Task Mapping on Supercomputers with Cellular Networks

A Dissertation Presented

by

Yongzhi Chen

to

The Graduate School

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

August 2008

Stony Brook University

The Graduate School

Yongzhi Chen

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Yuefan Deng - Dissertation Advisor
Professor, Department of Applied Mathematics and Statistics

W. Brent Lindquist - Chairperson of Defense
Professor, Department of Applied Mathematics and Statistics

John Reinitz - Committee Member
Professor, Department of Applied Mathematics and Statistics

James Davenport - Outside Committee Member
Director, Computational Science Center
Brookhaven National Laboratory

This dissertation is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Task Mapping on Supercomputers with Cellular Networks

by

Yongzhi Chen

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2008

This thesis focuses on techniques of task mapping for solving problems on parallel computers with hundreds of thousands of processors on cellular networks. Task mapping is a serious intellectual challenge and a practical tool for unleashing the potential power of supercomputers. It is challenging because of both the astronomical searching space and the high dependence on the exact nature of the applications and the computers. In this thesis, we propose two general static mapping models to optimize the assignment of tasks on heterogeneous, distributed-memory, ultra-scalable computers. In our models, the underlying application problems can be appropriately decomposed to subtasks with known computational load and known inter-task communicational demands. We also know, or can conveniently measure, the computing systems' specifications such as individual processor speed and inter-processor communication cost. Our models abstract an application as a demand matrix and a parallel computer as a load matrix and a supply matrix with which we construct our models as minimizing the objective function value for completing the application on the given computer.

We have tested several applications on Blue Gene/L supercomputer with 3D mesh and torus networks. For a 2D wave equation, the mappings generated by our models reduced communication by 51% for 3D-mesh and 31% for 3D-torus over the default MPI rank order mapping. For SMG2000 application, our mapping can reduce communication and total time by 16% and 5% over the default MPI rank order mapping, respectively. For NPB MG, we improve the communication time and benchmark result by 53% and 13%, respectively. For NPB CG, we improve the communication time and benchmark result by 43% and 22%, respectively. We believe that our models are useful for task assignment for broad applications on a family of supercomputers with cellular networks.

To My wife, Lujin Wang,

My Father, Zuqi Chen, and My Mother, Yuan Wang

with My Love!

Table of Contents

List of Figures

List of Tables

Acknowledgements

First of all, I would like to thank my parents, my wife, my brother and sister for their invaluable love and support for me in my life.

I want to express my gratitude to my advisor, Professor Yuefan Deng, for his years of guidance and encouragement. He helped me to set and achieve the higher research goals which makes this thesis possible.

I would like to thank Dr. James Davenport, for his years of support and being my external committee member. He helped me to get Blue Gene accounts in Argonne National Laboratory and Brookhaven National Laboratory which makes the experimental data collection possible.

I want to thank all current and past group members. Thanks go to Janet Laura Braunstein and Reid Powell for the thesis writing help, Xin Chen for joint work, Guowen Han, Yuxiang Gao, Bin Fang, Peter Rissland, Tatiana Polishchuk, Rui Feng, Peng Zhang and Zhihao Lou, for their help and friendship. Life at Stony Brook has been a great experience for me.

This research utilized resources at the New York Center for Computational Sciences at Stony Brook University/Brookhaven National Laboratory which is supported by the U.S. Department of Energy under Contract No. DE-AC02-98CH10886 and by the State of New York. We gratefully acknowledge use of “BGL,” a 1024-node IBM Blue Gene/L system operated by the Argonne Leadership Computing Facility at Argonne National Laboratory.

Publications

1. Yongzhi Chen and Yuefan Deng. **Task Mapping on Supercomputers with Cellular Networks**. *Computer Physics Communications* (2008), doi:10.1016/j.cpc.2008.04.011.

2. Yongzhi Chen and Yuefan Deng. **Detailed analysis of communication load balance for the overall performance gain on Blue Gene Supercomputer**. *Computer Physics Communications.* In progress.

Chapter 1

Introduction

1.1 Motivation

With the development of parallel computers, people with high expectations in shorter execution time for their applications on their computers often find such expectation unfulfillable due to two key reasons: First, the complexity of parallel algorithms and the associated software developments that never arise in sequential algorithms, such as synchronization, deadlock avoidance, load balancing, mapping, and communication, etc, is a serious obstacle. Second, the diversity of performance characteristics and programming interfaces that exists in the different parallel computers makes code portability nearly impossible [58]. These two challenges unique to parallel computing necessitate the creations of tools such as parallel compilers that may help decomposition, mapping and scheduling [57].

The mapping problem is known to be NP-Complete [14, 42, 55] and yet it becomes crucial for parallel computing as the systems become larger and more complicated:

First, previous research has demonstrated the sensitive dependance of the performance on the assignment of mapping the subtasks to the processors of new generation of supercomputers for some applications. A good mapping can reduces the average running time by efficient utilization of the network resource [33].

Second, the cost difference between a good and a poor mapping in a supercomputer increases with the number of processors [34].

Third, for an ultra-scalable supercomputer system, the searching space for an efficient mapping becomes astronomical.

Fourth, task mapping is highly dependent on both the nature of the underlying application and the properties of the machine. Re-mapping must be considered when a new application or a new machine is involved.

Fifth, the modern supercomputers may offer a mapping interface to support users to re-map jobs at run-time, allowing a convenient modification to generate a significant improvement.

1.2 Contributions

Our major contributions to static mapping research include mapping models and new insights to map reasonably arbitrary application problems onto parallel computers for achieve competitive improvement of system performance. Our models are designed to abstract an application as a demand matrix and the computer network as a supply matrix with the objective function value as minimum completion time. By adopting the MPI latency to construct the supply matrix and utilizing the proper profiling tool, our mapping models provide a productive tool for unleashing the potential of supercomputers. We verify our models by benchmarking a series of scientific applications. In summary, the main contributions in the research are:

- Two general mapping models are introduced. The schemes for mapping tasks of an arbitrary application onto a given network topology to reduce the execution time are proposed.

- A mapping suite is provided. Associated with the proper profiling tool, our models offer an efficient and productive mapping facility.

- Our models on scientific applications and multiple benchmarks are verified. Tests of 2D Wave equation, SMG2000, NPB MG and CG show that our models can improve performance significantly.

- Both theoretical and practical systems are analyzed. The exact model mappings for some cases and corresponding detailed timing analysis are given.

1.3 Thesis organization

In this thesis, practical static models are proposed and simulated annealing is adopted to solve the mapping problem in heterogeneous, distributed-memory parallel computers such as Blue Gene with cellular networks. The static mapping problem and its related work are reviewed in Chapter 2. In Chapter 3, a brief introduction of available parallel computers is given. Chapter 4 presents the models and demonstrates their theoretical superiority. Several realistic applications are presented and discussed in Chapter 5. The analyses of the experimental timing results are given in Chapter 6. Finally, conclusions and future work are described in Chapter 7.

Chapter 2

Static Mapping Problem

2.1 Mapping Problem

The mapping problem has not had a commonly accepted definition in the literature on parallel computing [62]. Some define it as the procedure of mapping each one of the interacting tasks to individual processors of the parallel computer system so as to minimize the total execution time [15, 16], while others claim that the mapping problem includes both task assignment and scheduling. In this thesis, we adopt Tabli's definition as briefly illustrated below [28].

The application and the parallel computer are represented as static graphs G_A and G_P, respectively. $G_A = (V_A, E_A)$ is a graph where V_A represents tasks and E_A means communication requests among them with weights being the communication loads. $G_P = (V_P, E_P)$ is a graph where V_P represents processors and E_P means the links among processors with weights being the per unit communication cost.

A mapping problem can then be defined as finding a mapping: $V_A \rightarrow V_P$

to minimize the objective function value that associates with each mapping.

In static mapping, the interaction pattern and execution time for each task on each processor are static and known prior to the execution and thus the tasks could be represented by a static task graph. There are two different types of graphs: task precedence graph (TPG) and task interaction graph (TIG). TPG is a directed graph in which directed edges describe execution dependencies while interaction patterns are represented by undirected edges in TIG. In TIG, tasks can be executed independently and simultaneously [16, 45]. This kind of independent set of tasks is also called a meta-task [41, 74]. In this thesis, we consider only the TIG cases for task assignment.

2.2 Related Work

Task mapping has been studied by many groups for a long time and many approaches are developed under different assumptions on applications and architectures [34]. We try to classify all these related research based on two aspects: models and techniques adopted in the models.

The model aspect concerns the criterion, the objective function and constraints. It reflects the assumptions regarding platform and decomposition strategy. The objective function can be the total execution time, or only the computational time or only the communication time. The decomposition strategy describes how the application is decomposed into subtasks. The most important assumption of a platform is its heterogeneity of computation and communication. The technique aspect considers the best approach for searching for optimal or near-optimal solutions. Solutions of the models are based on two types of algorithms, exact and heuristic.

2.2.1 Mapping Models

The field of static mapping can be divided into two categories. The communication costs are independent of the processors' locations in the first category, i.e., processor links are homogeneous. This type of model may include both computation and communication costs in their cost functions by an appropriate formulation. However, the communication costs in the second category are highly dependent on the location of the communicating processors, which are common in many ultra-scalable systems. This type of model usually involves minimization of the communication costs only.

We will introduce and discuss the most important static models illustrated as Fig. 2.1.

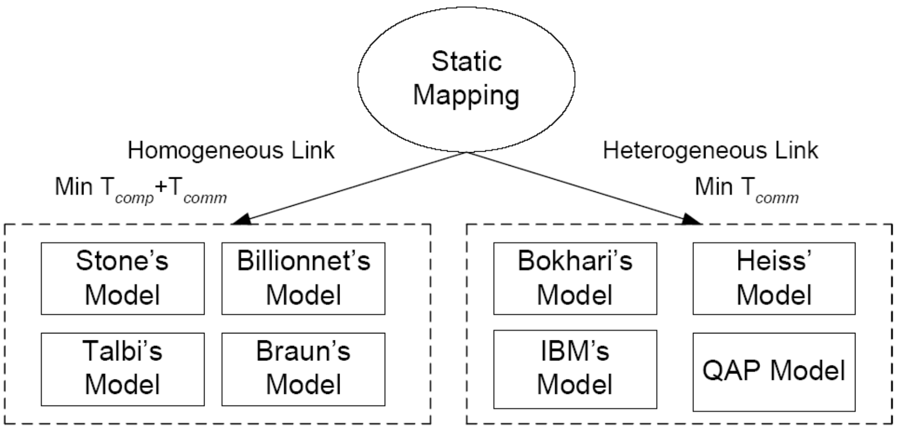


Figure 2.1: Static Mapping Models

Stone's 1977 Model

Stone's model [44] is formulated to find an optimal assignment of program modules onto a two-processor distributed computer system to minimize the cost of intermodule reference and running which can be represented as follows:

$$\min_{\Pi \in \Gamma} \sum_{t \in T} r[t, \Pi(t)] + \sum_{(t,t') \in E_A, \Pi(t) \neq \Pi(t')} c(t, t')$$

where Γ is the set of possible task mappings; Π is a mapping from tasks to processors; $r[t, \Pi(t)]$ denotes the time required to compute task t on the processor $\Pi(t)$ and $c(t, t')$ denotes the time associated with module t calling t' if they are mapped to different processors.

Stone assumes no parallelism in the system and the running times of a module on different processors are different (from very small value such as 2 to ∞) due to different FPU and memory situations. Stone's model considers only the case where computation and communication occur sequentially although it is easily overlooked [62].

Stone shows that the well-known maximum flow (Ford-Fulkerson) algorithm can be used to solve this model with $n = 2$. When $n = 2$, processors P_1 and P_2 are added into the original interconnection graph as the unique source node (S_1) and the unique sink node (S_2), respectively. For each node other than these two, an edge is added from that node to S_1 and S_2. The weight of the edge to S_1 carried the cost of executing the corresponding module on P_2 and vice versa.

Stone proves that the weight of a cut set of this modified graph is equal to the cost of the corresponding assignment. He also extends it to 3- and n-processors cases without a complete efficient solution. Fernández-Baca [22] shows that it is NP-complete in general. In [62], Norman also discussed the important extension to this model by adding an explicit parallel processing constraints [17].

Bokhari's 1981 model

Stone's model is not directly applicable when computation and communication are not incurred sequentially. As an alternative, Bokhari considers a model that maps n tasks to n processors to find the minimum communication cost independent of computation costs which can be represented as follows [15]:

$$\min \sum_{x \in V_p, y \in V_p} G_p(x, y) \cdot G_a(f_m(x), f_m(y))$$

In this model, the problem modules and the parallel computers are represented by two undirected graphs denoted by $G_p = \langle V_p, E_p \rangle$ and $G_a = \langle V_a, E_a \rangle$. Bokhari adopts *cardinality*, the number of module edges falling on processor edges, as the criterion. Therefore $G_a(f_m(x), f_m(y)) = 1$ only if the processors onto which x and y are mapped are directly connected. In [15], Bokhari proposes mapping a series of structured problems of 9 to 49 modules onto the finite element machine (FEM) of sizes 4×4 to 7×7, in which each processor is directly connected to its 8 neighbors. He uses a pairwise interchange algorithm with probabilistic jumps to solve the above problems and get good results. Bokhari also points out that this model can be transformed to a graph isomorphism problem, a bandwidth reduction problem for sparse matrices, or a quadratic assignment problem under proper conditions. Bokhari's model is used in the analysis of algorithms for SIMD architectures and is extended to consider contention in the processor network [62, 77]. This model is indeed very close to the graph isomorphism problem since the weights of both G_p and G_a are all equal to 1. That means it can not reflect the amount differences among intermodule communications and inter processor links. Therefore, it is inaccurate for many systems.

Billionnet's 1992 model

Stone's model assumes sequential execution and Bokhari's model considers only communication overheads. Many research efforts have extended these two models to integrate computation and communication costs into one model [62].

In [10], Billionnet considers the task assignment problem in the heterogeneous multiple processors system. The problem is formulated as the following 0-1 programming problem:

$$\min \sum_{t=1}^{n} \sum_{p=1}^{m} q_{tp} x_{tp} + \sum_{t,t' \in T, t<t'} \sum_{p=1}^{m} c_{tt'} x_{tp} \bar{x}_{t'p}$$

$$\text{Subject to: } \begin{cases} \sum\limits_{p=1}^{m} x_{tp} = 1 \quad (t = 1, \ldots, n) \\ x_{tp} \in 0, 1 \quad (t = 1, \ldots, n; p = 1, \ldots, m) \end{cases}$$

where $c_{tt'}$ is the communication cost between tasks t and t'; q_{tp} is the execution cost when task t is assigned to processor p and x_{tp} is a Boolean variable equal to 1 if task t is assigned to processor p and 0 otherwise [10]. The branch-and-bound algorithm is proved helpful when problem is small in size. As shown in [10], it produces good results for small numbers of heterogeneous processors. The largest size in that paper is mapping 53 tasks to 20 processors with 229 communication requests and the instances in that paper are all theoretical rather than experimental.

Taibi's 1993 model

In Taibi's model [28], the integrating computation and communication costs are represented by a weighted sum:

$$\min_{\Pi \in \Gamma} \left(\frac{1}{N} \sum_{k=1}^{N} L_k^2 - L^2 \right) + w \cdot \sum_{i,j \in V_p} c_{ij} d_{\Pi(i),\Pi(j)}$$

where

$$L = \frac{\sum_{i=1}^{M} e_i}{N} \qquad \text{and} \qquad L_k = \sum_{i=1,\Pi(i)=k}^{M} e_i$$

N is the number of processors; M is the number of tasks; e_i represents the computation cost of task t_i; $d_{\Pi(i),\Pi(j)}$ represents the hop distance between the processor $\Pi(i)$ and $\Pi(j)$; w is the weight of the contribution of the communication cost relative to the computational load balance across the system. Different computer architectures have different computation and communication costs, resulting in distinct optimal balancing between computation and communication [56]. A suitable value for w can be estimated by characteristics of the parallel architectures or empirical experiments.

Three general purpose heuristic algorithms, hill-climbing, simulated annealing and genetic algorithms, are adopted and evaluated by three benchmarks. A massively parallel genetic algorithm (PGA) is also used to solve such problems and is found to achieve near-linear speed-up [28]. A comparative study of the algorithms is carried out. Taibi claims the hill-climbing give the worst quality solutions but it is fastest. The PGA can give comparable quality solutions as simulated annealing with comparable search time as hill-climbing.

Heiss' 1996 model

Since both the communication pattern of the problem and the interconnection structure of the parallel computers can be represented by undirected graphs, the mapping problem can be regarded as a graph mapping or graph embedding problem [47]. Heiss and Dormanns formulate the mapping problem as to find a mapping, $T \rightarrow P$:

$$\min \quad CC = \sum_{(i,j)\in E^T} \alpha(i,j) \cdot d(\pi(i), \pi(j))$$

where $\alpha(i,j)$ is the amount of data needed to be exchanged and $d(\pi(i), \pi(j))$ represents the length of shortest path between i and j. When a graph embedding problem between a task interaction graph (TIG) and processor connection graph (PCG) is considered, it is likely that adjacent TIG nodes will be assigned to adjacent PCG nodes resulting in minimum communication costs [47]. Heiss and Dormanns introduce the Kohonen's algorithm [46] to realize this kind of topology-conserving mapping.

Braun's 1999 model

Heterogeneous computing (HC) is an efficient technique to solve computationally intensive problems that has several types of embedded parallelism [9]. The problem of mapping large and diverse groups of tasks onto the machines of an HC had been researched by Siegel et al. The mapping problem is formulated as follows [79]:

$$\min \ \max\ ct(i,j) = mat(j) + ETC(i,j)$$

where max $ct(i,j)$ is the completion time and known as makespan [63]; $mat(j)$ is the earliest time a machine j can complete the execution of all the tasks that had been assigned to it and $ETC(i,j)$ represents the estimated execution time for task i on machine j. The elements of ETC are varied in an attempt to represent task heterogeneity and machine heterogeneity. In total 11 heuristics, opportunistic load balancing (OLB), user-directed assignment (UDA), fast greedy, min-min, max-min, greedy, genetic algorithm (GA), simulated annealing (SA), genetic simulated annealing (GSA), Tabu search, and A* are examined and compared regarding performance in both solution time and solution quality.

The min-min algorithm is a simple heuristic and can be adopted if the execution time has high priority. It begins with the set of all unmapped tasks. Then the set of minimum completion times is found and the task with the overall minimum completion time is selected and assigned to the corresponding machine. Intuitively, it attempts to map as many tasks as possible to their first choice of machine. If more time is available for finding better mapping, a more complex heuristic such as GA and SA should be considered [79]. In [79], Braun et al. show that if GA can introduce a good starting mapping from other approach, for example, a min-min solution, as a seed, it can get better mapping than SA for allocating heterogeneous tasks to heterogeneous computing environment. Experimental results indicate, as a combination of GA and SA, in all cases of allocating heterogeneous tasks to heterogeneous computing environment, GSA always obtains a result between GA and SA. For the systems in [79], Tabu requirs the same time as GA but get worse result than GA. Depending on the situations, Braun et al. suggest to adopt the min-min if the execution time is very limited and use GA and A* to find better mapping if more execution time is available.

QAP model

Under the conditions minimizing only the inter-task communication, n being equal to m and assuming that the communication cost is highly dependent on the location of sender and receiver processors, the mapping problem reduces to a quadratic assignment problem (QAP). The QAP is stated as follows [69]:

$$\min_{p \in II_N} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)} + \sum_{i=1}^{n} c_{ip(i)}$$

where II_N is the set of permutations of N; f_{ij} represents the flow of material from facility i to facility j and $d_{p(i)p(j)}$ is the distance from site $p(i)$ to site $p(j)$. Therefore, the cost of simultaneously assigning facility i and j to site $p(i)$ and $p(j)$, respectively is $f_{ij}d_{p(i)p(j)}$. Pardalos et al. claim that there are three exact algorithms, dynamic programming, cutting plane and branch-and-bound, available when the problem size is small. They also introduced five types of heuristic algorithms, construction method, limited enumeration method, improvement method (Tabu search), simulated annealing and genetic algorithm and show that greedy randomized adaptive search procedures (GRASP) works well in computing optimal solutions when it is incorporated within a branch-and-bound algorithm [69]. Most exact solution methods for the QAP adopt a branch-and-bound algorithm. It is well known that a crucial factor in the performance of branch-and-bound algorithm for the QAP is the choice of lower-bounding method. Exact solution methods are often implemented on high-performance computers because of the extreme difficulty of the QAP. In [55], Linderoth et al. extended the power of branch-and-bound algorithm for solving QAP in two ways. The algorithm introduced a new lower-bounding technique quadratic programming bound (QPB) and is implemented on a large geographically distributed resources known as a computational grid. They believe that the potential of algorithm advances for the QAP can be realized by utilizing the power from computational grids.

