東海大學

資訊工程研究所

碩士論文

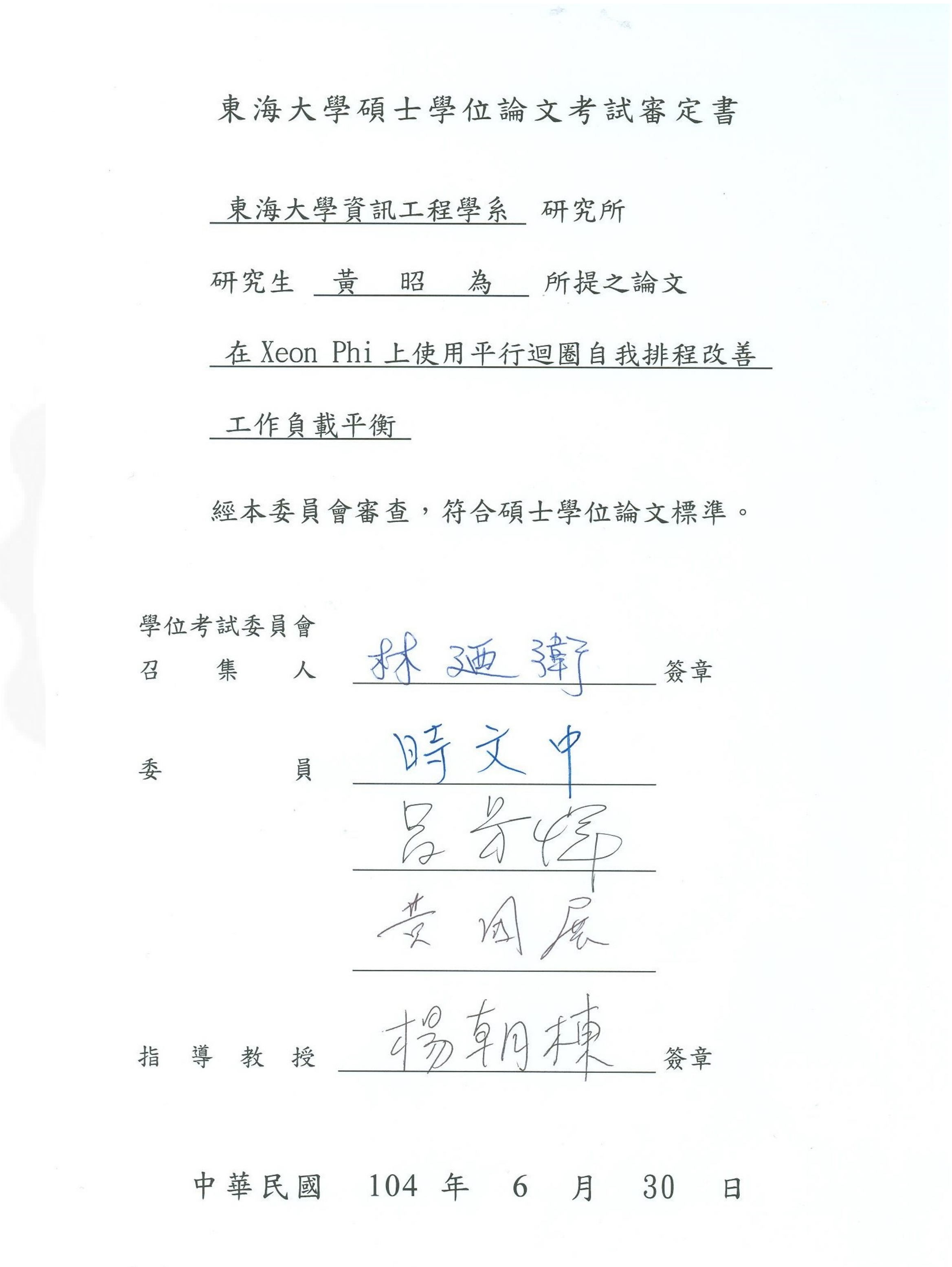
指導教授: 楊朝棟博士

在 Xeon Phi 上使用平行迴圈自我排程改善工作負載平衡

Improvement of Workload Balancing Using Parallel Loop Self-Scheduling on Xeon Phi

研究生: 黃昭為

中華民國一零四年



東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 黃 昭 為 所提之論文

在 Xeon Phi 上使用平行迴圈自我排程改善工作負載平衡

經本委員會審查，符合碩士學位論文標準。

學位考試委員會
召 集 人 林迺衛 簽章

委 員 時文中

呂芳懌

黃國展

指 導 教 授 楊朝棟 簽章

中華民國 104 年 6 月 30 日

摘　要

在本論文中，我們將研究如何改善計算機群的工作負載平衡，透過平行迴圈自我排程方法，我們使用混合 MPI 和 OpenMP 的 C 語言平行編程。根據計算節點性能權重為基礎分割迴圈的區塊。這個研究是使用 Xeon Phi 實施平行迴圈自我排程，藉由平行迴圈自我排程的特性改善異質節點之間的工作負載均衡。平行迴圈自我排程是由靜態排程和動態排程兩個部分所組成，在靜態的部分我們依照權重分配工作量的演算法，在動態的部分我們使用幾個知名的排程方法。Intel 近年來推出它們的新產品 Xeon Phi，它是類似 x86 架構的輔助處理器，它擁有大約 60 個核心且可以被當作單個計算節點，且擁有的計算能力不能忽視。在我們的實驗中我們將會使用多個計算節點。我們實驗四個應用，包括矩陣相乘、稀疏矩陣相乘、曼德博集合、電路滿足。結果將會列出使用平行迴圈自我排程，如何分配權重及排程方案能夠達到最好的性能。

關鍵字: Xeon Phi、Many-core、OpenMP、MPI、平行迴圈、自我排程

Abstract

In this paper, we will examine how to improve workload balancing on a computing cluster by a parallel loop self-scheduling scheme. We use hybrid MPI and OpenMP parallel programming in C language. The block partition loop is according to the performance weighting of compute nodes. This study implements parallel loop self-scheduling use Xeon Phi, with its characteristics to improve workload balancing between heterogeneous nodes. The parallel loop self-scheduling is composed of the static and dynamic allocation. A weighting algorithm is adopted in the static part while the well-known loop self-scheduling scheme is adopted in the dynamic part. In recent years, Intel promotes its new product Xeon Phi coprocessor, which is similar to the x86 architecture coprocessor. It has about 60 cores and can be regarded as a single computing node, with the computing power that cannot be ignored. In our experiment, we will use a plurality of computing nodes. We compute four applications, i.e., matrix multiplication, sparse matrix multiplication, Mandelbrot set computation, and the circuit satisfiability problem. Our results will show how to do the weight allocation and how to choose a scheduling scheme to achieve the best performance in the parallel loop self-scheduling.

Keyword: Xeon Phi, Many-core, OpenMP, MPI, Parallel Loop, Self-Scheduling

致謝詞

首先誠摯的感謝指導教授楊朝棟博士及劉榮春博士以及各位口試委員，兩位老師悉心的教導使我得以一窺高效能運算領域的深奧，不時的討論並指點我正確的方向，使我在這些年中獲益匪淺。老師對學問的嚴謹更是我輩學習的典範。

本論文的完成另外亦得感謝的大力協助。因為有你的體諒及幫忙，使得本論文能夠更完整而嚴謹。

兩年裡的日子，實驗室里共同的生活點滴，學術上的討論、言不及義的閒扯、讓人又愛又怕的宵夜、趕作業的革命情感、因為睡太晚而遮遮掩掩閃進實驗室...，感謝眾位學長姐、同學、學弟的共同砥礪，你們的陪伴讓兩年的研究生活變得絢麗多彩。

感謝不厭其煩的指出我研究中的缺失，且總能在我迷惘時為我解惑，也感謝同學的幫忙，恭喜我們順利走過這兩年。實驗室的當然也不能忘記，的幫忙及搞笑我銘感在心。

最後，謹以此文獻給我摯愛的雙親。

東海大學資訊工程學系 高效能實驗室 黃昭為 104 年 07 月

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the past few years, the CPU computing power keeps increasing. However, with more and more applications of scientific computing, improvement of CPU computing power alone appears inadequate. Recently, it is very common for people to use GPU as computing accelerator. To use GPU, one needs to know CUDA language; nevertheless, it is not easy to learn CUDA and algorithms written in CUDA are difficult to reuse [22]. Nowadays Intel launches Xeon Phi coprocessor families with x86-based core architecture. Each core of Xeon Phi supports 4 hardware threads. The feature is the usage of C or C ++ programming language. It can be performed on the Many Integrated Core (MIC) once the user adds a simple parameter using the compiler. In addition, it also supports for Open Multi-Processing (OpenMP), POSIX Threads (PThread), Message Passing Interface (MPI) and other parallel programming languages [21]. Compared to GPU, it only needs to pay a small amount of operation overhead to achieve the same performance.

Cluster computing is a computer system composed of numerous processors. When a processor in the cluster executes some application, it can communicate with other processors through message passing or memory sharing. Combining cluster computing with other technologies, especially cloud technology, creates a high-performance platform. In this fashion, the scalable computing cluster is becoming a high-performance and large-scale cloud computing platform [29].

The heterogeneous computer cluster consists of different kinds of CPUs for computing. If the assigned work is the same for each CPU in a heterogeneous computer cluster, the high performance CPUs usually complete the assigned work early and need to wait for low performance CPUs. Consequently, this situation results in a waste of waiting time. Therefore, CPU planning in a heterogeneous computer cluster is an important issue [23].

1.2 Thesis Goal and Contributions

The task loop self-scheduling is an important part in parallel and distributed systems [18] [19] [20]. Many Loop self-scheduling methods have been proposed, including static scheduling and dynamic scheduling [4] [9]. The former is not suitable for a dynamic environment and the latter is difficult to maintain workload balance. Therefore, the implementation of the parallel loop self-scheduling in the cluster environment is a challenge [12] [13] [15]. This work studies the weight of the parallel loop self-scheduling in the cluster environment to address the challenge.

In our previous work, we have improved the workload balancing of a system [8]. Now Intel introduces a new accelerator Xeon Phi with x86-based architecture. It has about 60 cores with independent streamlined Linux operating system; each can be set with an IP and regarded as a single computing node. Accordingly, we study the case of many-core in this work, and use a parallel loop self-scheduling scheme to achieve workload balancing.

1.3 Thesis Organization

The rest of the paper is as follows. In Section 2, we introduce Xeon Phi, HPL, OpenMP, MPI, parallel loop self-scheduling and Related Work. In Section 3, we introduce our system design, system implementation and experimental design. Section 4 lists our experimental environment, experimental results and discussion. In Section 5, some conclusions are given and future work is discussed.

Chapter 2

Background Review and Related Work

In this chapter, we will introduce the background related to our work. Section 2.1 introduces the Xeon Phi coprocessor. Section 2.2 introduces the HPL. Section 2.3 describes the OpenMP programming model. Section 2.4 describes the MPI programming model. Section 2.5 describes the parallel loop self-scheduling scheme. Section 2.6 describes the related work.

2.1 Background Review

2.1.1 Xeon Phi coprocessor

In the SC12 conference, Intel officially released the Xeon Phi using x86-based Intel Many Integrated Core (MIC). Intel MIC architecture uses a number of Intel-based central processing unit (CPU) cores onto a single chip. These cores can run standard C, C ++ and FORTRAN source code, and program source code designed for Intel MIC products can be compiled and executed on a standard Intel Xeon processors. The familiar programming model can eliminate learning barriers, allowing developers to focus on how to solve problems, rather than on software design. Intel MIC architecture products aim for highly parallel processing fields and applications in high-performance computing, workstations, and data centers. It supports multiple parallel models such as OpenMP, pThread, MPI and other parallel programming languages [30], as shown in FIG 2.1.

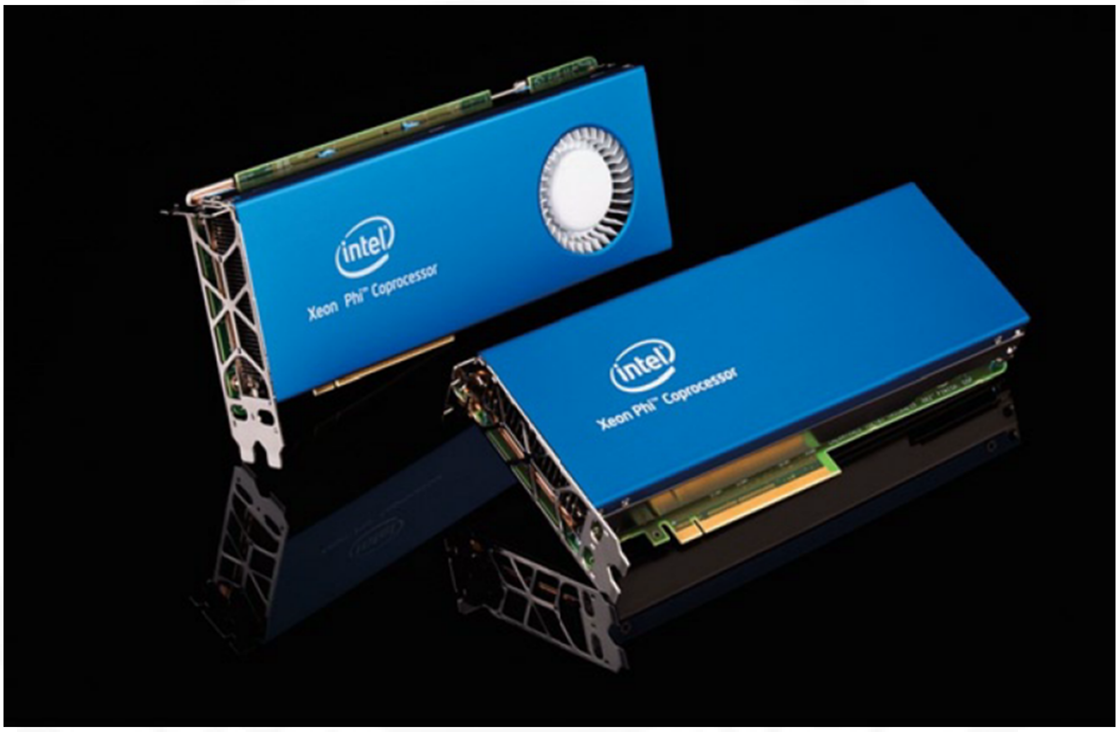


FIGURE 2.1: Xeon Phi Product

The Xeon Phi core boasts of more than 5 billion transistors using the 22nm tri-gate process. The Xeon Phi coprocessor consists of processing cores, cache, memory controller, PCI-E client logic, a very high bandwidth, and bi-directional ring bus components. Each core uses x86 instruction set architecture, and comes complete with a private L2 cache that is kept fully coherent by a global-distributed tag directory. The memory controller and PCI-E client logic separately provide direct interface to GDDR5 memory. All cores are connected to each other via a bidirectional ring bus, as shown in FIG 2.2.

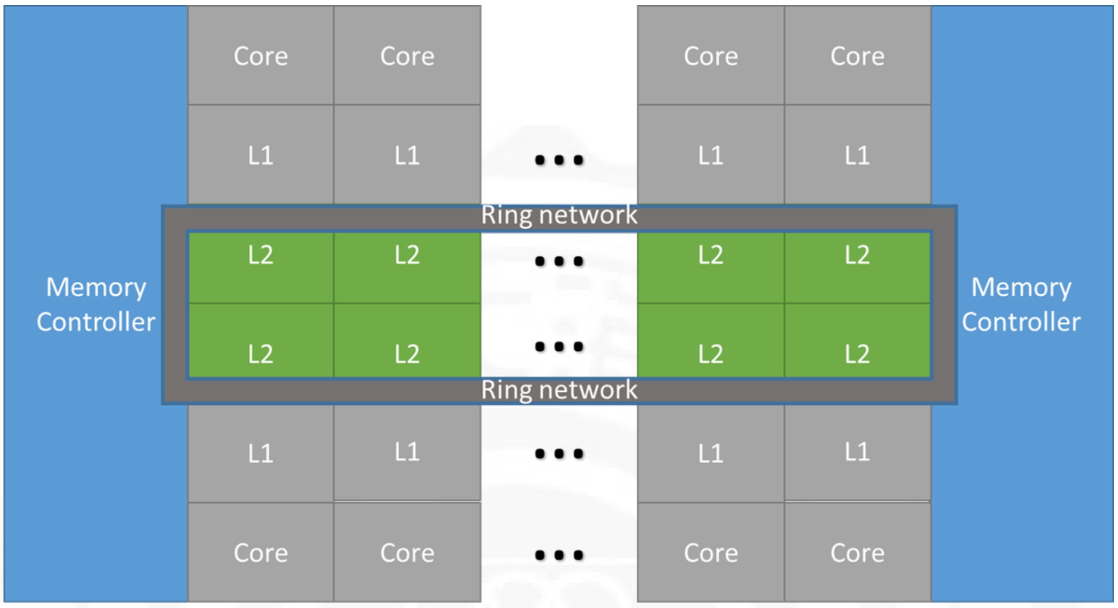


Figure 2.2: Xeon Phi Architecture

The Xeon Phi model is divided into 3100 series, 5100 series, and 7100 series, and with 57, 60, and 61 x86-based cores, respectively. Each core supporting 4 threads, containing a local L1 cache and a global L2 cache, with 32 512bit vector bandwidth registers to support 12 to 16 pairs of channel GDDR5 memory controller, and with memory capacity of 6GB to 16GB, supporting more than 2 Teraflops single precision floating-point operations and approaching 1 Teraflops double precision floating-point operations. The specifications of each version of Xeon Phi are listed in TABLE 2.1.

