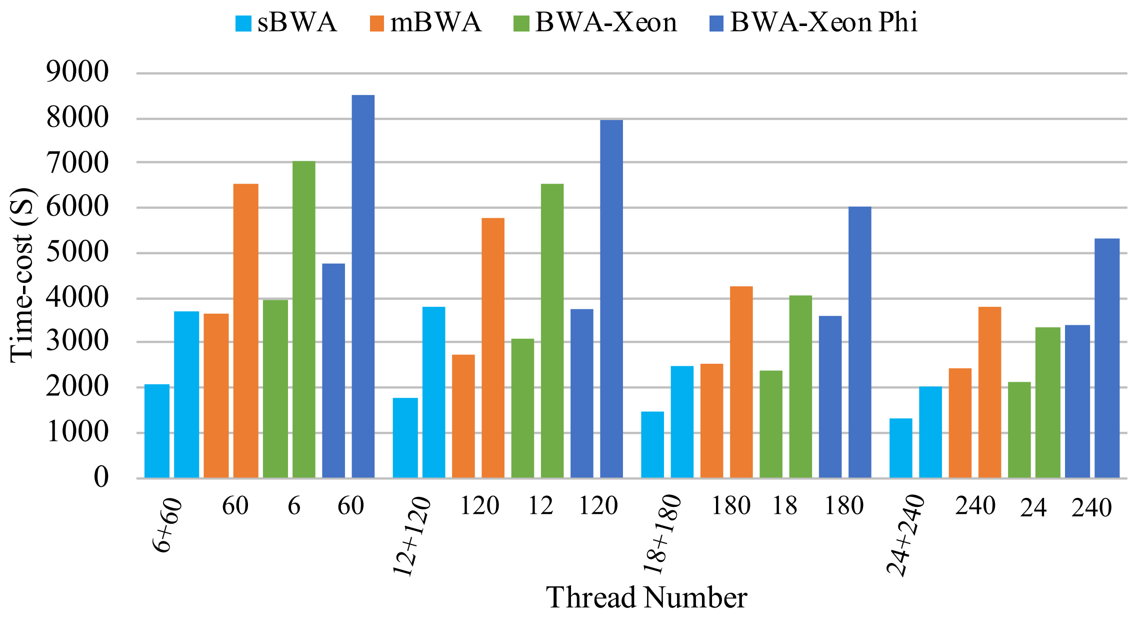


Fig. 7.1 Scalability of BWA aligners (identical color in short column: the results from dataset SRR, and long column: the results from dataset YH)

Figure 7.1 shows average execution times obtained by each BWA-based aligner for each dataset. The longer column in one color represents the results from dataset YH, while the shorter column in same color stands for the results from dataset SRR.

Maximal parallelism in our system is 24 threads in Xeon (Sandman) and 240 threads in Xeon Phi (Sandman-Xeon Phi). Hence, BWA-Xeon case illustrates scalability of 6, 12, 18 and 24 threads. BWA-Xeon Phi and mBWA cases utilize scalability of 60, 120, 180 and 240 threads. sBWA results for the following number of threads: 6 (on the Xeon) and 60 (on the Xeon Phi), 12 and 120, 18 and 180, and 24 and 240, respectively.

All strategies show a similar tendency in each dataset. Despite the high number of threads used, BWA-Xeon Phi exhibits the worst execution times, with sBWA being the best performing version. sBWA always outperforms native and offload cases. Although sBWA simultaneously takes advantage of available cores both on the Xeon and on the Xeon Phi, the performance improvement is not linear with the number of growing threads. You can observe that time-consuming decreases quite slowly when 12+120 threads and more are applied in the experiments.

This phenomena can be explained because sequence alignment process has large data I/O operations and is sensitive with the cost of data I/O operations. It consumes more time in the Xeon Phi part rather than in the Xeon part when the necessary reading datasets of genome reference and short read, and writing alignment results.

In the case of offload mode, mBWA performs main data I/O operation in the Xeon part instead of in the Xeon Phi part, and selects highly parallel loops offload into the Xeon Phi part. Thus when comparing the performance of the same amount of threads in mBWA with that of BWA-Xeon Phi, mBWA are approximately two third of execution time spend by the case of BWA-Xeon Phi and approaches a close performance with BWA-Xeon aligner.

The identical situation occurs in BWA-Xeon and BWA-Xeon Phi, the former case reaches better performance than the latter one, despite a lower amount of threads (1/10 X) applied in BWA-Xeon. In the case of sBWA, on the one hand, the asymmetric distribution of initial data compensates the loss in performance incurred by data I/O operations on the Xeon Phi. On the other hand, sBWA makes memory access only within its local memory bank of nodes, without any remote access within nodes.

Figure 7.1 also shows another problem that affects all four aligners: scalability does not improve significantly as a growing number of threads is used. As mentioned in Chapter 3, BWA uses a shared data structure containing the genome reference. This data structure is loaded at one particular memory bank of the NUMA system by the master thread according to first-touch policy. Thus this memory bank turns into a congestion bottleneck of performance as the number of threads grows. This memory bank needs to support more concurrent accesses coming from various threads to populate the corresponding caches. Multiple replicas of genome reference in NUMA node could be a convenient road to resolve this bottleneck.

7.2 Thread Parallelism Evaluation: Pthread, Intel Cilk and OpenMP

7.2.1 Implementation

As mentioned in Chapter 3, BWA aligner is an embarrassing tool in which the threads in parallel process chunks by repeating the following three principal actions:

1. Acquiring a fixed size of batch from short read, 256KB in default,
2. Aligning the batch in a round-robin pattern among threads against the genome reference,
3. Writing the alignment results to the output file.

BWA has multi-threading capability (using the Pthread library) to solve read mapping operations in parallel since there is no data dependency in these actions. The addition of multi-threading to an aligner does not guarantee that processing resources are used efficiently. In fact, it is quite common to observe in the experiment that efficiency decreases when the thread count grows large enough.

We denote the original version of BWA aligner as BWA-Pthread, which is implemented under the Pthread library and spawn t threads via the -t parameter in the master thread. A job distribution mechanism in threads, called round-robin pattern, activates dynamic load balance at some extent. However, we observed in VTune that BWA-Pthread does not preserve t threads when running. In fact, 3097 threads were observed(see Figure 7.6) in total during the alignment in the experiments of setting up 24 threads by parameter -t. This behavior indicates a considerable overhead expense on the management of thousands of threads.

For the purpose of avoid this overhead, we also evaluated two alternative version of BWA aligner utilizing different thread library: OpenMP (BWA-OpenMP) and Intel Cilk (BWA-Cilk). These two versions maintain a fixed number of threads (t threads) until BWA aligner completes the whole alignment. Short read is distributed in a fixed road among the threads in BWA-OpenMP with no stealing mechanism. Thus synchronization and overhead is lower at some cases.

Additional, BWA-Cilk is task-based parallelization under the usage of Intel Cilk Plus extension for C and C++ programming. BWA-Cilk threads work in the dynamic job-stealing schedule to achieve load balance, namely faster threads could steal jobs from other slower threads. But several machine instructions BWA-Cilk uses are not supported in the Xeon Phi part, we disable the unsupported instructions in MEM algorithm, thus ALN algorithm we work can be implemented in the Xeon Phi.

7.2.2 Performance Analysis

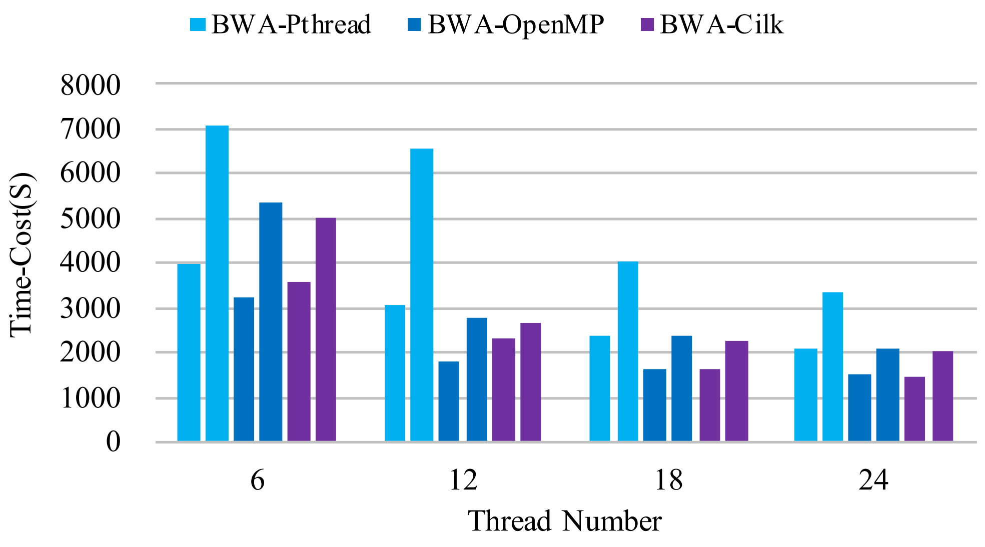


Fig. 7.2 Thread scalability in the Intel Xeon (Sandman). Identical color in short column: the results from dataset SRR, and long column: the results from dataset YH

Figures 7.2 and 7.3 display mean time of three thread versions of BWA aligners executed in the Intel Xeon processor and Xeon Phi coprocessor (S1) with two datasets respectively. Figures 7.4 and 7.5 illustrate another heterogeneous architecture of Intel Xeon and AMD Opteron (S2), four sockets multiple NUMA nodes with two datasets.

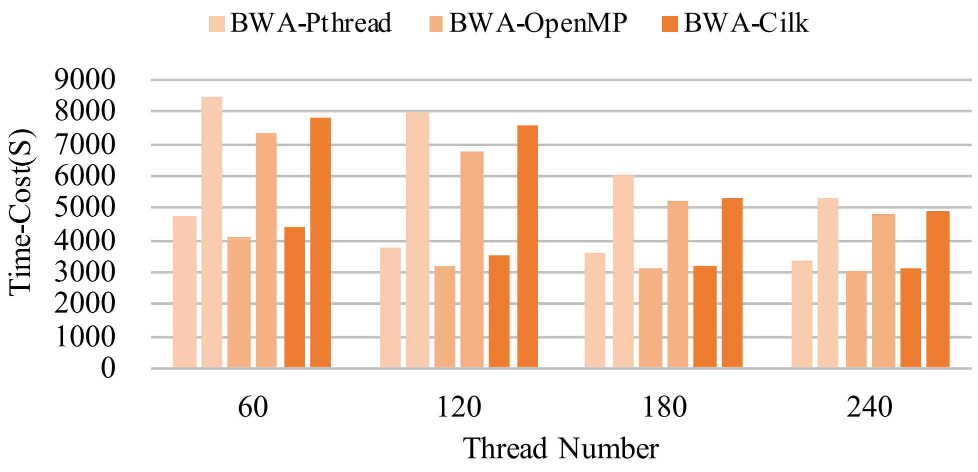


Fig. 7.3 Thread scalability in the Intel Xeon Phi (Sandman-Xeon Phi). Identical color in short column: the results from dataset SRR, and long column: the results from dataset YH

As indicated in the previous chapter, the longer column in one color represents the results from dataset YH, while the shorter column in same color stands for the results from dataset SRR. These BWA aligners are conducted in the system of S1(Sandman and Sandman-Xeon Phi), the number of threads varies from 6 to 24, 60 to 240, respectively.

Similar phenomena was observed in both systems. BWA-Pthread always performs worst performance which exhibits the highest execution time of all. We discovered that irregular waiting time and fluctuated CPU time by utilizing the performance software VTune. As shown in Figure 7.6, each CPU time of Pthread ranges from 4s to 11s when we applied 24 threads in Sandman with dataset SRR. This imbalance behavior in CPU time could occur due to the overhead in thousands threads synchronization and management.