IBM's 2005 model

Recently, a model is developed by Bhanot et al. to minimize only inter-task communication [34]. They neglect the actual computing cost when placing tasks on processors linked by mesh or torus by the following model:

$$\min \quad F = \sum_{i,j} C(i,j)H(i,j)$$

where $C(i, j)$ is communication data from domain i to j and $H(i, j)$ represents the smallest number of hops on Blue Gene/L (BG/L) torus between processors allocated domains i and j. A simulated annealing technique is used to map n tasks to n processors on the supercomputer for minimizing the communication time. Two tests for applications SAGE and UMT2000 on Blue Gene/L are reported in [34]. The method is applied up to 2048 processors and the communication efficiency is improved by 45% for a 512-node case [34]. The highlight of this model is that of mapping real large applications to a real massive supercomputer beyond theoretical exercise.

Yu's graph embedding model

As Heiss and Dormanns did before [47], Yu et al. treated the static mapping problem as a graph embedding problem since an embedding of a guest graph $G = (V_G, E_G)$ into a host graph $H = (V_H, E_H)$ is one-to-one mapping from V_G to V_H [48]. The quality of the embedding is measured by *dilation* and *expansion* in their research. The graph embedding problem is NP-hard requiring utilizing heuristics for its solution. They use a series of embedding and folding technologies such as pipe and ring folding, and paper folding to construct their topological mapping library. By evaluating their library against the NPB benchmarks, the mappings generated by their library can reduce the communication costs significantly and drastically improve the scalability [48].

Fixed mapping strategy for Blue Gene Supercomputer

Smith and Bode compare a series of predefined mappings with the default

MPI rank mapping [13]. The mapping set consists of Gray-code mappings with different aspect ratios and permutations of X, Y and Z coordinates. Results are presented for three NPB benchmarks: BT, CG and MG and such results showed that in special cases, the YZX and ZYX mappings may improve the performance by about 15% over the default XYZ mapping.

Stone's and Bokhari's models are fundamental and good starting points in the field. Billionnet's, Talbi's, and Braun's models attempt to include both computation and communication costs but they assume that communication costs are independent of locations of processors. This assumption is appreciate for Beowulf Clusters but is inadequate for parallel computers with more complex network topologies [34]. Heiss' model, QAP model, IBM's model and Yu's model can handle these complex cases but they lack an efficient way to integrate the computation and communication costs into one model. IBM's model, Yu's model and fixed mapping strategy are tested by executing popular benchmarks on Blue Gene/L.

2.2.2 Mapping Algorithms

Our goal is to solve the mapping problem for ultra-scalable supercomputer systems with thousands of processors. Optimization techniques often fail for problems of this size due to the exponentially growing computational effort. Several heuristic techniques have been developed for searching in large solution spaces, such as simulated annealing (SA), genetic algorithm (GA), evolution strategies (ES), genetic simulated annealing (GSA) and Tabu search

(TS). The most important techniques for the mapping problem can be illustrated in Fig. 2.2.

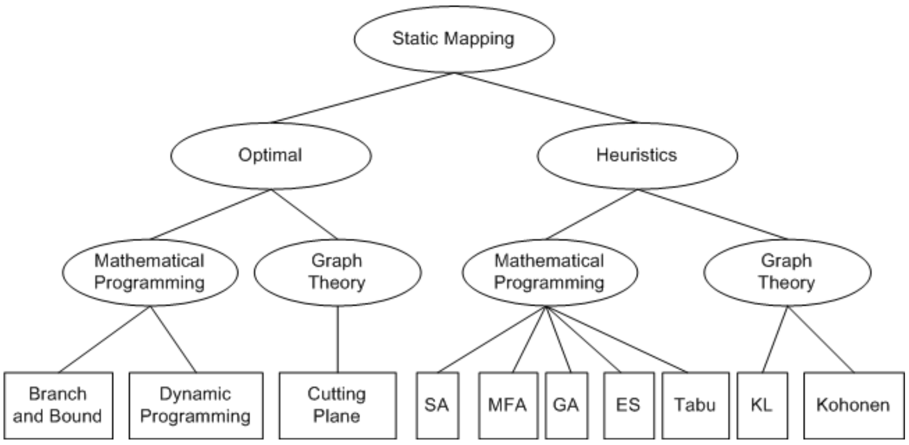


Figure 2.2: Techniques for the mapping problem [28, 59]

Simulated Annealing (SA)

SA is an optimization method that repeatedly attempts to improve a given configuration by making random changes. It works for large size problems due to its deep and useful connection to statistical mechanics and such connection suggests the validity of the method in handling thermodynamical problems for finding the optimal values of an objective function with many independent variables [73, 76]. The SA technique has been widely applied to many combinatorial optimization problems although it has serious drawbacks: slow and *inherently sequential* [23], making efficient parallelization a major challenge [23, 25].

A successful SA implementation must address three key issues: a cooling schedule, an objective function and a movement in a neighborhood structure [61]. The basic SA algorithm is illustrated in Fig. 2.3. In each iteration, the

```
INITIALIZE($i_0$, $T_0$, $L_0$);
$k := 0$;
$i := i_0$;
**While** (stopping criteria not met) **do**
**Begin**
        For $i := 1$ to $L_k$ do
        Begin
            GENERATE( $j$ from $s_i$);
            if $f(j) \leq f(i)$ then $i = j$;
            else
            if $\exp((f(i) - f(j))/T_k) > random[0, 1)$
            then $i = j$;
        End;
        $k := k + 1$;
        CALCULATE($L_k$);
        CALCULATE($T_k$);
**End**;
```

Figure 2.3: SA Procedure [73]

temperature for k^{th} step T_k, is evaluated according to a cooling schedule. There are at least two types of cooling schedule: predetermined and adaptive. The former is fixed before the calculation and is not influenced by the progress. For instance, a linear schedule $T_k = T_0 - \alpha k$ and an exponential schedule $T_k = T_0 \beta^k$ are such predetermined schedules and have been used widely [86]. In the formulas, k is the step count; T_0 is the initial temperature; α and β are constant factors. Another fixed schedule example is the logarithmic cooling scheme,

$$T_k = \frac{c}{\log(k + d)}$$

where d is usually set to one and c is no more than the largest energy barrier in the problem. This schedule is theoretically important but impractical due to the extremely slow temperature decrease [75]. In contrast, an analogous adaptive optimal schedule based on constant thermodynamic speed v_s is adopted in the finite-time thermodynamic optimization [12].

Defining a proper objective function is central to SA implementation. For most combinatorial optimization problems, the objective function is often calculated in conjunction with the selection of candidate solutions. Once an objective function is clearly defined, a reasonable neighborhood structure often results from the problems. These structures must allow SA to explore all solutions and the performance of SA depends on the quality of the neighborhood structure. Many researchers attempt to improve its performance by modifying the neighborhood size [61].

The theory behind evolutionary algorithms is the Darwinian theory of evolution in which the survival of the fittest is generally accepted. Evolutionary algorithm is a general term that consists of genetic algorithms and evolution strategies [59].

Genetic Algorithm (GA)

The genetic algorithm is a stochastic search technique and proposed by Holland in 1975 [52]. The basic GA algorithm is shown in Fig. 2.4.

The proportionate selection scheme is often used to create population, where the ratio of the fitness value of a string of a current population over the average fitness value represents the expected number of its offspring. Crossover comes after selection. Pairs of strings from the population are randomly picked up to be subjected to crossover. The portions of the two strings beyond a crossover point are exchanged to form two new strings whenever this point is

```
initial population generation
evaluation
**While** (stopping criteria not met) **do**
**Begin**
    selection
    crossover
    mutation
    evaluation
**End**;
```

Figure 2.4: GA Procedure [64]

chosen. The algorithm invokes crossover only if a randomly generated number in the range zero to one is greater than a parameter, the crossover rate p_c. Strings are subjected to mutation after crossover. Mutation of a bit means flipping it, changing a zero to one or vice versa. Another parameter, the mutation rate p_m, gives the probability that a bit will be flipped. Mutation is specially useful to restore lost genetic material when all strings in a population converged to zero at a given position but the optimal solution has a one at that position or vice versa [64].

Unlike SA, GA is *inherently parallel.* Cantú-Paz classifies parallel genetic algorithms into four categories: global master-slave parallelization, fine-grained algorithms, multiple-deme and hierarchical parallel [27].

Evolution Strategies (ES)

It is proved that evolution strategies can be used to solve many types of optimization problems [11, 38, 87]. The basic ES algorithm is shown in Fig. 2.5.

The initial population is randomly generated and uniformly distributed throughout the search space. Each generation is evaluated by computing the

```
$k := 0$
randomly generate population $P(k)$
evaluate $P(k)$
**While** (stopping criteria not met) **do**
**Begin**
    $k := k + 1$
    $T := P(k - 1)$
    use $\mu$ parents to create $\lambda$ children
    $T := T \cup \lambda$ children
    $P(k) := \mu$ fittest from $T$
    evaluate $P(k)$
**End**;
```

Figure 2.5: ES Procedure [38]

fitness of its structure. New generation are produced by copying the old generation of size μ to an intermediated population then using them as parents to produce λ children by crossover or mutation with probability p_c and p_m, respectively. The μ fittest structures become the new generation [38].

In spite of its origin from the GA, ES has important differences. Selection process for ES is deterministic while that of GA is stochastic. Good solutions will always survive to the next generation in ES while reproduction may eliminate good solutions in GA [38].

Genetic Simulated Annealing (GSA)

The genetic simulated annealing algorithm is a combination of GA and SA techniques [40, 72, 79]. In general, GSA follows procedures similar to GA. In the selection process, GSA accepts a new chromosome if the new fitness value is less than the sum of the old fitness value and the current temperature. As the system temperature decreases, it becomes more and more difficult to accept a poorer solution.

Tabu Search (TS)

The basic idea behind the Tabu search is to keep track of the regions of the solution space which have already been searched to avoid repeating a search near these areas [30, 49, 79]. It is implemented by making a new random mapping different from each mapping in the Tabu list by at least half.

Besides the above strategies, there are also other approaches for solving the mapping problems. In [80], a new approach is proposed to solve the problem of workload partitioning and assignment for very large distributed real-time systems, in which software components are hierarchically organized and hardware components potentially spanned several shared and dedicated links. An optimal solution is presented in [26] that allocates communicating periodic tasks to heterogeneous nodes in a distributed real-time system. A special case is studied in [60]. A new assignment policy is introduced to improve performance by unbalancing loads. Some work has been limited to handle special communication patterns. For example, an algorithm is developed to solve mapping problems having a rectilinear topology [24]. Graph partitioning techniques has been used in the load balancing and clustering tasks to processors [53]. It has also been used for task clustering and mapping on eight node hypercube [29]. Mean Field Annealing (MFA) is originally proposed for solving travelling salesperson problem and believed a similar strategy as SA. It is compared to SA in [16]. When it is applied to a randomly generated TIG model with 400 tasks, 4298 communication requests and 32 processors in hypercube and mesh topologies, MFA can obtain the results a little worse than the ones from SA but with less execution time.

In the above cases, the problem of finding a static mapping from uniform tasks of realistic parallel applications to thousands of processors of an ultra-scalable system where communication costs are location dependent to minimize the total execution time has not been carefully addressed so far. Most of these cases are limited to the improvement of the communication time. It is an interesting but unclear issue that to what extent such improvement in communication can bring to reduce the overall execution time. These issues are addressed in this thesis.

Chapter 3

Parallel Computing Systems

To achieve an optimal mapping from n domains to m processors, we need to consider each system component: processors, memory systems, networks and the relationships between these components.

First let us review the classical taxonomy [31], which characterizes computer architectures by the number of distinct instruction stream issued at a time and the number of data streams they operate on [21]:

- SISD (single instruction - single data) - sequential computers.
- MISD (multiple instruction - single data) - no real implementation [83].
- MIMD (multiple instruction - multiple data) - composing of multiple conventional processors.
- SIMD (single instruction - multiple data) - performing operations in parallel

Currently, most high-performance computers adopt MIMD model and are built from off-the-shelf components except, mostly, with special networks. It has a better compatibility with the common workstations with a lower price.

3.1 Memory systems for parallel computers

The memory aspect leads to two popular classes of parallel computers: shared-memory and distributed-memory.

3.1.1 Shared-memory systems

In shared-memory systems, the memory modules are connected to the processors by a bus, a multistage interconnection network, a crossbar or similar memory sharing systems that links any processor to any memory module; each processor can access equally fast any memory location. The programmer can directly use conventional memory access instructions to access data without the otherwise explicit calls for communication with other processors.

Shared memory multiprocessors provide convenient support of parallel programming and throughput on workloads. They are found across a wide range of scale, from a few to hundreds of processors. Such architecture where two or more identical processors are connected to a single shared memory is usually called a *symmetric multiprocessor* (SMP) [21]. Many common multiprocessor systems adopt an SMP architecture. Since SMP systems allow any processor to work on any task no matter where the data for the task are located in memory, they can easily distribute tasks between processors to balance the workload efficiently. Obviously, the different assignments will lead to the same communication costs in this type of system.

3.1.2 Distributed-memory systems

In these systems, each processor has its own local memory while the other processors can access the data in the local memory through the communication network. In practice, each node has a memory management unit directly connected to the communication network. It is obvious that the programming for these computers is more difficult than for a single-processor computer or for the shared memory computers because message passing through the communication network has to be explicitly programmed. The explicit message passing libraries such as Message Passing Interface (MPI) and Parallel Virtual Machine (PVM) are often used to alleviate such difficulties.

3.2 Beowulf computers

Named after the pioneer Beowulf project at NASA [81], a Beowulf cluster consists of several personal computers connected through a switch network, resulting in commercial, off-the-shelf components a cost-effective solution for supercomputing. Typical Beowulf systems communicate with a private intranet. Unfortunately, the inter-processor communication on a Beowulf system is inadequate for many applications. RPC (Remote Procedure Call), sockets and TCP/IP are used for communication and typically have slow throughput and high message latency. The communication among computers is handled by message passing libraries, such as PVM and MPI. Both Seawulf Cluster [6] and Galaxy system [85] are examples of Beowulf Clusters on the Stony Brook campus.

3.3 Distributed SMP

The IBM pSeries 655 is a 4-processor or 8-processor symmetric multiprocessing server. The basic computing component of the pSeries system is the Power4 chip. A single chip contains two independent processors, each having its own 32 KB L1 data cache. A 1.5 MB L2 cache is shared by the two processors. A 32MB L3 cache is located between each processor chip and main memory and operates at one-third of the chip frequency.

Multiple pSeries 655 servers can be configured in distributed clusters, to providing higher performance, ideal for computationally-intensive and data-intensive workloads in science, engineering, business intelligence and data warehousing [43]. Up to four Power4 chips can be joined to form a 8-way multi-chip module (MCM) of as many as eight processors (Fig. 3.1). An 8-way MCM contains a total of 6 MB of L2 cache and 128MB of L3 cache [43].

A distributed SMP system available for Stony Brook University graduate students is the IBM's SUR machine, IBM pSeries 655 cluster, awarded to Stony Brook University in 2005 through IBM's SUR (Shared University Research) Program. It consists of four modules and is linked to form a 32-way system. The intra-node connection is bus-based while the inter-node connection is switch-based, provided by the High Performance Switch (HPS) network, allowing such high-performance message passing fabric. Each point-to-point connection between nodes is comprised of two channels (full duplex) that can

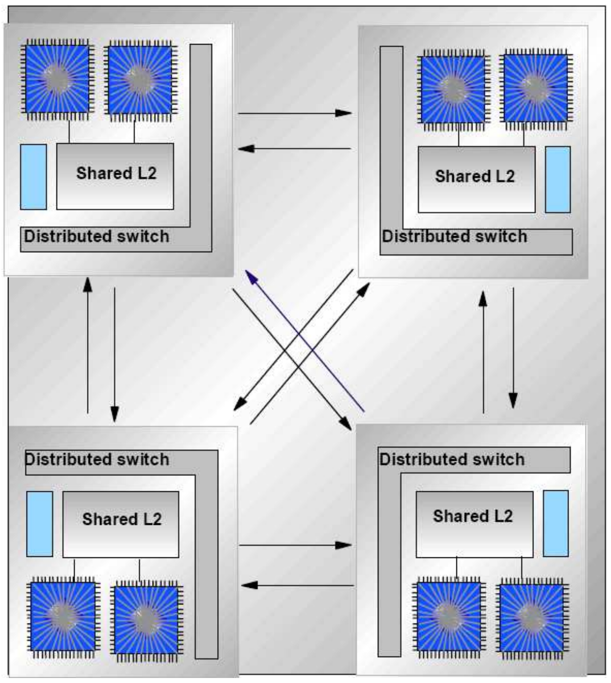


Figure 3.1: 8-way MCM [43]

carry data in opposite directions simultaneously [1]. The communication distance between any pair of processors on the same node is of equal, low-latency, high-bandwidth. Higher latency is found when processors on different nodes communicate because of the switch involved.

3.4 Ultra-scalable mesh or torus network systems

Both QCDOC and Blue Gene/L are examples of ultra-scalable supercomputer systems in which the communication costs are dependent on subtask placements.

3.4.1 QCDOC

QCDOC is a massively parallel and highly scalable computer using the system-on-a-chip technology (Fig. 3.2). The QCDOC processing component consists of a single ASIC (application specific integrated circuit) and a standard DDR (double data rate) RAM module. Larger machines can be built by connecting many of the smaller processing components [67, 68].

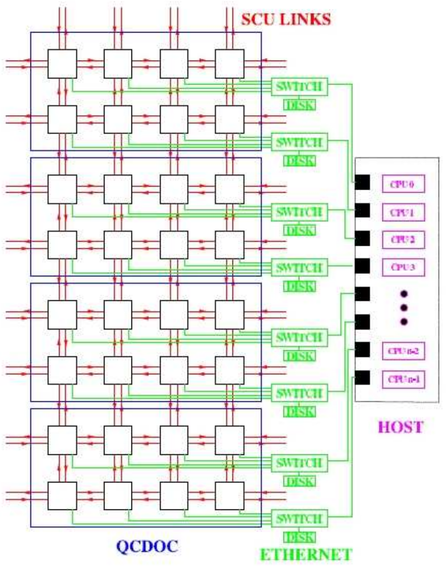


Figure 3.2: 2D version of the network connections in QCDOC [68]

The QCDOC ASIC contains a PowerPC 440 running at 500MHz , a 1Gflop 64-bit floating point unit (FPU), a 4MB of embedded DRAM and aggregate bandwidth of 12 Gbit/s in 12 independent directions [67, 68].

The network topology is set up as a 6D torus; three dimensions are closed on a motherboard while the other three are open to off board communication. Communication between nodes is managed by the SCU (serial communication unit) in the ASIC. QCDOC has implemented its own operating system called QOS to handle nearest neighbor communication calls, and global operations.

3.4.2 Blue Gene/L

The Blue Gene/L Supercomputer (BG/L) is a massively parallel computer with five communication networks. Among them a nearest neighbor network, with the topology of a 3D torus, and a global tree are the two networks for run-time data sharing. In normal usage, the torus is the primary network and is used both for point-to-point and for many global or collective communications. The tree is used for collective communications such as `MPI_Reduce` [54, 82].

Computation nodes on BG/L are logically arranged into a 3D lattice and the torus communications network provides physical links only between nearest neighbors in that lattice. Each node has six torus links connecting to its six nearest neighbors in the $\pm x$, $\pm y$, $\pm z$ directions with only one-hop distiance between any nearest neighbors (Fig. 3.3). All communications between nodes must therefore be routed to the available physical connections and the cost of communications between nodes varies depending on locations of the nodes.

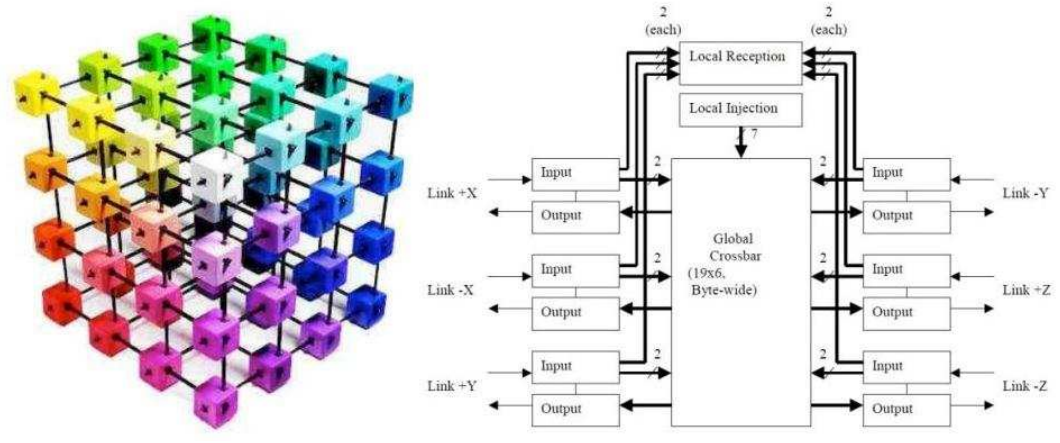


Figure 3.3: Blue Gene/L network (Left: Topology of a 3D torus; Right: Basic Architecture of the torus router [82])

There are two compute modes, the co-processor mode (CO) and the virtual node mode (VN) in the BG/L system. In the CO mode, the secondary CPU in the node works as an offload coprocessor for processing the I/O of the main CPU, while in the VN mode, both CPUs are used and the main memory is split between them [65]. All experiments conducted for this thesis are in the CO mode.