Table 2.1: Xeon Phi Specifications

	3100 Series	5100 Series	7100 Series
Cores	57	60	61
Processor Frequency	1.1 GHz	1.053 GHz	1.238 GHz
L2 Cache	28.5 MB	30 MB	30.5 MB
Memory Capacity	6 GB	8 GB	16 GB
Memory Channels	12	16	16
Memory Bandwidth	240 GB/s	320 GB/s	352 GB/s
Thermal Design Power	300 W	225 W	300 W
Turbo Boost Technology	No	No	Yes

In addition Xeon Phi also has many features listed below:

- MIC with the host via PCI-E connectivity, support for PCI-E X8 or X16 width configurations.
- Each MIC core can execute different instructions and supports up to four hardware threads, thus to hide the access latency.
- MIC has a stripped down Linux operating system, and can be set with an independent IP and treated as an independent node, but it cannot operate independently without a HOST.
- MIC has a 512-bit vector bandwidth registers, can handle 16 32-bit floating-point or 8 64-bit floating-point operations at the same time.
- MIC includes a local 32KB L1 instruction cache and a 32KB L1 data cache, and each core has a global 512KB L2 cache.

Programming between the MIC and the HOST is very flexible; HOST or MIC, or HOST and MIC can simultaneously launch the main function. HOST and MIC collaborative computing has several modes, as shown in FIG 2.3.

- Native Mode: A program can be complied by using -mmic compiler option to be compiled into an MIC executable program; the compiled program can only be executed on the MIC.
- Offload modes: HOST launches the main function and executes the program until the core computation part, which is transferred to be computed on the MIC and the computation results are transmitted back to the HOST later.
- Symmetric mode: HOST and MIC individually launch a main function, using the compiler options -mmic to get MIC executable programs and HOST executable programs, which is commonly achieved by the MPI programming model.

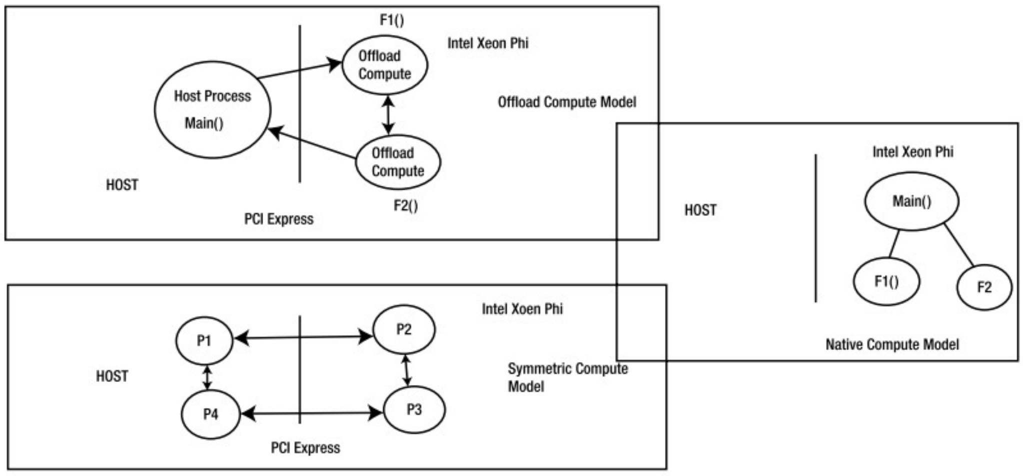


FIGURE 2.3: Xeon Phi execution model

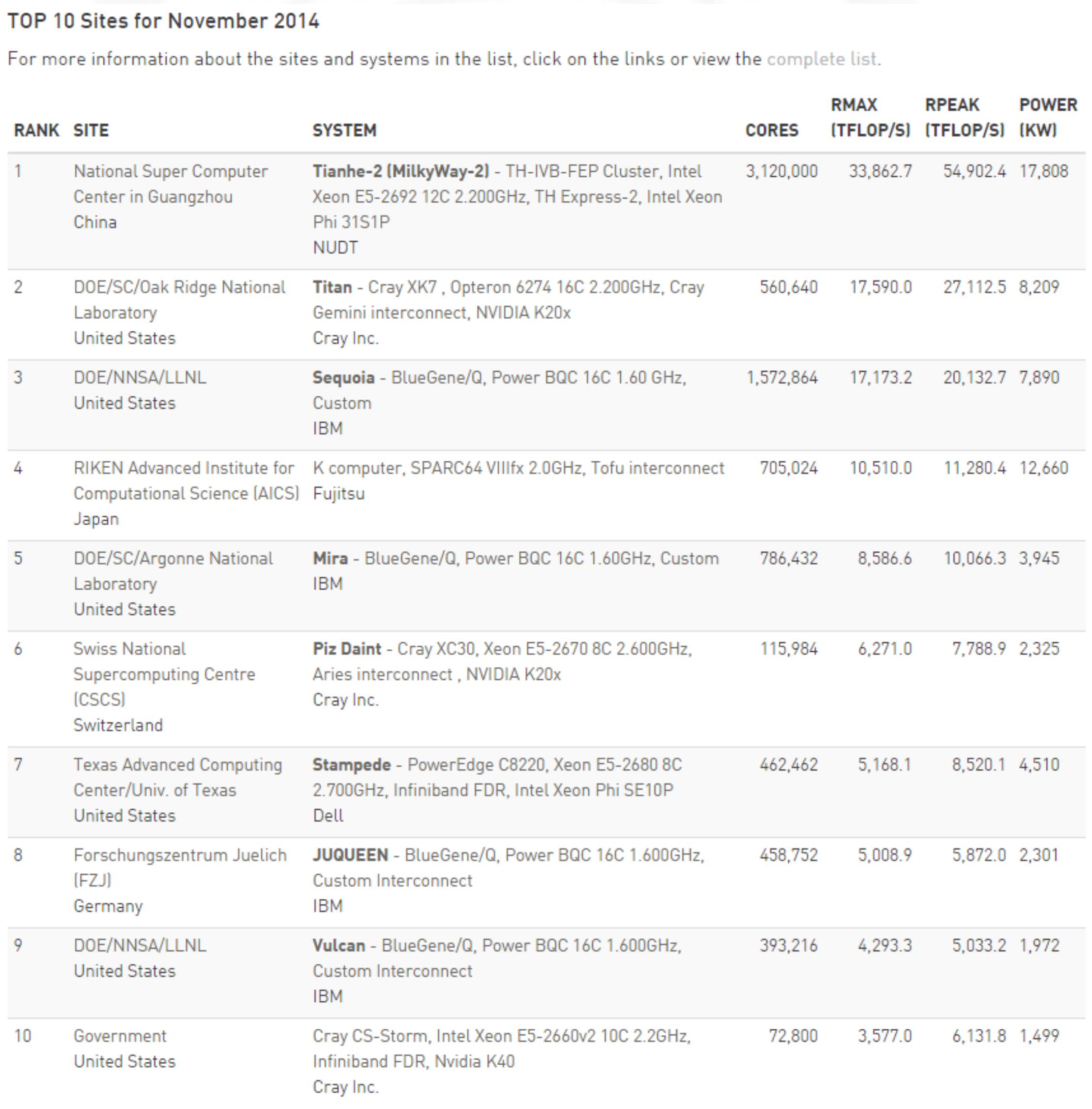
2.1.2 HPL (High Performance Linpack)

Constantly upgrading computer hardware is accompanied by continuous improvement of the performance of the system processing capability. The main goal of performance testing is to make right judgments of the platform performance. A variety of benchmarks are used in the industry. Some are based on the actual types of applications, such as TPC-C. Some test performance of a part of systems, such as "IOmeter" that tests performance of hardware, and "stream" that tests performance of bandwidth of memory. Linpack has become the most popular benchmark for testing high-performance floating-point performance of a computer system in the world. To evaluate floating-point performance of high-performance computers, the Gaussian elimination method is used for solving a dense n by n system of linear equations in high-performance computers [34].

The Linpack test includes three types, i.e., Linpack100, Linpack1000 and HPL. Linpack100 has a solution scale of 100 order dense linear algebraic equations; the code cannot be changed, it only allows using compiler optimization options to optimize the code. Linpack1000 has a solution scale of 1000 order linear algebraic equations; to achieve the specified accuracy requirements, the algorithms and code can be optimized under the premise of without changing the number of operations. HPL, also called highly parallel computing benchmark, does not has constraint on the array size N, i.e., N can be changed for solving the problems; besides, except the basic algorithm that cannot be changed, any optimization methods can be used. The scale of the first two tests is too small to be suitable for testing modern computers, so nowadays HPL test standard is used mostly, and parameter N, the order of linpack test, must be specified to do the test.

HPL is the test method proposed for testing modern parallel computers. Users cannot modify any part of the test program; but they can adjust or choose the problem size N (i.e., the matrix size), the number of used CPUs, and a variety of optimization methods to execute the test procedures to obtain the best performance. The measured peak floating-point value refers to the Linpack test value, i.e., the best test results obtained by the various optimizing methods of the Linpack test program running on the machine. In the actual running of the program, it is almost impossible to reach the theoretic peak floating-point value [33].

The TOP500 project provides a ranking and detailed list of the most powerful computer systems in the world. This program began in 1993 and publishes a latest ranking list of supercomputers twice a year. Every year, the annual ranking published for the first time is always in the International Supercomputing Conference in June, while the second ranking is published in supercomputing conference in November. The spirit of this program is to provide a reliable basis to track and detect trends of high-performance computing. The website of TOP500 ranks the 500 fastest supercomputers in the world based on the HPL benchmark. In recent years, the trend of accelerators used in supercomputers has changed from the NVidia CUDA card into the Intel Xeon Phi coprocessor [37]. As shown in FIG 2.4.



TOP 10 Sites for November 2014

For more information about the sites and systems in the list, click on the links or view the complete list.

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
9	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972
10	Government United States	Cray CS-Storm, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, Nvidia K40 Cray Inc.	72,800	3,577.0	6,131.8	1,499

Figure 2.4: TOP 10 Sites for November 2014

2.1.3 OpenMP (Open Multi-Processing)

OpenMP is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and it can run on most processor architectures and operating systems, including Solaris, AIX, HP-UX, GNU / Linux, Mac OS X, and Microsoft Windows. It includes compiler directives, library, and some environmental variables that can affect the runtime behavior. OpenMP uses a portable, scalable model to provide for the programmer a simple and flexible interface to develop parallel applications for platforms ranging from standard desktop computers to supercomputers [38].

OpenMP is proposed by the OpenMP Architecture Review Board, and has been widely accepted a multi-threaded programming paradigm for shared memory parallel system. OpenMP supports programming in C language, C ++ and FORTRAN. Many compilers support OpenMP, including Sun Studio compilers and Intel Compiler, open-source GCC compiler, and Open64. OpenMP provides high-level abstract description for parallel algorithms. Programmers can add dedicated pragmas to the source program to indicate their intentions. Whereby the compiler can automatically parallelize the program, adding synchronous mutex and communications when is necessary. When these pragmas are ignored as selected, or the compiler does not support OpenMP, the program can degenerate to the usual procedure and the code can still work properly, but cannot take advantage of multi-threading to speed up program execution. As shown in FIG 2.5.

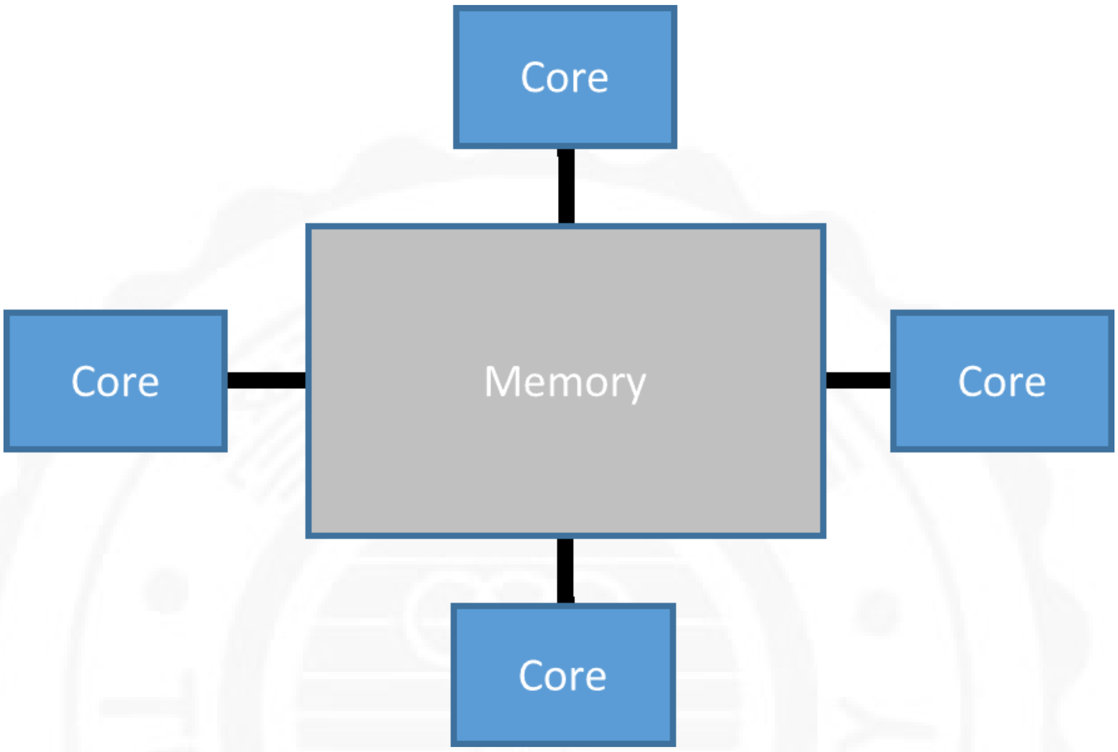


FIGURE 2.5: OpenMP Abstraction Architecture

OpenMP is a cross-platform, multi-threaded achieve. The Master thread generates a series of sub-threads and tasks allocated to these sub-threads for execution. These sub-threads run in parallel, and by the runtime environment these sub-threads are assigned to different processors for execution. To carry out the parallel execution of the code fragment, it needs to be labeled accordingly. With pre-compiled instructions the code fragment generates threads before being executed. Each thread is assigned with an id, which can be obtained by function omp_get_thread_num. Each id is an integer, and the id of the master thread is 0. At the end of parallel execution of the code, sub-threads join the master thread and only the master thread continues execution. By default, each independent execution threads to execute code in parallel region. Work-sharing constructs can be used to divide the tasks, so that each thread has its allocated portion of code execution. In this way, the parallel task and parallel data can be achieved using OpenMP. Run-time environment assigns the number of threads to each processor according to the usage, machine load, and other factors. The number of threads can be specified through the environment variable or function code. In C / C +

+, OpenMP functions are declared in the header file omp.h [35]. As shown in FIG 2.6.

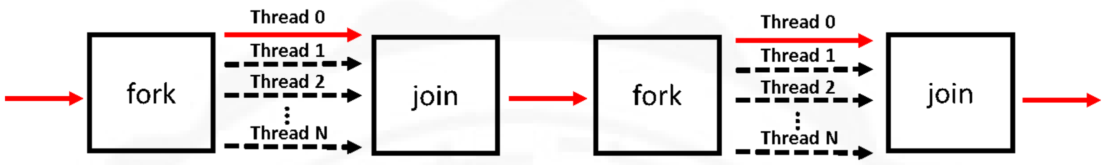


FIGURE 2.6: Threads Fork and Join

OpenMP provides a high level of abstraction in parallel description reduces the difficulty and complexity of parallel programming, so programmers can put more energy into the parallel algorithm itself, rather than specific details of implementation. Also, OpenMP also provides greater flexibility to more easily adapt to different parallel system configurations. Thread granularity and load balancing are traditional multi-threaded programming problems.

2.1.4 MPI (Message Passing Interface)

MPI initially was the standardized message delivery system in various parallel computers used by a group of researchers from academia and industry. MPI is a messaging application program interface, as well as agreements and semantic specification to achieve its function. It can be written in Fortran, C, C ++ and other languages. MPI aims to have high-performance, scalability, and portability. Most MPI implementations directly call a specific set of code, i.e. API, from C, C ++, FORTRAN, C #, Java, and Python. The popular open source MPI implementations include open-source software Open MPI, open-source based MPICH2 from Intel, and Intel MPI by MVAPICH2 [39].

MPI is a cross-language communications protocol for composing parallel computers. It supports point to point communications and broadcast. MPI is an information delivery API, including protocol and semantics to indicate how to realize its properties in a variety of implementations. MPI aims to have high-performance, large-scale and portability. MPI is still the main model for high-performance computing today. The main MPI-1 model does not include the concept of shared memory, and MPI-2 only has a limited concept of distributed shared memory. MPI programs are frequently performed on a shared memory machine. The MPI model design is better than the NUMA architecture. Most MPI implementations use specified APIs in C, C ++, FORTRAN, or languages with libraries in this category, such as C #, Java or Python. MPI is better than the old information delivery library because of his portability and speed.