BWA-OpenMP reaches the least time-cost of three BWA aligners in the cases of 6 and 12 threads in Sandman, and similar value in the case of 18 and 24 threads with BWA-Cilk version. According to these results illustrated in the figures, BWA-OpenMP we proposed always achieve less time-cost than the original Pthread version and not worse than Intel Cilk version. Another advantage BWA-OpenMP displays is that this version can be easily implemented in Xeon Phi part while BWA-Cilk does not.

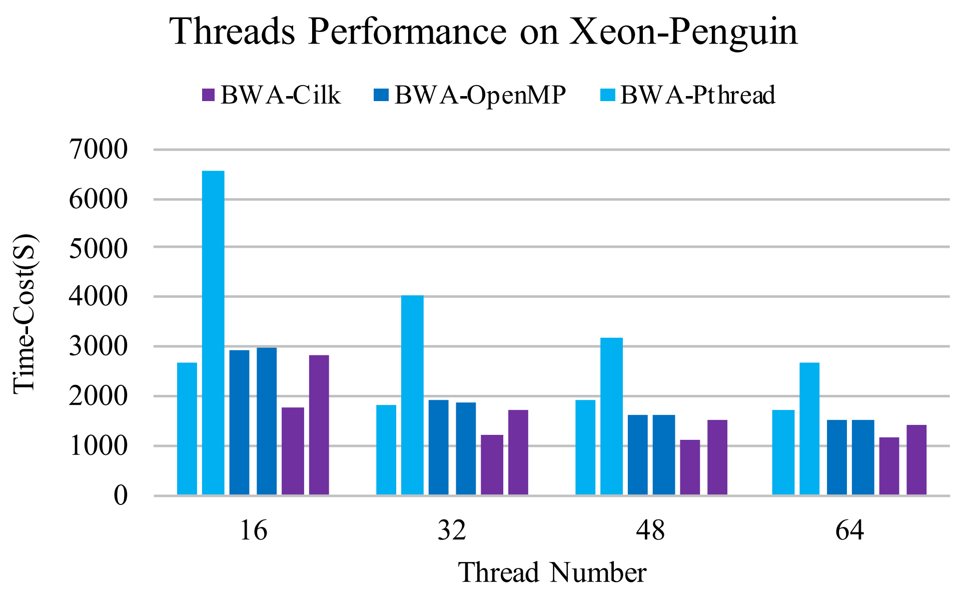


Fig. 7.4 Thread scalability in the 4-socket Intel Xeon Phi (Penguin). Identical color in short column: the results from dataset SRR, and long column: the results from dataset YH

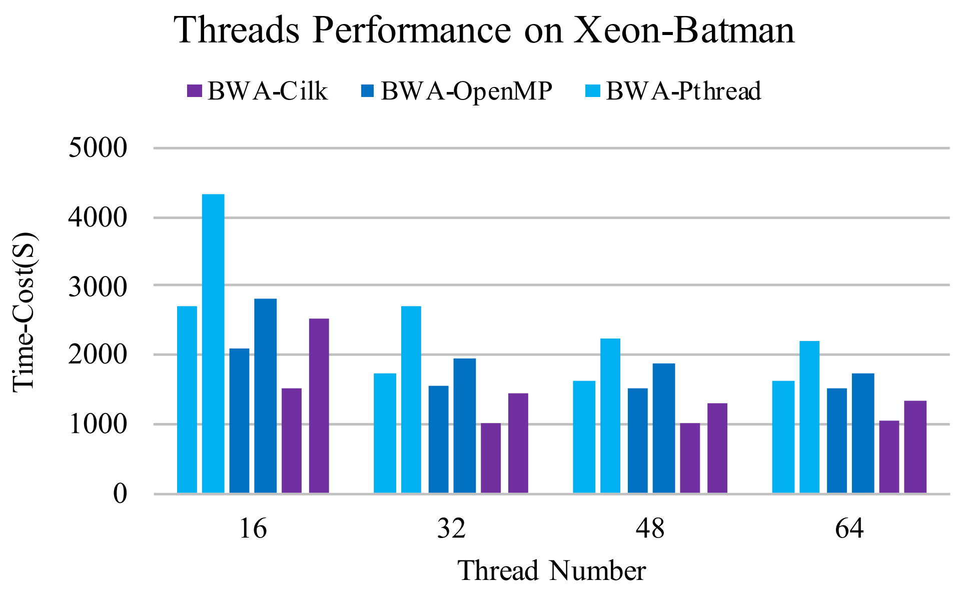


Fig. 7.5 Thread scalability in the AMD Opteron (Batman). Identical color in short column: the results from dataset SRR, and long column: the results from dataset YH

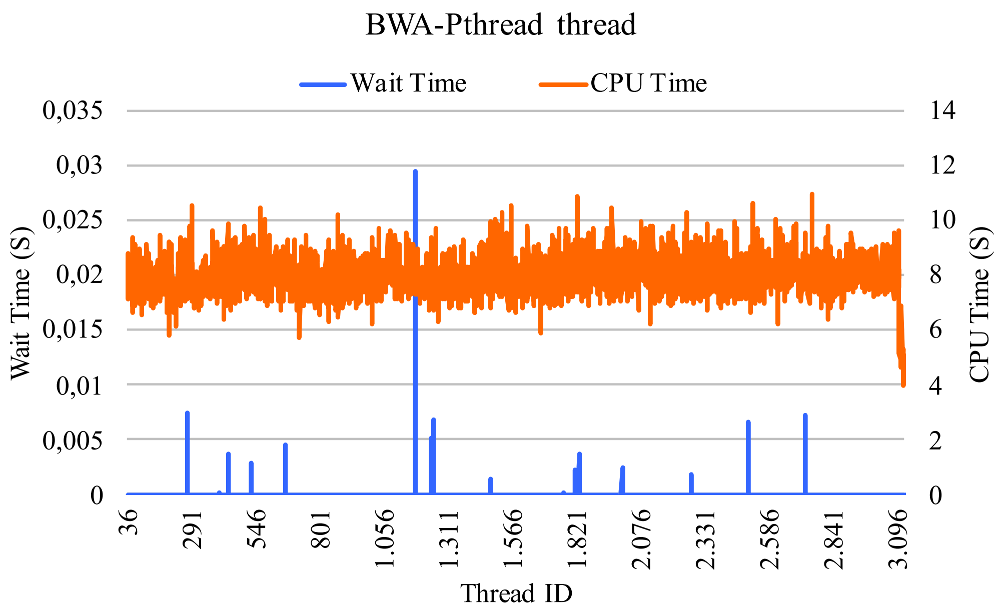


Fig. 7.6 Thread performance of BWA-Pthread in the Intel Xeon(Sandman) with dataset SRR by VTune

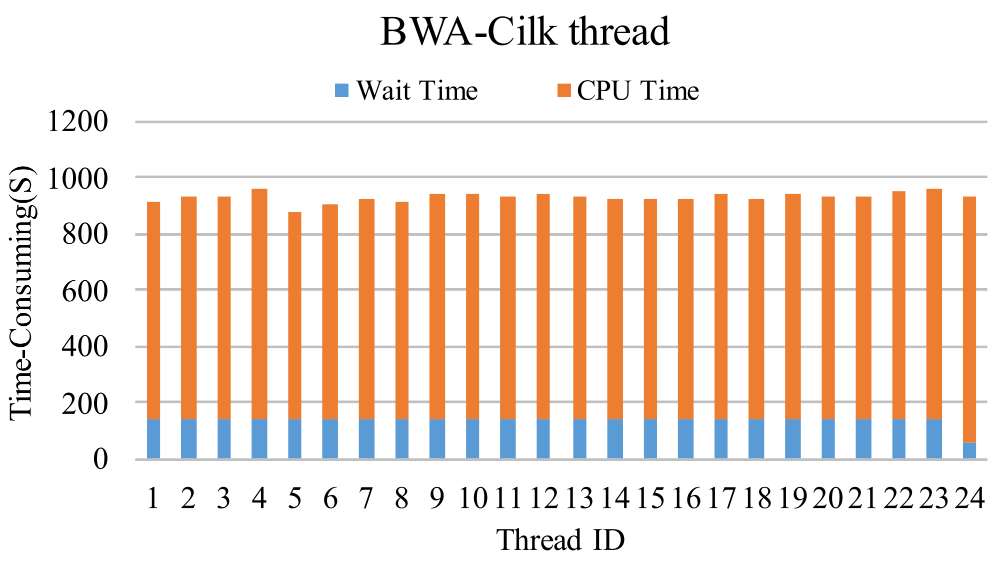


Fig. 7.7 Thread performance of BWA-Cilk in the Intel Xeon(Sandman) with dataset SRR by VTune

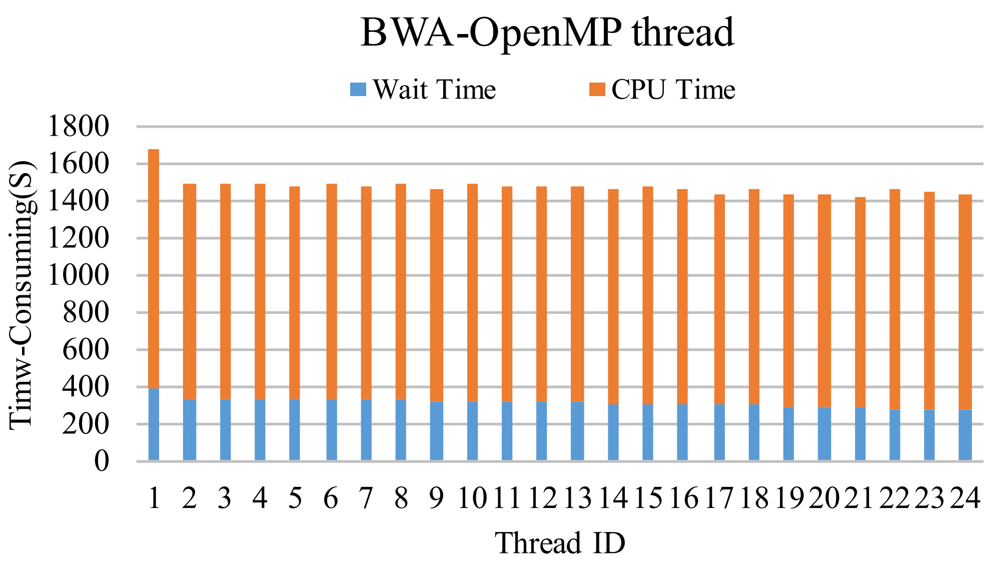


Fig. 7.8 Thread performance of BWA-OpenMP on the Intel Xeon(Sandman) with dataset SRR by VTune

We also evaluated thread time of 24 threads running in the Intel Xeon(Sandman) with dataset SRR, see Figures 7.6, 7.7 and 7.8. CPU Time is time during which the CPU is actively executing your application. Wait time is per-thread that occurs when threads are waiting due to APIs that block or cause synchronization.

The CPU usage of baseline, in Figures 7.6, is rather rough and fluctuated which means the threads occasionally work and much times for idle. Each CPU time of Pthread ranges from 4s to 11s, even we observed totally 3096 threads created in the whole alignment process when we set up parameter t as 24. While in the execution of BWA-OpenMP and BWA-Cilk, as depicted in the Figures 7.7 and 7.8, the threads work simultaneously on the vast majority of execution time and rarely rest. Hence, the CPU usage is very flat and tight which indicts load is much balanced among threads.

We observed similar behavior that happens in S2, whether Penguin or Batman. BWA-OpenMP version gains a more considerable improvement than BWA-Pthread, and could provide a balance performance among these three thread libraries. Thereby in the following experiments, if we do not refer the version of BWA aligner we used for implementation, BWA-OpenMP always the default one we applied in the experiments.