All data in this thesis are collected from the Blue Gene/L systems at Argonne and Brookhaven National Laboratories. A series of pre-defined partitions, 32-node ($4 \times 4 \times 2$), 64-node ($8 \times 4 \times 2$), 128-node ($8 \times 4 \times 4$), 256-node ($8 \times 4 \times 8$), 512-node ($8 \times 8 \times 8$), and 1024-node ($8 \times 8 \times 16$) are utilized to test the models at different sizes and structures of application. The partitions with fewer than 512 nodes are connected as 3D-meshes while 512-node and 1024-node are 3D-torus partitions.

We select the BG/L system as the testing platform because it meets all our major requirements: It is used worldwide and large partitions are readily available. It supports MPI [32, 33] and offers an MPI re-map interface [35].

Chapter 4

Mapping Models

In this section, we discuss the objective function with assumptions on the applications and computer platforms. We also verify models by a theoretical example of the hyperbolic equation.

4.1 Assumptions and notations

In our models, we consider more realistic and general cases compared to many appearing in literature. We assume the application has already been appropriately decomposed as n subtasks and the computing load of each subtask equal. The inter-subtask communication requirements are described by the demand matrix $D_{n\times n}$ whose entry $D(t,t')$ is the required data transfer from subtask t to subtask t'.

We suppose the parallel computer is heterogeneous, i.e., the computing speeds of processors different and the communication time cost between two processors depends on the exact pair. These heterogeneous properties are described by two matrices $L_{n\times m}$ and $S_{m\times m}$, where n is the number of subtasks and m is the number of processors. The computation cost is expressed as a load matrix $L_{n\times m}$ whose entry $L(t,p)$ is the computation cost of the subtask t when t is executed on the processor p. The communication cost is expressed as a supply matrix $S_{m\times m}$ whose entry $S(p,p')$ is the cost for communication between processors p and p'. These costs depend on processor locations, available buffer, network congestion, and other conditions. These matrices are assumed achievable from the characteristics or by standard testing, or by simple analysis.

4.2 Basic model

With assumptions and definitions stated earlier, we can formulate the mapping problem in the following way.

Let $\{t_1, t_2, \ldots, t_n\}$ be the set of subtasks of the problem and $\{p_1, p_2, \ldots, p_m\}$ be the set of heterogeneous processors of the parallel computers to which the subtasks are assigned. In general, $n \geq m$.

Let X_{tp} be the decision Boolean variable that is defined as:

$$x_{tp} = \begin{cases} 1, & \text{if subtask } t \text{ is assigned to processor } p \\ 0, & \text{otherwise} \end{cases}$$

Let $Y_{tt'pp'}$ be the decision Boolean variable that is defined as:

$$y_{tt'pp'} = \begin{cases} 1, & \text{if subtask } t \text{ and } t' \text{ are assigned to processor } p \text{ and } p' \text{ respectively} \\ 0, & \text{otherwise} \end{cases}$$

We adopt the total execution time, the sum of the computation and communication time as the criterion and formulate a 0-1 programming problem as:

$$min \ \left\{ \sum_{t=1}^{n} \sum_{p=1}^{m} L(t,p) \cdot x_{tp} + \sum_{t=1}^{n} \sum_{t'=1}^{n} D(t,t') \cdot \left(\sum_{p=1}^{m} \sum_{p'=1}^{m} S(p,p') \cdot y_{tt'pp'} \right) \right\} \tag{4.1}$$

$$\text{Subject to: } \begin{cases} \sum\limits_{p=1}^{m} x_{tp} = 1 \ \ (t = 1, \ldots, n) \\ \sum\limits_{t=1}^{n} x_{tp} \geq 1 \ \ (p = 1, \ldots, m) \\ \sum\limits_{t=1}^{n} x_{tp} \leq \left\lceil \frac{A_p}{A_T} \times n \right\rceil \ \ (p = 1, ..., m) \\ x_{tp} + x_{t'p'} \leq 1 + y_{tt'pp'} \ \ (t, t' = 1, \ldots, n, t < t'; p, p' = 1, \ldots, m) \end{cases}$$

The first term of the objective function represents the global computation cost and the second represents the inter-subtask communication cost. The assignment constraints require each subtask to be assigned to one and only one processor. To fully utilize the computing resource, each processor is assigned at least one subtask and no more than the ratio its ability to the total computing abilities. The term A_p denotes the computing ability of the p^{th} processor and A_T represents the total computing abilities of the given processor set. We set both low and upper bounds of computing load for each processor for achieving system load balance. We also introduce the Boolean variables x_{tp} and $y_{tt'pp'}$ and relative constraints to formulate this problem to an Integer Linear Programming (ILP) problem.

The rationale of this model is that the minimal execution time depends on uniform load of the processors and the inter-processor communication minimization. Although the criterion is not the more desirable total execution time, it is still a useful metric because we believe the mapping that minimizes the value of the criterion minimizes the value of the actual time. This observation has been demonstrated in a similar case by the results [34] from experiments on BG/L system.

This model can be easily implemented with the convenience of obtaining the demand matrix by a known pattern or profiling, but it lacks the sophistication in considering communication overlap, network congestion, and collision, resulting in misleading mappings for subtle cases.

4.3 Enhanced model

An enhanced model is proposed as:

$$min \quad \left\{ \sum_{t=1}^{n} \sum_{p=1}^{m} L(t,p) \cdot x_{tp} + \sum_{i=1}^{k} (D_i S)_{max} \right\} \tag{4.2}$$

$$\text{Subject to:} \begin{cases} \sum\limits_{p=1}^{m} x_{tp} = 1 \ \ (t = 1, \ldots, n) \\ \sum\limits_{t=1}^{n} x_{tp} \geq 1 \ \ (p = 1, \ldots, m) \\ \sum\limits_{t=1}^{n} x_{tp} \leq \left\lceil \frac{A_p}{A_T} \times n \right\rceil \ \ (p = 1, ..., m) \end{cases}$$

where $k \leq n^2$ is the total number of the communication batch. The term $(D_i S)_{max}$ represents the maximum value of the i^{th} batch communication. With more detailed communication information, the objective function only focuses on minimizing the sum of the dominant terms in each batch. With this added consideration of communication overlapping, we can model a more realistic communication scenario for more accurate mapping, although the overlapping information is difficult to obtain for many cases.

The basic and enhanced models are verified by a series of applications and benchmarks in Chapter 5. When the information of the communication overlapping is known, such as 2D wave equation, both models are utilized and verified. For general applications and benchmarks, such as SMG2000, NPB MG and NPB CG, only the basic model is used to generate mappings.

Given a real application on an ultra-scalable system whose processor count may exceed 1000, ILP solvers cannot solve the problem due to the memory limitation. Heuristic techniques are often becoming the practical tool for searching for near-optimal solutions [59]. Simulated Annealing (SA) is a good choice and it is indeed adopted in our study.

The corresponding key issues are described as follows. An exponential cooling schedule is adopted. Both basic model and enhanced model are utilized to evaluate the objective function values for 2D wave equation application while only basic model is used to evaluate the objective function values for SMG2000, NPB MG and NPB CG. In most cases, two types of exchanges, single-exchange and axis-exchange, are adopted to generate the neighborhood structure in each iteration. In single-exchange case, a subtask is randomly picked up, then it will be switched with another subtask which is located in proper distance along X, or Y, or Z dimension of BG/L machine where distance depends on a parameter which is based on the machine topology size. In axis-exchange case, a subtask is randomly selected, then the entire row or column which this subtask belongs to will be switched with the near row or column along X, or Y, or Z dimension. The dimension, direction and the mixture ratios of adopting above two types of new state generation are controlled by a series of parameters.

The implementation above models can be illustrated in Fig. 4.1.

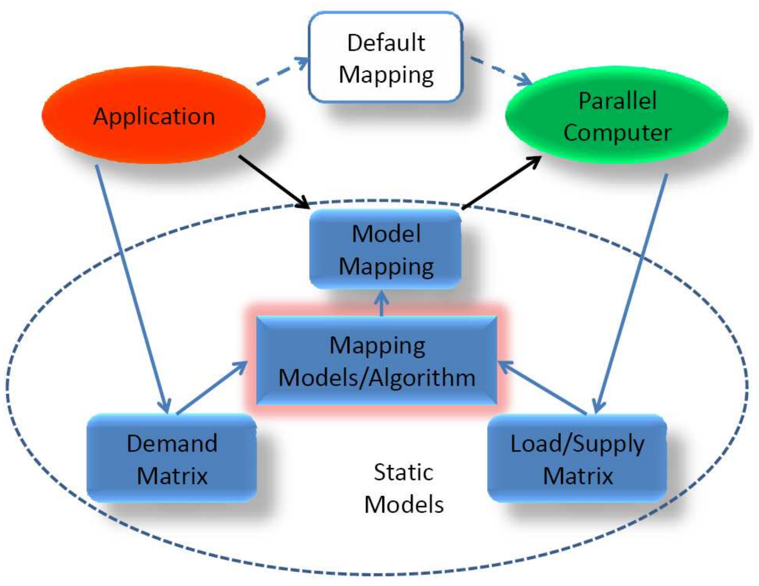


Figure 4.1: Static mapping model tool implementation

4.4 Latency and supply matrix

Our models introduce the latency matrix to quantify the communication cost instead of the hop matrix used in earlier models [34, 78] to reduce the inaccuracy of the hop measure.

The actual measured communication time for sending a 0 byte message from all 1024 nodes to all other nodes form a matrix as shown in Fig. 4.2. Obviously, the diagonal elements are 0 because there is no cost for self communication and the matrix is symmetrical.

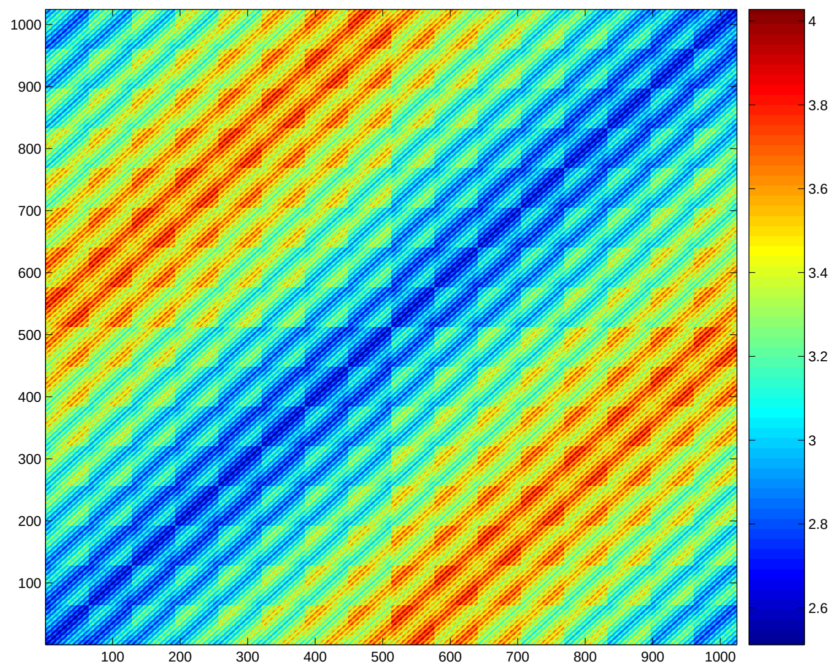


Figure 4.2: MPI latency(μs) of a 0 Byte packet on a $8 \times 8 \times 16$ BG/L system

Linear regression analysis of the latency with respect to the hop, illustrated in Fig. 4.3, shows there are many outliers (red dots) and these outliers may mislead the optimization if hops instead of latency or other more accurate measures are adopted to measure the communication cost. Most of them result from the torus in Z-dimension of the BG/L system (Fig. 4.4). For each hop number, there are many different values of latency, corresponding to different cases, such as relative locations. The hop measure fails to distinguish these multiple subtle states that the latency measure can.

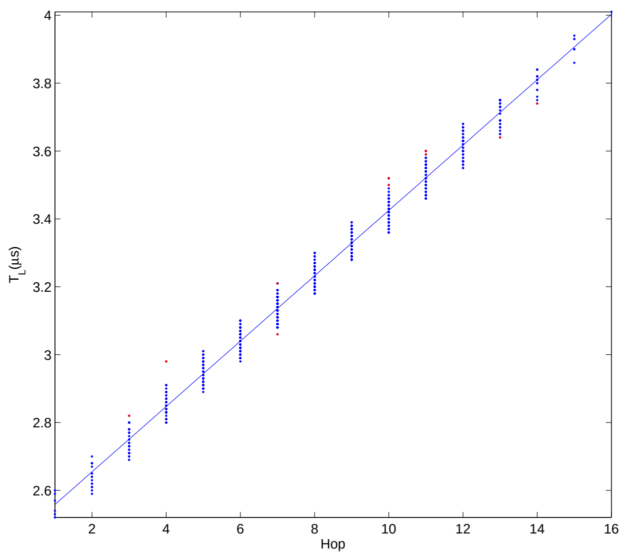


Figure 4.3: Linear Regression for $T_L = a * \text{Hop} + b$ (95% confidence level)

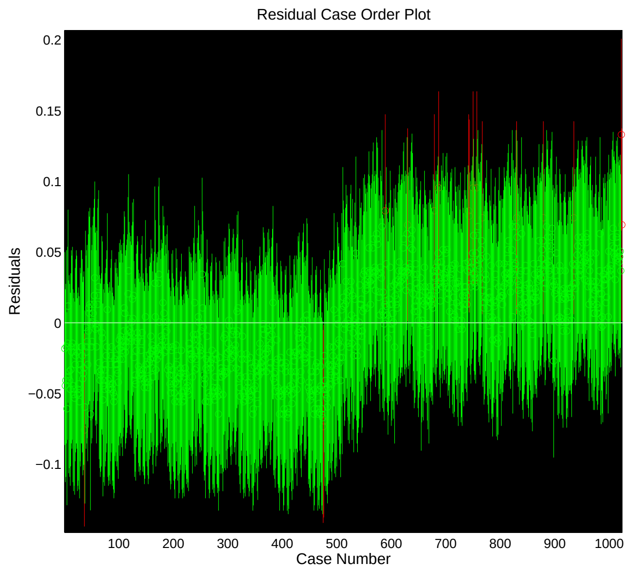


Figure 4.4: Regression residuals' error bars

In [13], Brian and Brett report that the timing results for NAS parallel benchmark gathered from the permutations of the X, Y and Z coordinates of BG/L system showed that YZX and ZYX mapping result in higher performance than the default XYZ mapping. Their observation suggests that the hop measures along different dimensions are not equivalent.

4.5 A theoretical example

One example is to find an efficient mapping of 18×12 subtasks resulting from a 2D hyperbolic equation (Fig. 4.5) onto a $3 \times 3 \times 3$ lattice (Fig. 4.6).

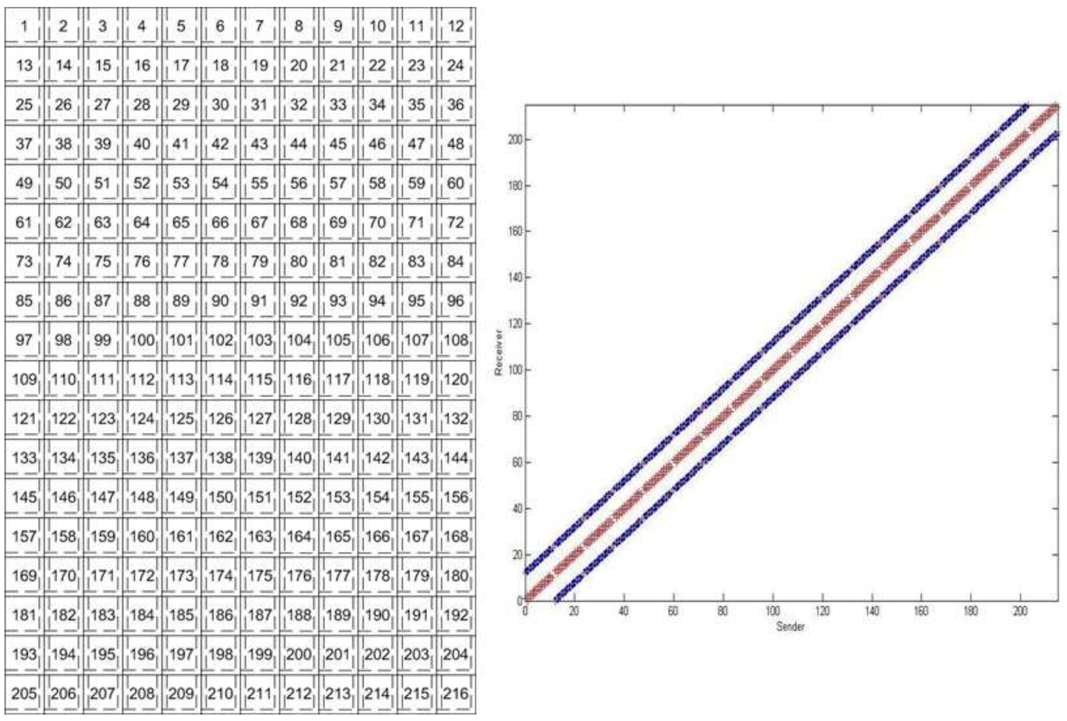


Figure 4.5: Left: Subtasks of a 2D hyperbolic equation; Right: Communication pattern for the 2D hyperbolic equation

For comparison, we consider the following four starting mappings: a ran-

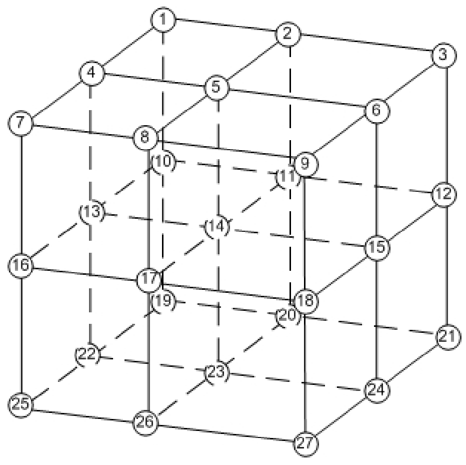
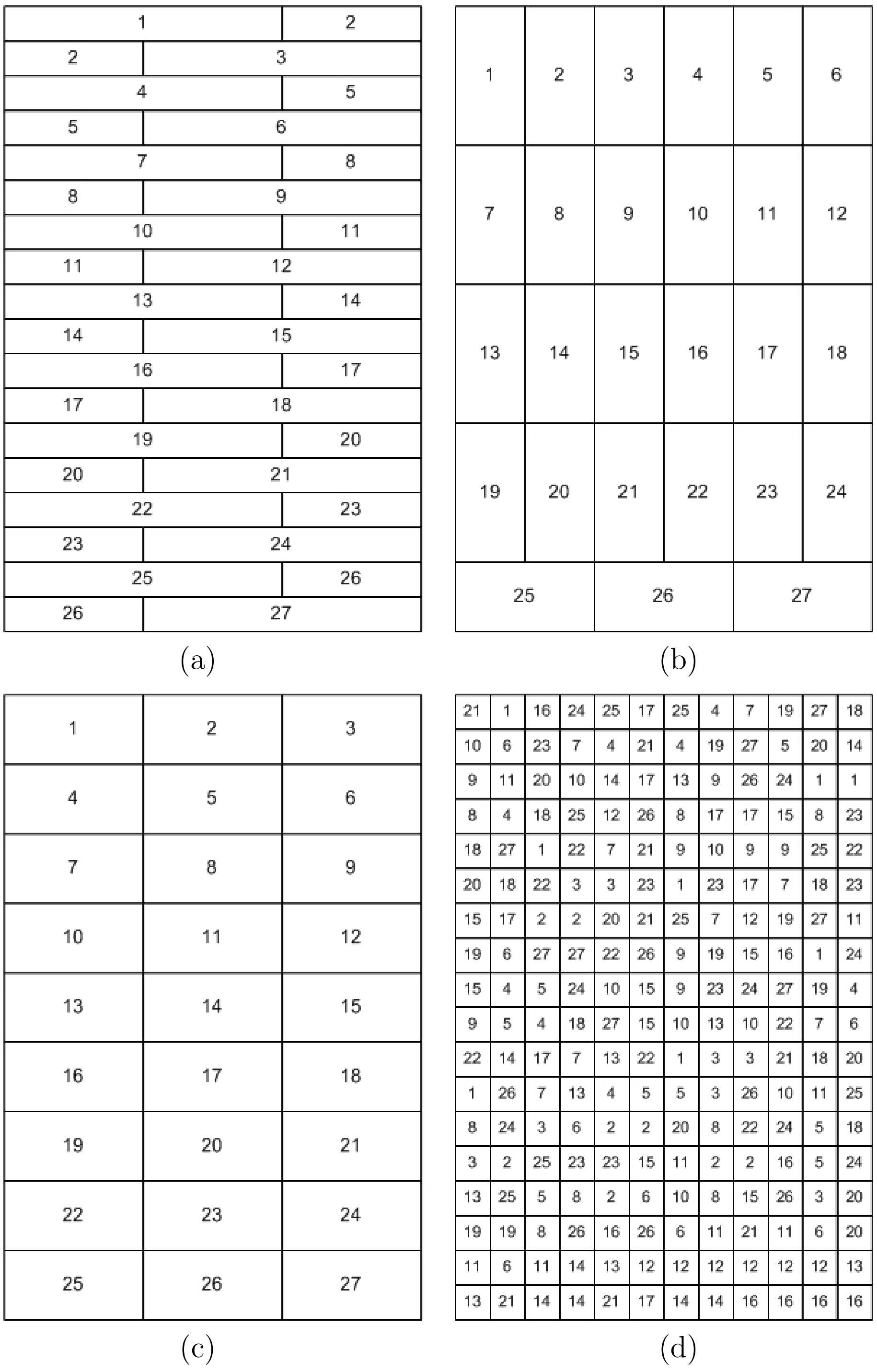


Figure 4.6: A grid from a 3D torus system

dom assignment, the Bar assignment, the Block Column assignment and the Block Row assignment. The subtask marked with a number means that subtask is assigned to that processor. The Bar assignment is often adopted as default, while the Block Column and Block Row assignment usually appear to be good assignments. Applying the SA algorithm to our basic model, the above mappings evolve in the way illustrated in Fig. 4.8 and the best mapping generated by our models is illustrated in Fig. 4.9.