MPI belongs to distributed memory architecture. The program in a node will be copied into the nodes of multiple memory blocks at the same time. Different nodes will be connected through the Internet and different procedures are used to exchange messages by MPI. Advantages of MPI are that all the machines can be easily connected together using the Internet and memory can be accessed fast in it. Drawbacks are that programmers spend more time in exchanging information among the programs and additional memory consumed in the multiple copies of programs [36]. As shown in FIG 2.7.

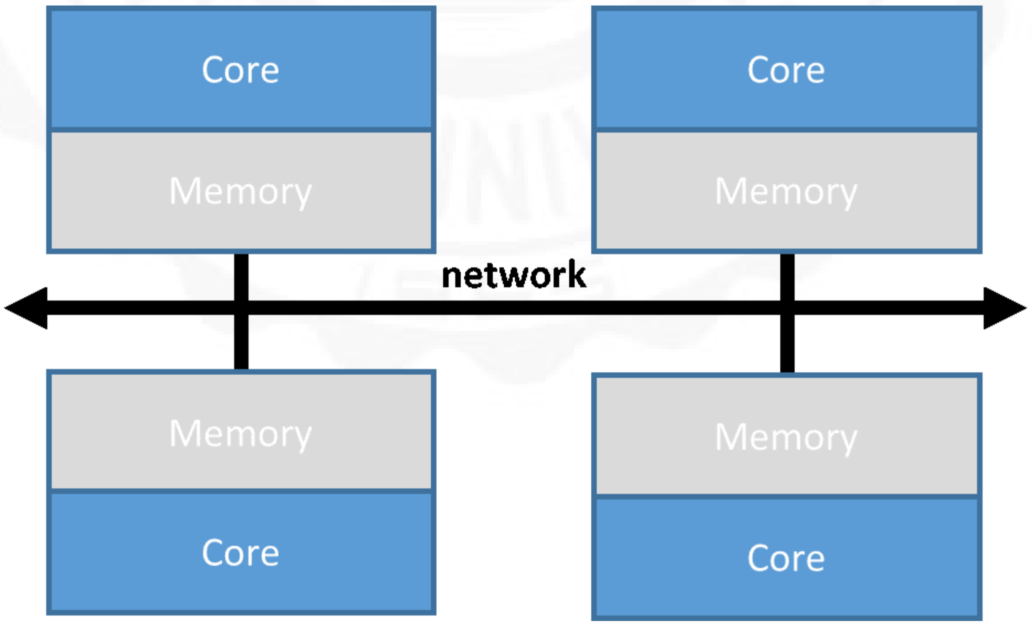


FIGURE 2.7: MPI Abstraction Architecture

Applications built by the hybrid parallel programming model by integrating MPI and OpenMP, can have advantages offered by MPI or OpenMP, or more transparently use the computer cluster running on the unshared memory system via OpenMP extensions [26] [24] [25] [27]. As shown in FIG 2.8.

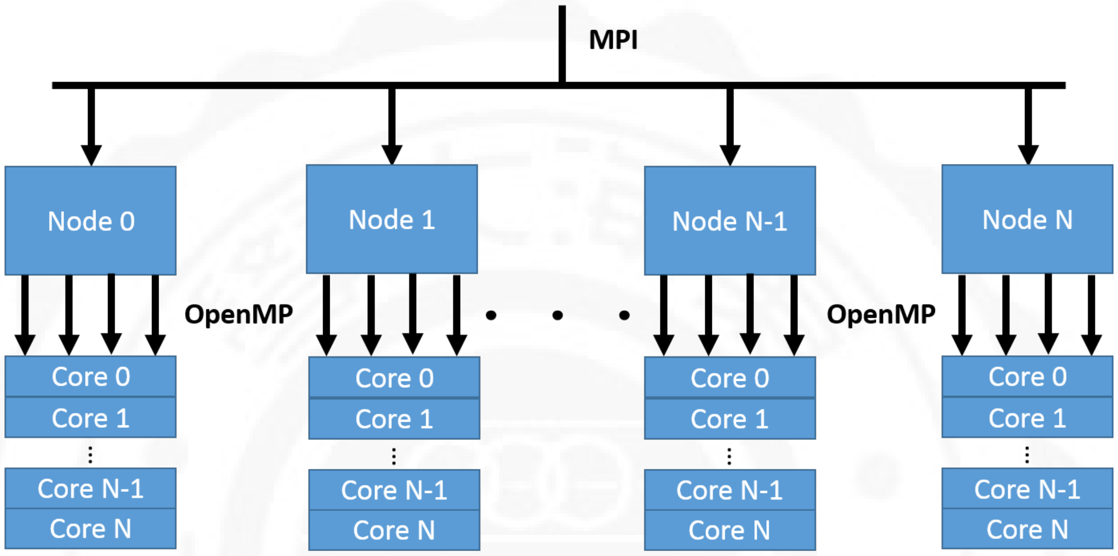


FIGURE 2.8: MPI Hybrid OpenMP Model

2.1.5 Parallel Loop Self-Scheduling

Self-scheduling schemes are mainly used to deal with load balancing and they can be divided into static and dynamic. The Static scheduling scheme determines how many loop iterations are assigned to each core before running. The advantage of these schemes is scheduled to run when there is no scheduling overhead. However, it is difficult to estimate the computing power of each core, resulting in load imbalance. In contrast, the dynamic scheduling is more suitable for load balancing, because its scheduling scheme is determined at running. It does not require estimation and projection. Initially part of the loop iterations are scheduled to all cores. To complete the work assigned, it will require repeated schedule. However, the runtime overhead must be considered and designed to use the dynamically self-scheduling scheme, because of the excessive runtime overhead may result in poor system performance [4] [5] [6] [7] [8] [9].

We introduce several loop self-scheduling scheme. Assume loop iterations N = 1000, the number of cores C = 4. As shown in TABLE 2.2.

- Chunk self-scheduling (CSS). The schedule for each iteration loop will assign tasks to each core, and each core will be assigned with the same amount of work. Assignment will repeat after the assigned work is completed and the workload in next round is assigned in the same way until the assignment is finished.
- Factoring self-scheduling (FSS). This schedule for each iteration loop will assign tasks to each core, and each core will be assigned with the same amount of work. Assignment will repeat after the assigned work is completed and the workload in next round is halved until the assignment is finished.
- Guide self-scheduling (GSS). This schedule for each iteration loop will assign tasks to each core, and the workload assigned to a core is equal to the amount of workload assigned to the core before it times N-1/N (N: the total number of cores). Assignment will repeat after the assigned work is completed and the workload in next round is halved until the assignment is finished.
- Trapezoid self-scheduling (TSS). The schedule for each iteration loop will assign tasks to each core, and the workload assigned to a core is equal to amount of workload assigned to the core before it deducted by an arithmetic difference. Assignment will repeat after the assigned work is completed and the workload in next round is halved until the assignment is finished.

Table 2.2: Examples of self-scheduling scheme

Scheme	Partition size
CSS	125,125,125,125,125,125,125,125
FSS	125,125,125,125,63,63,63,63,31, ...
GSS	250,188,141,106,79,59,45,33,25, ...
TSS	125,117,109,101,93,85,77,69,61, ...

2.2 Related Work

In the method proposed by Wu et. al. [20], parallel loop of self-scheduling scheme is applied to the grid system. In the static scheduling part, the formula W = CS/ CL (CS: CPU clock speed, CL: CPU loading) is calculated for each core utilization, and the authors used this formula as a basis for allocation. In the dynamic scheduling part, the comparison of scheduling methods PSS, CSS, FSS, GSS and TSS is discussed to decide which application is suitable for a dynamic scheduling scheme.

W is the assigned weight of each core in static scheduling
CS is the CPU clock
CL is the CPU loading
i is the node i

$$W_i = (CS_i/CL_i)/(\sum CS/\sum CL) \tag{2.1}$$

SWR is the static workload ratio
MIN is the minimum execution time of all sampled iterations
MAX is the maximum execution time of all sampled iterations

$$SWR = MIN/MAX \tag{2.2}$$

In the method proposed by Yang et. al. [8], the parallel loop of self-scheduling scheme is applied to SMP (symmetric multiprocessor) cluster. In the static scheduling part, CPU clock speed and HPL are utilized as the basis of their scheme. In the dynamic scheduling part, comparison of scheduling methods FSS, GSS and TSS are discussed. In their experiments, four applications are adopted to observe the effects of different types of applications on the proportion of static scheduling and dynamic scheduling. The results show that proportional allocation of static scheduling formula CPU clock speed and HPL obtained better performance.

T is the total number of loop iterations

α is the proportion of static scheduling to dynamic scheduling

$$T = \alpha \times Static\ Scheduling + (100 - \alpha) \times Dynamic\ Scheduling \tag{2.3}$$

W is the assigned weight of each core in static scheduling

β is the proportion of the CPU clock to the HPL between for static scheduling

$$W = \beta \times (CPU\ clock / \sum CPU\ clock) + (1 - \beta) \times (HPL / \sum HPL) \tag{2.4}$$

In the method proposed by Han and Chronopoulos [10], the parallel loop of self-scheduling scheme is applied to Large-Scale cluster. They used Hierarchical Architecture in stead of general self-scheduling method. The Hierarchical Architecture contains three models: Supermaster, Master, and Worker. In the experiments, three scheduling schemes FSS, GSS and TSS are compared and applied to two applications so as to realize the impact of number of master on performance.

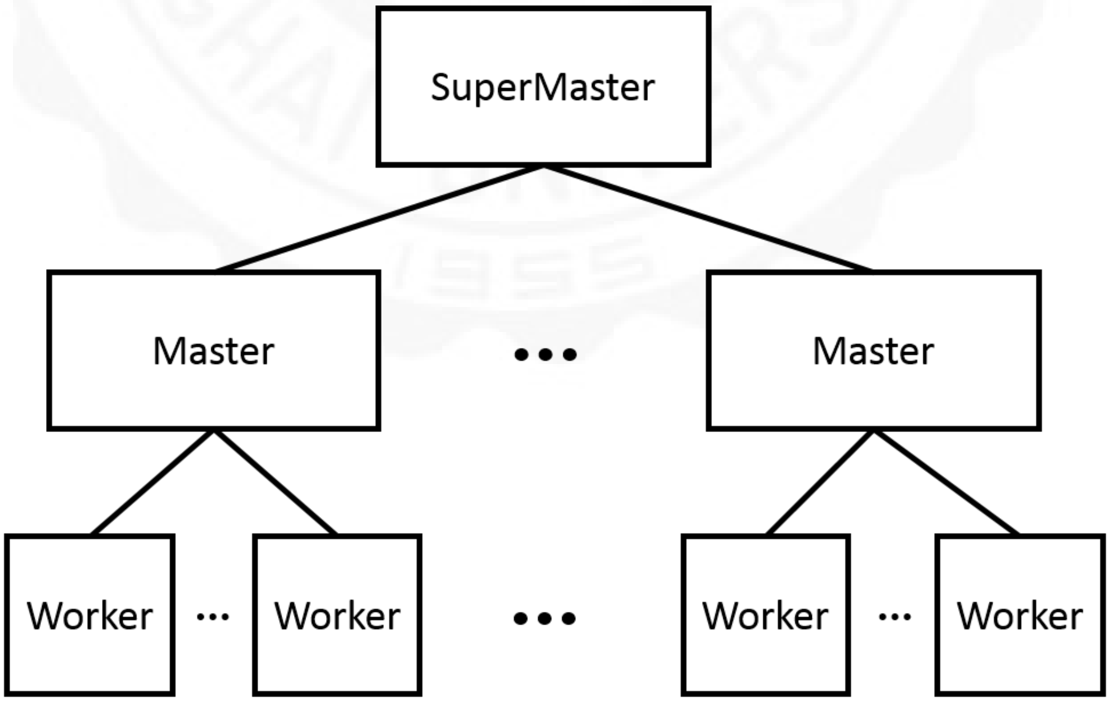


Figure 2.9: Hierarchical Architecture

Chapter 3

System Design and Implementation

In this section, we introduce the proposed system design and implementation. Section 3.1 describes the system design. Section 3.2 implements the system.

3.1 System Design

For the proposed system design, we first introduce the weighting algorithm and then describe the system flow.

3.1.1 Weight Algorithm

Without loss of generality, we can divide loop self-scheduling into static and dynamic parts. In the static part, the program assigns tasks to each core by our weighting formula before execution. In the dynamic part, the program assigns tasks in accordance with the results of the self-scheduling algorithm after execution. The formula for our loop self-scheduling is as follows:

T is the total number of loop iterations

α is the proportion of static scheduling to dynamic scheduling

$$T = \alpha \times Static\ Scheduling + (100 - \alpha) \times Dynamic\ Scheduling \quad (3.1)$$

The method proposed by C.-T. Yang et al. [8] has a poor performance of loop self-scheduling when the proportion of static scheduling is high. To solve this drawback, the estimation of weights should be improved. Therefore, we modify the formula by replacing the HPL with the Performance of small Size (PS for short) in each compute node to increase the accuracy of the weight estimation. The modified formula is as follows:

W is the assigned weight of each core in static scheduling
β is the proportion of the CPU clock to the PS between for static scheduling
PS is the performance of the small SIZE

$$W = \beta \times (CPU\ clock / \sum CPU\ clock) + (1 - \beta) \times (PS / \sum PS) \quad (3.2)$$

3.1.2 System Flow

Figure 3.1 shows the proposed system. In the first step, the information related to each node is collected and stored in that node. In the second step, the alpha value is set to find the best ratio of static scheduling over dynamic scheduling; and at the same time, the beta value is set to find the best performance. The second step is repeated until an optimal result is found. This above system flow is repeated for different scheduling methods and applications.

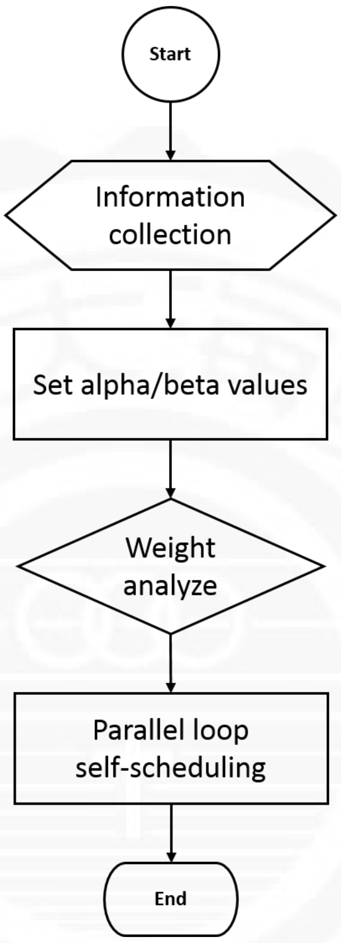


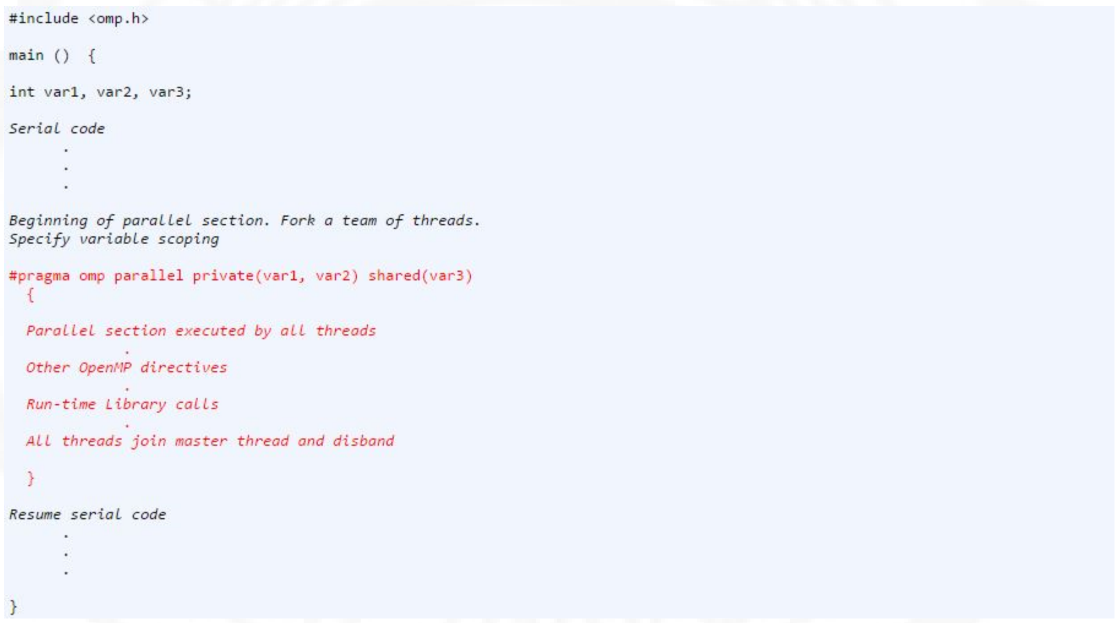
FIGURE 3.1: System Flow

3.2 System Implementation

3.2.1 OpenMP and MPI Programming Model

Generally, program parallelization can be implemented by OpenMP or MPI. The effectiveness of OpenMP parallel architecture is better than that of MPI parallel architecture. The loop parallelization can be achieved in OpenMP by just adding some simple syntax. Moreover, in OpenMP one can set the number of execution threads merely by modifying some variables. MPI is relatively complicated. In addition to use a function to start and end the parallel program, in MPI the parallelization of loop and messages transmitted between different procedures also need to be assigned by programmers themselves. However, OpenMP does not support cross-node so that it is necessary to apply MPI to use multiple processors. Most programmers today use MPI mixed with OpenMPI to develop parallel programs. OpenMP and MPI programs are introduced in the following:

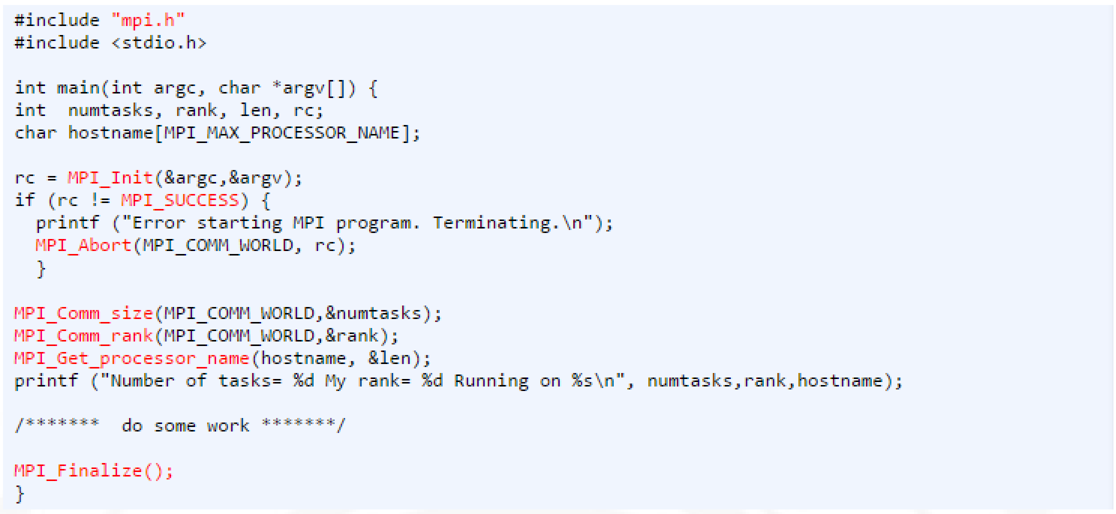
The program of OpenMP starts with a header file #include <omp.h> and then declares parallel blocks by "#pragma omp parallel" and parallel loops by "#pragma omp parallel for" , and finally completes with some directives and clauses. As shown in FIG 3.2.



```
#include <omp.h>

main () {

int var1, var2, var3;

Serial code
      .
      .
      .

Beginning of parallel section. Fork a team of threads.
Specify variable scoping

#pragma omp parallel private(var1, var2) shared(var3)
  {

  Parallel section executed by all threads
             .
  Other OpenMP directives
             .
  Run-time Library calls
             .
  All threads join master thread and disband

  }

Resume serial code
      .
      .
      .

}
```

Figure 3.2: OpenMP Example

The beginning of an MPI program is a header file #include "mpi.h". One needs to start an MPI program after initializing it by MPI_Init (& argc, & argv); and to end an MPI program by MPI_Finalize ( ). MPI_Comm_size (MPI_COMM_WORLD, & numtask) and MPI_Comm_rank (MPI_COMM_WORLD, & rank) are utilized to get the number of parallel and program thread ID, respectively. These two pieces of information are helpful to split blocks and transmit messages to other threads. As shown in FIG 3.3.

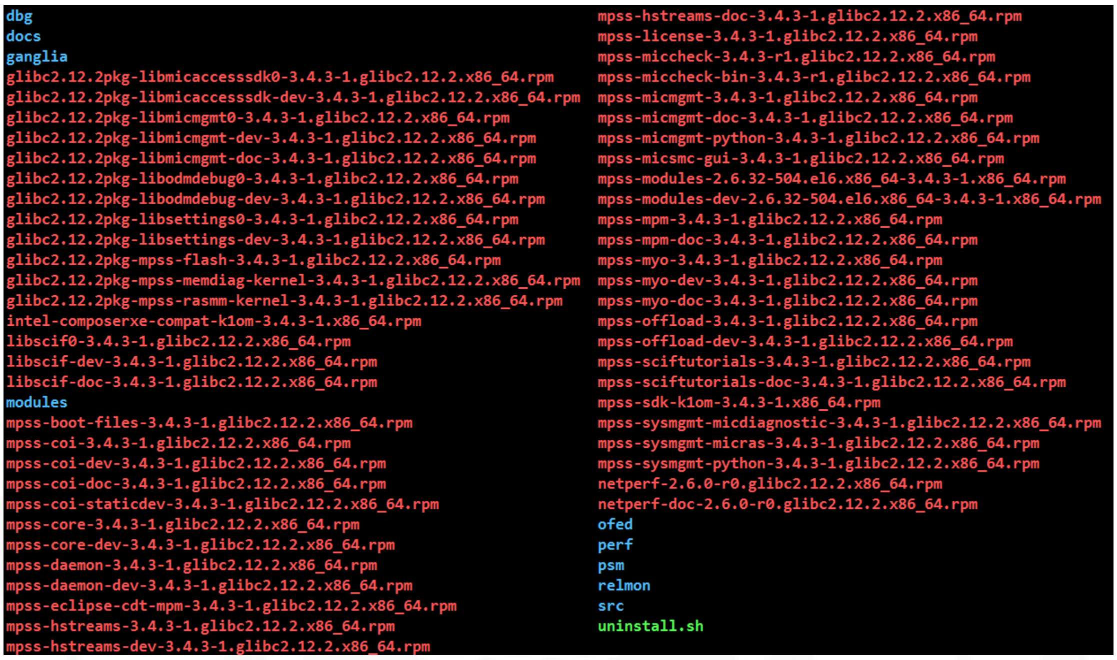


```
#include "mpi.h"
#include <stdio.h>

int main(int argc, char *argv[]) {
int  numtasks, rank, len, rc;
char hostname[MPI_MAX_PROCESSOR_NAME];

rc = MPI_Init(&argc,&argv);
if (rc != MPI_SUCCESS) {
  printf ("Error starting MPI program. Terminating.\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  }

MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Get_processor_name(hostname, &len);
printf ("Number of tasks= %d My rank= %d Running on %s\n", numtasks,rank,hostname);

/*******  do some work *******/

MPI_Finalize();
}
```

FIGURE 3.3: MPI Example

3.2.2 Intel Many-core Platform Software Stack

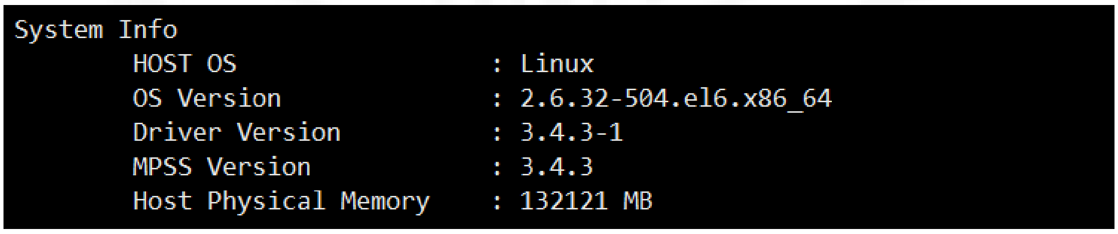
To speed up calculations using Intel Xeon Phi, a specified operating system and the installation of Intel Many-core Platform Software Stack (MPSS) are necessary. Since Xeon Phi executes program by the supported compiler, we adopt Intel Parallel Studio XE toolkit with Intel Parallel Studio XE source files compilervars.sh and mpivars.sh. Intel Parallel Studio XE provides C language compiler, MPI compiler, and Xeon Phi variety. Most importantly, it also provides VTune Amplifier XE performance profiler that can analyze the cost of parallel programs and help programmers develop optimized programs. The two files, compilervars.sh and mpivars.sh, can be used to setup the Intel compiler and Intel MPI environment variables. As shown in FIG 3.4.



```
dbg                                                                mpss-hstreams-doc-3.4.3-1.glibc2.12.2.x86_64.rpm
docs                                                               mpss-license-3.4.3-1.glibc2.12.2.x86_64.rpm
ganglia                                                            mpss-miccheck-3.4.3-r1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-libmicaccesssdk0-3.4.3-1.glibc2.12.2.x86_64.rpm     mpss-miccheck-bin-3.4.3-r1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-libmicaccesssdk-dev-3.4.3-1.glibc2.12.2.x86_64.rpm  mpss-micmgmt-3.4.3-1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-libmicmgmt0-3.4.3-1.glibc2.12.2.x86_64.rpm          mpss-micmgmt-doc-3.4.3-1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-libmicmgmt-dev-3.4.3-1.glibc2.12.2.x86_64.rpm       mpss-micmgmt-python-3.4.3-1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-libmicmgmt-doc-3.4.3-1.glibc2.12.2.x86_64.rpm       mpss-micsmc-gui-3.4.3-1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-libodmdebug0-3.4.3-1.glibc2.12.2.x86_64.rpm         mpss-modules-2.6.32-504.el6.x86_64-3.4.3-1.x86_64.rpm
glibc2.12.2pkg-libodmdebug-dev-3.4.3-1.glibc2.12.2.x86_64.rpm      mpss-modules-dev-2.6.32-504.el6.x86_64-3.4.3-1.x86_64.rpm
glibc2.12.2pkg-libsettings0-3.4.3-1.glibc2.12.2.x86_64.rpm         mpss-mpm-3.4.3-1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-libsettings-dev-3.4.3-1.glibc2.12.2.x86_64.rpm      mpss-mpm-doc-3.4.3-1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-mpss-flash-3.4.3-1.glibc2.12.2.x86_64.rpm           mpss-myo-3.4.3-1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-mpss-memdiag-kernel-3.4.3-1.glibc2.12.2.x86_64.rpm  mpss-myo-dev-3.4.3-1.glibc2.12.2.x86_64.rpm
glibc2.12.2pkg-mpss-rasmm-kernel-3.4.3-1.glibc2.12.2.x86_64.rpm    mpss-myo-doc-3.4.3-1.glibc2.12.2.x86_64.rpm
intel-composerxe-compat-k1om-3.4.3-1.x86_64.rpm                    mpss-offload-3.4.3-1.glibc2.12.2.x86_64.rpm
libscif0-3.4.3-1.glibc2.12.2.x86_64.rpm                            mpss-offload-dev-3.4.3-1.glibc2.12.2.x86_64.rpm
libscif-dev-3.4.3-1.glibc2.12.2.x86_64.rpm                         mpss-sciftutorials-3.4.3-1.glibc2.12.2.x86_64.rpm
libscif-doc-3.4.3-1.glibc2.12.2.x86_64.rpm                         mpss-sciftutorials-doc-3.4.3-1.glibc2.12.2.x86_64.rpm
modules                                                            mpss-sdk-k1om-3.4.3-1.x86_64.rpm
mpss-boot-files-3.4.3-1.glibc2.12.2.x86_64.rpm                     mpss-sysmgmt-micdiagnostic-3.4.3-1.glibc2.12.2.x86_64.rpm
mpss-coi-3.4.3-1.glibc2.12.2.x86_64.rpm                            mpss-sysmgmt-micras-3.4.3-1.glibc2.12.2.x86_64.rpm
mpss-coi-dev-3.4.3-1.glibc2.12.2.x86_64.rpm                        mpss-sysmgmt-python-3.4.3-1.glibc2.12.2.x86_64.rpm
mpss-coi-doc-3.4.3-1.glibc2.12.2.x86_64.rpm                        netperf-2.6.0-r0.glibc2.12.2.x86_64.rpm
mpss-coi-staticdev-3.4.3-1.glibc2.12.2.x86_64.rpm                  netperf-doc-2.6.0-r0.glibc2.12.2.x86_64.rpm
mpss-core-3.4.3-1.glibc2.12.2.x86_64.rpm                           ofed
mpss-core-dev-3.4.3-1.glibc2.12.2.x86_64.rpm                       perf
mpss-daemon-3.4.3-1.glibc2.12.2.x86_64.rpm                         psm
mpss-daemon-dev-3.4.3-1.glibc2.12.2.x86_64.rpm                     relmon
mpss-eclipse-cdt-mpm-3.4.3-1.glibc2.12.2.x86_64.rpm                src
mpss-hstreams-3.4.3-1.glibc2.12.2.x86_64.rpm                       uninstall.sh
mpss-hstreams-dev-3.4.3-1.glibc2.12.2.x86_64.rpm
```

FIGURE 3.4: MPSS Installation Toolkit

The Intel Many-core Platform Software Stack installation kit is installed by Administrator RPM packages. After the installation, by the micinfo instruction one can see the system information, including the HOST operating system, operating system version, driver version, MPSS version, and HOST physical memory capacity. As shown in FIG 3.5.



```
System Info
        HOST OS                 : Linux
        OS Version              : 2.6.32-504.el6.x86_64
        Driver Version          : 3.4.3-1
        MPSS Version            : 3.4.3
        Host Physical Memory    : 132121 MB
```

FIGURE 3.5: MPSS System Information

This observation tool for Intel Many-core Platform Software Stack installment lets the user control the use of Intel Xeon Phi, and it provides various visual data, including core utilization rate, the temperature of the card, memory usage, and power usage. As shown in FIG 3.6.

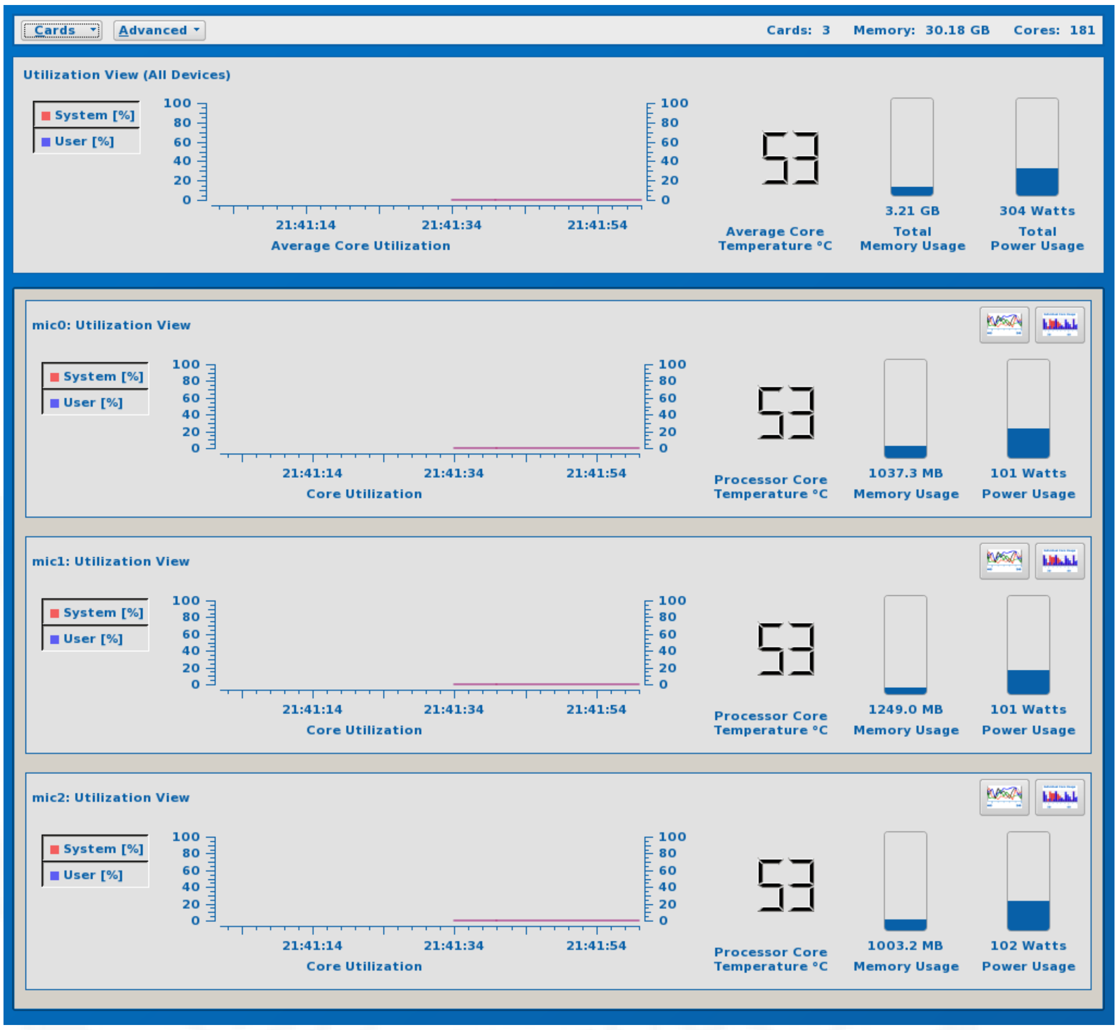
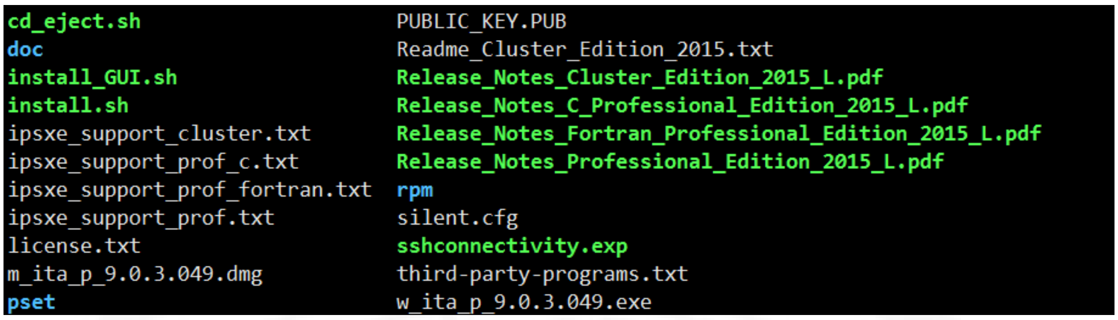