7.3 Scalability Evaluation of DP, DR and DPR

7.3.1 Implementation

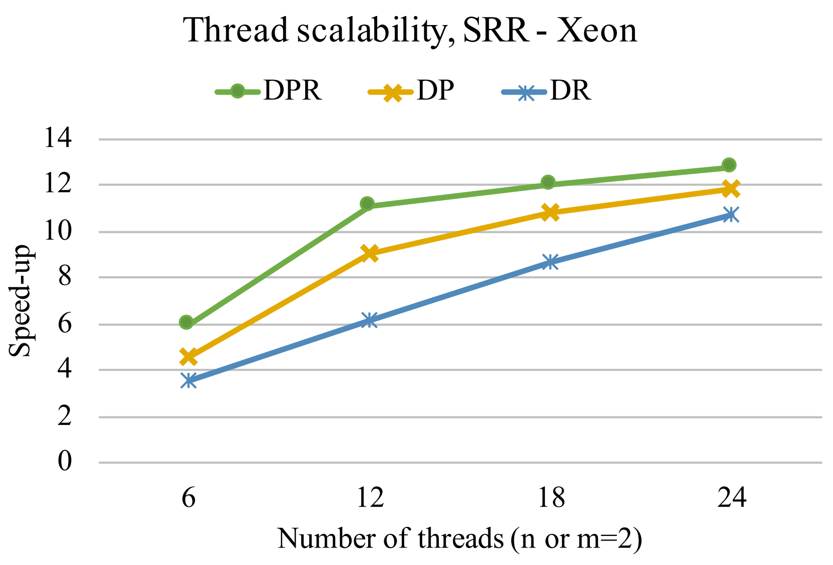
In this experiments, we set up two thread groups for execution in the strategy of DP, DR and DPR, namely $n = 2$ in DP, $m = 2$ in DR and $n = m = 2$ in DPR. Figure 7.9 and 7.10 show relative speedups in Intel Xeon and Xeon Phi, respectively.

In terms of available threads in Sandman and Sandman-Xeon Phi, thread number of 6, 12, 18 and 24 are tested in the evaluation of scalability in former system and thread number of 60, 120, 180 and 240 are tested in latter one. The speed-up is calculated using a baseline case that the execution time of original BWA with one thread running on Sandman under dataset SRR.

7.3.2 Performance Analysis

In general, the three strategies have a consistent behavior in all cases and it can be concluded that DPR obtains the best performance, followed by DP and DR. DPR achieves best speed-up in all cases no matter in the Xeon or Xeon Phi platform because hybrid MPI and OpenMP is applied in this strategy for solving the congestion of accessing genome index and improving the parallelization of short read. However, in the case of strategies of DP and DR, although some parallelism methods are utilized in the implementation, it is not better for improvement of overall performance.

There still crucial bottlenecks of performance existed in the implementation of BWA aligner. DP strategy solves short read in parallel subset simultaneously, but it does not



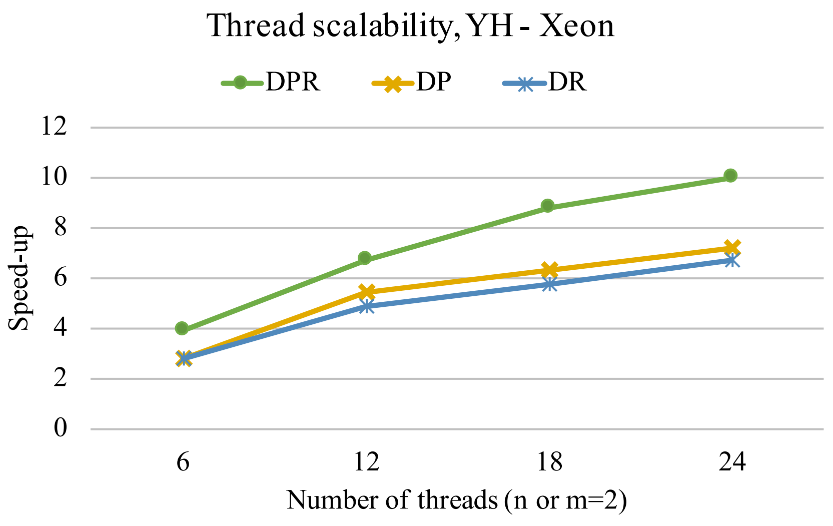
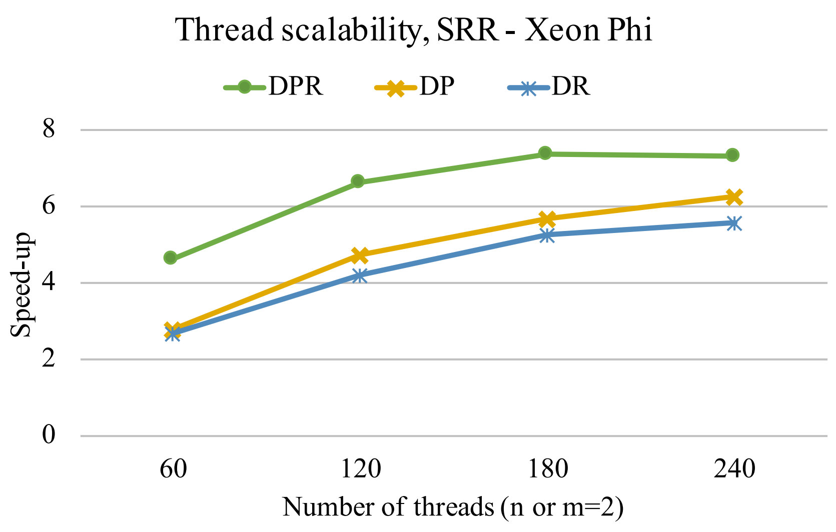


Fig. 7.9 DP, DR and DPR scalability comparison on Sandman



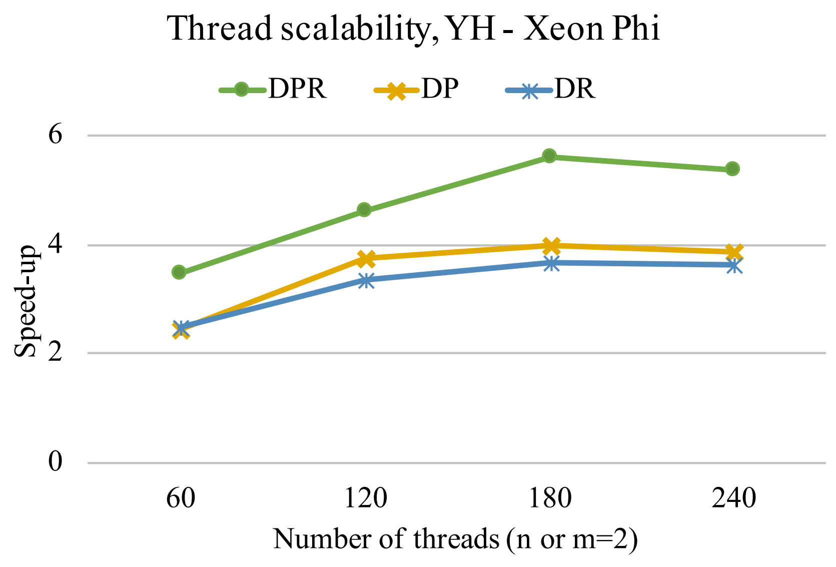


Fig. 7.10 DP, DR and DPR scalability comparison on Sandman-Xeon Phi

reach much performance cause only the batch (256KB) is read each time. DR strategy at some extent resolves the bottleneck of congestion of genome index dataset, but it does not compensate the remote access of it. DP and DR show a quite close speed-up line in the experiments.

Results in the Xeon exceed by a factor of 10 at the number of threads that those obtained in the Xeon Phi. The overhead of I/O explains part of these lack of performance in the Xeon Phi. On the one side, BWA does not take advantage of the 512-bit SIMD units available in the Xeon Phi. On the other side, the effect of memory containment is attenuated thanks to the usage of replicas of the reference genome. This effect is greater in the case of the Xeon system because it has two clearly separated NUMA nodes.

In the case of Xeon Phi, the use of replicas allows increasing the number of concurrent accesses. But in this case, the improvement is not so significant because there are no clearly-separated NUMA nodes and the performance is improved until a point is reached where the ring connecting all the memory banks is saturated.

7.4 Analysis of Thread Groups and Data Replicas in DP, DR and DPR

7.4.1 Implementation

One of the factors associated with the design of the strategies of DP, DR and DPR is the number of thread groups in them. With these experiments we want to answer questions like what is the ideal number of groups that should be used in a particular architecture? or can we use any practical rule that can be applied to deduce such number?

We executed DP, DR and DPR varying the number of thread groups that were used (x groups in Xeon and y groups in Xeon Phi). In all cases, we used the maximal number of available cores. This means that 24 and 240 threads were used in all the cases in the Xeon part (Sandman) and in the Xeon Phi part (Sandman-Xeon Phi), respectively. The number of OpenMP threads used at each instance (tx and ty) in the Xeon and Xeon Phi are $tx = 24/x$ and $ty = 240/y$. For instance, if 2 thread groups are applied in the whole system, each thread group in Xeon utilizes 12 OpenMP threads and each thread group in Xeon Phi utilizes 120 OpenMP threads. If 3 thread groups are tested in the system, each thread group in Xeon uses 8 threads and each thread group in Xeon Phi uses 80 threads.

According to these discussion above, thus, the number of thread groups, 2, 3, 4, 6 and 12, could stand for the number of OpenMP threads 12, 8, 6, 4 and 2 in each thread group in Figure 7.11 and 120, 80, 60, 40 and 20 in each thread group in Figure 7.12.

For DR and DPR, the number of thread groups is limited by memory consumption in each alignment process because we replicated the genome index in memory. Basically, we measured that one completed instance of BWA aligner consumes about 5 GB memory, which is mainly required to store the human genome sequence reference (4.4GB). According to the memory size available on the Xeon part(64 GB) and on the Xeon Phi part(16 GB), 12 and 3 should be the maximum number of thread groups that can be launched at each parts. We tested DP, DR and DPR using thread groups of 2, 3, 4, 6 and 12 on both systems, but in the cases of DPR and DR only 3 thread groups were applied in the experiment of Xeon Phi due to the memory limitations in the system of S1.

7.4.2 Performance Analysis

Results are shown in Figures 7.11 and 7.12. It can be observed that the best performance is achieved when a small number of groups are used (2 or 3). Beyond these values, adding more groups worsens performance. These results seem coherent if we take into account the number of NUMA nodes in the system (2 in the case of Sandman and 1 in the case of Sandman Phi). Having more replicas helps distribute traffic between different memory banks and prevents a single bank from becoming congested.