21	1	16	24	25	17	25	4	7	19	27	18
10	6	23	7	4	21	4	19	27	5	20	14
9	11	20	10	14	17	13	9	26	24	1	1
8	4	18	25	12	26	8	17	17	15	8	23
18	27	1	22	7	21	9	10	9	9	25	22
20	18	22	3	3	23	1	23	17	7	18	23
15	17	2	2	20	21	25	7	12	19	27	11
19	6	27	27	22	26	9	19	15	16	1	24
15	4	5	24	10	15	9	23	24	27	19	4
9	5	4	18	27	15	10	13	10	22	7	6
22	14	17	7	13	22	1	3	3	21	18	20
1	26	7	13	4	5	5	3	26	10	11	25
8	24	3	6	2	2	20	8	22	24	5	18
3	2	25	23	23	15	11	2	2	16	5	24
13	25	5	8	2	6	10	8	15	26	3	20
19	19	8	26	16	26	6	11	21	11	6	20
11	6	11	14	13	12	12	12	12	12	12	13
13	21	14	14	21	17	14	14	16	16	16	16

Figure 4.7: (a) Bar Assignment (M_{Bar}), (b) Block Column Assignment (M_{BC}), (c) Block Row Assignment (M_{BR}), (d) Random Assignment (M_R).

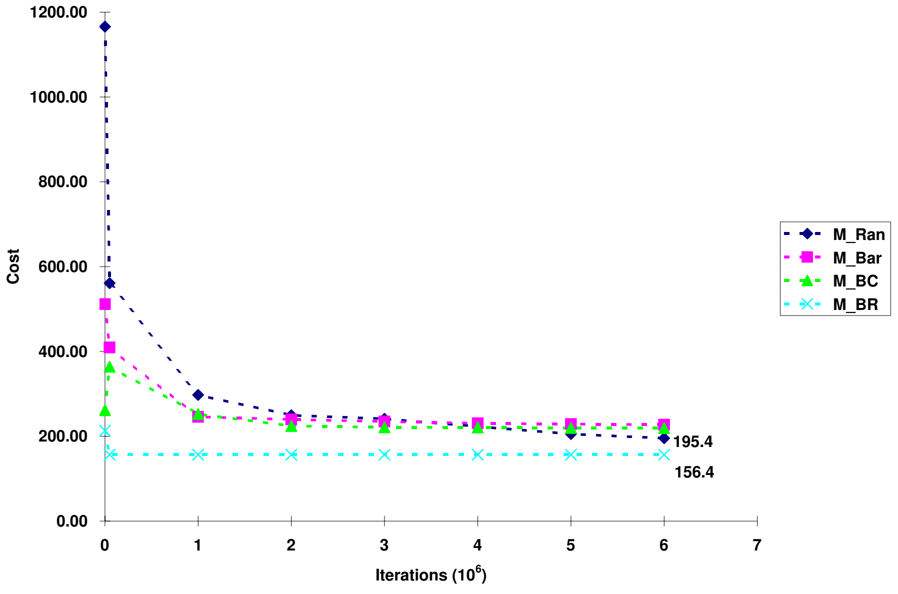


Figure 4.8: SA procedure for 2D hyperbolic equation with different starting mappings

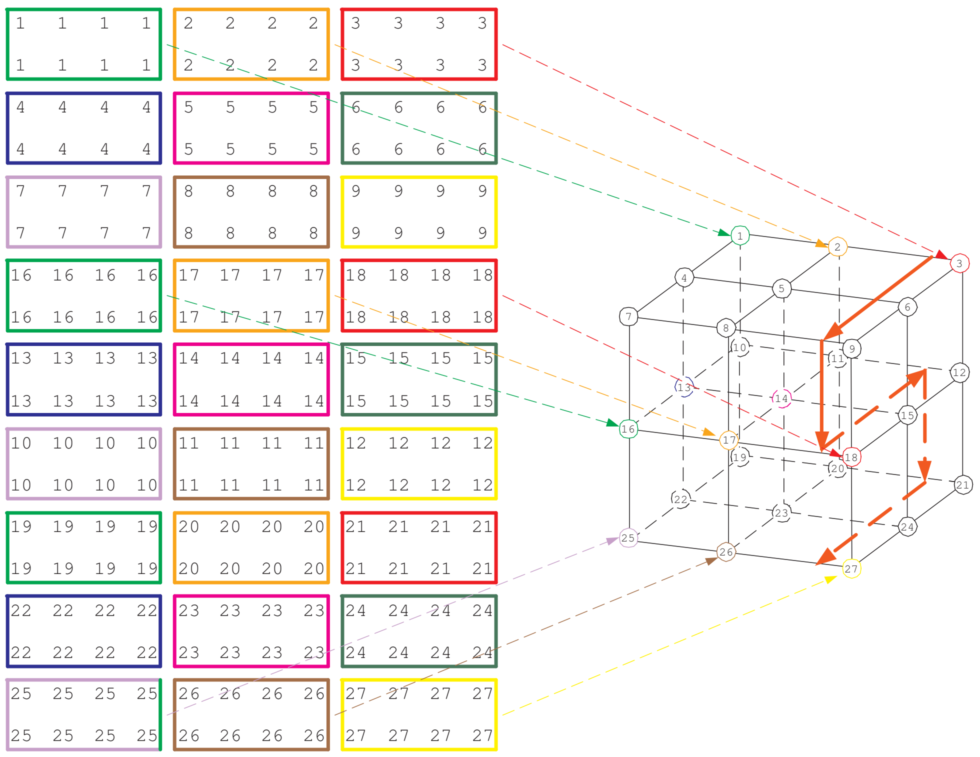


Figure 4.9: The model mapping for the 2D hyperbolic equation

Chapter 5

Applications

In this section, we continue to verify our models by executing four realistic applications on BG/L system.

5.1 2D hyperbolic partial-differential equations

We consider the numerical solution to the 2D wave equation, an example of a 2D hyperbolic partial-differential equation. The wave equation is given by the differential equation

$$\frac{\partial^2 u}{\partial t^2}(x,y,t) = \alpha^2 \left(\frac{\partial^2 u}{\partial x^2}(x,y,t) + \frac{\partial^2 u}{\partial y^2}(x,y,t) \right), \ \ 0 < x < l_x, \ \ 0 < y < l_y, \ \ t > 0, \tag{5.1}$$

with periodic boundary conditions and initial conditions

$$u(x,y,0) = \sin(2\pi x)\sin(2\pi y), \ \ 0 \le x \le l_x, \ \ 0 \le y \le l_y$$

and

$$\frac{\partial u}{\partial t}(x,y,0)=0,\ \ 0\le x\le l_x,\ \ 0\le y\le l_y,\ \ t>0,$$

where α, l_x and l_y are constants. To set up the difference equation, select a integer $m>0$ and time-step size $n>0$. With $h_x=l_x/m$ and $h_y=l_y/m$, the mesh points (x_i,y_j,t_k) are

$$x_i=ih_x\ \ \forall i=0,1,\ldots,m,$$

$$y_j=jh_y\ \ \forall j=0,1,\ldots,m,$$

and

$$t_k=kn\ \ \forall k=0,1,\ldots.$$

The difference equation is obtained by using the centered-difference for the second order derivatives given by

$$\frac{\partial^2 u}{\partial t^2}(x_i,y_j,t_k)=\frac{u(x_i,y_j,t_{k+1})-2u(x_i,y_j,t_k)+u(x_i,y_j,t_{k-1})}{n^2}-\frac{n^2}{12}\frac{\partial^4 u}{\partial t^4}(x_i,y_j,\mu_k)$$

where

$$\mu_k\in(t_{k-1},t_{k+1}),$$

$$\frac{\partial^2 u}{\partial x^2}(x_i,y_j,t_k)=\frac{u(x_{i+1},y_j,t_k)-2u(x_i,y_j,t_k)+u(x_{i-1},y_j,t_k)}{h_x^2}-\frac{h_x^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i,y_j,t_k)$$

where

$$\xi_i\in(x_{i-1},x_{i+1}),$$

and

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j, t_k) = \frac{u(x_i, y_{j+1}, t_k) - 2u(x_i, y_j, t_k) + u(x_i, y_{j-1}, t_k)}{h_y^2} - \frac{h_y^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j, t_k)$$

where

$$\eta_j \in (y_{j-1}, y_{j+1}).$$

Substituting these into Equation 5.1 gives

$$\frac{u(x_i, y_j, t_{k+1}) - 2u(x_i, y_j, t_k) + u(x_i, y_j, t_{k-1})}{n^2} - \alpha^2 [\frac{u(x_{i+1}, y_j, t_k) - 2u(x_i, y_j, t_k) + u(x_{i-1}, y_j, t_k)}{h_x^2}$$

$$+ \frac{u(x_i, y_{j+1}, t_k) - 2u(x_i, y_j, t_k) + u(x_i, y_{j-1}, t_k)}{h_y^2}]$$

$$= \frac{1}{12}[n^2 \frac{\partial^4 u}{\partial t^4}(x_i, y_j, \mu_k) - \alpha^2 \frac{\partial^4 u}{\partial x^4}(\xi_i, y_j, t_k) - \alpha^2 \frac{\partial^4 u}{\partial y^4}(x_i, \eta_j, t_k)].$$

Neglecting the truncation error,

$$\tau_{i,j,k} = \frac{1}{12}[n^2 \frac{\partial^4 u}{\partial t^4}(x_i, y_j, \mu_k) - \alpha^2 \frac{\partial^4 u}{\partial x^4}(\xi_i, y_j, t_k) - \alpha^2 \frac{\partial^4 u}{\partial y^4}(x_i, \eta_j, t_k)]$$

and letting $n = \Delta t$, $h_x = \Delta x$, $h_y = \Delta y$, and $u(x_i, y_j, t_k) = w_{i,j}^k$, leads to the difference equation

$$\frac{w_{i,j}^{k+1} - 2w_{i,j}^k + w_{i,j}^{k-1}}{\Delta t^2} - \alpha^2 [\frac{w_{i+1,j}^k - 2w_{i,j}^k + w_{i-1,j}^k}{\Delta x^2} + \frac{w_{i,j+1}^k - 2w_{i,j}^k + w_{i,j-1}^k}{\Delta y^2}] = 0.$$

By multiplying Δt^2, we obtain

$$w_{i,j}^{k+1} - 2w_{i,j}^{k} + w_{i,j}^{k-1} = \Delta t^2\alpha^2[\frac{w_{i+1,j}^{k} - 2w_{i,j}^{k} + w_{i-1,j}^{k}}{\Delta x^2} + \frac{w_{i,j+1}^{k} - 2w_{i,j}^{k} + w_{i,j-1}^{k}}{\Delta y^2}].$$

After removing two terms to the right hand side, we obtain

$$w_{i,j}^{k+1} = 2(1-\frac{\Delta t^2\alpha^2}{\Delta x^2}-\frac{\Delta t^2\alpha^2}{\Delta y^2})w_{i,j}^{k}+\frac{\Delta t^2\alpha^2}{\Delta x^2}w_{i+1,j}^{k}+\frac{\Delta t^2\alpha^2}{\Delta x^2}w_{i-1,j}^{k}+\frac{\Delta t^2\alpha^2}{\Delta y^2}w_{i,j+1}^{k}+\frac{\Delta t^2\alpha^2}{\Delta y^2}w_{i,j-1}^{k}-w_{i,j}^{k-1}.$$

Letting $\frac{\Delta t^2\alpha^2}{\Delta x^2} = \lambda_x^2$ and $\frac{\Delta t^2\alpha^2}{\Delta y^2} = \lambda_y^2$, we get the equation 5.2

$$w_{i,j}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)w_{i,j}^{k} + \lambda_x^2(w_{i+1,j}^{k} + w_{i-1,j}^{k}) + \lambda_y^2(w_{i,j+1}^{k} + w_{i,j-1}^{k}) - w_{i,j}^{k-1}. \quad (5.2)$$

Equation 5.2 implies that to evaluate the $(k+1)^{\text{st}}$ time step value requires five values from the k^{th} and one $(k-1)^{\text{th}}$ time steps (Fig. 5.1). Its communication relationship is illustrated in Fig. 5.2. The application is decomposed into $N_X \times N_Y$ subtasks for assigning to $N_X \times N_Y$ processors. Each subtask communicates only with its four ($\pm X$ and $\pm Y$ direction) nearest neighbors and the boundary cases is handled by the periodic boundary conditions (PBC). One example communication pattern for $N_X = 32$ and $N_Y = 4$ is illustrated in Fig. 5.3. Each cross represents a communication request from the sender to the receiver.

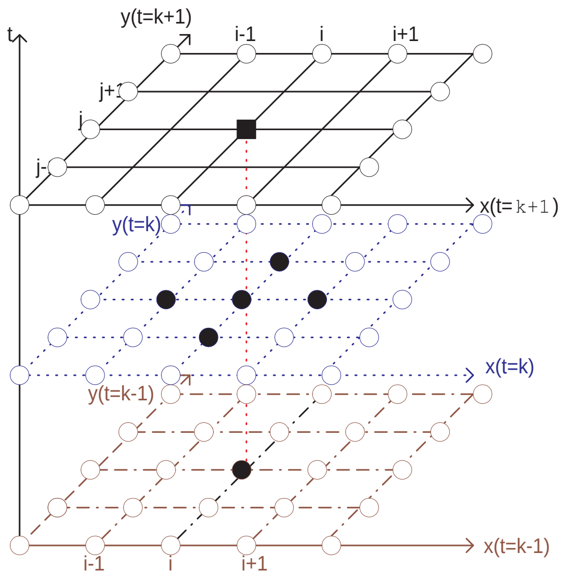


Figure 5.1: Using finite-difference method to solve 2D hyperbolic partial-differential equation

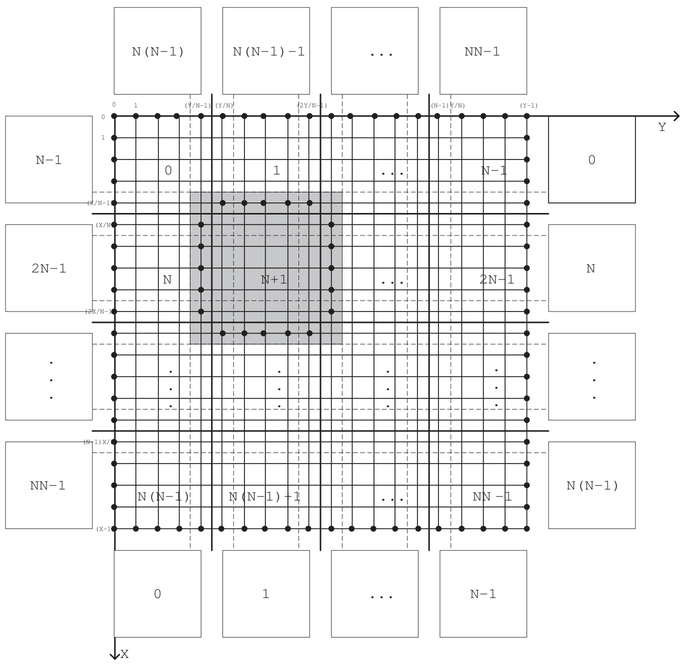


Figure 5.2: The ghost region of a 2D hyperbolic equation

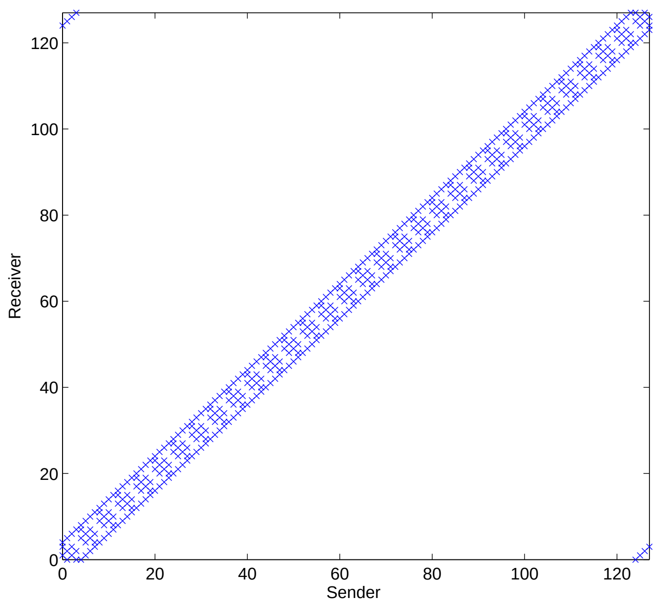


Figure 5.3: The communication pattern of 2D wave equation decomposed as 32×4 subtasks

5.1.1 BG/L re-map interface and mapping strategy

When an application is implemented in MPI and executed on a given parallel computer, there are many choices implementing this mapping.

A built-in mapping strategy in the parallel program assigns subtasks with properly assigned task ID to the MPI processes. Usually, the MPI rank and the physical processor assignment depends on the MPI implement on the machine. The ultimate mapping from subtasks to processors follows: Task_ID $\rightarrow$ MPI_ID $\rightarrow$ Processor_ID. After fixing one of the two steps, we can achieve the model mapping by changing the other relationship.

In the BG/L system, by using the command:

```
cqsub -t ⟨time⟩ -n ⟨nodecount⟩ -e BGLMPI_MAPPING=explicit mapping file ⟨exe⟩
```

a user can determine the mapping between MPI_ID and Processor_ID [35], so the model can generate an explicit mapping file to realize an assignment. This method has the advantage of not requiring modification of the original program for achieving performance benefit.

5.1.2 Subtasks in 2D-Mesh to machine in 3D-Mesh

Suppose that a 2D Wave equation, is evenly decomposed to $N_X \times N_Y$ subtasks and executed on a $X_{Size} \times Y_{Size}$ computing grids. Tables 5.1, 5.2, 5.3, and 5.4 show the point-to-point communication time of running 1000 iterations of the wave equation on 16-, 32-, 64- and 128-node BG/L system, respectively. The last column shows the communication efficiency gain over the default rank order mapping. We adopt the formula

$$\frac{t_{MPI} - t_{Model}}{t_{MPI}}$$

to calculate the improvement in this thesis. The basic or enhanced models are used to evaluate the cost of each mapping, the cost value in each mapping column is the corresponding normalized value which is evaluated under the condition that the communication demand is one.