FIGURE 3.6: Xeon Phi Platform Control Panel

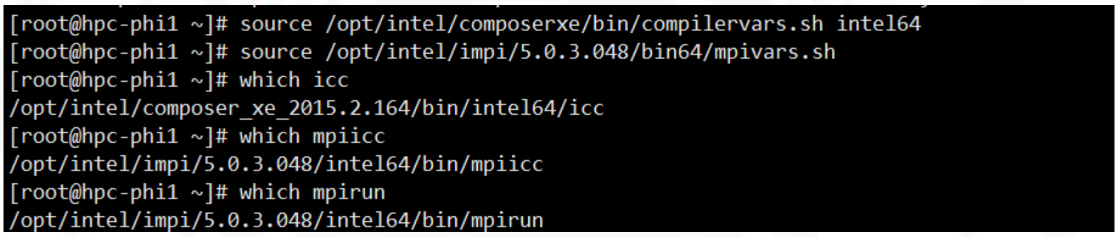
3.2.3 Intel Parallel Studio XE

For the Intel Parallel Studio XE installation kit, Intel provides automated scripts so that the user just needs to complete the installation using C / C ++ compiler by step-by-step confirmations, otherwise it will not execute the script. After completing the installation, Intel Parallel Studio XE will automatically set the environmental parameters needed using the source instructions compilervars.sh and mpivars.sh. In addition, one can use the commands, icc, mpiicc, mpirun to verify the path connection. As shown in FIG 3.7 3.8.



```
cd_eject.sh                     PUBLIC_KEY.PUB
doc                             Readme_Cluster_Edition_2015.txt
install_GUI.sh                  Release_Notes_Cluster_Edition_2015_L.pdf
install.sh                      Release_Notes_C_Professional_Edition_2015_L.pdf
ipsxe_support_cluster.txt       Release_Notes_Fortran_Professional_Edition_2015_L.pdf
ipsxe_support_prof_c.txt        Release_Notes_Professional_Edition_2015_L.pdf
ipsxe_support_prof_fortran.txt  rpm
ipsxe_support_prof.txt          silent.cfg
license.txt                     sshconnectivity.exp
m_ita_p_9.0.3.049.dmg           third-party-programs.txt
pset                            w_ita_p_9.0.3.049.exe
```

Figure 3.7: Intel Parallel Studio XE Installation Toolkit

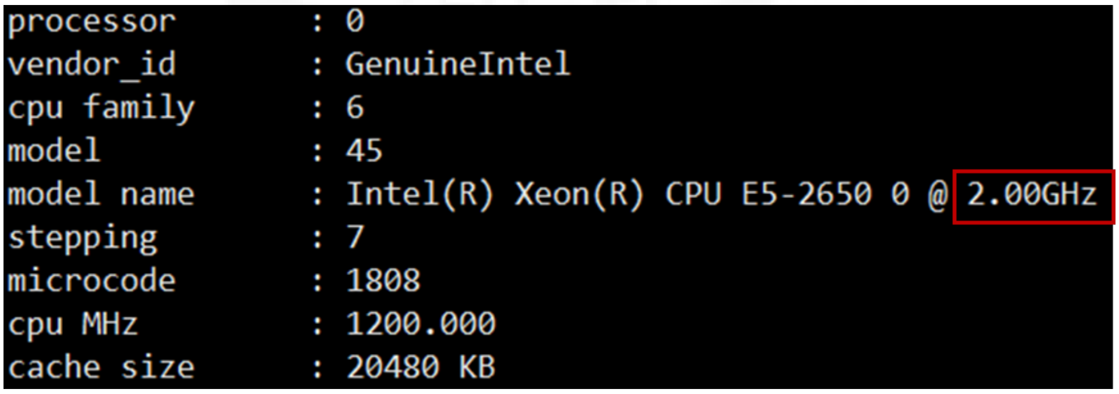


```
[root@hpc-phi1 ~]# source /opt/intel/composerxe/bin/compilervars.sh intel64
[root@hpc-phi1 ~]# source /opt/intel/impi/5.0.3.048/bin64/mpivars.sh
[root@hpc-phi1 ~]# which icc
/opt/intel/composer_xe_2015.2.164/bin/intel64/icc
[root@hpc-phi1 ~]# which mpiicc
/opt/intel/impi/5.0.3.048/intel64/bin/mpiicc
[root@hpc-phi1 ~]# which mpirun
/opt/intel/impi/5.0.3.048/intel64/bin/mpirun
```

Figure 3.8: Source compilervars.sh and mpivars.sh

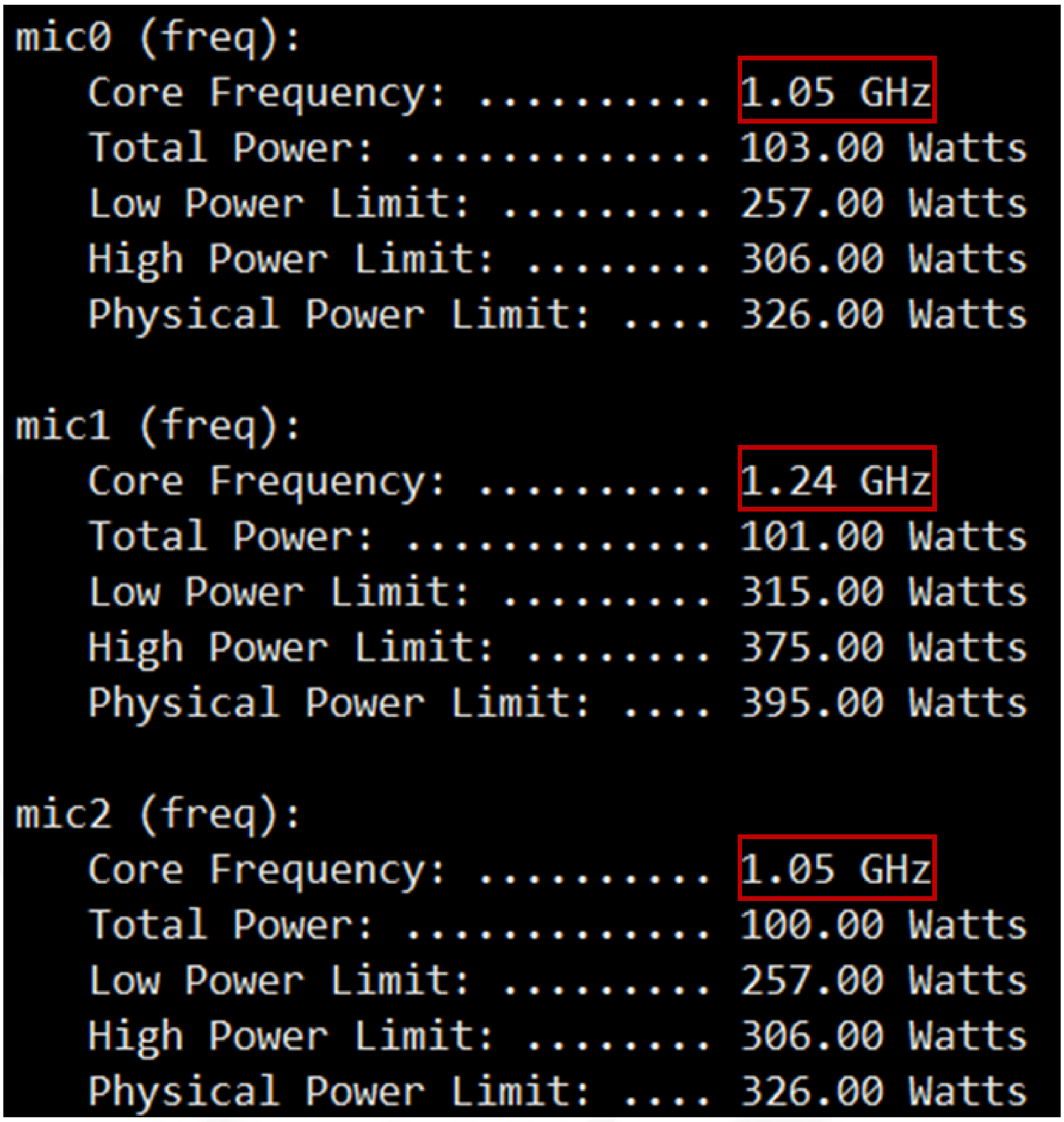
3.2.4 CPU Clock Measuring

The CPU clock speed value is obtained by checking the /proc/cpuinfo file, as shown in FIG 3.9. To obtain the MIC clock speed value, one needs to install Intel Many-core Platform Software Stack and uses the micsmc instruction, as shown in FIG 3.10.



```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 45
model name      : Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz
stepping        : 7
microcode       : 1808
cpu MHz         : 1200.000
cache size      : 20480 KB
```

Figure 3.9: CPU Clock Measuring



```
mic0 (freq):
   Core Frequency: .......... 1.05 GHz
   Total Power: ............. 103.00 Watts
   Low Power Limit: ......... 257.00 Watts
   High Power Limit: ........ 306.00 Watts
   Physical Power Limit: .... 326.00 Watts

mic1 (freq):
   Core Frequency: .......... 1.24 GHz
   Total Power: ............. 101.00 Watts
   Low Power Limit: ......... 315.00 Watts
   High Power Limit: ........ 375.00 Watts
   Physical Power Limit: .... 395.00 Watts

mic2 (freq):
   Core Frequency: .......... 1.05 GHz
   Total Power: ............. 100.00 Watts
   Low Power Limit: ......... 257.00 Watts
   High Power Limit: ........ 306.00 Watts
   Physical Power Limit: .... 326.00 Watts
```

FIGURE 3.10: MIC Clock Measuring

3.2.5 HPL Measuring

The HPL value can be obtained by a formula, but it is a theoretical value and does not necessarily represent the actual performance. To measure the actual value of HPL, we use Intel Math Kernel Library-LINPACK. It is an optimized version by Intel. We measure HPL of CPU by using the runme_xeon64 file. We also measure HPL of MIC by the using runme_mic file, as listed in TABLE 3.1.

Table 3.1: HPL Measuring

Node	HPL
phi1	277.2789 GFlops
phi1-mic0	547.2142 GFlops
phi1-mic1	874.6949 GFlops
phi1-mic2	551.0072 GFlops

3.2.6 Bandwidth Measuring

We measure the bandwidth speed of message transmission between cores, as shown in TABLE 3.2.

Table 3.2: Bandwidth Measuring

	phi1	phi1-mic0	phi1-mic1	phi1-mic2
phi1	34294 Mbps			
phi1-mic0	897 Mbps	13767 Mbps		
phi1-mic1	992 Mbps	577 Mbps	14905 Mbps	
phi1-mic2	1050 Mbps	609 Mbps	625 Mbps	13791 Mbps

Chapter 4

Experiments

In this section, we will describe the experimental environment and results. Section 4.1 introduces the experimental environment, including hardware and software environments. Section 4.2 shows the experimental results, including the performance comparison of matrix multiplication, Sparse Matrix Multiplication, Mandelbrot set computation, and the circuit satisfiability problem in static and dynamic ways.

4.1 Experimental Environment

Our experimental hardware and software environments are listed in Table 4.1 and Table 4.2, respectively. The experimental hardware includes one server and three Xeon Phi coprocessors, a total of four nodes. Messages are transmitted over nodes via the PCI-E bus. The computing power of these nodes is varied as to compose a heterogeneous environment as listed in Table 4.1. Table 4.2 lists the software version and functions used in the experiment.

4.1.1 Experimental Hardware

We list each node hardware details. As shown in the table 4.1.

Table 4.1: Hardware Specification

	phi1	phi1-mic0, phi1-mic2	phi1-mic1
CPU	Intel Xeon E5-2650 * 2	Intel Xeon Phi 5110P	Intel Xeon Phi 7120P
CPU Clock	2 GHz	1.05 GHz	1.24 GHz
CPU Core	16	60	61
RAM	132 GB	8 GB	16 GB
Disk	1 TB	NONE	NONE
OS	CentOS 6.6	Intel uOS	Intel uOS
Linux kernel	2.6.32-504.el6.x86_64	2.6.38.8	2.6.38.8

4.1.2 Experimental Software

We list the software version used in the experiment, and describe its function. As shown in the table 4.2.

Table 4.2: Software Specification

Name	Version	Description
Intel Composer XE	2015.2.164	Includes compilers, performance libraries, and parallel models optimized to build fast parallel code.
Intel Advisor XE	2015.1.10.380555	Intel Advisor XE is a threading prototyping tool for C, C++, C# and Fortran software architects.
Intel Inspector XE	2015.1.2.379161	Intel Inspector XE is an easy to use memory and threading error debugger for C, C++, C# and Fortran applications that run.
Intel VTune Amplifier	2015.2.0.393444	Intel VTune Amplifier provides a rich set of performance insight into hotspots, threading, locks & waits, OpenCL, bandwidth and more.
Intel MPI	5.0.3	MPI library, along with MPI error checking and tuning to design, build, debug and tune fast parallel code that includes MPI.
Intel MPSS	3.4.3	Is necessary to run the Intel Xeon Phi Coprocessor.

4.1.3 Experimental Design

In order to validate our approach, illustrate our environment, and describe the terms of our application, we achieved with the MPI / OpenMP program execution in our test platform. We then compare the performances of the matrix multiplication, Sparse Matrix Multiplication, Mandelbrot set computation, and circuit satisfaction in static and dynamic ways.

We implemented four applications, matrix multiplication, Sparse Matrix Multiplication, Mandelbrot set computation, and circuit satisfaction in C language. MPI / OpenMP parallel code segments are applied to implement our testing platform. Matrix multiplication has regular workload distribution and data communications needs of each step in the schedule. Sparse matrix multiplication has irregular workload distribution and data communications needs of each step in the schedule. Mandelbrot Set Computation has irregular workload distribution and no data communications needs of each step in the schedule. Circuit Satisfiability has regular workload distribution and data communications needs of each step in the schedule. The implementation of all programs is divided into four groups, Matrix multiplication (mat-), sparse matrix multiplication (smat-), Mandelbrot Set Computation (man-) and Circuit Satisfiability (sat-).

4.2 Experimental Results

We collect CPU clock speed and HPL values in all computing nodes, and study the effect by adjusting parameters α and β. Parameters α and β are set by the programmer. But it is difficult to select the appropriate α andβ during the experiment. If parameter α is too large, the computer will not be able to complete the work within a specific time; and if it is too small, the dynamic scheduling overhead becomes significant.

From the experimental results, we found that when α value is too high, the effectiveness of the implementation is poor due to the fact that HPL values do not represent the actual performance. Therefore, we made changes in the formula, i.e., we used the performance of execution in each node of a small size to do assessments.

4.2.1 Application 1: Matrix Multiplication

The most important method of matrix multiplication is the general method used to find the product of two matrixes. The definition of matrix multiplication is defined under the premise that that the size of columns of the first matrix should be equal to the size of rows of the second matrix.

In the experiment of matrix multiplication, the used size is 10240. As shown in the figure, factoring self-scheduling (the blue curve) has the best performance at α = 30, guide self-scheduling (the red curve) has the best performance at α = 50, and trapezoid self-scheduling (the green curve) has the best performance at α = 50. As shown in FIG 4.1 TABLE 4.3.

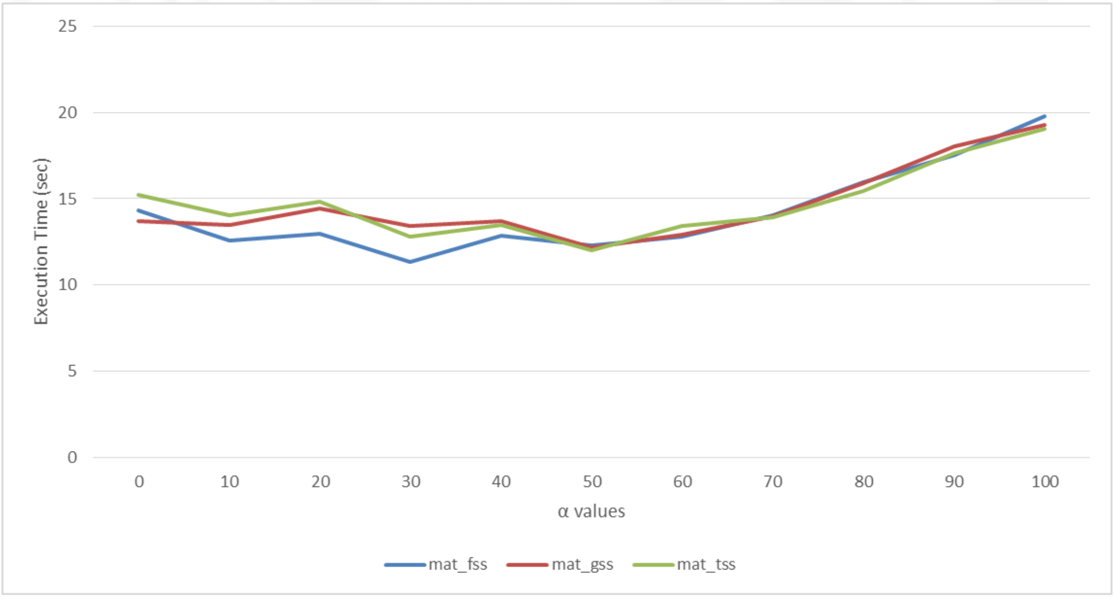


FIGURE 4.1: Matrix Multiplication adaptive α scheduling

Table 4.3: Matrix Multiplication execution time

	0	10	20	30	40	50	60	70	80	90	100
fss	14.34	12.57	12.99	11.34	12.85	12.27	12.79	14.06	15.93	17.50	19.78
gss	13.70	13.49	14.45	13.40	13.72	12.21	12.89	13.98	15.91	18.01	19.26
tss	15.25	14.06	14.83	12.80	13.46	12.04	13.41	13.90	15.45	17.62	19.02

We used the modified formula and repeated the experiment. We found that when the value of α increases, the improvement of performance increases, with a maximal speedup of 1.41 when α is 100 (the blue bar). As shown in FIG 4.2 TABLE 4.4.