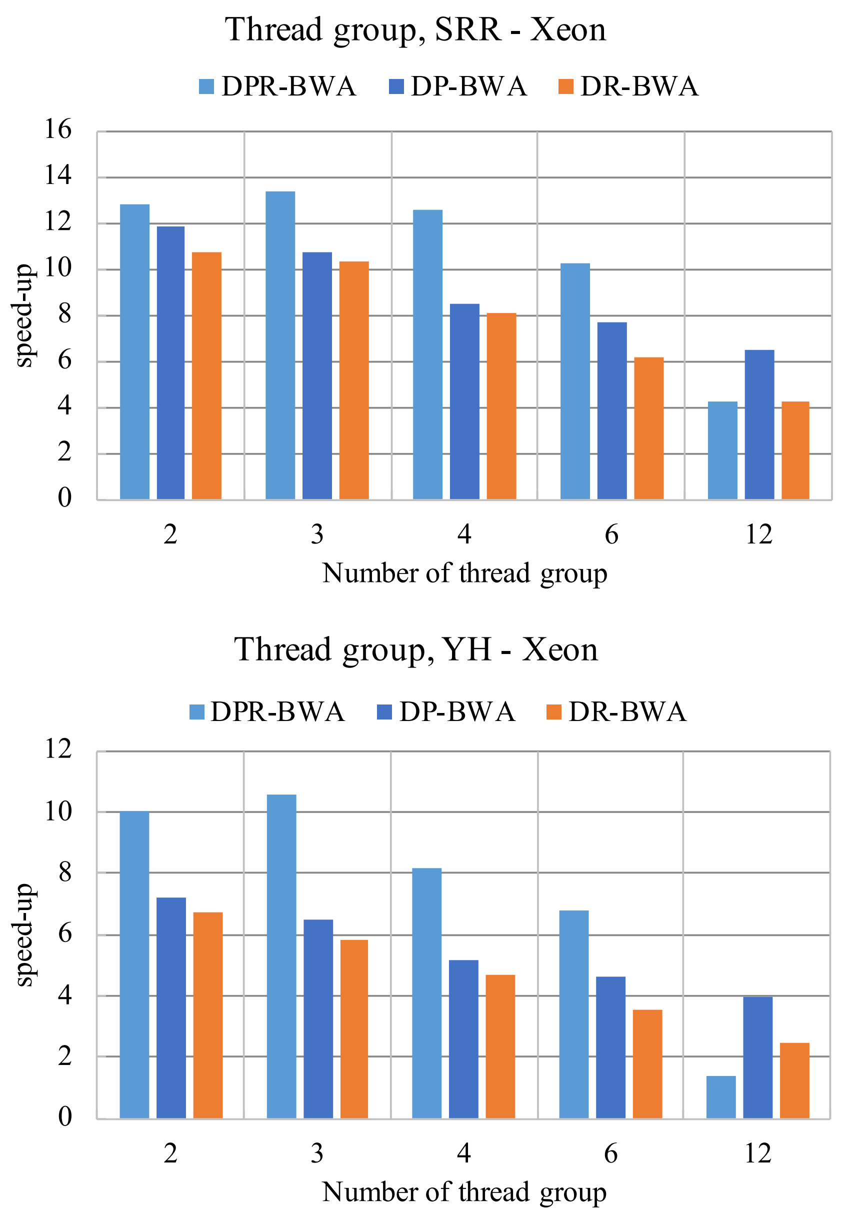


Fig. 7.11 DP, DR and DPR performance comparison on Sandman

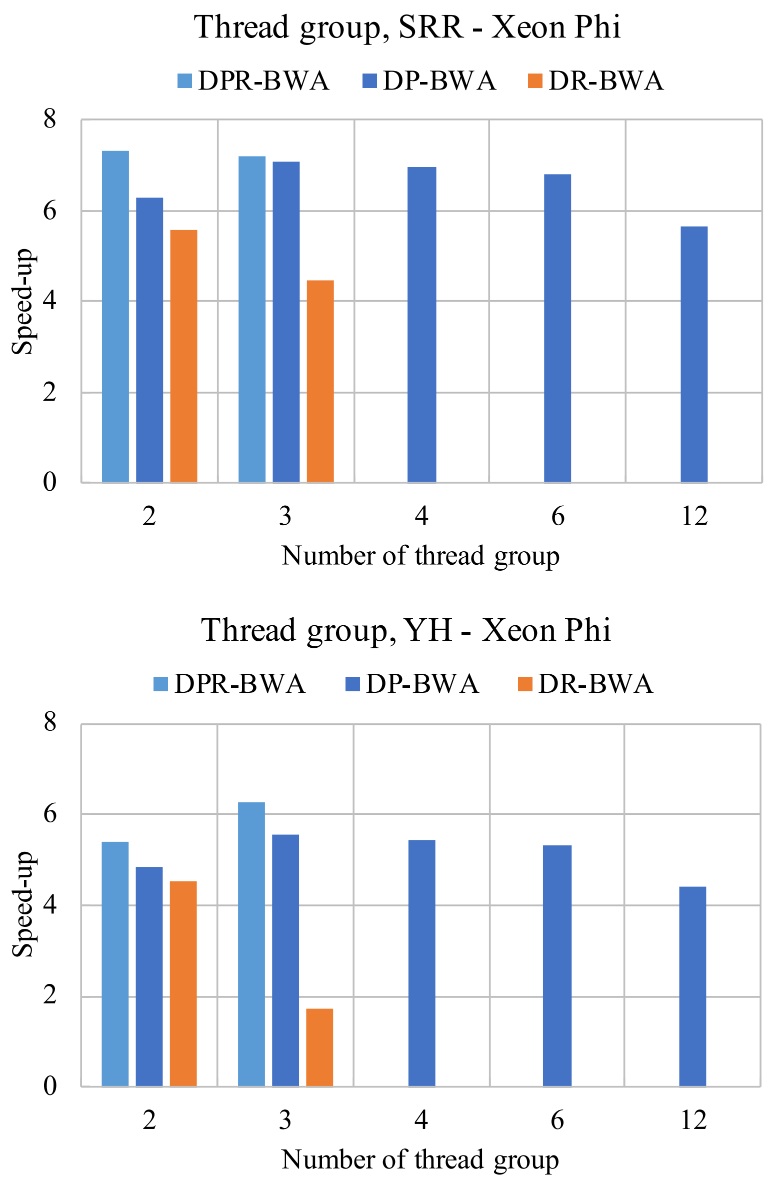


Fig. 7.12 DP, DR and DPR performance comparison on Sandman-Xeon Phi

However, adding more replicas increases the overhead and reduces the performance of the cache because there is more competition between different groups of threads without increasing the transference capacity of each memory bank. Threads that belong to each group also take advantage of those cache entries that contain the index positions which are read in the first stages of the alignment. By increasing the number of thread groups, the usage of index entries in the cache memory gets worsen and there is an increasing number of cache misses that does not compensate for the reduction of memory congestion.

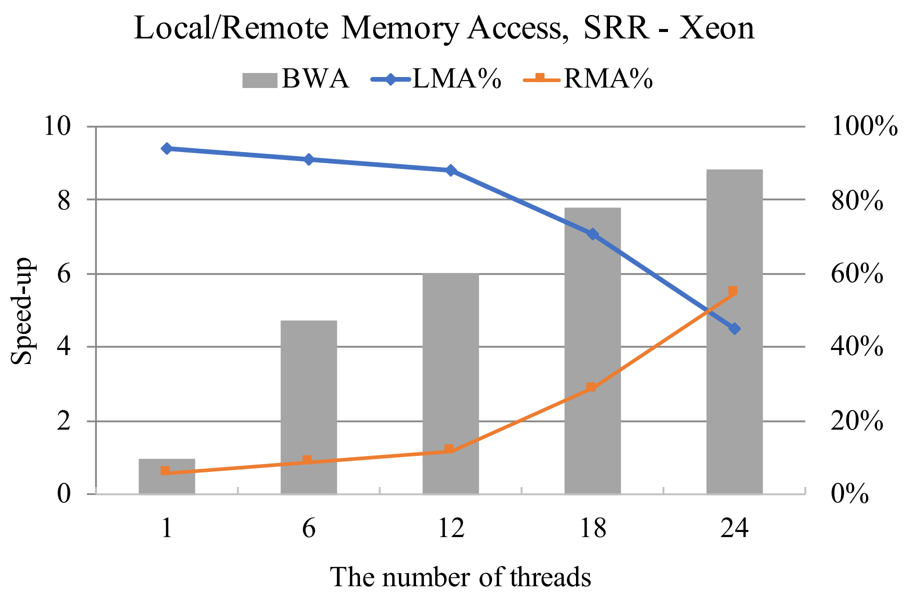


Fig. 7.13 BWA Memory Access

The performance improvements observed in these experiments are based on the idea that each group of threads will make accesses to the memory node of its NUMA domain, thus taking advantage of the greatest available bandwidth. Using the NUMATOP tool, we obtained measurements on the distribution of memory accesses between the different memory nodes. Figures 7.13 and 7.14 show the behavior of the original version of BWA and our DPR strategy when they were executed in Sandman, a system that has 2 NUMA nodes formed by a processor and a memory bank. The two figures show, on the one hand, the proportion of local memory accesses (LMA) and the speedup of the application compared to the sequential

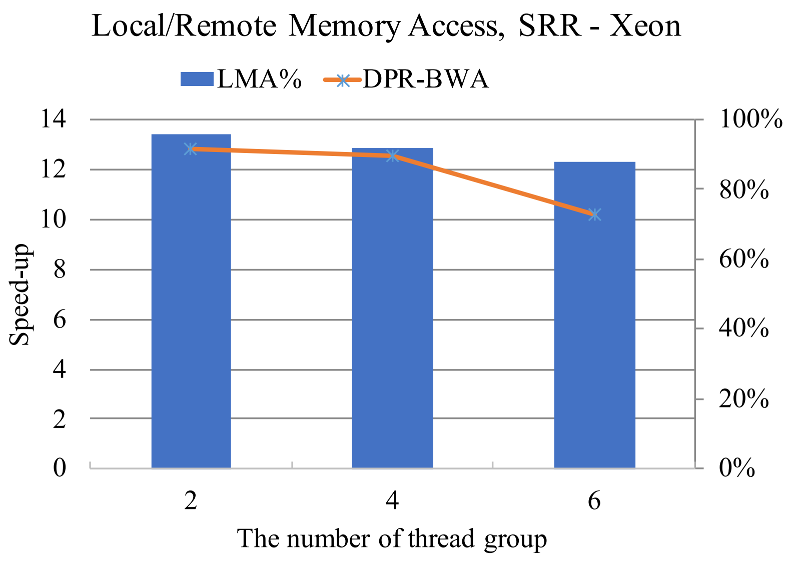


Fig. 7.14 DPR-BWA Memory Access

execution. As can be seen in figure 7.13, BWA exhibits a proportion of local memory access greater than 80% if it is executed in a single processor (12 threads).

When increasing the parallelism, the additional threads are executed in the second processor and all of them will generate remote accesses of memory because they have to access the node where the reference index is located. The LMA index decreases to just over 40%. As a result, remote accesses have a negative effect on the execution time of the application because latency is higher for remote accesses and they increase congestion in the node where the index is located. This negative impact on the execution time explains that when doubling from 12 to 24 threads the speed-up only improves by a factor of just over 2.

On the contrary, the use of instances (as shown in figure 7.14) in DPR always maintains a proportion of local accesses above 80% although the number of instances used is increased. There is no negative impact on memory access times nor does congestion occur. Anyway, as we saw earlier, the best speedup (factor 12, approximately) is obtained when the number of instances is low (2 or 4, which means one or two instances executed in each NUMA node). If the number of instances is too high (6 or more) the groups of threads compete and interfere with the use of the cache memory without improving the access times to the main memory.

7.5 Performance Evaluation of MDPR

7.5.1 Implementation

This last set of experiments analyze the performance of MDPR strategy, evaluating the three data distribution policies described in Chapter 5.

MDPR allows not only the execution of BWA aligner in the symmetric mode, but also in the distributed memory systems. The best execution time should be achieved if reads are distributed so that all the threads finish its jobs at the same time. The distribution of reads can be done in different ways taking into account that there is a trade-off between the overhead in which it is incurred and the load balance that can be achieved.

In our case, we evaluated a static mechanism that requires some executions in the native mode beforehand but does not incur much overheads at run time. We also tested two dynamic methods (even distribution and round-robin fashion), which incur in some overheads in execution time but do not require previous executions in advance.

In this experiments, we utilized the same datasets used in previous experiments (SRR+YH) and two different heterogeneous systems(S1+S2). The first one system (referred as S1) is the one equipped with an Intel Xeon and Xeon Phi(Sandman and Sandman-Xeon Phi). The second one system (referred as S2) is made of an Intel-based server and an AMD-based server(Penguin and Batman). Furthermore, we also tested in a large-scale system (S3) that comprises 6 nodes 17 sockets 25 NUMA and 252 threads made of Intel Xeon and AMD Opteron.