Dimensions		**Mappings (Unit:ms)**			**Gain (%)**
(N_X,N_Y)	(X_{Size},Y_{Size})	Random (Cost=23.27)	MPI Rank (Cost=12.80)	Model (Cost=8.39)	
(8,2)	(600,600)	152	142	133	6
(8,2)	(800,800)	159	153	142	7
(8,2)	(1600,800)	177	163	152	7
(8,2)	(4800,800)	292	267	254	5

Table 5.1: Communication timing results for running 1000 iterations 2D wave equation on a $4 \times 4 \times 1$ BG/L mesh

Dimensions		**Mappings (Unit:ms)**			**Gain (%)**
(N_X,N_Y)	(X_{Size},Y_{Size})	Random (Cost=53.87)	MPI Rank (Cost=26.24)	Model (Cost=15.96)	
(8,4)	(176,176)	36	26	26	0
(8,4)	(336,336)	57	33	31	6
(8,4)	(1000,1000)	167	141	130	8
(8,4)	(2000,2000)	285	253	231	9
(8,4)	(2400,2400)	297	264	237	10

Table 5.2: Communication timing results for running 1000 iterations 2D wave equation on a $4 \times 4 \times 2$ BG/L mesh

Dimensions		**Mappings (Unit:ms)**			**Gain (%)**
(N_X,N_Y)	(X_{Size},Y_{Size})	Random (Cost=148.24)	MPI Rank (Cost=43.06)	Model (Cost=34.44)	
(8,8)	(600,600)	98	58	58	0
(8,8)	(1000,1000)	129	66	65	2
(8,8)	(1600,1600)	160	69	68	1
(8,8)	(2400,2400)	376	242	239	1
(N_X,N_Y)	(X_{Size},Y_{Size})	Random (Cost=150.23)	MPI Rank (Cost=76.86)	Model (Cost=31.16)	
(16,4)	(1200,300)	103	71	55	23
(16,4)	(2000,500)	136	88	63	28
(16,4)	(3200,800)	170	102	66	35
(16,4)	(4800,1200)	378	306	237	23

Table 5.3: Communication timing results for running 1000 iterations 2D wave equation on a $8 \times 4 \times 2$ BG/L mesh

Dimensions		**Mappings (Unit:ms)**			**Gain (%)**
(N_X,N_Y)	(X_{Size},Y_{Size})	Random (Cost=357.22)	MPI Rank (Cost=111.17)	Model (Cost=71.30)	
(16,8)	(800,800)	79	36	33	8
(16,8)	(1600,1600)	130	47	40	15
(16,8)	(2000,2000)	230	142	128	10
(16,8)	(2400,2400)	236	145	128	12
(N_X,N_Y)	(X_{Size},Y_{Size})	Random (Cost=319.40)	MPI Rank (Cost=155.12)	Model (Cost=63.41)	
(32,4)	(1600,400)	77	53	31	42
(32,4)	(3200,800)	126	80	39	51
(32,4)	(4800,1200)	230	203	129	36

Table 5.4: Communication timing results for running 1000 iterations 2D wave equation on a $8 \times 4 \times 4$ BG/L mesh

The model mapping (Fig. 5.4) that assigns 32×4 subtasks to $8 \times 4 \times 4$ partitions improves point-to-point communication by up to 51%. The four columns are distinguished by colors. The subtasks in the same column are assigned to the processors marked with a dotted straight line of the same color while the dotted arc shows the continuation of the assignment.

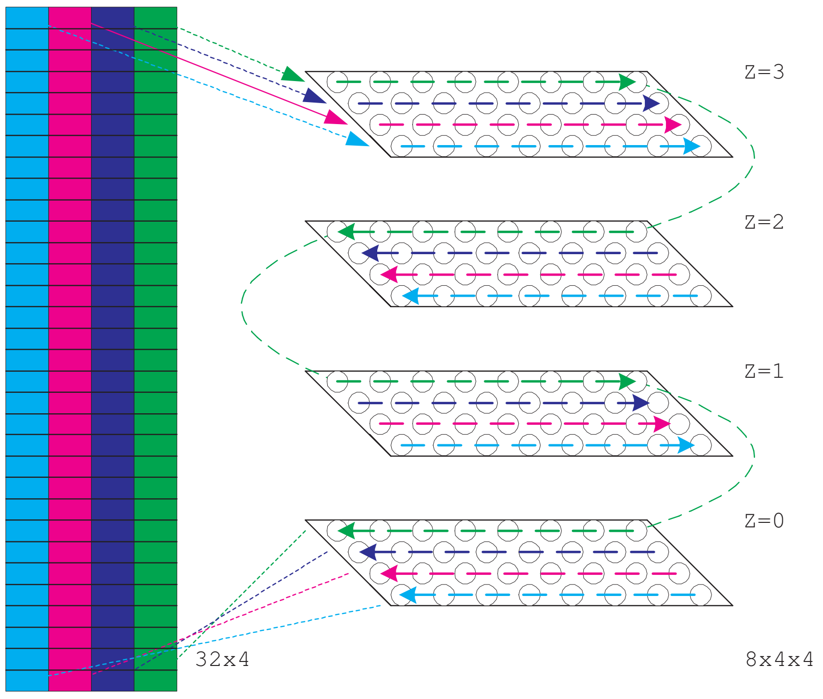


Figure 5.4: Mapping 32×4 subtasks onto $8 \times 4 \times 4$ mesh

5.1.3 Subtasks in 2D-Mesh to machine in 3D-Torus

Tables 5.5 and 5.6 show the point-to-point communication timing results of running 1000 iterations of the 2D wave equation on 512- and 1024-node BG/L torus network.

Dimensions		**Mappings (Unit:ms)**			**Gain (%)**
(N_X, N_Y)	(X_{Size}, Y_{Size})	Random (Cost=1654.29)	MPI Rank (Cost=540.55)	Model (Cost=199.52)	
(16,32)	(1600,1600)	94	37	34	8
(16,32)	(1600,3200)	113	45	40	11
(16,32)	(2400,4800)	156	54	49	9

Table 5.5: Communication time iterations for running 1000 iterations 2D wave equation on a $8 \times 8 \times 8$ BG/L torus

Dimensions		**Mappings (Unit:ms)**			**Gain (%)**
(N_X, N_Y)	(X_{Size}, Y_{Size})	Random (Cost=3880.33)	MPI Rank (Cost=570.69)	Model (Cost=333.53)	
(64,16)	(6400,1600)	130	44	34	23
(64,16)	(6400,3200)	182	58	40	31
(64,16)	(5120,3840)	182	58	40	31
(64,16)	(3200,6400)	277	172	135	22

Table 5.6: Communication time results for running 1000 iterations 2D wave equation on a $8 \times 8 \times 16$ BG/L torus

The model mapping (Fig. 5.5) that assigns 64×16 subtasks to $8 \times 8 \times 16$ partition improves point-to-point communication by up to 31%. Different columns are distinguished by colors. The subtasks in the same column are mapped to the processors marked dotted straight line of the same color while the colored arcs represent the torus connection along the Y axis and the black arcs represent the torus connection along the Z axis.

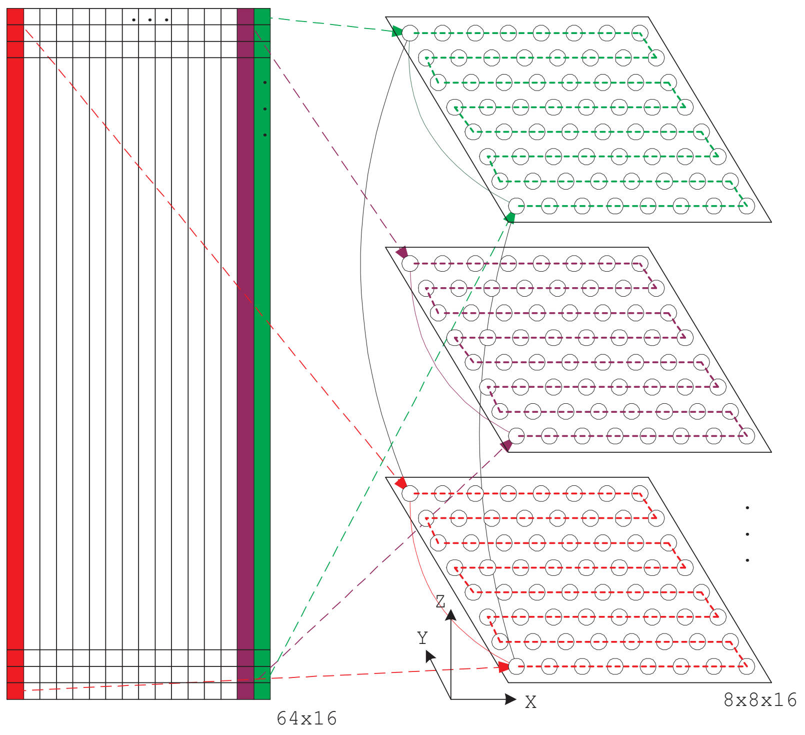


Figure 5.5: Mapping 64 × 16 subtasks onto 8 × 8 × 16 torus

5.2 SMG2000

The SMG2000 benchmark is an important component in the Lawrence Livermoore National Laboratory (LLNL) Purple benchmark package for scientific applications. More specially, SMG2000 is a parallel semicoarsening multigrid solver for the linear systems arising from finite difference, finite volume, or finite element discretizations of the diffusion equation,

$$-\nabla \cdot (D\nabla u) + \sigma u = f \tag{5.3}$$

on logically rectangular grids [70]. The code solves both 2D and 3D problems with discretization stencils of up to 9-point in 2D and up to 27-point in 3D. This solver can serve as a key component for achieving scalability in radiation diffusion simulations.

Multigrid methods are a class of techniques for computing fast, iterative solutions of linear systems in linear time and space. These methods improve the convergence rate of classic iterative methods by using a hierarchy of grids at different resolutions. Coarser levels of the grid hierarchy are effective for quickly eliminating low frequency components of solution error. As a result, the total computational work required by a multigrid method to achieve a prescribed level of accuracy is proportional to the grid size.

In the SMG2000 benchmark, the solver is based on Eq.(5.3) which focussed on the scalablility for a three dimensional semicoarsening multigrid solver on a distributed-memory computer. In the best case, the efficiency

$$E(N, P) = \frac{T(N, 1)}{T(PN, P)}$$

should be equal to 1, where $T(N, P)$ represents the time to solve a linear system with N unknowns on a computer using P processors.

SMG2000 is written in ISO-C with MPI. Parallelism is achieved by data decomposition. The driver provided with SMG2000 achieves this decomposition by simply subdividing the grid into logical $PX \times PY \times PZ$ chunks of equal size. SMG2000 is a highly synchronous code. The communication and computation patterns exhibit the surface-to-volume relationship common to many parallel scientific codes. Hence, parallel efficiency is largely determined by the size of $PX \times PY \times PZ$, and the speed of communications and computations on the machine. SMG2000 is also memory-access bound, doing only about 1-2 computations per memory access, so memory-access speeds will also have a large impact on performance.

For SMG2000, the communication patterns depends on both machine size and problem size, i.e., the mapping model needs to recalculate if the problem size changes. For example, when PX=PY=PZ=4, Figs. 5.6 and 5.8 illustrate the communication patterns for NX=NY=NZ=10 and NX=100, NY=100, NZ=80, respectively. Figs. 5.7 and 5.9 illustrate the corresponding sending bytes for each case. Based on our models, the re-mapping is needed when the demand matrix changes.

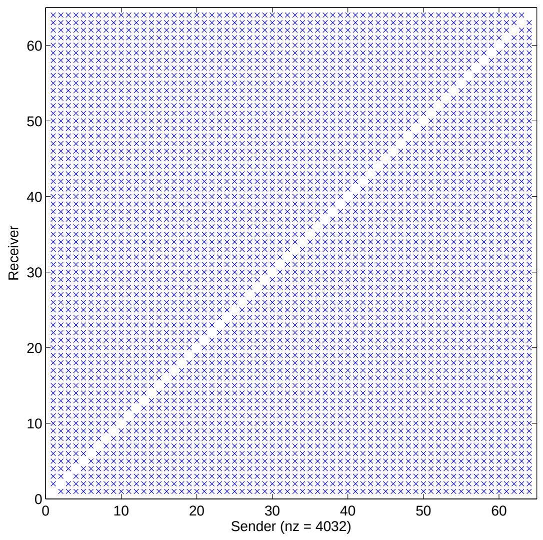


Figure 5.6: The communication pattern of SMG2000 when PX=PY=PZ=4 and NX=NY=NZ=10

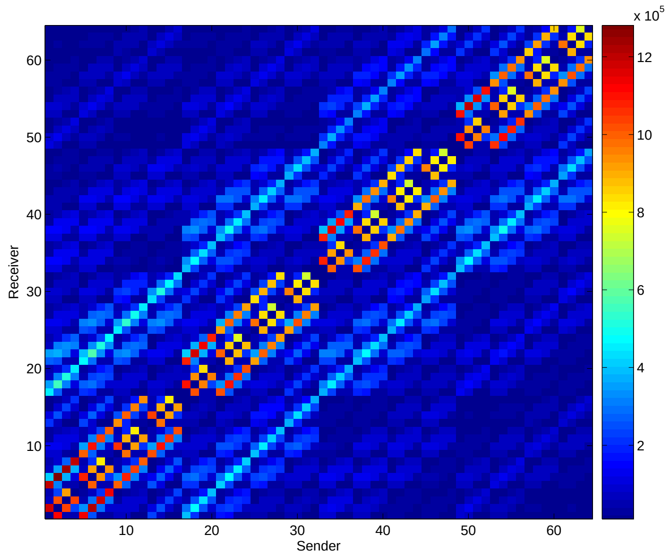


Figure 5.7: The MPI data transfer of SMG2000 when PX=PY=PZ=4 and NX=NY=NZ=10

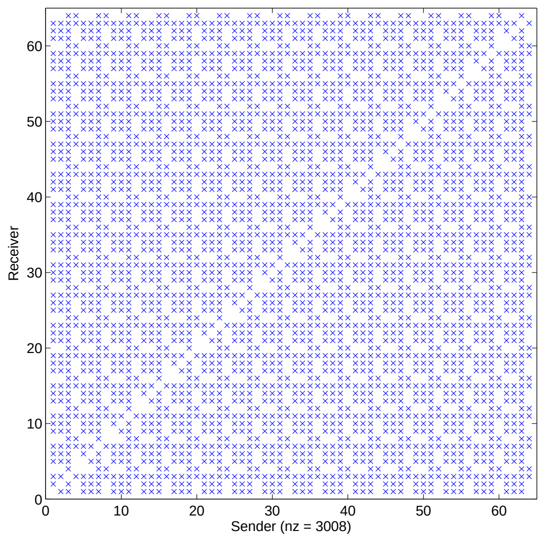


Figure 5.8: The communication pattern of SMG2000 when PX=PY=PZ=4, and NX=100,NY=100,NZ=80

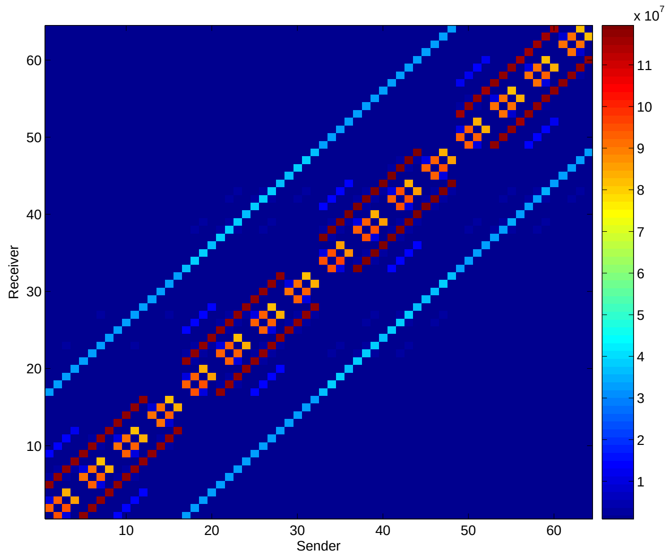


Figure 5.9: The MPI Sending (Bytes) of SMG2000 when PX=PY=PZ=4, and NX=100,NY=100,NZ=80

Tables 5.7, 5.8, and 5.9 show timing results of running SMG2000 benchmark on 64-, 128- and 256-node BG/L system, respectively. They are all instances that map 3D-mesh communication pattern onto 3D-mesh parallel computers. Without the information of the communication overlapping, only the basic model is used to evaluate cost for each mapping. The timing results in the "SMG Solve" column come from the SMG2000 standard output file which indicate the overall time that the adopted solver needs to solve the given problem (default in 2 digit decimal), while the timing results in the "Comm." and "Total" columns collected by the MPI profiling tool, MPI_Trace, represent the total communication time and total elapsed time in a MPI view (default in 3 digit decimal), respectively. The timing results from the "Comm." and "Total" columns are used to evaluate the improvement for each and cross check the standard output in the "SMG solve" column. The corresponding performance improvements are showed in Tables 5.10, 5.11 and 5.12, respectively.

Dimensions		**Rank Mapping**			**Model Mapping**		
(PX,PY,PZ)	(NX,NY,NZ)	SMG Solve (Sec.)	Comm. (Sec.)	Total (Sec.)	SMG Solve (Sec.)	Comm. (Sec.)	Total (Sec.)
(4,4,4)	(10,10,10)	1.05	1.620	2.343	1.05	1.578	2.302
(4,4,4)	(40,40,40)	10.05	6.924	14.605	9.79	6.515	14.195
(4,4,4)	(80,80,80)	52.72	20.069	66.604	50.94	17.905	64.427
(4,4,4)	(100,100,80)	77.25	24.803	95.789	74.71	21.677	92.659

Table 5.7: Timing results for running SMG2000 on a $8 \times 4 \times 2$ BG/L mesh

Dimensions		**Rank Mapping**			**Model Mapping**		
(PX,PY,PZ)	(NX,NY,NZ)	SMG Solve (Sec.)	Comm. (Sec.)	Total (Sec.)	SMG Solve (Sec.)	Comm. (Sec.)	Total (Sec.)
(4,8,4)	(10,10,10)	1.21	2.102	2.928	1.21	2.053	2.879
(4,8,4)	(40,40,40)	10.63	8.065	16.015	10.34	7.610	15.561
(4,8,4)	(80,80,80)	54.23	22.367	69.468	52.14	19.829	66.941
(4,8,4)	(100,100,80)	79.57	27.924	99.581	76.24	23.984	95.644
(4,8,4)	(100,80,100)	81.17	28.342	101.675	77.20	23.698	97.033

Table 5.8: Timing results for running SMG2000 on a $8 \times 4 \times 4$ BG/L mesh

Dimensions		**Rank Mapping**			**Model Mapping**		
(PX,PY,PZ)	(NX,NY,NZ)	SMG Solve (Sec.)	Comm. (Sec.)	Total (Sec.)	SMG Solve (Sec.)	Comm. (Sec.)	Total (Sec.)
(4,8,8)	(10,10,10)	1.23	2.614	3.461	1.22	2.557	3.403
(4,8,8)	(40,40,40)	11.13	9.706	17.734	10.86	9.246	17.277
(4,8,8)	(80,80,80)	62.92	27.282	79.883	60.62	24.528	77.134
(4,8,8)	(80,100,100)	94.34	34.088	116.778	91.08	30.235	112.936

Table 5.9: Timing results for running SMG2000 on a $8 \times 4 \times 8$ BG/L mesh

Dimensions		**Efficiency Improvement(%)**		
(PX,PY,PZ)	(NX,NY,NZ)	SMG Solve	Communication	Total
(4,4,4)	(10,10,10)	0	2.59	1.75
(4,4,4)	(40,40,40)	2.59	5.91	2.81
(4,4,4)	(80,80,80)	3.38	10.78	3.27
(4,4,4)	(100,100,80)	3.29	12.60	3.27

Table 5.10: The improvement over default layout of SMG2000 on a $8 \times 4 \times 2$ BG/L mesh

Dimensions		**Efficiency Improvement(%)**		
(PX,PY,PZ)	(NX,NY,NZ)	SMG Solve	Communication	Total
(4,8,4)	(10,10,10)	0	2.33	1.67
(4,8,4)	(40,40,40)	2.73	5.64	2.83
(4,8,4)	(80,80,80)	3.85	11.35	3.64
(4,8,4)	(100,100,80)	4.18	14.11	3.95
(4,8,4)	(100,80,100)	4.89	16.39	4.57

Table 5.11: The improvement over default layout of SMG2000 on a $8 \times 4 \times 4$ BG/L mesh

Dimensions		**Efficiency Improvement(%)**		
(PX,PY,PZ)	(NX,NY,NZ)	SMG Solve	Communication	Total
(4,8,8)	(10,10,10)	0.81	2.18	1.68
(4,8,8)	(40,40,40)	2.43	4.74	2.58
(4,8,8)	(80,80,80)	3.66	10.09	3.44
(4,8,8)	(80,100,100)	3.46	11.30	3.29

Table 5.12: The improvement over default layout of SMG2000 on a $8 \times 4 \times 8$ BG/L mesh

From Tables 5.10, 5.11 and 5.12, we observe that our mapping models achieve the SMG Solve benchmark gain, the total time, and communication time by up to 5%, 5%, and 16%, respectively.

5.3 NPB Benchmarks

Administration Advanced Supercomputing (NAS) division of NASA Ames Research Center focuses on computational fluid dynamics and related aero-science disciplines. To measure objectively the performance of highly parallel computers and compare them with that of conventional supercomputers, NAS developed the first version NAS Parallel Benchmarks (NPB1.0) in 1991. The NPB consist of five parallel kernels and three simulated application benchmarks that are based on Fortran 77 and the MPI. Those benchmarks are derived from computational fluid dynamics codes and widely accepted as a standard indicator of supercomputer performance [20].

In NPB1.0, only "class A" and "class B" sizes are defined. The larger sizes such as "class C" and "class D" are introduced in the later versions to suit the new generation of supercomputers.