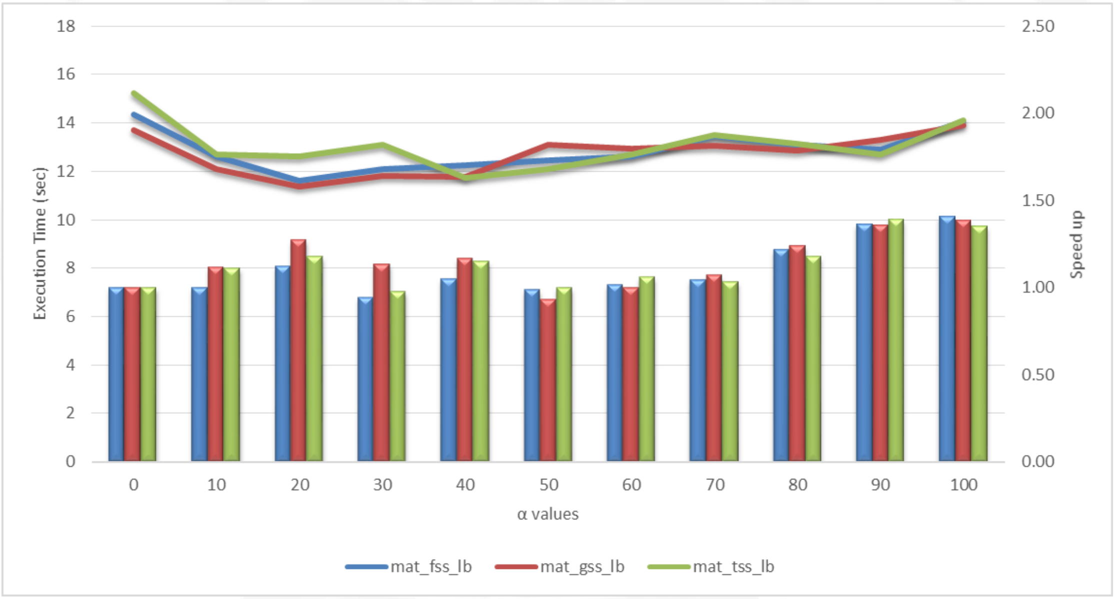


Figure 4.2: Improvement of Workload Balancing, Matrix Multiplication adaptive α scheduling

Table 4.4: Improvement of Workload Balancing, Matrix Multiplication execution time

	0	10	20	30	40	50	60	70	80	90	100
fss	14.34	12.62	11.62	12.07	12.26	12.43	12.62	13.47	13.11	12.88	14.07
gss	13.70	12.09	11.35	11.82	11.78	13.10	12.92	13.05	12.87	13.28	13.90
tss	15.25	12.68	12.60	13.10	11.73	12.07	12.68	13.48	13.12	12.69	14.08

4.2.2 Application 2: Sparse Matrix Multiplication

The multiplication of two matrixes is defined under the premise that the size of columns of the first matrix and the size of rows of the second matrix are the same. Sparse matrix multiplication is calculated only when the condition is satisfied, and, most of the results are zeros.

In the experiment of sparse matrix multiplication, the used size is 10240. As shown in the figure, factoring self-scheduling (the blue curve) has the best performance at $\alpha = 50$, guide self-scheduling (the red curve) has the best performance at $\alpha = 40$, and trapezoid self-scheduling (the green curve) has the best performance at $\alpha = 40$. As shown in FIG 4.3 TABLE 4.3.

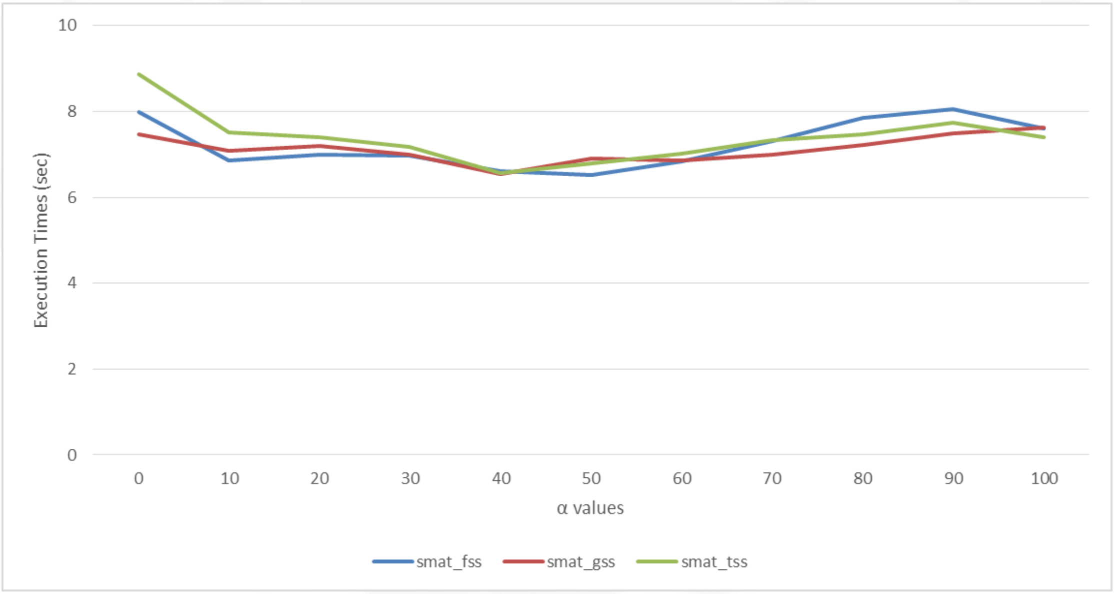


Figure 4.3: Sparse Matrix Multiplication adaptive α scheduling

Table 4.5: Sparse Matrix Multiplication execution time

	0	10	20	30	40	50	60	70	80	90	100
fss	7.99	6.86	6.99	6.96	6.61	6.52	6.82	7.30	7.85	8.05	7.61
gss	7.46	7.08	7.20	6.98	6.55	6.90	6.85	6.98	7.21	7.48	7.62
tss	8.86	7.50	7.40	7.18	6.57	6.78	7.02	7.34	7.47	7.73	7.40

We used the modified formula and repeated the experiment. We found the performance is almost the same. Just very small differences of performances of the two sets of experiments are found, thus the improvement of performance obtained by using modified formula is ineffective for the experiment of sparse matrix multiplication. As shown in FIG 4.4 TABLE 4.6.

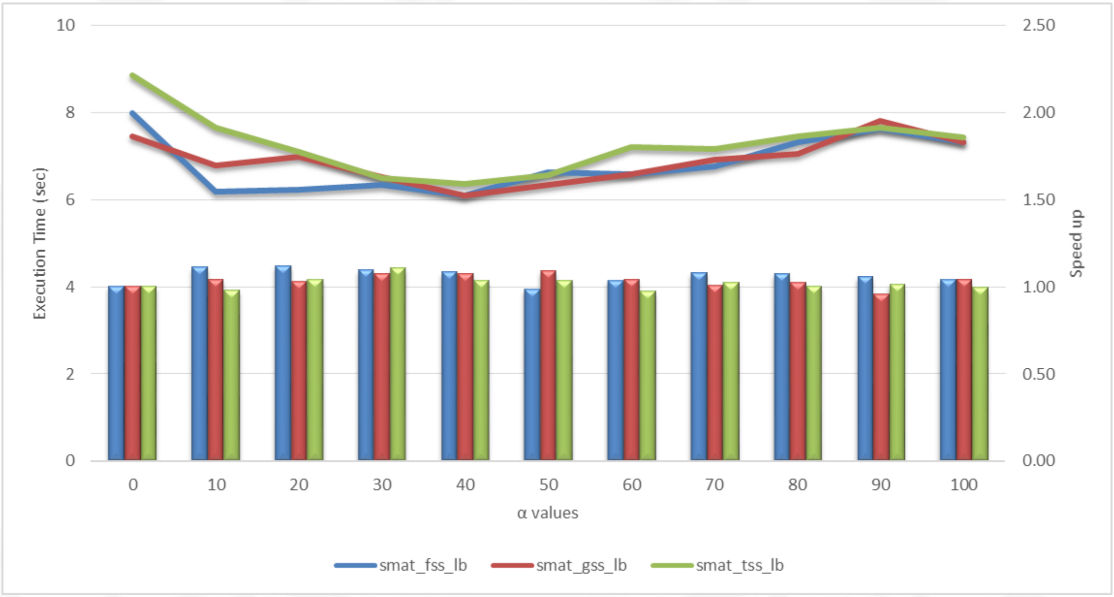


FIGURE 4.4: Improvement of Workload Balancing, Sparse Matrix Multiplication adaptive α scheduling

TABLE 4.6: Improvement of Workload Balancing, Sparse Matrix Multiplication execution time

	0	10	20	30	40	50	60	70	80	90	100
fss	7.99	6.18	6.24	6.34	6.09	6.63	6.59	6.76	7.33	7.62	7.32
gss	7.46	6.79	6.98	6.51	6.10	6.34	6.58	6.93	7.04	7.81	7.32
tss	8.86	7.65	7.09	6.49	6.36	6.57	7.21	7.17	7.46	7.65	7.44

4.2.3 Application 3: Mandelbrot Set Computation

The Mandelbrot set is a collection of fractals composed of points in the complex plane. Mandelbrot set is similar to Julia set in the use of iteration of a complex quadratic polynomial.

In the experiment of Mandelbrot set, the used size is 2048. As shown in the figure, factoring self-scheduling has the best performance at $\alpha = 10$ (the blue curve), guide self-scheduling has the best performance at $\alpha = 50$ (the blue curve), and trapezoid self-scheduling has the best performance at $\alpha = 20$ (the green curve). As shown in FIG 4.5 TABLE 4.7.

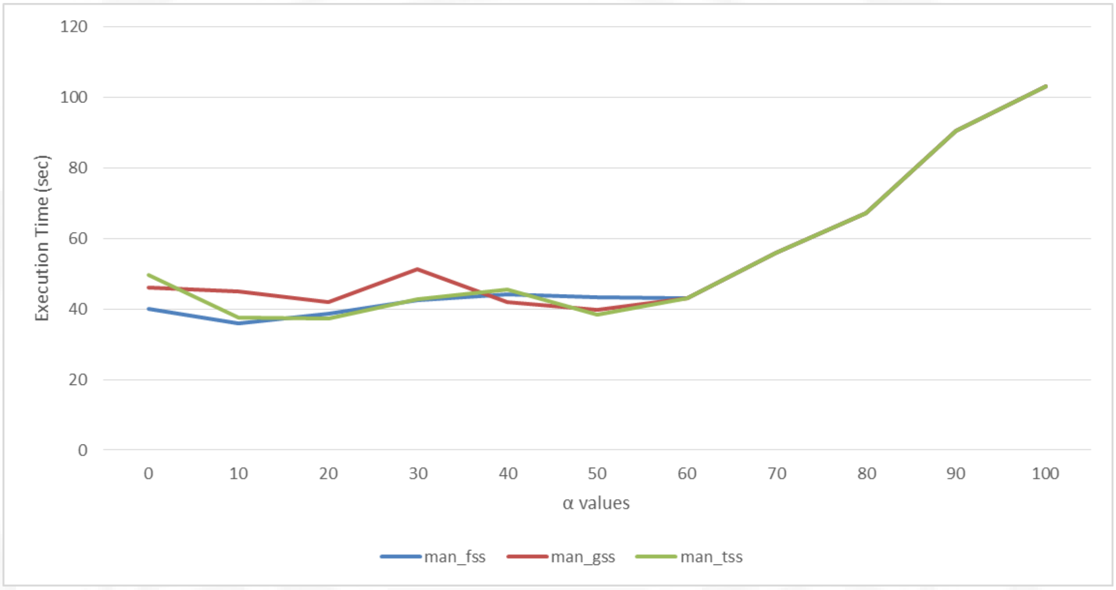


FIGURE 4.5: Mandelbrot Set Computation adaptive α scheduling

TABLE 4.7: Mandelbrot Set Computation execution time

	0	10	20	30	40	50	60	70	80	90	100
fss	40.03	35.76	38.54	42.60	44.04	43.31	42.92	55.93	67.18	90.53	102.9
gss	45.93	44.87	41.90	51.33	41.92	39.72	42.93	55.95	67.18	90.58	102.9
tss	49.73	37.63	37.36	42.89	45.50	38.46	42.90	55.93	67.18	90.56	102.9

We used the modified formula and repeated the experiment. We found when the value of α increases, the improvement of performance tends to increase, with a maximal speedup of 1.74 when α is 90 (the red bar). As shown in FIG 4.6 TABLE 4.8.

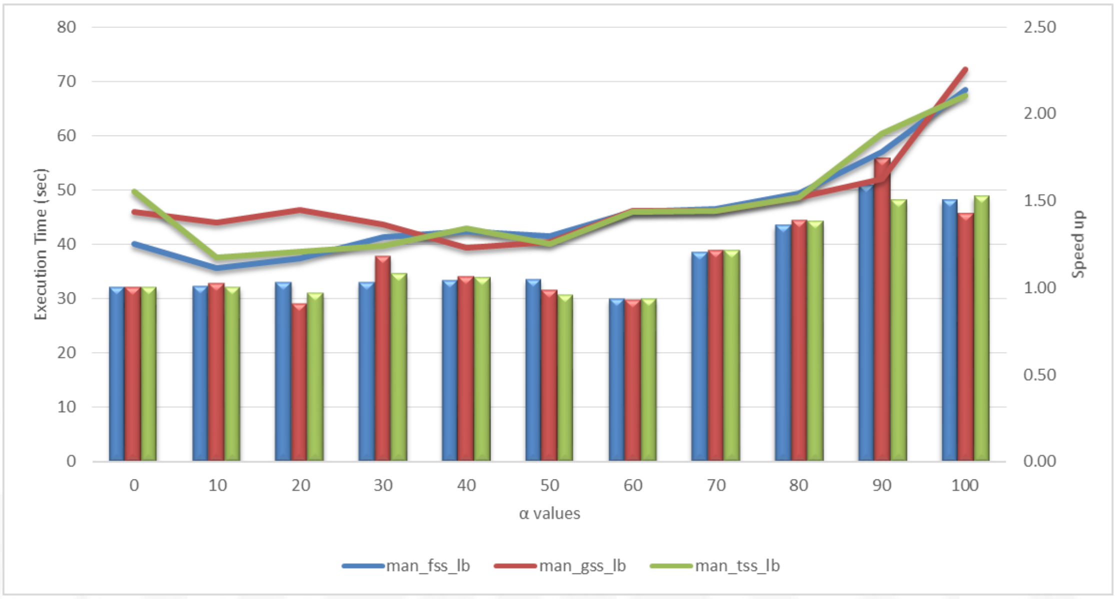


FIGURE 4.6: Improvement of Workload Balancing, Mandelbrot Set Computation adaptive α scheduling

TABLE 4.8: Improvement of Workload Balancing, Mandelbrot Set Computation execution time

	0	10	20	30	40	50	60	70	80	90	100
fss	40.03	35.55	37.35	41.29	42.32	41.43	45.97	46.56	49.42	57.02	68.37
gss	45.93	43.93	46.27	43.58	39.39	40.42	46.08	46.14	48.56	52.00	72.24
tss	49.73	37.53	38.60	39.70	42.98	40.07	45.97	46.19	48.61	60.35	67.39

4.2.4 Application 4: Circuit Satisfiability

In theoretical computer science, the circuit satisfiability problem is a decision-making problem which is to determine whether a given combinational logic circuit has an assignment of its inputs that makes the output true.

In the experiment of circuit satisfiability problem, the used size is 20 variables. As shown in the figure, factoring self-scheduling (the blue curve) has the best performance at $\alpha = 30$, guide self-scheduling (the red curve) has the best performance at $\alpha = 30$, and trapezoid self-scheduling (the green curve) has the best performance at $\alpha = 40$. As shown in FIG 4.7 TABLE 4.9.