According to the results obtained in the evaluation of DPR with different number of thread groups, and system S1 has 2+1 sockets and 2+1 NUMA nodes, the following configurations of instance number were investigated in the system S1: 1+1(2), 2+1(3), 2+2(4), 2+3(5) and 4+2(6). The first value corresponds to the number of thread groups in the Xeon part and the second value corresponds to the number of thread groups in the Xeon Phi.

Similarly, system S2 and S3 possesses 4+4 sockets and 4+8 NUMA nodes, 6 nodes 17 sockets and 25 NUMA, respectively. Thus, the configurations evaluated in S2 are 1+1(2), 2+2(4), 4+4(8) and 4+8(12), in S3 are 6, 9, 17, 25. All available hardware threads were used in all cases of MDPR experiments (i.e., Intel Xeon and Xeon Phi could use 24 threads plus 240 threads, Intel server and AMD server could support 64 threads each, S3 could provide 252 threads).

7.5.2 Performance Analysis

In these figures, the cases of 4(2+2) and 6(4+2) instances in Figure 7.15, and the cases of 8(4+4) and 12(4+8) instances in Figure 7.16, we tried overload more than one instance in each socket in Sandman(2 sockets) and Batman(4 sockets). These columns in figures illustrate a small fluctuation in overall time-cost among these cases. Thus the bandwidth between sockets and local memory bank is high enough for one more instance of BWA aligner. Identical behavior with Intel Xeon Phi architecture. All available cores in the Intel Xeon Phi are expert in computation, and already utilized in all the cases. With more instances applied in one NUMA node in Xeon Phi, it does not increase memory locality and decrease memory congestion at all.

Memory congestion becomes a crucial influence under multiple instances in the system. In the cases of 2(1+1) and 3(2+1) instances in Figure 7.15, and 2(1+1), 4(2+2) and 8(4+4) instances in Figure 7.16, they show the process of each socket saturates with one instance in Sandman(2 sockets) and Batman(4 sockets). Cause of first-touch policy in the systems, the instance would consume its relative memory band in local socket when first thread is initial in system. More remote memory access exists in the cases of 2 instances in Figure 7.15, and 2, 4 instances in Figure 7.16. This explains why a decreasing time-cost in the cases of 3 instances in S1 and 8 instances in S2, when comparing them to previous cases in figures.

The case of 2(1+1) instance in static mode in Figure 7.15 represents sBWA 24+240 threads case in figure 7.1. Comparing to other static cases of MDPR (MDPR-static) with

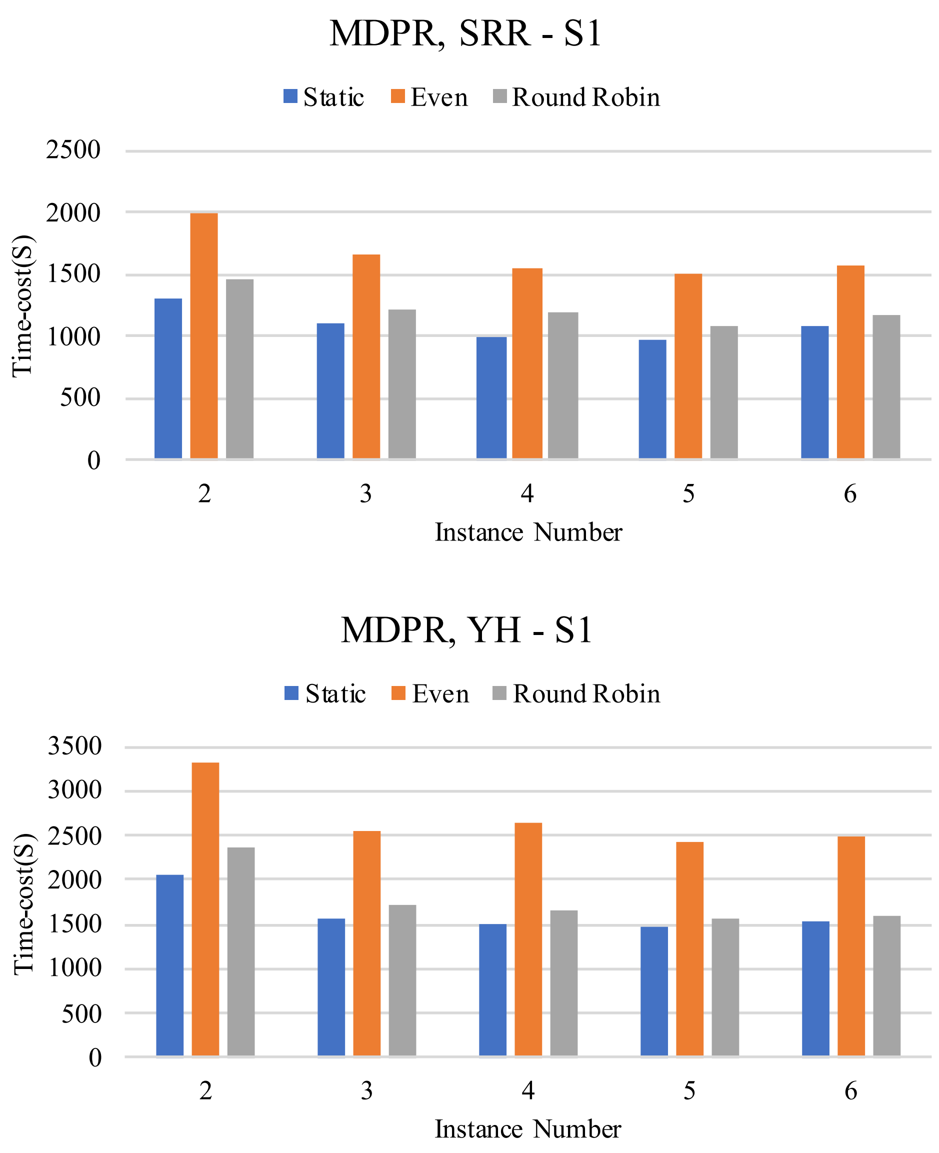
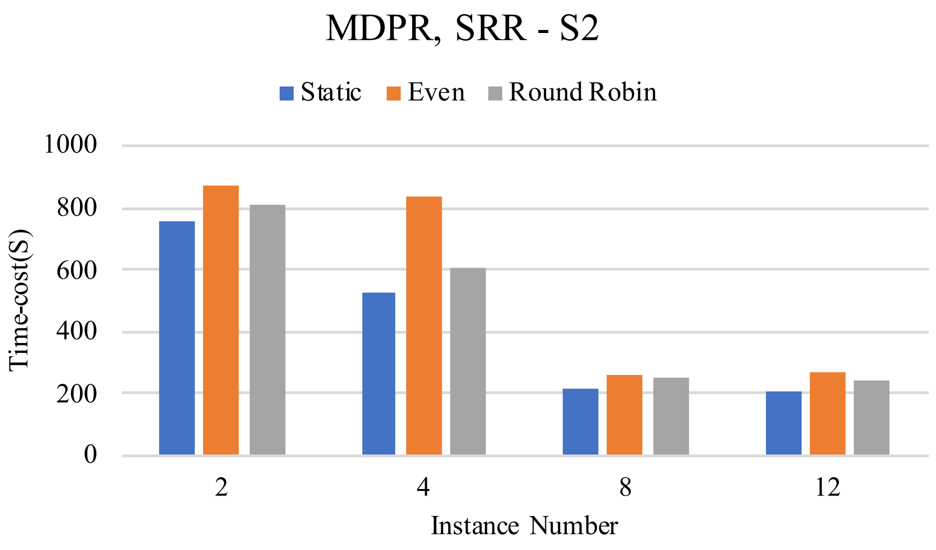


Fig. 7.15 Static, even and round-robin schedule distribution of MDPR strategy, performance comparison on the Intel Xeon and Xeon Phi(S1)



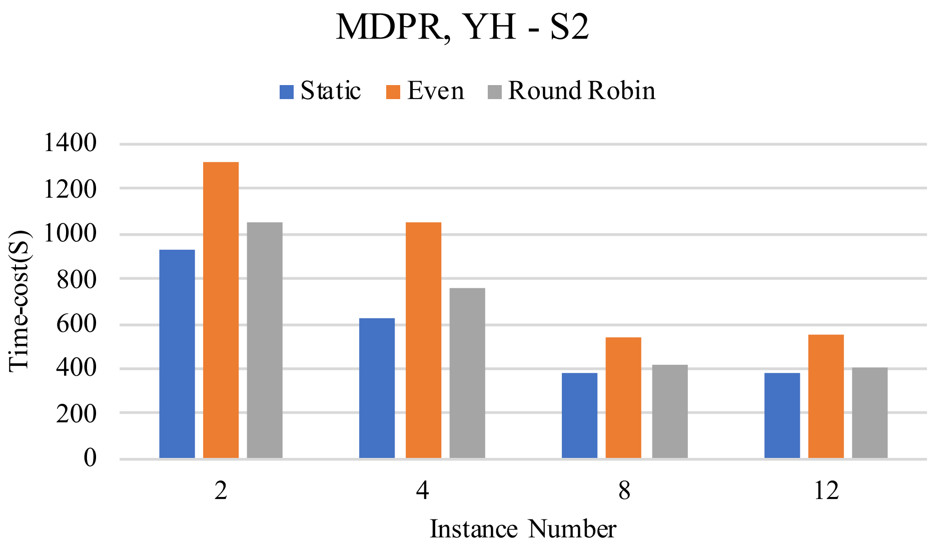


Fig. 7.16 Static, even and round-robin schedule distribution of MDPR strategy, performance comparison on the Intel and AMD NUMA nodes(S2)

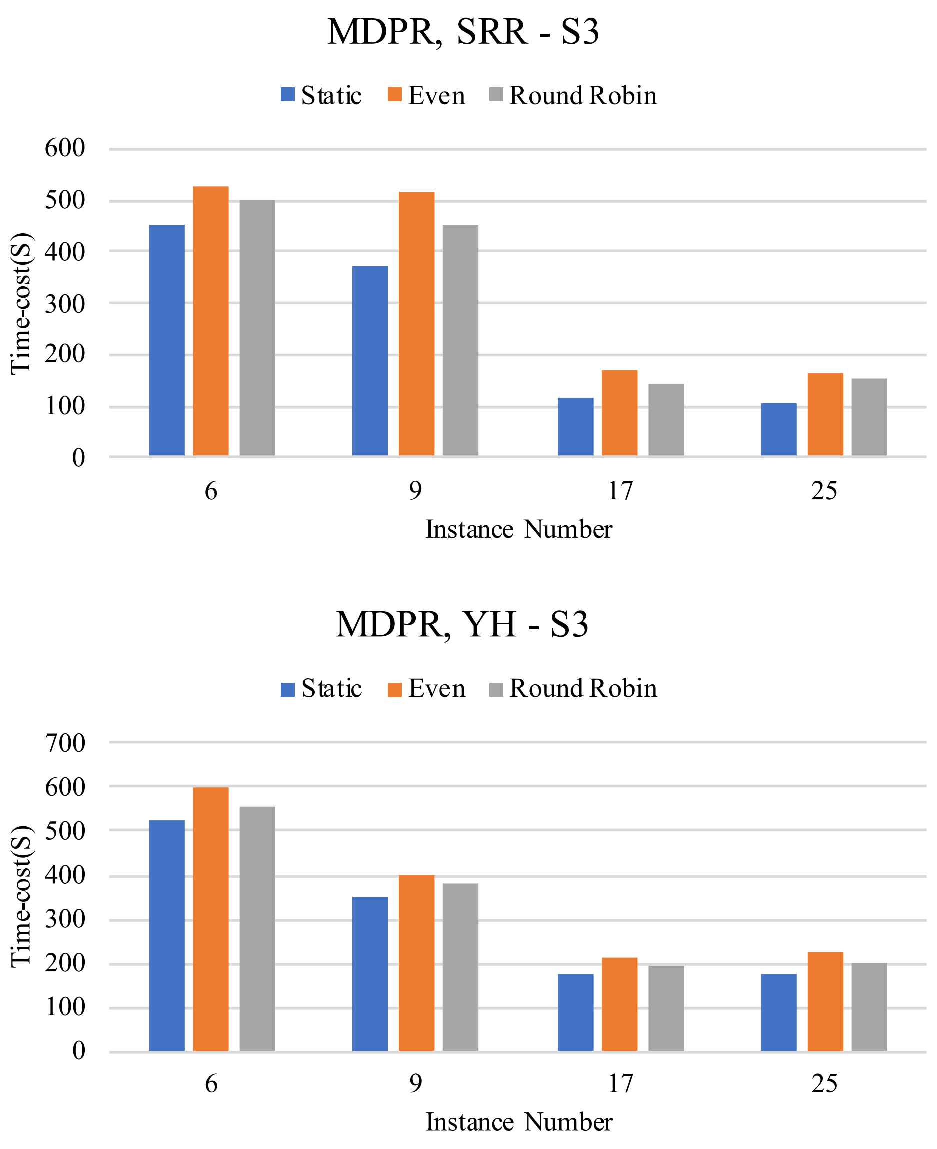


Fig. 7.17 Static, even and round-robin schedule distribution of MDPR strategy, performance comparison on multiple nodes(S3)

SRR dataset in S1 system, sBWA always costs the most wall time and MDPR-static reaches approximately 15-30% improvement in speed up with instance number gradually increases. This behavior could be seen as well as the case of MDPR-even and MDPR-round robin in S1 system.