NPB contains several features. The benchmarks must be compiled for a specific grid size and number of processors. The timing results are valid only when the execution configurations are identical to those specified at the compile time. Self-verification is contained in the code to determine if each run has completed with the correct results. If possible, the code runs for one time step and then reinitializes before timing begins. The purpose of this extra iteration is to eliminate startup costs associated with demand paging and cache loading by making sure that all code and data has been touched. One important measure, Mop/s, millions of operations per second, indicates the sysetem performance. The estimate of Mop/s rates is based on actual operation counts without compiler optimizations [20].

5.3.1 NPB MG

Multigrid (MG) is a simplified multigrid kernel for solving a 3D Poisson PDE. It is one kernel of the NPB. The communication pattern when MG is executed on 256 processors is illustrated in Fig. 5.10. This code requires a power-of-two number of processors. The partitioning of the grid onto processors occurs such that the grid is successively halved, starting from the Z dimension, then Y and then X dimension, and repeating this process until all processors are assigned [20]. The class A MG problem is briefly defined as follows [19]:

Equation

Four iterations of the V-cycle multigrid algorithm are used to obtain an approximate solution u to the discrete Poisson problem

$$\nabla^2 u = v \tag{5.4}$$

on a $256 \times 256 \times 256$ grid with periodic boundary conditions.

Algorithm

Set $v = 0$ except at the 20 points where $v = \pm 1$. These 20 points are determined as the locations of the ten largest and ten smallest pseudorandom numbers.

Begin the iterative solution with $u = 0$. Each of the four iterations consists of the following two steps, in which $k = 8 = \log_2 256$:

$$r = v - Au \qquad \text{(evaluated residual)}$$

$$u = u + M^k r \qquad \text{(apply correction)}$$

Here M^k denotes the V-cycle multigrid operator.

Timing

clocking starts after the initializing u and v, but before evaluating the residual for the first time. It is stopped after evaluating the norm of the final residual, but before displaying or printing values.

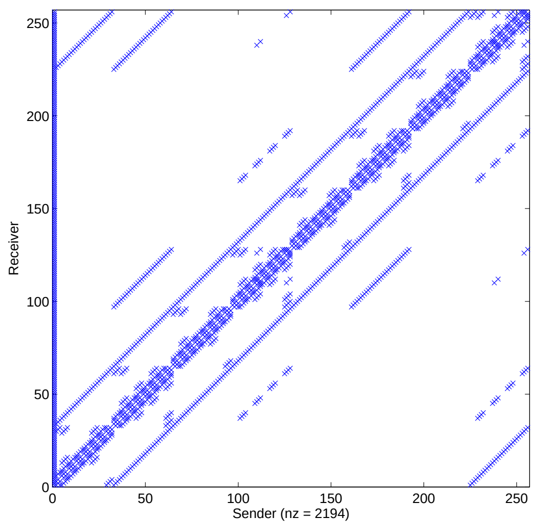


Figure 5.10: The communication pattern of MG on 256 Processors (Class D)

Tables 5.13, 5.14, and 5.15 show timing results of running NPB MG benchmark on 64-, 128- and 256-node BG/L system, respectively. They are all instances that map 3D-mesh communication pattern onto 3D-mesh parallel computers. Without the information of the communication overlapping, only the basic model is used to evaluate cost for each mapping. The results in the "MOps/P" column and "Time" come from the NPB MG standard output file which indicate system performance (default in 2 digit decimal), while the timing results in the "Comm." and "Total" columns collected by MPI_Trace, represent the total MPI communication time and total elapsed time (default in 3 digit decimal). The corresponding performance improvements are showed in Tables 5.16, 5.17 and 5.18, respectively.

Problem Size	**Rank Mapping**				**Model Mapping**			
NX×NY×NZ (Class)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)
$256 \times 256 \times 256$(A)	168.63	0.36	0.096	0.530	181.64	0.33	0.061	0.495
$256 \times 256 \times 256$(B)	179.31	1.70	0.359	1.865	193.23	1.57	0.226	1.733
$512 \times 512 \times 512$(C)	216.80	11.22	1.255	12.375	225.70	10.78	0.771	11.891
$1024 \times 1024 \times 1024$(D)	237.88	204.54	11.234	213.117	242.93	200.28	6.810	208.700

Table 5.13: Timing results for running NPB MG benchmark on a $8 \times 4 \times 2$ BG/L mesh

Problem Size	**Rank Mapping**				**Model Mapping**			
NX×NY×NZ(Class)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)
$256 \times 256 \times 256$(A)	150.87	0.20	0.073	0.295	165.05	0.18	0.048	0.270
$256 \times 256 \times 256$(B)	160.22	0.95	0.271	1.043	175.38	0.87	0.182	0.953
$512 \times 512 \times 512$(C)	203.45	5.98	0.891	6.585	214.28	5.68	0.558	6.253
$1024 \times 1024 \times 1024$(D)	230.86	105.38	7.802	109.760	236.73	102.76	5.066	107.029

Table 5.14: Timing results for running NPB MG benchmark on a $8 \times 4 \times 4$ BG/L mesh

Problem Size	**Rank Mapping**				**Model Mapping**			
NX×NY×NZ(Class)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)
256 × 256 × 256(A)	125.17	0.12	0.054	0.177	141.73	0.11	0.034	0.157
256 × 256 × 256(B)	132.29	0.57	0.201	0.630	149.13	0.51	0.131	0.559
512 × 512 × 512(C)	180.63	3.37	0.641	3.693	196.72	3.09	0.345	3.396
1024 × 1024 × 1024(D)	223.99	54.30	5.476	56.532	236.16	51.51	2.581	53.633

Table 5.15: Timing results for running NPB MG benchmark on a $8 \times 4 \times 8$ BG/L mesh

Problem Size	**Efficiency Improvement(%)**			
NX×NY×NZ(Class)	MOps/P	Time	Communication	Total
256 × 256 × 256 (A)	7.72	8.33	36.46	6.60
256 × 256 × 256 (B)	7.76	7.65	37.05	7.08
512 × 512 × 512 (C)	4.11	3.92	38.57	3.91
1024 × 1024 × 1024 (D)	2.11	2.08	39.38	2.07

Table 5.16: The improvement over default layout of NPB MG on a $8 \times 4 \times 2$ BG/L mesh

Problem Size	**Efficiency Improvement(%)**			
NX×NY×NZ(Class)	MOps/P	Time	Communication	Total
256 × 256 × 256 (A)	9.40	10.00	34.25	8.47
256 × 256 × 256 (B)	9.46	8.42	32.84	8.63
512 × 512 × 512 (C)	5.32	5.02	37.37	5.04
1024 × 1024 × 1024 (D)	2.54	2.49	35.07	2.49

Table 5.17: The improvement over default layout of NPB MG on a $8 \times 4 \times 4$ BG/L mesh

In [13], the best improvement when MG benchmark is tested on 128 processors is 5.83%. With the same measure, we boost it to 9.4% here (Tables 5.14 and 5.17), almost doubling the improvement of the previous work.

Problem Size	**Efficiency Improvement(%)**			
NX×NY×NZ(Class)	MOps/P	Time	Communication	Total
$256 \times 256 \times 256$ (A)	13.23	8.33	37.04	11.30
$256 \times 256 \times 256$ (B)	12.73	10.53	34.83	11.27
$512 \times 512 \times 512$ (C)	8.91	8.31	46.18	8.04
$1024 \times 1024 \times 1024$ (D)	5.43	5.14	52.87	5.13

Table 5.18: The improvement over default layout of NPB MG on a $8 \times 4 \times 8$ BG/L mesh

In summary, when NPB MG benchmark is executed on 3D-mesh BG/L with up to 256 processors, our models improve the performance by up to 13% and reduce the total time and communication time by up to 11% and 53%, respectively.

5.3.2 NPB CG

Conjugate gradient (CG) is another kernel of the NPB, that finds an estimate of the largest eigenvalue of a symmetric positive-definite sparse matrix by the conjugate gradient method [19]. Fig. 5.11 illustrates the communication pattern when CG is executed on 256 processors.

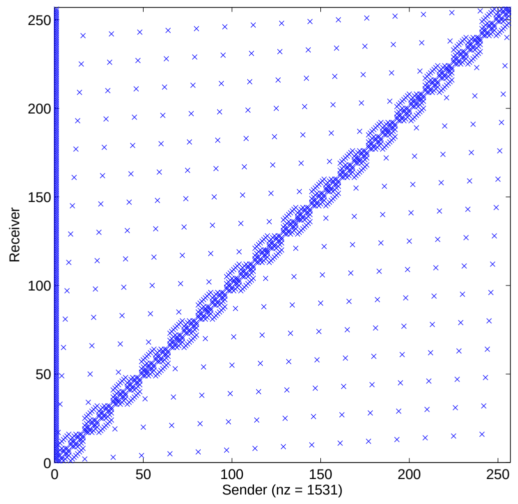


Figure 5.11: The communication pattern of CG on 256 Processors (Class D)

Tables 5.19, 5.20, 5.21 and 5.22 show timing results of running NPB CG benchmark on 64-, 128-, 256-, and 512-node BG/L system, respectively. The first three cases map 3D-mesh communication pattern onto 3D-mesh parallel computer while the last maps 3D-mesh pattern onto 3D-torus system. Without the information of the communication overlapping, only the basic model is used to evaluate cost for each mapping. The results in the "MOps/P" column and "Time" come from the NPB CG standard output file summarizing the system performance (default in 2 digit decimal), while the timing results in the "Comm." and "Total" columns collected by MPI_Trace, represent the total MPI communication time and total elapsed time (default in 3 digit decimal). The corresponding performance improvements are tabulated in Tables 5.23, 5.24, 5.25, and 5.26, respectively.

Problem Size	**Rank Mapping**				**Model Mapping**			
NX×NY×NZ(Class)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)
$256 \times 256 \times 256$(A)	25.65	0.91	0.546	1.141	31.31	0.75	0.373	0.965
$256 \times 256 \times 256$(B)	23.71	36.06	11.613	38.059	26.66	32.07	7.558	34.011
$512 \times 512 \times 512$(C)	22.50	99.55	23.017	104.421	24.95	89.78	13.122	94.520
$1024 \times 1024 \times 1024$(D)	22.61	2517.51	308.038	2586.478	23.62	2410.22	199.741	2478.184

Table 5.19: Timing results for running NPB CG benchmark on a $8 \times 4 \times 2$ BG/L mesh

Problem Size	**Rank Mapping**				**Model Mapping**			
NX×NY×NZ(Class)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)
$256 \times 256 \times 256$(A)	24.72	0.47	0.301	0.666	26.13	0.45	0.276	0.640
$256 \times 256 \times 256$(B)	25.85	16.54	6.102	18.224	28.43	15.03	4.578	16.701
$512 \times 512 \times 512$(C)	24.69	45.35	12.473	49.370	26.51	42.25	9.468	46.221
$1024 \times 1024 \times 1024$(D)	22.63	1257.45	163.894	1311.463	23.43	1214.70	120.712	1268.251

Table 5.20: Timing results for running NPB CG benchmark on a $8 \times 4 \times 4$ BG/L mesh

Problem Size	**Rank Mapping**				**Model Mapping**			
NX×NY×NZ(Class)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)
$256 \times 256 \times 256$(A)	15.02	0.39	0.287	0.575	16.94	0.35	0.240	0.528
$256 \times 256 \times 256$(B)	17.79	12.01	6.072	13.628	19.53	10.94	4.988	12.544
$512 \times 512 \times 512$(C)	18.89	29.64	12.050	33.397	20.63	27.15	9.522	30.871
$1024 \times 1024 \times 1024$(D)	19.63	725.05	167.374	772.556	20.62	690.25	132.203	737.388

Table 5.21: Timing results for running NPB CG benchmark on a $8 \times 4 \times 8$ BG/L mesh

Problem Size	**Rank Mapping**				**Model Mapping**			
NX×NY×NZ(Class)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)
$256 \times 256 \times 256$(A)	11.88	0.25	0.191	0.420	12.44	0.23	0.178	0.409
$256 \times 256 \times 256$(B)	17.67	6.05	3.446	7.566	19.92	5.37	2.792	6.874
$512 \times 512 \times 512$(C)	19.43	14.41	6.582	17.913	21.51	13.02	5.176	16.507
$1024 \times 1024 \times 1024$(D)	19.42	366.38	85.308	409.349	20.49	347.28	66.035	390.060

Table 5.22: Timing results for running NPB CG benchmark on a $8 \times 8 \times 8$ BG/L mesh

Problem Size	**Efficiency Improvement(%)**			
NX×NY×NZ(Class)	MOps/P	Time	Communication	Total
$256 \times 256 \times 256$ (A)	22.07	17.58	31.68	15.43
$256 \times 256 \times 256$ (B)	12.44	11.06	34.92	10.64
$512 \times 512 \times 512$ (C)	10.89	9.81	42.99	9.48
$1024 \times 1024 \times 1024$ (D)	4.47	4.26	35.16	4.19

Table 5.23: The improvement over default layout of NPB CG on a $8 \times 4 \times 2$ BG/L mesh

Problem Size	**Efficiency Improvement(%)**			
NX×NY×NZ(Class)	MOps/P	Time	Communication	Total
$256 \times 256 \times 256$ (A)	5.70	4.26	8.31	3.90
$256 \times 256 \times 256$ (B)	9.98	9.13	24.98	8.36
$512 \times 512 \times 512$ (C)	7.37	6.84	24.09	6.38
$1024 \times 1024 \times 1024$ (D)	3.54	3.40	26.35	3.29

Table 5.24: The improvement over default layout of NPB CG on a $8 \times 4 \times 4$ BG/L mesh

Problem Size	**Efficiency Improvement(%)**			
NX×NY×NZ(Class)	MOps/P	Time	Communication	Total
$256 \times 256 \times 256$ (A)	12.78	10.26	16.38	8.17
$256 \times 256 \times 256$ (B)	9.78	8.91	17.85	7.95
$512 \times 512 \times 512$ (C)	9.21	8.40	20.98	7.56
$1024 \times 1024 \times 1024$ (D)	5.04	4.80	21.01	4.55

Table 5.25: The improvement over default layout of NPB CG on a $8 \times 4 \times 8$ BG/L mesh

Problem Size	**Efficiency Improvement(%)**			
NX×NY×NZ(Class)	MOps/P	Time	Communication	Total
$256 \times 256 \times 256$ (A)	4.71	8.00	6.81	2.62
$256 \times 256 \times 256$ (B)	12.73	11.24	18.98	9.15
$512 \times 512 \times 512$ (C)	10.71	9.65	21.36	7.85
$1024 \times 1024 \times 1024$ (D)	5.51	5.21	22.59	4.71

Table 5.26: The improvement over default layout of NPB CG on a $8 \times 8 \times 8$ BG/L mesh

In summary, when NPB CG benchmark is executed on 3D mesh and torus BG/L up to 512 processors, our models improve the system performance by up to 22% and reduce the total time and communication time by up to 15% and 43%, respectively.

Chapter 6

Performance Analysis and Tools

We provide analysis of the performance of our models on several benchmarks with BG/L system.

6.1 MPI performance analysis tools

Many MPI performance analysis tools are available for collecting information at run-time to help identify bottlenecks and improve the performance. The two types of performance tools include profiling and tracing. The profiling tools, such as mpiP [3] and MPI_Trace [65], generate cumulative timing results while tracing tools, such as Vampir [8], KOJAK [2], Paraver [4, 84] and PAPI [5], can be used to collect and display a sequence of time-stamped events [50, 51]. Some other tools like IBM HPCT [36, 37, 39] and TAU [7] may perform both profiling and tracing.

6.1.1 MPI profiling interface

MPI implements its profiling interface by offering two distinct libraries of functions: a production library in which each function is optimized and a profiling library in which each function has a built-in counter or timer. The MPI Standards require each implementation to allow each MPI function to be called by its usual name and modified name which is the corresponding usual name preceded by a capital "P". For example, a process can send the float type data stored in x to process 0 with either

```
MPI_Send(&x, 1, MPI_FLOAT, 0, 0, MPI_COMM_WORLD)
```

or

```
PMPI_Send(&x, 1, MPI_FLOAT, 0, 0, MPI_COMM_WORLD)
```

which allows users to implement their own profiling with link to the execution code [71].

For example, the MPI_Trace wrappers use the MPI profiling interface in the following form:

```
int MPI_Send(...) {
      start_timing();
      PMPI_Send(...);
      stop_timing();
      log_the_event();
}
```

6.1.2 MPI_Trace tool

MPI_Trace collects timing summary and prints out the individual profiling file. It is installed on Argonne National Laboratory BG/L system and also integrated in the IBM HPCT package installed on BG/P system. Fig. 6.1 shows a typical profiling output for the MPI Rank 0. The file, `mpi_profile.0`, collects much useful information:

```
elapsed time from clock-cycles using freq = 700.0 MHz
-----------------------------------------------------------------
MPI Routine                  #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                     1            0.0          0.000
MPI_Comm_rank                     1            0.0          0.000
MPI_Send                      19988        29662.7          3.678
MPI_Irecv                     19988        29662.7          0.017
MPI_Wait                      19988            0.0          7.918
MPI_Barrier                       1            0.0          0.000
MPI_Reduce                        1            8.0          0.000
-----------------------------------------------------------------
total communication time = 11.613 seconds.
total elapsed time       = 38.059 seconds.
top of the heap address  = 23.008 MBytes.

-----------------------------------------------------------------
Message size distributions:

MPI_Send                  #calls    avg. bytes      time(sec)
                           11856           8.0          0.023
                             228          16.0          0.000
                            7904       75000.0          3.655

MPI_Irecv                 #calls    avg. bytes      time(sec)
                           11856           8.0          0.009
                             228          16.0          0.000
                            7904       75000.0          0.008

MPI_Reduce                #calls    avg. bytes      time(sec)
                               1           8.0          0.000

-----------------------------------------------------------------

Communication summary for all tasks:

  minimum communication time = 11.312 sec for task 9
  median  communication time = 12.175 sec for task 62
  maximum communication time = 12.374 sec for task 58

taskid  xcoord  ycoord  zcoord  procid   total_comm(sec)   avg_hops
     0       0       0       0       0        11.613          1.75
     1       1       0       0       0        11.974          2.25
     2       2       0       0       0        12.251          2.75
     3       3       0       0       0        12.257          2.75
     4       4       0       0       0        12.139          3.00
     5       5       0       0       0        12.150          3.00
     6       6       0       0       0        12.088          3.00
```

Figure 6.1: Segment of the `mpi_profile.0` for NPB CG (Class B) benchmark

It lists all MPI communication functions with the number of times that they are called and the corresponding cumulative time. This helps identify the message passing method and the communication time distribution. It reports both total communication time and total elapsed time. This helps estimate the ratio of the communication time over the execution time and make a reasonable improvement expectation. It also offers a rough analysis data, average hops, for each MPI process. This help predict the performance of the mapping.

The average hops is evaluated by [50]:

$$\text{AverageHops} = \frac{\sum_i Hops_i \times Bytes_i}{\sum_i Bytes_i}$$

where the *Hops* between two processors p and q with physical coordinates (x_p, y_p, z_p) and (x_q, y_q, z_q) is calculated by:

$$Hops(p,q) = |x_p - y_p| + |y_p - y_q| + |z_p - z_q|$$

measuring the average hop distance between two processors.

When the proper variable is set, the wrappers keep track of how many bytes are sent to each task, and a matrix is written during `MPI_Finalize` which lists how many bytes are sent from each task to all other tasks. This matrix is just the demand matrix in our models.

6.1.3 Computational overhead of the profiling tool

The previous timing results are collected by using the profiling tool, MPI_Trace, to gather the profiling data. Whenever a MPI function is invoked, a light weight profiling code is executed, introducing small computational overhead to the timing results of both default MPI rank mappings and model mappings. The overhead mainly depends on the total times of MPI functions being called. In theory, this extra overhead decreases the improvement a little. For NPB benchmarks, the total times is less than $100,000$ and the corresponding overhead is ignorable. For SMG2000, the total times of the MPI functions being called is about half million and the corresponding gain change is about 0.1% and hence negligible. For example, when a SMG2000 with logical processor topology $4 \times 8 \times 4$ and dimension (NX,NY,NZ) being (100,80,100) is assigned onto $8 \times 4 \times 4$ BG/L mesh, the overhead may decrease the SMG Solve gain from 4.98% to 4.89% (see the last row in Table 5.11).

6.2 The validation of model metric

In Chapter 4, we discuss the objective functions in our models and the metric is evaluated by these functions associated with the latency measurement.

To validate the model metric, we study a series of mappings in a SA process for NPB CG. Figs. 6.2 and 6.3 illustrated with the decreasing of the model cost, the communication times decreases, and the system performance increases. The model metric is valid that the mappings generated by our models can significantly reduce the communication time and improve the performance.