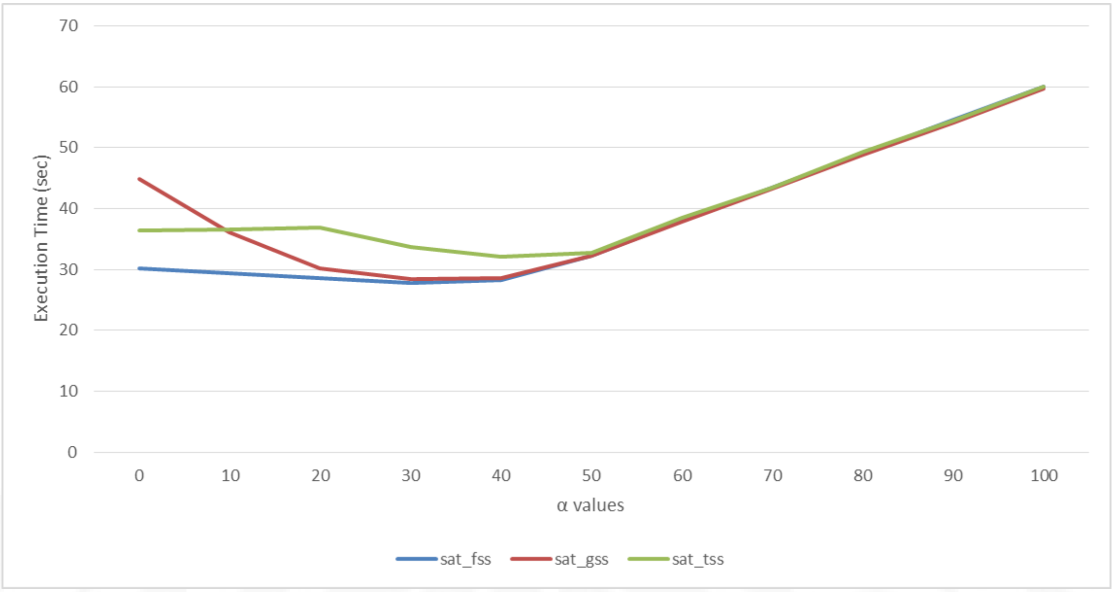


FIGURE 4.7: Circuit Satisfiability adaptive α scheduling

TABLE 4.9: Circuit Satisfiability execution time

	0	10	20	30	40	50	60	70	80	90	100
fss	30.23	29.30	28.65	27.83	28.28	32.31	37.97	43.39	49.09	54.63	60.01
gss	44.81	36.09	30.26	28.40	28.65	32.27	37.89	43.28	48.92	54.10	59.76
tss	36.36	36.58	36.93	33.64	32.11	32.66	38.52	43.48	49.42	54.45	60.08

We used the modified formula and repeated the experiment. We found when the value of α increases, the improvement of performance tends to increase, with a maximal speedup of 1.63 when α is 100 (the green bar). As shown in FIG 4.8 TABLE 4.10.

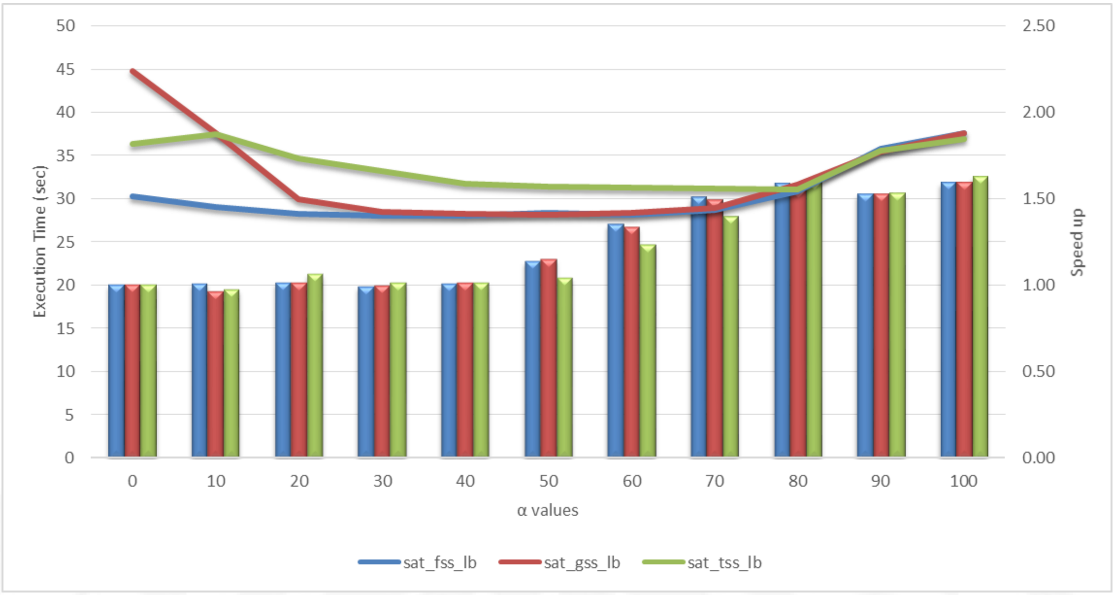


FIGURE 4.8: Improvement of Workload Balancing, Circuit Satisfiability adaptive α scheduling

TABLE 4.10: Improvement of Workload Balancing, Circuit Satisfiability execution time

	0	10	20	30	40	50	60	70	80	90	100
fss	30.23	29.08	28.25	28.06	28.03	28.38	28.13	28.72	30.89	35.84	37.61
gss	44.81	37.53	29.89	28.46	28.21	28.13	28.35	28.97	31.57	35.48	37.56
tss	36.36	37.47	34.64	33.16	31.68	31.40	31.29	31.18	31.06	35.56	36.93

4.3 Discussion

The experimental results are summarized in this section. For regular workloads, better performances can be achieved with a low α value; whereas, for irregular workloads, better performances can be achieved with a higher α value. However, poor performance was found when increased significantly high. We think the reason for this result is inaccurate estimates of the static weights for irregular workloads as well as the regular workloads. Therefore, we used a new formula to estimate weights. The experimental results show that in the case of regular workloads, performance does not degrade when increases; besides, in the case of irregular workloads, performance significantly improves when increases, with a maximal speedup of 1.74 times.

In the four experiments, only the performance of sparse matrix multiplication is not affected much from adopting the modified formula. We think this may be because the originally weights of it has been accurately estimated. Although the weights have been correctly estimated, its performance is not improved much. Part of the reason is that it has a low proportion of static scheduling; thus the effect of adjustment is relatively little.

Chapter 5

Conclusions and Future Work

In this work, we use a hybrid programming model with OpenMP and MPI to implement loop self-scheduling in the Xeon Phi environment. We then compare its performance of static and dynamic projects, such as the matrix multiplication, Sparse Matrix Multiplication, Mandelbrot set computation, and circuit satisfiability problems.

5.1 Conclusions

We combine the static and dynamic loop self-scheduling programs to execute on Xeon Phi. The loop iteration is based on the performance of our adaptively weighting assigned to the compute nodes. We study and compare our algorithm with four types of applications. These four applications have different characteristics: some applications have regular workload distributions and data communications needs in each step of the schedule; some have irregular workload distributions and do not need to do data communication in each step of the schedule. However, in most cases, we obtained better performance by our method of improvement than that of the previous schemes. Our method can be used in the case of irregular workloads; in addition, our method provides a better workload balancing according to the performance-based scheduling policy for the application program.

5.2 Future Work

Although our method can effectively workload balancing, but still some shortcomings. For example, it must be performed once at each node and the best performance does not improve and how do we prediction best alpha and so on. Find new ways to improve the shortcomings will be the future focus. For the future work, we plan to include more types of applications to validate our approach. We will also expand the experimental environment. In addition, we hope to be able to implement the method in a dynamic environment and investigate dynamic resource usages, such as CPU usage, memory usage, and network bandwidth.

References

[1] Benche M Grosu D Chronopoulos AT, Andonie R. A class of loop self-scheduling for heterogeneous clusters. In *Proceedings of the 2001 IEEE International Conference on Cluster Computing*, pages 282–291, 2001.

[2] Stumm M Sevcik KC Li H, Tandri S. Locality and loop scheduling on numa multiprocessors. In *Proceedings of the 1993 International Conference on Parallel Processing*, pages 140–147, 1993.

[3] Kuck D Polychronopoulos CD. Guided self-scheduling: a practical scheduling scheme for parallel supercomputers. In *IEEE Transactions on Computers*, pages 1425–1439, 1987.

[4] Tseng S-S Shih W-C, Yang C-T. A performance-based parallel loop scheduling on grid environments. *The Journal of Supercomputing*, 41(3):247–267, 2007.

[5] Li K-C Yang C-T, Cheng K-W. An enhanced parallel loop self-scheduling scheme for cluster environments. *The Journal of Supercomputing*, 34(3):315–335, 2005.

[6] Shun-Chyi Chang Yang C-T. A parallel loop self-scheduling on extremely heterogeneous pc clusters. *Journal of Information Science and Engineering*, 20(2):263–273, 2004.

[7] Shih W-C Yang C-T, Cheng K-W. On development of an efficient parallel loop self-scheduling for grid computing environments. *Parallel Computing*, 33(7-8):467–487, 2007.

[8] Chao-Tung Yang, Chao-Chin Wu, and Jen-Hsiang Chang. Performance-based parallel loop self-scheduling using hybrid openmp and mpi programming on multicore smp clusters. *Concurrency and Computation: Practice and Experience*, 23(8):721–744, 2011.

[9] Chao-Tung Yang, Wen-Chung Shih, and Lung-Hsing Cheng. Performance-based dynamic loop scheduling in heterogeneous computing environments. *The Journal of Supercomputing*, 59(1):414–442, 2012.

[10] Yiming Han and Anthony T. Chronopoulos. Scalable loop self-scheduling schemes implemented on large-scale clusters. In *IEEE International Symposium on Parallel & Distributed Processing*, pages 1735–1742, 2013.

[11] Legrand A Robert Y-Yang Y Beaumont O, Casanova H. Scheduling divisible loads on star and tree networks: Results and open problems. In *IEEE Transactions on Parallel and Distributed Systems*, pages 207–218, 2005.

[12] Lamont GB Bohn CA. Load balancing for heterogeneous clusters of pcs. *Future Generation Computer Systems*, 18:389–400, 2002.

[13] Slimani Y Yagoubi B. Load balancing strategy in grid environment. *Journal of Information Technology and Applications*, 1(4):285–296, 2007.

[14] Chao-Chin Wu, Lien-Fu Lai, Chao-Tung Yang, and Po-Hsun Chiu. Using hybrid mpi and openmp programming to optimize communications in parallel loop self-scheduling schemes for multicore pc clusters. *The Journal of Supercomputing*, 60(1):31–61, 2012.

[15] Fatma Omara Doaa M. Abdelkader. Dynamic task scheduling algorithm with load balancing for heterogeneous computing system. *Egyptian Informatics Journal*, 13(2):135–145, 2012.

[16] Francisco Almeida Alejandro Acosta, Vicente Blanco. Dynamic load balancing on heterogeneous multi-gpu systems. *Computers & Electrical Engineering*, 39(8):2591–2602, 2013.

[17] Michael Lysaght Gilles Civario. Dynamic load balancing using openmp 4.0. *High Performance Parallelism Pearls*, pages 185–200, 2015.

[18] Tseng S-S Yang C-T, Shih W-C. Dynamic partitioning of loop iterations on heterogeneous pc clusters. *The Journal of Supercomputing*, 44(1):1–23, 2007.

[19] Cheng L-H Yang C-T, Shih W-C. A performance-based dynamic loop scheduling on heterogeneous clusters. *Journal of Supercomputing*, 2010.

[20] Chao-Chin Wu, Chao-Tung Yang, Kuan-Chou Lai, and Po-Hsun Chiu. Designing parallel loop self-scheduling schemes using the hybrid mpi and openmp programming model for multi-core grid systems. *The Journal of Supercomputing*, 59(1):42–60, 2012.

[21] C. Rosales. Porting to the intel xeon phi: Opportunities and challenges. In *2013 Extreme Scaling Workshop*, pages 1 – 7, 2013.

[22] A. Heinecke. Accelerators in scientific computing is it worth the effort? In *High Performance Computing and Simulation (HPCS), 2013 International Conference on*, page 504, 2013.

[23] Andrew Milluzzi, Justin Richardson, Alan George, and Herman Lam. A multi-tiered optimization framework for heterogeneous computing. In *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, pages 1 – 6, 2014.

[24] Piyush Mehrotra Rupak Biswas-Lei Huang Barbara Chapman Haoqiang Jin, Dennis Jespersen. High performance computing using mpi and openmp on multi-core parallel systems. *Parallel Computing*, 37(9):562–575, 2011.

[25] David W. Walker Martin J. Chorley. Performance analysis of a hybrid mpi/openmp application on multi-core clusters. *Journal of Computational Science*, 1(3):168–174, 2010.

[26] Bernd Mohr Felix Wolf. Automatic performance analysis of hybrid mpi/openmp applications. *Journal of Systems Architecture*, 49(10-11):421–439, 2003.

[27] Guang Lin Xiaoliang Wan. Hybrid parallel computing of minimum action method. *Parallel Computing*, 39(10):638–651, 2013.

[28] Kent Milfeld Jerome Vienne, Carlos Rosales. Heterogeneous computing with mpi. *High Performance Parallelism Pearls*, pages 225–238, 2015.

[29] Wen mei Hwu. What is ahead for parallel computing. *Journal of Parallel and Distributed Computing*, 74(7):2574–2581, 2014.

[30] Intel xeon phi. http://www.intel.com.tw/content/www/tw/zh/processors/xeon/xeon-phi-detail.html.

[31] Intel manycore platform software stack (mpss). https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss.

[32] Intel parallel studio xe. https://software.intel.com/en-us/intel-parallel-studio-xe.

[33] Intel ark. http://ark.intel.com.

[34] Intel math kernel library - linpack. https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download.

[35] Openmp. http://openmp.org/wp.

[36] Open mpi. http://www.open-mpi.org.

[37] Top500. http://www.top500.org/.

[38] Openmp wiki. http://en.wikipedia.org/wiki/OpenMP.

[39] Mpi wiki. http://en.wikipedia.org/wiki/Message_Passing_Interface.

Appendix A

MPSS 3.4.4 Installation and Setup

1. Requirements

```
  1) Super-user privileges are required to install the Intel(R) MPSS 3.4.4
     release.

  2) Valid SSH keys are required for users (including "root" user) that need
     SSH access to each Intel(R) Xeon Phi(TM) coprocessor. To set SSH access,
     see Section 2.5, "SSH Access and Configuration for the Intel(R) Xeon Phi(TM)
     Coprocessor".

  3) Supported hardware platform with at least one Intel(R) Xeon Phi(TM)
     coprocessor installed.

  4) Linux* host operating system (default configuration).

NOTE: On both RHEL* and SLES*, issues were encountered in configuring the
      virtual network interfaces for the Intel(R) Xeon Phi(TM) coprocessors when
      "NetworkManager" is used. It is
      strongly recommended to use the "network daemon" instead.

      To switch to the network daemon:

             [host]# chkconfig NetworkManager off
             [host]# chkconfig network on
             [host]# service NetworkManager stop
             [host]# service network start
```

```
WARNING: Host OS kernel updates beyond the "officially supported" initial
release versions specified in the table above, may prevent the pre-built
Intel(R) MPSS 3.4 driver modules from loading.  Most Linux distributions
provide minor kernel updates and patches.  To ensure compatibility,
disable kernel updates on your host OS.

If the host system must run with an updated kernel, use the following
steps to re-build the MPSS modules for the updated kernel version prior
to installing MPSS.  If using Infiniband as an interconnect, refer to
section 9.1 of the MPSS User's guide, "Recompiling the Intel(R) MPSS
RPM specifically for OFED".

  O To recompile the MPSS host kernel modules:
    1) Ensure the prerequisites are installed:
       a. For Red Hat* Enterprise Linux
          [host]#  yum install kernel-headers kernel-devel
       b. For SUSE* Linux Enterprise Server
          [host]# zypper install kernel-default-devel rpm

    2) Regenerate the Intel(R) MPSS driver module package:
          [host]$  cd <path to extracted MPSS-3.x.-Linux.tar>/src/
          [host]$  rpmbuild --rebuild mpss-modules-*.rpm

    3) The newly built mpss-modules and mpss-modules-dev rpms will be
       located at:
       a. Red Hat* Enterprise Linux
          $HOME/rpmbuild/RPMS/x86_64/
       b. SUSE* Linux Enterprise Server
          /usr/src/packages/RPMS/x86_64/

    4) Copy the re-built mpss-modules and mpss-modules-dev RPMs from the
       respective directory specified in step 3 into the /modules
       directory of the extracted MPSS tar file. Continue to section 2.2
       to install MPSS.
```