As we know that even distribution disperses dataset in an even size among all instances no matter which system we utilize. The overall time-cost is determined by the slowest system. As previous experiments illustrated, Intel Xeon cores almost approaches 10X+ speed up than Xeon Phi cores in alignment. S1 has an absolutely different hardware architecture for alignment while S2 remains much similar in NUMA nodes. This is the reason which explains the large difference existed in time-cost between static distribution and even distribution in S1.

For the cases of round robin fashion, which performs between static distribution and even distribution, and approaches closely with static distribution in the overall time-cost. Round robin fashion is an automatic distributed mode among the instances in the MDPR, which takes advantages of the previous two distribution modes. A fixed size of chunk is assigned to each instance, and one more chunks will be automatically dispersed to the free-job instances when it completes the previous chunk. Round robin fashion could keep each instance occupying and least time-waiting. But it has one disadvantage that some resource is consumed for management of this fashion.

Finally, as shown in Figures 7.15, 7.16 and 7.17, the static distribution of MDPR achieves the best execution times in all cases. The even distribution is the worst strategy since it only makes an initial distribution of the data without subsequent corrections according the relative performance of the nodes. As an intermediate strategy, the round-robin method obtains results close to the static one without the need for preliminary setup computations. On the other hand, if we analyze the number of thread groups used, it is observed that the best results are obtained when the number of thread groups is adjusted to the number of NUMA nodes available in the system. This behavior has already been observed in the DPR experiments in the manycore system and is confirmed in the S2 system formed by two multicore servers.

In large-scale implementation of S3, it displays a identical phenomenon as argued above. Because we only communicate data start and data ending position with each instances, load balancing is obvious.

Chapter 8

Discussion and Conclusion

8.1 Conclusion

In this dissertation, we have evaluated different data parallelism strategies to execute BWA aligner in heterogeneous architectures made of multicore and/or manycore nodes. BWA has scalability limitations when large number of threads are used. The use of a shared reference genome generates problems of memory contention that explain such scalability problems. Firstly, we evaluated different versions of BWA that run on manycore systems in native mode, offload mode and symmetric mode. Secondly, we compared three different thread libraries and thread allocation affinity mechanisms that could play a significant impact in system architecture. Then, we analyzed different versions of BWA that use different strategies to parallelize the distribution of short reads or to replicate the reference genome. Finallly, we proposed and evaluated a generalized strategy (named MDPR) that uses two levels of parallelism which combine data replication and data parallelization. MDPR can be used both in manycore-based and multicore-based systems.

In our study we found that the symmetric mode execution strategy, sBWA, outperforms native and offload cases in all experiments, but sBWA still has a degraded scalability. Additionally, with the combined utilization of thread parallelism with OpenMP groups and multiple instances with MPI, we explored several data parallelization strategies on homogeneous architectures; namely, DP, DR and DPR, and we found out that DPR exhibits an excellent scalability and performance. Furthermore, we extended DPR to MDPR, a multiple process of BWA that uses DPR with OpenMP thread library in the symmetric mode. Static and dynamic schedule data distributions were evaluated in the MDPR strategy.

Table 8.1 BWA Performance Summary on Intel Xeon

Aligner	System	Dataset	Time-cost(s)	Speed-up	Group Num	Thread Num
BWA-Pthread	S1-Xeon	SRR	18658	1X	1	1
BWA-ALN-Xeon-Phi	S1-Xeon	SRR	1873	10X	1	24
		YH	2985	6.3X	1	24
DP	S1-Xeon	SRR	1580	11.8X	2	24
		YH	2591	7.2X	2	24
DR	S1-Xeon	SRR	1740	10.7X	2	24
		YH	2784	6.7X	2	24
DPR	S1-Xeon	SRR	1392	13.4X	3	24
		YH	1777	10.5X	3	24

Table 8.2 BWA Performance Summary on Intel Xeon Phi

Aligner	System	Dataset	Time-cost(s)	Speed-up	Group Num	Thread Num
BWA-ALN-Xeon-Phi	S1-Xeon Phi	SRR	2895	6.4X	1	240
		YH	4623	4X	1	240
mBWA	S1-Xeon Phi	SRR	2427	7.7X	1	240
		YH	3817	4.9X	1	240
DP	S1-Xeon Phi	SRR	2630	7.1X	3	240
		YH	3395	5.5X	3	240
DR	S1-Xeon Phi	SRR	3347	5.6X	2	240
		YH	4118	4.5X	2	240
DPR	S1-Xeon Phi	SRR	2555	7.3X	2	240
		YH	2955	6.3X	3	240

Tables 8.1, 8.2, 8.3 and 8.4 summarize the best results achieved in all our experiments by each BWA-based aligner presented in this work when they were executed with the maximum number of resources. Each entry in the tables shows the aligner that was used, the system on which it was executed, the dataset, the average execution time that was obtained, the corresponding speed-up, the number of groups or instances used and the total number of threads. The speed up is obtained by comparison with the original implementation of BWA (BWA-Pthread) using one thread with SRR dataset (first entry in table 8.1).

Table 8.3 BWA Performance Summary on S1

Aligner	System	Dataset	Time-cost(s)	Speed-up	Group Num	Thread Num
BWA-ALN-Xeon-Phi	S1	SRR	1156	16.1X	2(1+1)	24+240
		YH	1733	10.8X	2(1+1)	24+240
sBWA	S1	SRR	1301	14.3X	2(1+1)	24+240
		YH	2047	9.1X	2(1+1)	24+240
MDPR-static	S1	SRR	966	19.3X	5(2+3)	24+240
		YH	1477	12.6X	5(2+3)	24+240
MDPR-even	S1	SRR	1501	12.4X	5(2+3)	24+240
		YH	2443	7.6X	5(2+3)	24+240
MDPR-roundrobin	S1	SRR	1086	17.2X	5(2+3)	24+240
		YH	1584	11.8X	5(2+3)	24+240

Table 8.4 BWA Performance Summary on S2 and S3

Aligner	System	Dataset	Time-cost(s)	Speed-up	Group Num	Thread Num
MDPR-static	S2	SRR	204	91.5X	12(4+8)	64+64
		YH	385	48.5X	12(4+8)	64+64
	S3	SRR	109	171.2X	25	252
		YH	175	106.6X	25	252
MDPR-even	S2	SRR	260	71.8X	8(4+4)	64+64
		YH	535	34.9X	8(4+4)	64+64
	S3	SRR	167	111.7X	25	252
		YH	212	88X	17	252
MDPR-roundrobin	S2	SRR	245	76.2X	12(4+8)	64+64
		YH	411	45.4X	12(4+8)	64+64
	S3	SRR	145	128.7X	17	252
		YH	195	95.7X	17	252

Tables 8.1 and 8.2 show the results obtained by different BWA versions executed using a native or an offload method in the Xeon part or in the Xeon-Phi part, respectively, of our S1 system. In both cases we have also compared an official BWA implementation (referred as BWA-ALN-Xeon-Phi) from Intel Corporation [105]. BWA-ALN-Xeon-Phi was tested in native mode (Tables 8.1 and 8.2) and it was also tested in symmetric mode in S1 cluster, as shown in Table 8.3.

As seen in Table 8.1 and 8.2, our strategy DPR that uses data replication and data parallelization obtained better execution times in general. DPR achieved a speed-up of 13.4 and 10.5 for SRR and YH, respectively, in the Xeon case, and a speed-up of 7.3 and 6.3 in the Xeon-Phi case. Only mBWA achieved a slightly better result in the case of SRR dataset with a seep-up of 7.7.

Table 8.3 shows the best results obtained by the strategies running in symmetric mode on the entire S1 system. In this case, two versions of MDPR (with static distribution and with round-robin) obtain better results than the other two strategies that do not use data replication (BWA-ALN-Xeon-Phi) and sBWA. The best strategy is MDPR-static with speed-ups of 19.3 and 12.6, respectively, while MDPR-roundrobin achieves values of 17.2 and 11.8. It is worth noting that although our MDPR approach allows a significant improvement of performance in manycore systems based on Intel Xeon Phi, the characteristics of this accelerator (limited I/O capacity and main memory interconnected by a ring bus) limit its potential performance in comparison with multicore systems with NUMA architecture, if we take into account the total number of cores used. Speed-up values obtained at the S1 system are significantly lower that those obtained at S2 and S3 systems (see Table 8.4), but the total number of cores in S1 was greater than that used in S2 and S3.

Finally, Table 8.4 shows the results of the different MDPR variants when running on larger systems formed by several heterogeneous multicore nodes. MDPR with static distribution achieves the best results, followed by the round-robin version.

In summary, strategies that use data replication and data parallelization obtain better execution times in general. Given a certain architecture with a number of available threads, DPR strategies get better results than other strategies that do not use data replication. Data replication reduces memory contention and data parallelization increases memory locality that improves performance. In particular, MDPR (using either a static distribution or a dynamic round-robin distribution) constitutes a promising strategy for both multicore and manycore systems, and it outperforms Intel official version BWA-ALN-Xeon-Phi. On the other hand, although our approach allows the improvement of performance in manycore systems based on Intel Xeon Phi, the characteristics of this accelerator (limited I/O capacity, main memory interconnected by ring bus) limit its potential performance in comparison with multicore systems with NUMA architecture.

Comparing to other existing works, our research could be convenient for transiting to other programs without complicated coding skill. Although coding can be simple, achieving good performance is not easy in this type of systems. We have generated and evaluated a series of general strategies that could be applied to other sequence alignment tools that have similar working principles to the ones exhibited by BWA.

8.2 Future Work

As future lines we see several possible open paths:

1. A basic future work line could be to carry out a more detailed study of some parameter configurations that might play a significant role in the performance of MDPR. An example of this type of study would be the size of data chuncks, which could have different sizes depending on the characteristics of the system nodes.

2. The basic principles on which the MDPR strategy is based (distribution and data replication) could be adapted to other popular sequence alignment tools (such as BOWTIE or GEM), which also use a reference genome index as a basic element on which perform allocation operations..

3. The MDPR strategy is based on a hierarchical architecture that has shown its efficiency in medium-sized heterogeneous systems and has focused on reducing the cost of accesses to main memory. In a larger system, it is foreseeable that another source of delays will appear: file operations performed through a centralized/distributed file system. An interesting line of study would be to extend MDPR to efficiently access all files that contain sequence data and reference indexes, either by applying data replication techniques at the file level or by distributing data files between different file server nodes according to their proximity to computing nodes.