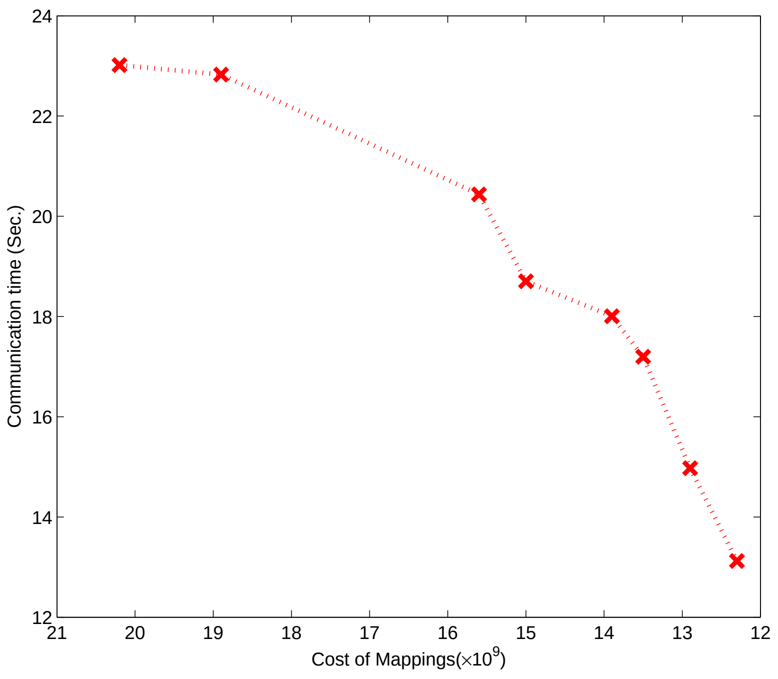


Figure 6.2: Communication time change trend with the cost of mapping change when NPB CG executed on 64processors (Class C)

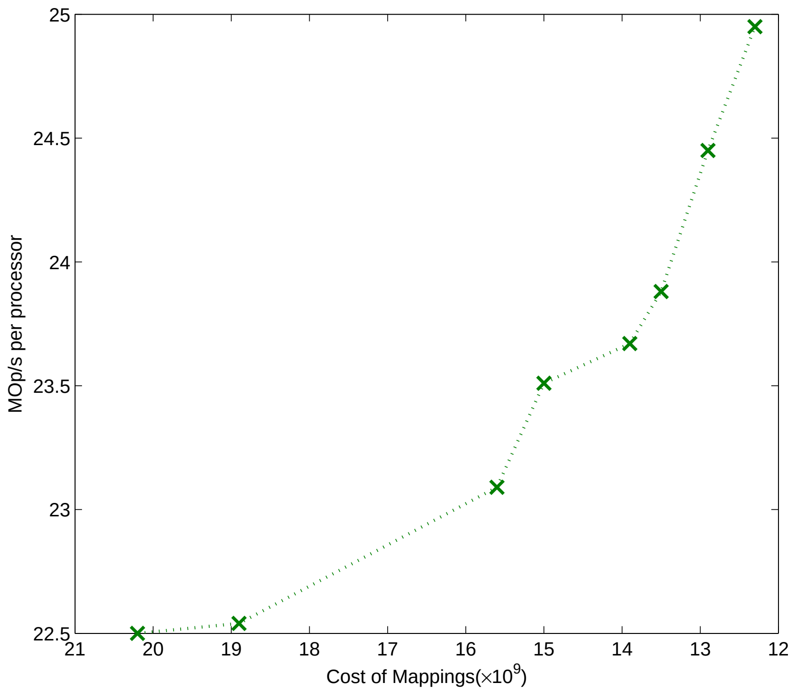


Figure 6.3: Performance change trend with the cost of mapping change when NPB CG executed on 64processors (Class C)

6.3 Fluctuation tests

For the time and resource limit, each timing result in this thesis is from one actual execution. All results from the default mapping and the model mapping are collected with identical environment parameters, isolating the mapping from other distractions.

To study the fluctuation on Blue Gene/L supercomputer, we repeat one test for NPB MG (Class D) on $8\times4\times8$ BG/L ten times, with the first five times only associated with ANL $R001\_J203-256$ partition while the last five times associated with both partition $R001\_J203-256$ and partition $R001\_J102-256$ (Table 6.1).

Run	**Rank Mapping**				**Model Mapping**			
	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)	MOps/P	Time (Sec.)	Comm. (Sec.)	Total (Sec.)
1	223.89	54.33	5.501	56.557	236.18	51.50	2.576	53.627
2	223.86	54.34	5.510	56.565	236.20	51.50	2.571	53.625
3	223.98	54.31	5.476	56.532	236.13	51.51	2.585	53.637
4	224.04	54.29	5.466	56.522	236.22	51.49	2.565	53.618
5	223.95	54.31	5.488	56.544	236.19	51.50	2.571	53.625
6	224.02	54.30	5.467	56.522	236.22	51.49	2.565	53.619
7	223.95	54.31	5.484	56.541	236.21	51.49	2.566	53.620
8	223.99	54.30	5.485	56.543	236.25	51.49	2.560	53.613
9	224.03	54.30	5.467	56.523	236.14	51.51	2.582	53.635
10	223.96	54.31	5.481	56.537	236.10	51.52	2.591	53.645
Average	223.97	54.31	5.483	56.539	236.18	51.50	2.573	53.626
STD	0.06	0.01	0.01	0.01	0.05	0.01	0.01	0.01

Table 6.1: Timing results for multiple running NPB MG (Class D) on a $8 \times 4 \times 8$ BG/L mesh

Comparing the average value of 10 runs with the one-time data (Table 5.15 last row), we find no difference. The small standard deviation (STD) values are another proof. Thus, it is safe to use one-time data to represent the average value by ignoring fluctuation.

6.4 Performance analysis of 2D wave equation

In this section, we want to answer the following two basic questions: (1) if model can always result in mapping for higher performance than the default mapping and (2) how much improvement. By analyzing the experimental results, we observed as follows:

First, in all experiments, our models always produce mappings requiring shorter communication time than those from a random mapping or the MPI rank order mapping. In many cases, the basic model also works because the problem of the network congestion neglected in that model is considered, implicitly.

Second, the amount of improvement of the model mapping over the default one depends on: the communication patterns, the network diameter, the size of machine and the message being passed, etc. For example, many partitions (from 64-node to 1024-node) with structure $8 \times Y \times Z$, when the application is decomposed to $N_X \times N_Y$ subtasks and N_Y=8, the 3D machine network matched with the 2D application pattern precisely and thus rendered little difference of the default mapping from model mapping (Table 5.3). For 512- and 1024-node cases with reduced network diameter, smaller improvement is observed.

Third, our model can always map 2D applications to 3D computer with significant communication gain. For the case of assigning 32×4 subtasks to a $8 \times 4 \times 4$ BG/L mesh, Figs. 6.4 and 6.5 illustrate the point-to-point communication details in each iteration associated with default mapping and model mapping, where S_R represents sending to the right; I_S_R represents idling after sending to the right; R_L represents receiving from the left, and so on. We suppose the blocking message passing routines `MPI_Send` and `MPI_Recv` are adopted to implement the data transfer between each subtask and its four neighbors: right, left, top and bottom.

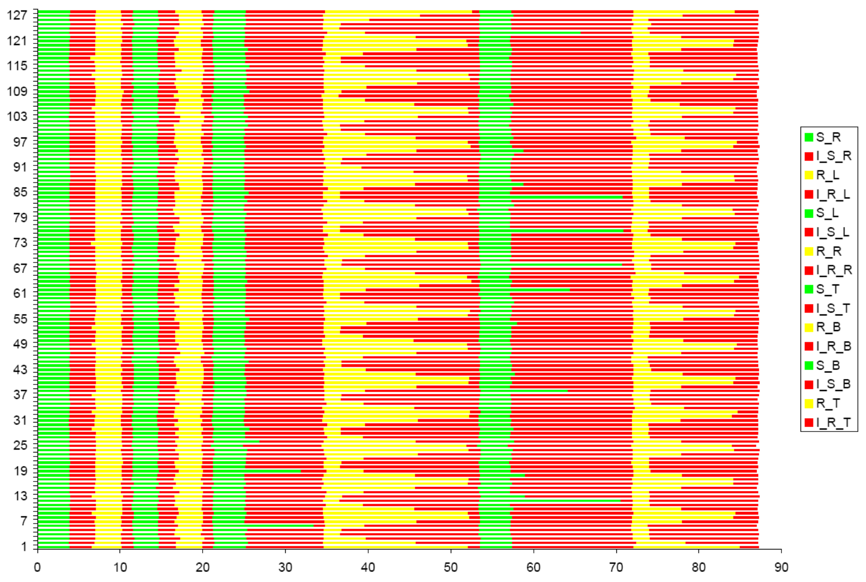


Figure 6.4: The trace of point-to-point communication of default mapping for 2D Wave Equation

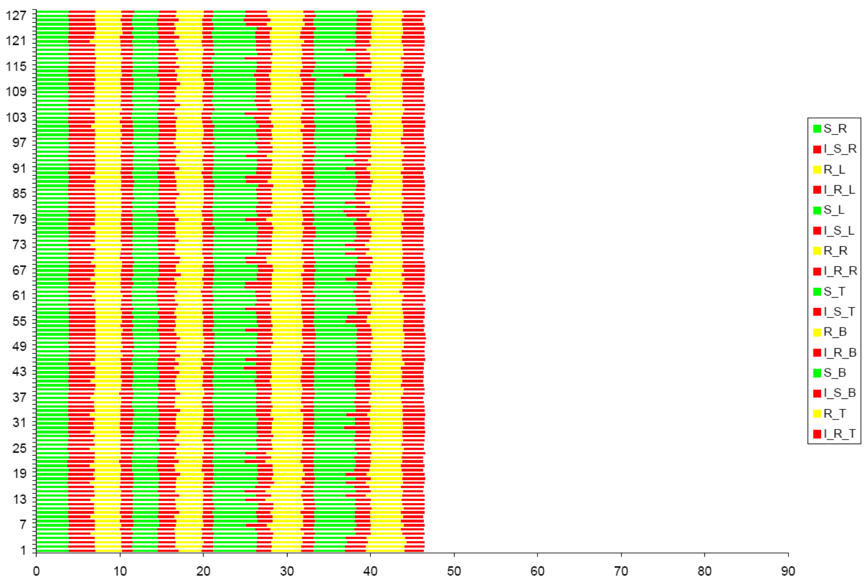


Figure 6.5: The trace of point-to-point communication of model mapping for 2D Wave Equation

It is obvious that data switching for each subtask with its right and left neighbors is similar; all of sending, receiving and idling times are short, taking about 22ms. However, for default mapping, subtasks cause many traffic jams while transferring data to or from its top or bottom neighbor. Since the logical top and bottom neighbors are no longer the physical neighbors, it takes 65ms for these two steps in default mapping. However, the model mapping generates a jam-free environment to all data transfer process, reducing the time to 25ms for these two steps. The IBM routing technology can further explain such improvement [18, 66].

If the point-to-point communication is implemented by other mode, for example, nonblocking message passing routine or the mixed mode, the timing results may vary a little. But the trend that the model mapping reduces the traffic jams for higher performance remains.

When mapping 64×16 subtasks to a $8 \times 8 \times 16$ torus, our models generate excellent improvement. The $8 \times 8 \times 16$ torus can be treated as 16 sheets (along Z dimension) and each sheet consist of 8×8 2D-torus (in the X-Y plane). In each such sheet, the assignment zigzags and winds off the 2D-torus to a 1D-torus, resulting in a precise mapping that matches exactly the 2D-torus communication pattern.

6.5 Performance analysis of SMG2000

We further analyze the timing results in Tables 5.7, 5.8, 5.9, and summarize the data in Table 6.2. The ΔSMG Solve column shows the benchmark timing difference between the model mapping and the default mapping.

The column ΔComm. represents the communication difference while the last column shows the difference of the previous two columns, **ΔΔ=ΔComm.-ΔSMG Solve**. From Table 6.2, we can draw the following conclusions. The overall improvement results mainly from that of the communication. We also observed the overlapping of computation and communication because the numbers in the column ΔΔ show the Δcomm. number is much bigger than the ΔSMG Solve. The communication improvement come mostly from the non-blocking message passing functions: `MPI_Waitall` (Fig. 6.6).

Problem Size $NX \times NY \times NZ$	Processor Topology $PX \times PY \times PZ$	ΔSMG Solve (Sec.)	ΔComm. (Sec.)	ΔΔ (Sec.)
$10 \times 10 \times 10$	$8 \times 4 \times 2$	0	0.042	0.042
$40 \times 40 \times 40$	$8 \times 4 \times 2$	0.26	0.409	0.149
$80 \times 80 \times 80$	$8 \times 4 \times 2$	1.78	2.164	0.384
$100 \times 100 \times 80$	$8 \times 4 \times 2$	2.54	3.126	0.586
$10 \times 10 \times 10$	$8 \times 4 \times 4$	0	0.049	0.049
$40 \times 40 \times 40$	$8 \times 4 \times 4$	0.29	0.455	0.165
$40 \times 40 \times 40$	$8 \times 4 \times 4$	2.09	2.538	0.448
$100 \times 100 \times 80$	$8 \times 4 \times 4$	3.33	3.94	0.61
$100 \times 80 \times 100$	$8 \times 4 \times 4$	3.97	4.644	0.674
$40 \times 40 \times 40$	$8 \times 4 \times 8$	0.01	0.057	0.047
$40 \times 40 \times 40$	$8 \times 4 \times 8$	0.27	0.46	0.19
$80 \times 80 \times 80$	$8 \times 4 \times 8$	2.3	2.754	0.454
$80 \times 100 \times 100$	$8 \times 4 \times 8$	3.26	3.853	0.593

Table 6.2: Improvement analysis of model mapping over the default layout for SMG2000

```
-------------------------------------------- --------------------------------------------
MPI Routine   #calls   avg. bytes  time(sec) MPI Routine   #calls   avg. bytes  time(sec)
-------------------------------------------- --------------------------------------------
MPI_Comm_size  24658          0.0      0.005 MPI_Comm_size  24658          0.0      0.005
MPI_Comm_rank  28230          0.0      0.005 MPI_Comm_rank  28230          0.0      0.005
MPI_Isend     166784       1970.8      2.354 MPI_Isend     166784       1970.8      2.359
MPI_Irecv     166602       2006.1      0.264 MPI_Irecv     166602       2006.1      0.278
MPI_Waitall   180138          0.0     25.411 MPI_Waitall   180138          0.0     20.785
MPI_Barrier        1          0.0      0.000 MPI_Barrier        1          0.0      0.000
MPI_Allgather      1          4.0      0.000 MPI_Allgather      1          4.0      0.000
MPI_Allgatherv     1         28.0      0.000 MPI_Allgatherv     1         28.0      0.000
MPI_Allreduce     15          8.0      0.302 MPI_Allreduce     15          8.0      0.267
-------------------------------------------- --------------------------------------------
total communication time = 28.342 seconds.   total communication time = 23.698 seconds.
total elapsed time       = 101.675 seconds.  total elapsed time       = 97.033 seconds.
top of the heap address  = 480.488 MBytes.   top of the heap address  = 480.488 MBytes.
```
Figure 6.6: Profile segments for SMG2000 benchmark on $8 \times 4 \times 4$ BG/L mesh (Left: Default mapping; Right: Model Mapping)

6.6 Performance analysis of NPB Benchmarks

We analyze the NPB MG timing results in Tables 5.13, 5.14, 5.15, and NPB CG timing results in Tables 5.19, 5.20, 5.21, 5.22, then summarize the data in Tables 6.3 and 6.4, respectively.

The numbers in the column $\Delta\Delta$ is very small, most of them being less than 0.1 second, meaning the ΔComm. number is slightly bigger than the Δtime. The improvement of the benchmark results mainly from the improvement of the communication. The communication improvement comes mostly from the functions: `MPI_Send` and `MPI_Wait` (Figs. 6.7 and 6.8).

Problem Size (Class)	Processor Topology	ΔTime	ΔComm.	ΔΔ
	$PX \times PY \times PZ$	(Sec.)	(Sec.)	(Sec.)
$256 \times 256 \times 256$(A)	$8 \times 4 \times 2$	0.03	0.035	0.005
$256 \times 256 \times 256$(B)	$8 \times 4 \times 2$	0.13	0.133	0.003
$512 \times 512 \times 512$(C)	$8 \times 4 \times 2$	0.44	0.484	0.044
$1024 \times 1024 \times 1024$(D)	$8 \times 4 \times 2$	4.26	4.424	0.164
$256 \times 256 \times 256$(A)	$8 \times 4 \times 4$	0.02	0.025	0.005
$256 \times 256 \times 256$(B)	$8 \times 4 \times 4$	0.08	0.089	0.009
$512 \times 512 \times 512$(C)	$8 \times 4 \times 4$	0.30	0.333	0.033
$1024 \times 1024 \times 1024$(D)	$8 \times 4 \times 4$	2.62	2.736	0.116
$256 \times 256 \times 256$(A)	$8 \times 4 \times 8$	0.01	0.018	0.008
$256 \times 256 \times 256$(B)	$8 \times 4 \times 8$	0.05	0.063	0.013
$512 \times 512 \times 512$(C)	$8 \times 4 \times 8$	0.22	0.237	0.017
$1024 \times 1024 \times 1024$(D)	$8 \times 4 \times 8$	1.99	2.077	0.087

Table 6.3: Improvement analysis of model mapping over the default layout for NPB MG

Problem Size (Class)	Processor Topology	ΔTime	ΔComm.	ΔΔ
	$PX \times PY \times PZ$	(Sec.)	(Sec.)	(Sec.)
$256 \times 256 \times 256$(A)	$8 \times 4 \times 2$	0.16	0.173	0.013
$256 \times 256 \times 256$(B)	$8 \times 4 \times 2$	3.99	4.055	0.065
$512 \times 512 \times 512$(C)	$8 \times 4 \times 2$	9.77	9.895	0.125
$1024 \times 1024 \times 1024$(D)	$8 \times 4 \times 2$	107.29	108.297	1.007
$256 \times 256 \times 256$(A)	$8 \times 4 \times 4$	0.02	0.025	0.005
$256 \times 256 \times 256$(B)	$8 \times 4 \times 4$	1.51	1.524	0.014
$512 \times 512 \times 512$(C)	$8 \times 4 \times 4$	3.10	3.005	-0.095
$1024 \times 1024 \times 1024$(D)	$8 \times 4 \times 4$	42.75	43.182	0.432
$256 \times 256 \times 256$(A)	$8 \times 4 \times 8$	0.04	0.047	0.007
$256 \times 256 \times 256$(B)	$8 \times 4 \times 8$	1.07	1.084	0.014
$512 \times 512 \times 512$(C)	$8 \times 4 \times 8$	2.49	2.528	0.038
$1024 \times 1024 \times 1024$(D)	$8 \times 4 \times 8$	34.80	35.171	0.371
$256 \times 256 \times 256$(A)	$8 \times 8 \times 8$	0.02	0.013	-0.007
$256 \times 256 \times 256$(B)	$8 \times 8 \times 8$	0.68	0.654	-0.026
$512 \times 512 \times 512$(C)	$8 \times 8 \times 8$	1.39	1.406	0.016
$1024 \times 1024 \times 1024$(D)	$8 \times 8 \times 8$	19.10	19.273	0.173

Table 6.4: Improvement analysis of model mapping over the default layout for NPB CG

```
---------------------------------------------   ---------------------------------------------
MPI Routine    #calls    avg. bytes   time(sec)  MPI Routine    #calls    avg. bytes   time(sec)
---------------------------------------------   ---------------------------------------------
MPI_Comm_size       1           0.0      0.000   MPI_Comm_size       1           0.0      0.000
MPI_Comm_rank       1           0.0      0.000   MPI_Comm_rank       1           0.0      0.000
MPI_Send         7368       37698.7      3.931   MPI_Send         7368       37698.7      2.026
MPI_Irecv        7674     1065024.0      0.013   MPI_Irecv        7674     1065024.0      0.009
MPI_Wait         7674           0.0      1.502   MPI_Wait         7674           0.0      0.545
MPI_Bcast           6           8.7      0.000   MPI_Bcast           6           8.7      0.000
MPI_Barrier         6           0.0      0.009   MPI_Barrier         6           0.0      0.000
MPI_Reduce          1           8.0      0.000   MPI_Reduce          1           8.0      0.000
MPI_Allreduce      88          11.6      0.022   MPI_Allreduce      88          11.6      0.001
--------------------------------------------    ---------------------------------------------
total communication time = 5.476 seconds.       total communication time = 2.581 seconds.
total elapsed time       = 56.532 seconds.      total elapsed time       = 53.633 seconds.
top of the heap address  = 125.926 MBytes.      top of the heap address  = 125.926 MBytes.
```

Figure 6.7: Profile segments for NPB MG benchmark(Class_D) on $8 \times 4 \times 8$ BG/L mesh (Left: Default mapping; Right: Model Mapping)

```
--------------------------------------------    ---------------------------------------------
MPI Routine    #calls   avg. bytes   time(sec)  MPI Routine   #calls   avg. bytes   time(sec)
--------------------------------------------    ---------------------------------------------
MPI_Comm_size      1          0.0       0.000   MPI_Comm_size     1          0.0       0.000
MPI_Comm_rank      1          0.0       0.000   MPI_Comm_rank     1          0.0       0.000
MPI_Send       19988      59320.5      10.349   MPI_Send      19988      59320.5       6.559
MPI_Irecv      19988      59320.5       0.022   MPI_Irecv     19988      59320.5       0.019
MPI_Wait       19988          0.0      12.645   MPI_Wait      19988          0.0       6.543
MPI_Barrier        1          0.0       0.000   MPI_Barrier       1          0.0       0.000
MPI_Reduce         1          8.0       0.000   MPI_Reduce        1          8.0       0.000
--------------------------------------------    --------------------------------------------
total communication time = 23.017 seconds.      total communication time = 13.122 seconds.
total elapsed time       = 104.421 seconds.     total elapsed time       = 94.520 seconds.
top of the heap address  = 39.551 MBytes.       top of the heap address  = 39.516 MBytes.
```

Figure 6.8: Profile segments for NPB CG benchmark(Class_C) on $8 \times 4 \times 2$ BG/L mesh (Left: Default mapping; Right: Model Mapping).