2. Steps to Install Intel(R) MPSS

```
Note:
  o In SUSE*, "/sbin" and "/usr/sbin" are by default not in the user's execution
    search path. Attempts to run micctrl and various other commands discussed
     in this document will fail with the "command not found" messages from
    the shell.

    To avoid this in the current shell session execute the following prior to
    execution:

    export PATH=$PATH:/usr/sbin:/sbin
```

```

   To prevent the Intell MPSS service from starting add that line to the user's
   shell start up files.

   For Example: for bash the .bashrc file.

Steps:
  1) Remove any previous installations of Intel(R) MPSS. Refer to Section 2.3.
     To check for previous installed version of Intel(R) MPSS package:
            [host]# rpm -qa | grep -e intel-mic -e mpss

     *Note: Both yum and zypper support software upgrades and downgrades.
            However, it is necessary that Intel(R) MPSS upgrades and downgrades
            be carried out by complete uninstallation of existing software
            followed by a clean installation of the replacement software.
            Section 2.3 describes uninstalls.

  2) Disarm Module security policies:
        a) The SUSE* Linux* Enterprise Server release requires setting the kernel
        to allow non-SUSE* driver modules to be loaded. Edit the file
        "/etc/modprobe.d unsupported -modules" and set the value of
        "allow_unsupported_modules" to 1.

        b) If SELinux is installed, disable SELinux before installing Intel(R)
        MPSS software, to prevent SELinux from overriding standard Linux*
        permissions settings.

  3) Download the Linux tar file for this release,
     then untar and install the Intel(R) MPSS package.
            [host]$ tar xvf mpss-3.4.4-linux.tar

     *Note: At this point a "modules" directory is created containing all
            kernel specific modules.

            [host]$ cd mpss-3.4.4
            [host]$ cp ./modules/*`uname -r`*.rpm .

        o Red Hat* Enterprise Linux*
            [host]# yum install *.rpm

        o SUSE* Linux* Enterprise Server
            [host]# zypper install *.rpm

     *Note: yum and zypper are the recommended tools for installing MPSS RPMs.

     *Note: MPSS packages are not GPG signed. If local package GPG check
            (localpkg_gpgcheck) is enabled in yum.conf, the --nogpgcheck option
            must be used:
```

```
          [host]# yum install --nogpgcheck *.rpm

  4) Load the mic.ko driver, and then initialize MPSS Default Settings.
          [host]# modprobe mic
          [host]# micctrl --initdefaults

  4b) Configure MPSS Via .conf Files (optional).
        o The MPSS_Users_Guide.pdf document explains in detail
          how to modify Intel(R) MPSS configuration files.

  5) Update Flash & SMC (see Section 2.4). Flash and SMC updates to versions
     specified in this release are required.

  6) Start Intel(R) MPSS by using the Linux* service command:
        RHEL 6.x and SUSE
          [host]# service mpss start
        RHEL 7.x
          [host]# systemctl start mpss

        RHEL 6.x and SUSE
        o To configure the Intel(R) MPSS service to start when the host OS boots:
          [host]# chkconfig mpss on

        o To disable the Intel(R) MPSS service from starting when the host OS
        boots:

          [host]# chkconfig mpss off

        RHEL 7.x,
        o To enable or disable service start on boot, the command is:
          [host]# systemctl enable mpss
          [host]# systemctl disable mpss
```

3. Steps to Uninstall Intel(R) MPSS

```
Note: The uninstall script will not uninstall rebuilt drivers. You must remove
      them manually before running the uninstall.sh script.
      On RedHat: [host]$yum remove <name of rebuilt package rpm>
      On SUSE Linux: [host]$ zypper remove <name of rebuilt package rpm>

  o To uninstall 3.x-based builds:

     1) Unload the MPSS driver using:
        RHEL 6.x and SUSE
            [host]# service mpss unload
        RHEL 7.x
            [host]# systemctl disable mpss
            [host]# modprobe -r mic

     2) Uninstall:
        a. To uninstall MPSS-3.x:
           [host]$ cd mpss-3.4.4
           [host]$ ./uninstall.sh

        b. To uninstall MPSS-2.x:
        o Red Hat* Enterprise Linux*
            [host]# yum remove intel-mic\*

        o SUSE* Linux* Enterprise Server
            [host]# zypper remove intel-mic\*
```

Appendix B

Intel Parallel Studio XE Installation and Setup

```
 I.      Download `Intel Parallel Studio XE' from Intel website


II.      Install `Intel Parallel Studio XE'
# tar -xvf parallel_studio_xe_2015_update3.tgz
# cd parallel_studio_xe_2015_update3
# ./install.sh

# vim /.bashrc

    source /opt/intel/composerxe/bin/compilervars.sh intel64
    source /opt/intel/impi/4.1.1.036/bin64/mpivars.sh

# source /.bashrc
```

Appendix C

HPL Installation and Setup

```
I.      Download `Intel® Math Kernel Library - LINPACK' from website
Download it ,then move to `/home/'.
# tar -xvf l\_lpk\_p\_11.3.0.004

II.     Setting `HPL'
# vim /root/l_lpk_p_11.3.0.004/compilers_and_libraries_2016.0.038/ \\
linux/mkl/benchmark/mp_linpack

##******************************************************************************
##  Copyright(C) 2005-2015 Intel Corporation. All Rights Reserved.
##
##  The source code, information  and  material ("Material") contained herein is
##  owned  by Intel Corporation or its suppliers or licensors, and title to such
##  Material remains  with Intel Corporation  or its suppliers or licensors. The
##  Material  contains proprietary information  of  Intel or  its  suppliers and
##  licensors. The  Material is protected by worldwide copyright laws and treaty
##  provisions. No  part  of  the  Material  may  be  used,  copied, reproduced,
##  modified, published, uploaded, posted, transmitted, distributed or disclosed
##  in any way  without Intel's  prior  express written  permission. No  license
##  under  any patent, copyright  or  other intellectual property rights  in the
##  Material  is  granted  to  or  conferred  upon  you,  either  expressly,  by
##  implication, inducement,  estoppel or  otherwise.  Any  license  under  such
##  intellectual  property  rights must  be express  and  approved  by  Intel in
##  writing.
##
##  *Third Party trademarks are the property of their respective owners.
##
##  Unless otherwise  agreed  by Intel  in writing, you may not remove  or alter
##  this  notice or  any other notice embedded  in Materials by Intel or Intel's
##  suppliers or licensors in any way.
##
```

```
##*****************************************************************************
##  Content:
##      Intel(R) Math Kernel Library MP LINPACK tests creation
##
##*****************************************************************************
#
#  -- High Performance Computing Linpack Benchmark (HPL)
#     HPL - 2.1 - October 26, 2012
#     Antoine P. Petitet
#     University of Tennessee, Knoxville
#     Innovative Computing Laboratory
#     (C) Copyright 2000-2008 All Rights Reserved
#
#  -- Copyright notice and Licensing terms:
#
#  Redistribution  and  use in  source and binary forms, with or without
#  modification, are  permitted provided  that the following  conditions
#  are met:
#
#  1. Redistributions  of  source  code  must retain the above copyright
#  notice, this list of conditions and the following disclaimer.
#
#  2. Redistributions in binary form must reproduce  the above copyright
#  notice, this list of conditions,  and the following disclaimer in the
#  documentation and/or other materials provided with the distribution.
#
#  3. All  advertising  materials  mentioning  features  or  use of this
#  software must display the following acknowledgement:
#  This  product  includes  software  developed  at  the  University  of
#  Tennessee, Knoxville, Innovative Computing Laboratory.
#
#  4. The name of the  University,  the name of the  Laboratory,  or the
#  names  of  its  contributors  may  not  be used to endorse or promote
#  products  derived   from   this  software  without  specific  written
#  permission.
#
#  -- Disclaimer:
#
#  THIS  SOFTWARE  IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#  ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,  INCLUDING,  BUT NOT
#  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE UNIVERSITY
#  OR  CONTRIBUTORS  BE  LIABLE FOR ANY  DIRECT,  INDIRECT,  INCIDENTAL,
#  SPECIAL,  EXEMPLARY,  OR  CONSEQUENTIAL DAMAGES  (INCLUDING,  BUT NOT
#  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#  DATA OR PROFITS; OR BUSINESS INTERRUPTION)  HOWEVER CAUSED AND ON ANY
#  THEORY OF LIABILITY, WHETHER IN CONTRACT,  STRICT LIABILITY,  OR TORT
#  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
# ######################################################################
#
# ----------------------------------------------------------------------
# - shell --------------------------------------------------------------
# ----------------------------------------------------------------------
#
SHELL        = /bin/sh
#
CD           = cd
CP           = cp
LN_S         = ln -fs
MKDIR        = mkdir -p
RM           = /bin/rm -f
TOUCH        = touch
#
# ----------------------------------------------------------------------
# - Platform identifier ------------------------------------------------
# ----------------------------------------------------------------------
#
ARCH         = mic
#
# ----------------------------------------------------------------------
# - HPL Directory Structure / HPL library ------------------------------
# ----------------------------------------------------------------------
#
# Set TOPdir to the location of where this is being built
ifndef  TOPdir
TOPdir = `pwd`
endif
INCdir       = $(TOPdir)/include
BINdir       = $(TOPdir)/bin/$(ARCH)
LIBdir       = $(TOPdir)/lib/$(ARCH)
#
HPLlib       = $(LIBdir)/libhpl.a
ifeq "$(version)" "hybrid"
HPLlibHybrid =  $(TOPdir)/lib_hybrid/$(ARCH)/libhpl_hybrid.a
else
HPLlibHybrid =
endif
ifeq "$(static)" "y"
STATICFLAG=-i-static -z noexecstack -z relro -z now -static_mpi
else
STATICFLAG=-i-static -z noexecstack -z relro -z now
endif
#
# ----------------------------------------------------------------------
# - Message Passing library (MPI) --------------------------------------
# ----------------------------------------------------------------------
# MPinc tells the  C  compiler where to find the Message Passing library
# header files,  MPlib  is defined  to be the name of  the library to be
# used. The variable MPdir is only used for defining MPinc and MPlib.
#
#MPdir        = /opt/intel/mpi/3.0
#MPinc        = -I$(MPdir)/include64
#MPlib        = $(MPdir)/lib64/libmpi.a
#MPlib        = $(MPdir)/lib64/libmpich.a
#
# ----------------------------------------------------------------------
# - Linear Algebra library (BLAS) -------------------------------------
# ----------------------------------------------------------------------
# LAinc tells the  C  compiler where to find the Linear Algebra  library
# header files,  LAlib  is defined  to be the name of  the library to be
# used. The variable LAdir is only used for defining LAinc and  LAlib.
#
ifndef  LAdir
LAdir        = $(TOPdir)/../../..
endif
ifndef  LAinc
LAinc        = $(LAdir)/mkl/include
endif
ifndef  LAlib
ifeq "$(version)" "hybrid"
LAlib        = -L$(LAdir)/mkl/lib/mic \
               -Wl,--start-group \
                   $(LAdir)/mkl/lib/mic/libmkl_intel_lp64.a \
                   $(LAdir)/mkl/lib/mic/libmkl_intel_thread.a \
                   $(LAdir)/mkl/lib/mic/libmkl_core.a \
               -Wl,--end-group \
               -lpthread -ldl $(HPLlibHybrid)
else
LAlib        = -L$(LAdir)/mkl/lib/mic \
               -Wl,--start-group \
                   $(LAdir)/mkl/lib/mic/libmkl_intel_lp64.a \
                   $(LAdir)/mkl/lib/mic/libmkl_sequential.a \
                   $(LAdir)/mkl/lib/mic/libmkl_core.a \
               -Wl,--end-group \
               -ldl
endif
endif
#
# ----------------------------------------------------------------------
# - F77 / C interface --------------------------------------------------
# ----------------------------------------------------------------------
# You can skip this section  if and only if  you are not planning to use
# a  BLAS  library featuring a Fortran 77 interface.  Otherwise,  it  is
# necessary  to  fill out the  F2CDEFS  variable  with  the  appropriate
# options.  **One and only one**  option should be chosen in **each** of
# the 3 following categories:
#
# 1) name space (How C calls a Fortran 77 routine)
#
# -DAdd_              : all lower case and a suffixed underscore  (Suns,
#                       Intel, ...),                          [default]
# -DNoChange          : all lower case (IBM RS6000),
# -DUpCase            : all upper case (Cray),
# -DAdd__             : the FORTRAN compiler in use is f2c.
#
# 2) C and Fortran 77 integer mapping
#
# -DF77_INTEGER=int   : Fortran 77 INTEGER is a C int,         [default]
# -DF77_INTEGER=long  : Fortran 77 INTEGER is a C long,
# -DF77_INTEGER=short : Fortran 77 INTEGER is a C short.
#
# 3) Fortran 77 string handling
#
# -DStringSunStyle    : The string address is passed at the string loca-
#                       tion on the stack, and the string length is then
#                       passed as  an  F77_INTEGER  after  all  explicit
#                       stack arguments,                       [default]
# -DStringStructPtr   : The address  of  a  structure  is  passed  by  a
#                       Fortran 77  string,  and the structure is of the
#                       form: struct {char *cp; F77_INTEGER len;},
# -DStringStructVal   : A structure is passed by value for each  Fortran
#                       77 string,  and  the  structure is  of the form:
#                       struct {char *cp; F77_INTEGER len;},
# -DStringCrayStyle   : Special option for  Cray  machines,  which  uses
#                       Cray  fcd  (fortran  character  descriptor)  for
#                       interoperation.
#
F2CDEFS      = -DAdd__ -DF77_INTEGER=int -DStringSunStyle
#
# ----------------------------------------------------------------------
# - HPL includes / libraries / specifics -------------------------------
# ----------------------------------------------------------------------
#
HPL_INCLUDES = -I$(INCdir) -I$(INCdir)/$(ARCH) -I$(LAinc) $(MPinc)
HPL_LIBS     = $(HPLlib) $(LAlib) $(MPlib)
#
# - Compile time options -----------------------------------------------
#
# -DHPL_COPY_L           force the copy of the panel L before bcast;
# -DHPL_CALL_CBLAS       call the cblas interface;
# -DHPL_DETAILED_TIMING  enable detailed timers;
# -DASYOUGO              enable timing information as you go (nonintrusive)
# -DASYOUGO2             slightly intrusive timing information
# -DASYOUGO2_DISPLAY     display detailed DGEMM information
# -DENDEARLY             end the problem early
# -DFASTSWAP             insert to use DLASWP instead of HPL code
# -DHYBRID               use for Hybrid OpenMP/MPI mode
#
# By default HPL will:
#    *) not copy L before broadcast,
#    *) call the BLAS Fortran 77 interface,
#    *) not display detailed timing information.
#
HPL_OPTS     = -DHYBRID
#

ifeq "$(version)" "hybrid"
HPL_OPTS     = -DHYBRID
else
HPL_OPTS     =
endif
# ----------------------------------------------------------------------
#
HPL_DEFS     = $(F2CDEFS) $(HPL_OPTS) $(HPL_INCLUDES)
#
# ----------------------------------------------------------------------
# - Compilers / linkers - Optimization flags ---------------------------
# ----------------------------------------------------------------------
#
# Next two lines should be commented in case of using Intel Compilers:
# CC      = mpicc
# CCFLAGS = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops -W -Wall
# Next nine lines should be commented in case of using GNU compilers:
  CC      = mpiicc
ifeq "$(version)" "hybrid"

ICCVERSION:= $(shell expr `icc -dumpversion | cut -f1 -d.` \>= 15)
ifeq "$(ICCVERSION)" "1"
        OMP_DEFS = -qopenmp
else
        OMP_DEFS = -openmp
endif

ifndef MKLINCDIR
        MKLINCDIR = -I"$(TOPdir)/../../include"
endif
        CCFLAGS = $(HPL_DEFS) $(MKLINCDIR) -O3  -mmic -w -ansi-alias -i-static \\
        -z noexecstack -z relro -z now $(OMP_DEFS) -nocompchk
else
```

```
        CCFLAGS = $(HPL_DEFS) -O3 -mmic -w -nocompchk
endif

#
CCNOOPT      = $(HPL_DEFS) -O0 -mmic -w -nocompchk
#
# On some platforms,  it is necessary  to use the Fortran linker to find
# the Fortran internals used in the BLAS library.
#
LINKER       = $(CC)
ifeq "$(version)" "hybrid"
        LINKFLAGS    = $(CCFLAGS) $(OMP_DEFS) -mmic -mt_mpi $(STATICFLAG) -nocompchk
else
        LINKFLAGS    = $(CCFLAGS) -mmic -nocompchk $(STATICFLAG)
endif

#
ARCHIVER     = ar
ARFLAGS      = r
RANLIB       = echo
#
# ----------------------------------------------------------------------
MAKE = make TOPdir="$(TOPdir)" LAdir="$(LAdir)" LAinc="$(LAinc)"
            LAlib="$(LAlib)" MKLINCDIR="$(MKLINCDIR)"

Basically you are ready to go to compile

# cd /root/l_lpk_p_11.3.0.004/compilers_and_libraries_2016.0.038/linux/mkl/benchmark/ \\
mp_linpack
# make ARCH=mic

# cd /root/l_lpk_p_11.3.0.004/compilers_and_libraries_2016.0.038/linux/mkl/benchmark/ \\
mp_linpack/bin/mic
# vim HPL.dat
```

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
29696 28672 24576        Ns
1            # of NBs
240          NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
1            Ps
1            Qs
```

```
16.0         threshold
1            # of panel fact
1            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
5            NBMINs (>= 1)
1            # of panels in recursion
3            NDIVs
1            # of recursive panel fact.
2            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0 1          L1 in (0=transposed,1=no-transposed) form
0 1          U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
# ----------------------------------------------------------------------

# cd /root/l_lpk\_p_11.3.0.004/compilers_and_libraries_2016.0.038/linux/mkl/benchmark/ \\
mp_linpack
# scp -r mp_linpack mic0:
# ssh mic0
# cd mp_linpack
# mpirun ./xhpl
```