4. In general, genomic workflows are usually composed of different phases. In this work we have focused our study on BWA, a tool that is used in the initial phase of those

workflows. The following phases are usually carried out using other tools that have different degrees of parallelization. It would be interesting, however, to carry out a global study through all the workflow to establish a degree of parallelism for all the phases so that the entire workflow was executed with maximum efficiency on a set of nodes assigned from beginning to end of the workflow.

8.3 List of Publications

The work for this thesis has been published in the following papers:

1. **Chen, Shaolong, and Miquel A. Senar. “Accelerating BWA Aligner Using Multistage Data Parallelization on Multicore and Manycore Architectures.” ICCS 2016, Proceeding Computer Science 80 (2016): 2438-2442.**

   Our critical contributions in this paper concentrate on putting forward the strategies of multistage data parallelization for accelerating sequence alignment on heterogeneous cluster. We aim at the most prevalent sequence alignment application BWA. Firstly, parallelization is powered by hardware. We apply an heterogeneous cluster, the Intel multicore Xeon host server equipped with manycore coprocessor, Xeon Phi. Secondly, we have introduced the strategies of two-dimensional parallelization on the data, including genome short-read data splitting and index parallelization of reference. The experiments’ results on the heterogeneous cluster shows a highest speed-up of 4-fold by contrast with BWA aligner which is implemented on the Intel Xeon processor E5 with 24 threads.

2. **Chen, Shaolong, and Miquel A. Senar. "Improving Performance of Genomic Aligners on Intel Xeon Phi-Based Architectures." 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2018.**

   In this paper, We studied different solutions to efficiently execute BWA aligner on a heterogeneous system that consists of multicore processors and manycore accelerators. We proposed a multi-level strategy (MDPR) based on data parallelization and data replication. The MDPR strategy was executed using several hierarchical levels of parallelism in systems that contain multicore processors (Xeon) and manycore accelerators (Xeon Phi). Through a multi-process and multi-thread scheme which comprises replication and parallelization of certain data structures, we had obtained substantial improvements in BWA execution time. The strategy runs in symmetric mode using a hierarchical scheme that improves memory locality and reduces memory congestion on NUMA architectures as well as on Xeon Phi processors. Significant improvements were obtained in our experiments, comparing our proposal to other strategies that are executed in native, offload and symmetric mode.

3. **Chen, Shaolong, and Miquel A. Senar. "Exploring Efficient Data Parallelism for Genome Read Mapping on Multicore and Manycore Architectures." Parallel Computing (2018): Submitted, under the 2nd review.**

   In this study we have analyzed different modes of execution of BWA on a system that incorporates an accelerator. Although coding can be simple, achieving good performance is not easy in this type of systems, as our results show. From all these strategies, we have found one that combines data parallelization and data replication (named MDPR) that provides the best performance. We have applied this strategy on a heterogeneous system consisting on a server with Intel Xeon multicore processors and Xeon Phi manycore accelerator. MDPR is not limited to systems with accelerators only. Its generic design has allowed us to use it on another heterogeneous system based on two multicore servers (one equipped with AMD nodes and one equipped with Intel nodes). In all these hardware configurations, we have tested two dynamic and one static modes of data distribution in MDPR. Our experimental results show that best results are obtained for MDPR when static mode of data distribution is applied. Additional, round-robin fashion in MDPR achieves approximate performance with static distribution with less configurations in advance.

References

[1] Aisling O'Driscoll, Jurate Daugelaite, and Roy D Sleator. 'big data', hadoop and cloud computing in genomics. *Journal of biomedical informatics*, 46(5):774–781, 2013.

[2] Crystal Boddie, Tara Kirk Sell, and Matthew Watson. Federal funding for health security in fy2015. *Biosecurity and bioterrorism: biodefense strategy, practice, and science*, 12(4):163–177, 2014.

[3] Kris A Wetterstrand. Dna sequencing costs: data from the nhgri genome sequencing program (gsp). 2013. *URL http://www. genome. gov/sequencingcosts*, 2016.

[4] Christine F Baes, Marlies A Dolezal, James E Koltes, Beat Bapst, Eric Fritz-Waters, Sandra Jansen, Christine Flury, Heidi Signer-Hasler, Christian Stricker, Rohan Fernando, et al. Evaluation of variant identification methods for whole genome sequencing data in dairy cattle. *BMC genomics*, 15(1):948, 2014.

[5] Ian Buck. Gpubench: Evaluating gpu performance for numerical and scientific application. In *Proc. 1st ACM Workshop General-Purpose Computing on Graphics Processors (GPˆ 2'04)*, 2004.

[6] Lincoln Stein. Genome annotation: from sequence to biology. *Nature reviews genetics*, 2(7):493, 2001.

[7] César Allande Álvarez. Modeling performance degradation in openmp memory bound applications on multicore multisocket systems. 2015.

[8] Rolf Rabenseifner, Georg Hager, Gabriele Jost, and Rainer Keller. Hybrid mpi and openmp parallel programming. In *PVM/MPI*, page 11, 2006.

[9] Jeffrey G Reid, Andrew Carroll, Narayanan Veeraraghavan, Mahmoud Dahdouli, Andreas Sundquist, Adam English, Matthew Bainbridge, Simon White, William Salerno, Christian Buhay, et al. Launching genomics into the cloud: deployment of mercury, a next generation sequence analysis pipeline. *BMC bioinformatics*, 15(1):30, 2014.

[10] Stephan Pabinger, Andreas Dander, Maria Fischer, Rene Snajder, Michael Sperk, Mirjana Efremova, Birgit Krabichler, Michael R Speicher, Johannes Zschocke, and Zlatko Trajanoski. A survey of tools for variant analysis of next-generation genome sequencing data. *Briefings in bioinformatics*, 15(2):256–278, 2014.

[11] Yingbo Cui, Xiangke Liao, Xiaoqian Zhu, Bingqiang Wang, and Shaoliang Peng. mbwa: A massively parallel sequence reads aligner. In *8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014)*, pages 113–120. Springer, 2014.

[12] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.

[13] Xinmin Tian, Hideki Saito, Serguei V Preis, Eric N Garcia, Sergey S Kozhukhov, Matt Masten, Aleksei G Cherkasov, and Nikolay Panchenko. Practical simd vectorization techniques for intel® xeon phi coprocessors. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1149–1158. IEEE, 2013.

[14] Liang Ping. Speeding up large-scale next generation sequencing data analysis with pbwa. *Journal of Applied Bioinformatics & Computational Biology*, 2012.

[15] Josefina Lenis and Miquel Angel Senar. A performance comparison of data and memory allocation strategies for sequence aligners on numa architectures. *Cluster Computing*, 20(3):1909–1924, 2017.

[16] Shaolong Chen and Miquel A Senar. Accelerating bwa aligner using multistage data parallelization on multicore and manycore architectures. *Procedia Computer Science*, 80:2438–2442, 2016.

[17] Shaolong Chen and Miquel A Senar. Improving performance of genomic aligners on intel xeon phi-based architectures. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 570–578. IEEE, 2018.

[18] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[19] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10(3):R25, 2009.

[20] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[21] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.

[22] Jing Zhang, Heshan Lin, Pavan Balaji, and Wu-chun Feng. Optimizing burrows-wheeler transform-based sequence alignment on multicore architectures. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 377–384. IEEE, 2013.

[23] Stephen L Olivier, Allan K Porterfield, Kyle B Wheeler, and Jan F Prins. Scheduling task parallelism on multi-socket multicore systems. In *Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers*, pages 49–56. ACM, 2011.

[24] Emerson de Araujo Macedo and Azzedine Boukerche. Hybrid mpi/openmp strategy for biological multiple sequence alignment with dialign-tx in heterogeneous multicore clusters. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 418–425. IEEE, 2011.

[25] Nagarajan Kathiresan, Mohamed Ramzi Temanni, and Rashid Al-Ali. Performance improvement of bwa mem algorithm using data-parallel with concurrent parallelization. In *Parallel, Distributed and Grid Computing (PDGC), 2014 International Conference on*, pages 406–411. IEEE, 2014.

[26] Yu-Ting Chen, Jason Cong, Jie Lei, and Peng Wei. A novel high-throughput acceleration engine for read alignment. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 199–202. IEEE, 2015.

[27] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. Gpu-accelerated bwa-mem genomic mapping algorithm using adaptive load balancing. In *International Conference on Architecture of Computing Systems*, pages 130–142. Springer, 2016.

[28] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. An fpga-based systolic array to accelerate the bwa-mem genomic mapping algorithm. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*, pages 221–227. IEEE, 2015.

[29] Ho-Cheung Ng, Shuanglong Liu, and Wayne Luk. Reconfigurable acceleration of genetic sequence alignment: A survey of two decades of efforts. In *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pages 1–8. IEEE, 2017.

[30] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

[31] Ernst Joachim Houtgast, Vlad-Mihai Sima, Giacomo Marchiori, Koen Bertels, and Zaid Al-Ars. Power-efficiency analysis of accelerated bwa-mem implementations on heterogeneous computing platforms. In *ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on*, pages 1–8. IEEE, 2016.

[32] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. Gpu-accelerated bwa-mem genomic mapping algorithm using adaptive load balancing. In *International Conference on Architecture of Computing Systems*, pages 130–142. Springer, 2016.

[33] Petr Klus, Simon Lam, Dag Lyberg, Ming Sin Cheung, Graham Pullan, Ian McFarlane, Giles SH Yeo, and Brian YH Lam. Barracuda-a fast short read sequence aligner using graphics processing units. *BMC research notes*, 5(1):27, 2012.

[34] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. Cushaw: a cuda compatible short read aligner to large genomes based on the burrows–wheeler transform. *Bioinformatics*, 28(14):1830–1837, 2012.

[35] Chi-Man Liu, Thomas Wong, Edward Wu, Ruibang Luo, Siu-Ming Yiu, Yingrui Li, Bingqiang Wang, Chang Yu, Xiaowen Chu, Kaiyong Zhao, et al. Soap3: ultra-fast gpu-based parallel alignment tool for short reads. *Bioinformatics*, 28(6):878–879, 2012.

[36] José M Abuín, Juan C Pichel, Tomás F Pena, and Jorge Amigo. Bigbwa: approaching the burrows–wheeler aligner to big data technologies. *Bioinformatics*, 31(24):4003–4005, 2015.

[37] José M Abuín, Juan C Pichel, Tomás F Pena, and Jorge Amigo. Sparkbwa: speeding up the alignment of high-throughput dna sequencing data. *PloS one*, 11(5):e0155461, 2016.

[38] Suejb Memeti and Sabri Pllana. Accelerating dna sequence analysis using intel (r) xeon phi (tm). In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 3, pages 222–227. IEEE, 2015.

[39] Lipeng Wang, Yuandong Chan, Xiaohui Duan, Haidong Lan, Xiangxu Meng, and Weiguo Liu. Xsw: Accelerating biological database search on xeon phi. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 950–957. IEEE, 2014.

[40] Sze-Hang Chan, Jeanno Cheung, Edward Wu, Heng Wang, Chi-Man Liu, Xiaoqian Zhu, Shaoliang Peng, Ruibang Luo, and Tak-Wah Lam. Mica: A fast short-read aligner that takes full advantage of intel many integrated core architecture (mic). *arXiv preprint arXiv:1402.4876*, 2014.

[41] Charlotte Herzeel, Thomas J Ashby, Pascal Costanza, and Wolfgang De Meuter. Resolving load balancing issues in bwa on numa multicore architectures. In *International Conference on Parallel Processing and Applied Mathematics*, pages 227–236. Springer, 2013.

[42] Alireza Mashaghi and Allard Katan. A physicist's view of dna. *arXiv preprint arXiv:1311.2545*, 2013.

[43] Wolfram Saenger. *Principles of nucleic acid structure*. Springer Science & Business Media, 2013.

[44] Tyra G Wolfsberg, Johanna McEntyre, and Gregory D Schuler. Guide to the draft human genome. *Nature*, 409(6822):824, 2001.

[45] ENCODE Project Consortium et al. Identification and analysis of functional elements in 1% of the human genome by the encode pilot project. *Nature*, 447(7146):799, 2007.

[46] Michael A Quail, Miriam Smith, Paul Coupland, Thomas D Otto, Simon R Harris, Thomas R Connor, Anna Bertoni, Harold P Swerdlow, and Yong Gu. A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers. *BMC genomics*, 13(1):341, 2012.

[47] Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of next-generation sequencing systems. *BioMed Research International*, 2012, 2012.

[48] The project WITDOM. Witdom use cases. *URL http://www.witdom.eu/content/witdom-use-cases*, 2018.

[49] Robert C Edgar and Serafim Batzoglou. Multiple sequence alignment. *Current opinion in structural biology*, 16(3):368–373, 2006.

[50] Vijini Mallawaarachchi. Pairwise sequence alignment using biopython. *URL https://towardsdatascience.com/pairwise-sequence-alignment-using-biopython-d1a9d0ba861f*, 2017.

[51] SB Needleman. Needleman-wunsch algorithm for sequence similarity searches. *J Mol Biol*, 48:443–453, 1970.

[52] Richard Mott. Smith–waterman algorithm. *e LS*, 2001.

[53] Vijini Mallawaarachchi. Pairwise sequence alignment using biopython. *URL https://towardsdatascience.com/pairwise-sequence-alignment-using-biopython-d1a9d0ba861f*, 2017.

[54] Heng Li, Jue Ruan, and Richard Durbin. Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome research*, pages gr–078212, 2008.

[55] Ruiqiang Li, Yingrui Li, Karsten Kristiansen, and Jun Wang. Soap: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.

[56] Heshan Lin, Xiaosong Ma, Praveen Chandramohan, Al Geist, and Nagiza Samatova. Efficient data access for parallel blast. In *null*, page 72b. IEEE, 2005.

[57] Faraz Hach, Fereydoun Hormozdiari, Can Alkan, Farhad Hormozdiari, Inanc Birol, Evan E Eichler, and S Cenk Sahinalp. mrsfast: a cache-oblivious algorithm for short-read mapping. *Nature methods*, 7(8):576, 2010.

[58] Hao Lin, Zefeng Zhang, Michael Q Zhang, Bin Ma, and Ming Li. Zoom! zillions of oligos mapped. *Bioinformatics*, 24(21):2431–2437, 2008.

[59] Stephen M Rumble, Phil Lacroute, Adrian V Dalca, Marc Fiume, Arend Sidow, and Michael Brudno. Shrimp: accurate mapping of short color-space reads. *PLoS computational biology*, 5(5):e1000386, 2009.

[60] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and Inanç Birol. Abyss: a parallel assembler for short read sequence data. *Genome research*, pages gr–089532, 2009.

[61] Shaun D Jackman, Benjamin P Vandervalk, Hamid Mohamadi, Justin Chu, Sarah Yeo, S Austin Hammond, Golnaz Jahesh, Hamza Khan, Lauren Coombe, Rene L Warren, et al. Abyss 2.0: resource-efficient assembly of large genomes using a bloom filter. *Genome research*, pages gr–214346, 2017.

[62] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.

[63] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10(3):R25, 2009.

[64] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357, 2012.

[65] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[66] Christian Otto, Peter F Stadler, and Steve Hoffmann. Fast and sensitive mapping of bisulfite-treated sequencing data. *Bioinformatics*, 28(13):1698–1704, 2012.

[67] Yingbo Cui, Xiangke Liao, Xiaoqian Zhu, Bingqiang Wang, and Shaoliang Peng. B-mic: an ultrafast three-level parallel sequence aligner using mic. *Interdisciplinary Sciences: Computational Life Sciences*, 8(1):28–34, 2016.

[68] Charlotte Herzeel, Pascal Costanza, T Ashby, and Roel Wuyts. Performance analysis of bwa alignment. Technical report, Technical Report Exascience Life Lab, 2013.

[69] Jason O'Rawe, Tao Jiang, Guangqing Sun, Yiyang Wu, Wei Wang, Jingchu Hu, Paul Bodily, Lifeng Tian, Hakon Hakonarson, W Evan Johnson, et al. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing. *Genome medicine*, 5(3):28, 2013.

[70] Diksha Garg, Ankita Jiwan, and Shailendra Singh. Computational approaches for variant identification. *International Journal of Computer Applications*, 165(8), 2017.

[71] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 2010.

[72] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[73] Julio Frenk, Lincoln Chen, Zulfiqar A Bhutta, Jordan Cohen, Nigel Crisp, Timothy Evans, Harvey Fineberg, Patricia Garcia, Yang Ke, Patrick Kelley, et al. Health professionals for a new century: transforming education to strengthen health systems in an interdependent world. *The lancet*, 376(9756):1923–1958, 2010.

[74] Ken Chen, John W Wallis, Michael D McLellan, David E Larson, Joelle M Kalicki, Craig S Pohl, Sean D McGrath, Michael C Wendl, Qunyuan Zhang, Devin P Locke,

et al. Breakdancer: an algorithm for high-resolution mapping of genomic structural variation. *Nature methods*, 6(9):677, 2009.

[75] Hui Yang and Kai Wang. Genomic variant annotation and prioritization with annovar and wannovar. *Nature protocols*, 10(10):1556, 2015.

[76] Wikipedia. Next generation tools for the annotation of human snps. *URL https://en.wikipedia.org/wiki/SNP$_a$nnotation*, 2018.

[77] Pablo Cingolani, Adrian Platts, Le Lily Wang, Melissa Coon, Tung Nguyen, Luan Wang, Susan J Land, Xiangyi Lu, and Douglas M Ruden. A program for annotating and predicting the effects of single nucleotide polymorphisms, snpeff: Snps in the genome of drosophila melanogaster strain w1118; iso-2; iso-3. *Fly*, 6(2):80–92, 2012.

[78] Kai Wang, Mingyao Li, and Hakon Hakonarson. Annovar: functional annotation of genetic variants from high-throughput sequencing data. *Nucleic acids research*, 38(16):e164–e164, 2010.

[79] Hashem A Shihab, Julian Gough, David N Cooper, Ian NM Day, and Tom R Gaunt. Predicting the functional consequences of cancer-associated amino acid substitutions. *Bioinformatics*, 29(12):1504–1510, 2013.

[80] Emidio Capriotti, Remo Calabrese, and Rita Casadio. Predicting the insurgence of human genetic diseases associated to single point protein mutations with support vector machines and evolutionary information. *Bioinformatics*, 22(22):2729–2734, 2006.

[81] Vladimir Makarov, Tina O'grady, Guiqing Cai, Jayon Lihm, Joseph D Buxbaum, and Seungtai Yoon. Anntools: a comprehensive and versatile annotation toolkit for genomic variants. *Bioinformatics*, 28(5):724–725, 2012.

[82] Jana Marie Schwarz, Christian Rödelsperger, Markus Schuelke, and Dominik Seelow. Mutationtaster evaluates disease-causing potential of sequence alterations. *Nature methods*, 7(8):575, 2010.

[83] Anthony G Doran and Christopher J Creevey. Snpdat: easy and rapid annotation of results from de novo snp discovery projects for model and non-model organisms. *BMC bioinformatics*, 14(1):45, 2013.

[84] Hsiang-Yu Yuan, Jen-Jie Chiou, Wen-Hsien Tseng, Chia-Hung Liu, Chuan-Kun Liu, Yi-Jung Lin, Hui-Hung Wang, Adam Yao, Yuan-Tsong Chen, and Chun-Nan Hsu. Fastsnp: an always up-to-date and extendable service for snp function analysis and prioritization. *Nucleic acids research*, 34(suppl_2):W635–W641, 2006.

[85] Christoph Lameter. Numa (non-uniform memory access): An overview. *Queue*, 11(7):40, 2013.

[86] Iraklis Psaroudakis, Stefan Kaestle, Matthias Grimmer, Daniel Goodman, Jean-Pierre Lozi, and Tim Harris. Analytics with smart arrays: adaptive and efficient language-independent data. In *Proceedings of the Thirteenth EuroSys Conference*, page 17. ACM, 2018.

[87] Tim Cramer, Dirk Schmidl, Michael Klemm, and Dieter an Mey. Openmp programming on intel r xeon phi tm coprocessors: An early performance comparison. In *Proc. Many Core Appl. Res. Community (MARC) Symp*, pages 38–44, 2012.

[88] Rezaur Rahman. *Intel® Xeon Phi™ Coprocessor Architecture and Tools: The Guide for Application Developers*. Apress, 2013.

[89] James Reinders. An overview of programming for intel xeon processors and intel xeon phi coprocessors. *Intel Corporation, Santa Clara*, 2012.

[90] Jianmin Chen, Xi Tao, Zhen Yang, Jih-Kwon Peir, Xiaoyuan Li, and Shih-Lien Lu. Guided region-based gpu scheduling: utilizing multi-thread parallelism to hide memory latency. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 441–451. IEEE, 2013.

[91] Michela Becchi and Patrick Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers*, pages 29–40. ACM, 2006.

[92] Jun Nakashima and Kenjiro Taura. Massivethreads: A thread library for high productivity languages. In *Concurrent Objects and Beyond*, pages 222–238. Springer, 2014.

[93] Kyle B Wheeler, Richard C Murphy, and Douglas Thain. Qthreads: An api for programming with millions of lightweight threads. 2008.

[94] Robert D Blumofe and Dionisios Papadopoulos. Hood: A user-level threads library for multiprogrammed multiprocessors. Technical report, Tech. rep., Department of Computer Science, University of Texas at Austin.[], 1999.

[95] Robert D Blumofe, Christopher F Joerg, Bradley C Kuszmaul, Charles E Leiserson, Keith H Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *Journal of parallel and distributed computing*, 37(1):55–69, 1996.

[96] Wikipedia. Fork–join model. *URL https://en.wikipedia.org/wiki/Fork2018.*

[97] Intel Cilk Plus. A brief history of cilk. 2018. URL https://www.cilkplus.org/cilk-history, *2016.*

[98] Edsger W Dijkstra. Solution of a problem in concurrent programming control. In Pioneers and Their Contributions to Software Engineering*, pages 289–294. Springer, 2001.*

[99] Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. Journal of the ACM (JACM)*, 46(5):720–748, 1999.*

[100] Shaolong Chen and Miquel A Senar. Exploring efficient data parallelism for genome read mapping on multicore and manycore architectures. Parallel Computing.

[101] *1000 Genomes Project Consortium et al. A global reference for human genetic variation.* Nature*, 526(7571):68, 2015.*

[102] *J Wang, Y Li, R Luo, B Liu, Y Xie, Z Li, X Fang, H Zheng, J Qin, B Yang, et al. Updated genome assembly of yh: the first diploid genome sequence of a han chinese individual (version 2, 07/2012).* GigaScience Database*, 2012.*

[103] *1000 Genomes Project. 1000 genomes project.* URL https://en.wikipedia.org/wiki/1000$_{Genomes Project}$, 2018.

[104] *Na Cai, Tim B Bigdeli, Warren W Kretzschmar, Yihan Li, Jieqin Liang, Jingchu Hu, Roseann E Peterson, Silviu Bacanu, Bradley Todd Webb, Brien Riley, et al. 11,670 whole-genome sequences representative of the han chinese population from the converge project.* Scientific data*, 4:170011, 2017.*

[105] *Charles (Intel) You, Liang (Intel); Congdon. Building and optimizing bwa* aln 0.5.10 for intel® xeon phi™ coprocessors.* URL https://github.com/intel-mic/bwa-aln-xeon-phi-0.5.10*, 2014.*

[106] *Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In* European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 97–104. Springer, 2004.*

[107] *Intel Corporation. Intel vtune amplifier.* URL https://en.wikipedia.org/wiki/VTune*, 2017.*