The corresponding performance improvements for NPB CG (Class A-D) tested on 64-, 128-, 256-, and 512-node machine can be illustrated in Figs. 6.9 and 6.10, showing the extents of our models' improvement of system performance for different cases.

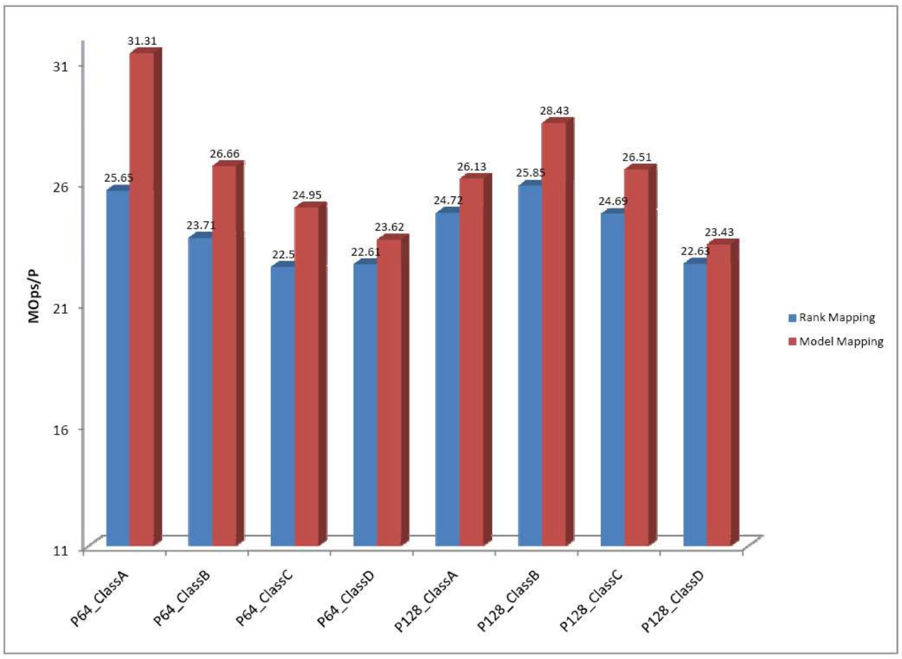


Figure 6.9: NPB CG performance improvements for 64- and 128- node

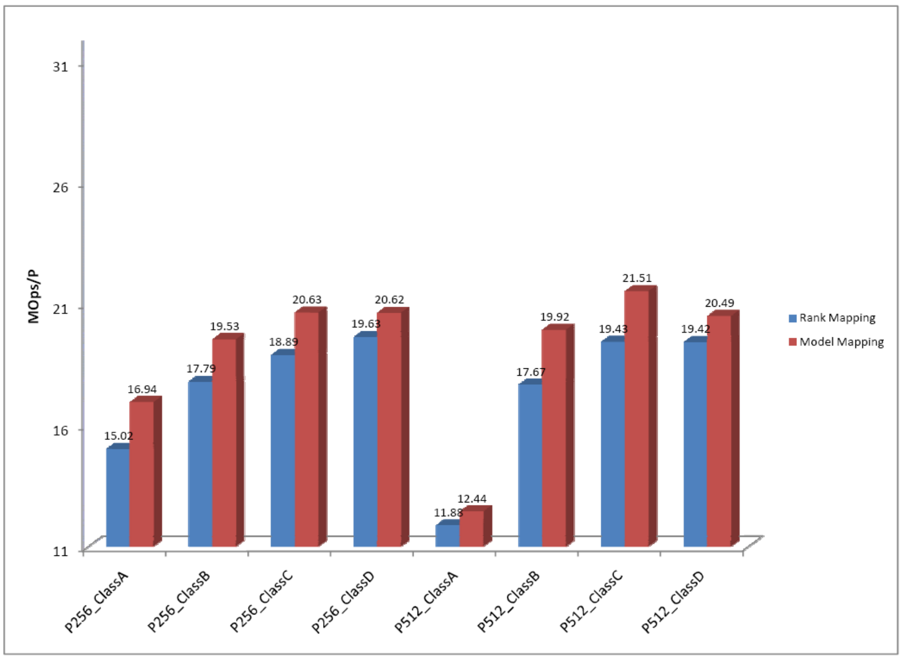


Figure 6.10: NPB CG performance improvements for 256- and 512- node

Chapter 7

Conclusions

In this chapter, we will summarize our research and discuss the next task in the future.

7.1 Conclusions

Our work contributes to reduce the complexity of parallel programming and to achieve high performance of ultra-scalable computer systems. In many cases, we can expect a much higher performance and parallel efficiency from model mapping than from the default MPI rank mapping. The method works well in 2D hyperbolic equation system, finite element discretizations of the diffusion equation and 3D Poisson PDE application.

Task mapping is sophisticated due to huge searching space and its intricate dependence on both the applications and the platforms. To solve this problem, we have proposed two general static mapping models to help optimize the assignment of applications onto a large heterogeneous parallel computer. The characteristics of the application and the computer are abstracted as proper matrixes. Every time a new application is introduced, we generate a new demand matrix while a parallel computer network is quantified as a supply matrix. The supply matrix is measured by MPI latency matrix to avoid inaccuracy of previous measurement such as hop or its variant. Solutions of our models can achieve the near-optimal performance of the application on the given platform efficiently.

Without an appropriate assignment, even an ultra-scalable supercomputer system such as Blue Gene/L with huge computing power will only realize a small portion of its potential [33]. Our models are great effective tools to avoid this kind of power waste. The models are verified by achieving near-optimal values when assigning 2D and 3D communication patterns occurring in solution of partial differential equations onto 3D mesh and torus BG/L networks. The timing results from a series of benchmarks and scientific applications suggests convenient extendability of our models to actual applications for achieve competitive improvement.

7.2 Future Work

We believe our models can be generalized as a general task assignment optimization tool for producing good mappings from arbitrary problems to parallel computers with reasonable complexities. Profiling tool will add the functioning and power of our static models to enable them to be an efficient and practical for wider selection of problems and more complex computer systems. A natural next task would be to refine our models to incorporate the profiling and tracing tools for additional values, especially on the field that generates more efficient task assignment for the pre-exist MPI codes. Our final goal is in the automatic parallelization, the parallel code generation starting from a sequential program for scalability, a goal that has attracted but trapped several generations of researchers .

To reduce the difference between the theoretical system and a real situation, we need to develop more accurate models to formulate the real computation and communication in a ultra-scalable supercomputer system. In general, we can modify the objective function and assumptions appropriately to handle the global collective communication and other factors.

Although many issues have been discussed, we are still left with many promising fields to discover since there are so many aspects in task mapping problem: the application, the objective function and the metric, the heuristic algorithm, the problem size, the computer size, the computing mode, the generating method and measurement of demand and supply matrixes, the utilizing of the profiling tool and the tracing tool, etc. It is impossible to examine every feasible combination in this thesis.

Further tests of our models on other realistic problems such as CPMD, other important benchmarks and the applications, other compute mode such as VN mode in BG/L system, and on larger sizes of processors and on more available supercomputers, Blue Gene/P, Cray XT4, and SGI Altix supercomputers are very desirable extension of our work.

Bibliography

[1] IBM POWER Systems Overview.
URL: https://computing.llnl.gov/tutorials/ibm_sp.

[2] Kit for Objective Judgement and Knowledge-based Detection of Performance Bottlenecks.
URL: http://www.fz-juelich.de/jsc/kojak/.

[3] mpiP: Lightweight, Scalable MPI Profiling.
URL: http://mpip.sourceforge.net/.

[4] Obtain Detailed Information from Raw Performance Traces.
URL: http://www.cepba.upc.es/paraver.

[5] Performance Application Programming Interface.
URL: http://icl.cs.utk.edu/papi/.

[6] Seawulf Cluster.
URL: http://www.sunysb.edu/seawulfcluster.

[7] Tuning and Analysis Utilities.
URL: http://www.cs.uoregon.edu/research/tau/home.php.

[8] Vampir - Performance Optimization.
URL: http://www.vampir.eu/.

[9] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban and C.-L. Wang. Heterogeneous Computing: Challenges and Opportunities. *Computer*, 26(6):18–27, 1993.

[10] A. Billionnet, M. C. Costa and A. Sutter. An Efficient Algorithm for a Task Allocation Problem. *J.ACM.*, 39(3):502–518, 1992.

[11] A. K. Gupta and G. W. Greenwood. Static Task Allocation Using (μ, λ) Evolutionary Strategies. *Information Sciences*, 94(1-4):141–150, 1996.

[12] B. Andresen. Finite-Time Thermodynamics and Simulated Annealing. *Entropy and Entropy Generation*, 18:111–127, 1996.

[13] B. E. Smith and B. Bode. Performance Effects of Node Mappings on the IBM BlueGene/L Machine. In *Euro-Par 2005 Parallel Processing*, volume 3648.

[14] B. Indurkhya, H. S. Stone, and L. Xi-Cheng. Optimal partitioning of randomly generated distributed programs. *IEEE Trans. Software Engrg.*, 12(3):483–495, 1986.

[15] S. H. Bokhari. On the mapping problem. *IEEE Trans. Comput.*, 30(3):207–214, 1981.

[16] T. Bultan and C. Aykanat. A New Mapping Heuristic Based on Mean Field Annealing. *Journal of Parallel and Distributed Computing*, 16(4):292–305, 1992.

[17] C. E. Houstis. Allocating Modules to Processors in a Distributed System. *IEEE Transactions on Software Engineering*, 16(7):699–709, 1990.

[18] C. P. Sosa. *IBM System Blue Gene Solution: Blue Gene/P Application Development.* IBM, 2007.

[19] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga. The NAS Parallel Benchmarks. Technical report, NASA Advanced Supercomputing Division, Ames Reserach Center, Moffett field, CA 94035-1000, 1994.

[20] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical report, NASA Advanced Supercomputing Division, Ames Reserach Center, Moffett field, CA 94035-1000, 1995.

[21] D. E. Culler, J. P. Singh and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann Publication, 1998.

[22] D. Fernández-Baca. Allocating Modules to Processors in a Distributed System. *IEEE Trans. on Software Engineering*, 15(11):1427–1436, 1989.

[23] D. J. Ram, T. H. Sreenivas and K. G. Subramaniam. Parallel Simulated Annealing Algorithms. *Parallel and Distributed Computing*, 37:207–212, 1996.

[24] D. M. Nicol. Rectilinear Partitioning of Irregular Data Parallel Computation. *Journal of Parallel and Distributed Computin*, 23(2):119–134, 1994.

[25] D. R. Greening. Parallel simulated annealing techniques. *Physica D*, 42(1-3):293–306, 1990.

[26] D.-T. Peng, K. G. Shin and T. F. Abdelzaher. Assignment and Scheduling Communicating Periodic tasks in Distributed Real Time Systems. *IEEE Transactions on Software Engineeering*, 23(12):745–758, 1997.

[27] E. Cantú-Paz. A Survey of Parallel Genetic Algorithms. *Technical Report IlliGAL #97003*, 1997.

[28] E.-G. Talbi and T. Muntean. General Heuristics for the Mapping Problem. *World Transputer Conf*, 1993.

[29] F. Ercal, J. Ramanujam and P. Sadayappan. Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning. *Parallel and Distributed Computing*, 10(1):35–44, 1990.

[30] F. W. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publisher, 1997.

[31] Flynn, M. J. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computing*, C(21):948–960, 1972.

[32] G. Almási, C. Archer, J. G. Castaños, M. Gupta, X. Martorell, J. E. Moreira, W. Gropp, S. Rus and B. Toonen. MPI on BlueGene/L: Designing an Efficient General Purpose Messaging Solution for a Large Cellular System. In *PVM/MPI*, pages 352–361, 2003.

[33] G. Almasi, G. Bhanot, A. Gara, M. Gupta, J. Sexton, B. Walkup, V. V. Bulatov, A. W. Cook, B. R. de Supinski, J. N. Glosli, J. A. Greenough, F. Gygi, A. Kubota, S. Louis, T. E. Spelce, F. H. Streitz, P. L. Williams, R. K. Yates, C. Archer, J. Moreira and C. Rendleman. Scaling physics and material science applications on a massively parallel Blue Gene/L system. In *ICS '05: Proceedings of the 19th Annual International Conference on Supercomputing*, pages 246–252, 2005.

[34] G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J. C. Sexton, and R. Walkup. Optimizing Task Layout on the Blue Gene/L Supercomputer. *IBM Journal of Research and Development*, 49(2/3):489–500, 2005.

[35] G. Lakner, E. W. Barnard and G. L. Mullen-Schultz. Blue Gene/L: System Administration. Technical report, April 2007.

[36] G. Lakner, I-H. Chung, G. Cong, D. Klepacki, C. Pospiech, S. R. Seelam and H.-F. Wen. *IBM System Blue Gene Solution:High Performance Computing Toolkit for Blue Gene/P.* IBM, 2007.

[37] G. L.Mullen-Schultz. *Blue Gene/L:Application Development.* IBM, 2005.

[38] G. W. Greenwood, A. Gupta and K. McSweeney. Scheduling Tasks in Multiprocessor Systems Using Evolutionary Strategies. In *Proc. of 1st IEEE International Conference on Evolutionary Computation*, pages 345–349, 1994.

[39] Gary Mullen-Schultz. *Blue Gene/L: Performance Analysis Tools.* IBM, 2006.

[40] H. Chen, N. S. Flann and D. W. Watson. Parallel genetic simulated annealing: A massively parallel SIMD approach. *IEEE Transactions on Parallel and Distributed Computing*, 9(2):126–136, 1998.

[41] H. J. Siegel and S. Ali. Techniques for mapping tasks to machines in heterogeneous computing systems. *System Architecutre*, 46:627–639, 2000.

[42] H. Kasahara and S. Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. Comput.*, 33(11):1023–1029, 1984.

[43] H. M. Mathis, J. D. McCalpin and J. Thomas. IBM eServer pSeries 655 for HPC and BI White Paper. Technical report, January 2004.

[44] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. on Software Engineering*, 3(1):85–93, 1977.

[45] H.-U. Heiss. Mapping Tasks to Processors at Run-time. In *Proc. Int. Symp. On Comp and Information Sciences*, 1992.

[46] H.-U. Heiss and M. Dormanns. Mapping Tasks to Processors with the Aid of Kohonen Networks. In *Proc. High Performance Computing Conference '94*, pages 133–143, 1994.

[47] H.-U. Heiss and M. Dormanns. Partitioning and mapping of parallel programs by self-organization. *Concurrency - Practice and Experience*, 8(9):685–706, 1996.

[48] H. Yu, I-H. Chung and J. Moreira. Topology Mapping for Blue Gene/L Supercomputer. In *Proc. of ACM/IEEE conference on Supercomputing '06*, pages 52–52, 2006.

[49] I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro. Improving search by incorportating evolution principles in parallel tabu search. In *Proc. 1994 IEEE Conference on Evolutionary Computation*, volume 2, pages 823–828, 1994.

[50] I-H. Chung, R. E. Walkup, H.-F. Wen and H. Yu. MPI Performance Analysis Tools on Blue Gene/L. In *Proc. of Supercomputing, 2006*, pages 16–16, Nov. 2006.

[51] I-H. Chung, R. E. Walkup, H.-F. Wen, H. Yu. A study of MPI performance analysis tools on Blue Gene/L. In *Proc. of IEEE International Parallel and Distributed Processing symposium*, pages 25–29, April 2006.

[52] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, 1975.

[53] J. L. Träff. Implementing the MPI Process Topology Mechanism. In *Proc. of ACM/IEEE conference on Supercomputing '02*, pages 1–14, 2002.

[54] J. Milano, G. L. Mullen-Schultz, G. Lakner. *Blue Gene/L: Hardware Overview and Planning.* IBM, 2006.

[55] K. Anstreicher, N. Brixius, J.-P. Goux and J. Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91(3):563–588, 2002.

[56] K. Kennedy and U. Kremer. Automatic data layout for High Performance Fortran. In *Proc. of the 1995 ACM/IEEE Supercomputing Conference*, 1995.

[57] L. V. Kalé, B. Ramkumar, A. B. Sinha and A. Gursoy. The CHARM Parallel Programming Language and System: Part I – Description of Language Features. *Parallel Programming Laboratory Technical Report #95-02*, 1994.

[58] L. V. Kalé, B. Ramkumar, A. B. Sinha and V. A. Saletore. The CHARM Parallel Programming Language and System: Part II – The Runtime system. *Parallel Programming Laboratory Technical Report #95-03*, 1994.

[59] M. Affenzeller and R. Mayrhofer. Generic Heuristics for Combinatorial Optimization Problems. In *Proc. of the 9th International Conference on Operational Research*, pages 83–92, 2002.

[60] M. E. Crovella, M. Harchol-Balter and C. D. Murta. Task assignment in a distributed system: Improving performance by unbalancing load. In *Proc. of ACM Conf. on Measurement and Modeling of Comp. Sys.*, pages 268–269, 1998.

[61] M. Fleischer. Simulated annealing: past, present, and future. In *Proc. of the 27th Conference on Winter simulation*, pages 155–161, 1995.

[62] M. G. Norman and P. Thanisch. Models of Machines and Computation for Mapping in Multicomputers. *ACM Computing Surveys*, 25(3):263–302, 1993.

[63] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 1995.

[64] M. Srinivas and L. M. Patnaik. Genetic algorithms: A Survey. *IEEE Computer*, 27(6):17–26, 1994.

[65] N. Allsopp, A. Tabary, J. Follows, P. Vezolle, M. Hennecke, H. Reddy, F. Ishibashi, C. Sosa, M. Paolini,S. Prakash, D. Quintero and O. Lascu. *Unfolding the IBM eServer Blue Gene Solution.* IBM, 2005.

[66] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao and P. Vranas. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49(2/3):265–275, 2005.

[67] P. A. Boyle, C. Jung and T. Wettig for the QCDOC Colaboration. The QCDOC supercomputer: Hardware, Software and Performance. In *Proc. of CHEP03*, pages 24–28, 2003.

[68] P. A. Boyle, D. Chen, N. H. Christ, M. Clark, S. D. Cohen, C. Cristian, Z. Dong, A. Gara, B. Jo´, C. Jung, C. Kim, L. Levkova, X. Liao, G. Liu, R. D. Mawhinney, S. Ohta, K. Petrov, T. Wettig, and A. Yamaguchi. Hardware and software status of QCDOC. In *Proc. of of the 21st International Symposium on Lattice Field Theory*, pages 838–843, 2003.

[69] P. M. Pardalos, F. Rendl and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In *Proc. of the DIMACS*

Workshop on Quadratic Assignment Problems, volume 16, pages 1–41, 1994.

[70] R. D. Falgout P. N. Brown and J. E. Jones. Semicoarsening Multigrid on Distributed Memory Machines. *SIAM Journal on Scientific Computing*, 21(5):1823–1834, 2000.

[71] P. S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufmann, 1996.

[72] P. Shroff, D. W. Watson, N. S. Flann and R. F. Freund. Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. In *Proc. of 5th IEEE Heterogeneous Computing Workshop (HCW'96)*, pages 98–104, 1996.

[73] R. H. J. M. Otten and L. P. P. P. van Ginneken. *The Annealing Algorithm.* Kluwer Academic Publishers, 1989.

[74] R.F. Freund, T. Kidd, D. Hensgen and L. Moore. SmartNet: A scheduling framework for metacomputing. *Second International Sysposium on Parallel Architectures, Algorithms, and Networks*, pages 514–521, 1996.

[75] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. on Pattern Anal. Machine Intell.*, 6:721–741, 1984.

[76] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[77] S.-Y. Lee and J.K. Aggarwal. A mapping strategy for parallel computing. *IEEE Trans. Comput.*, C-36(4):433–442, 1987.

[78] T. Agarwal, A. Sharma, L. V. Kalé. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proc. of IEEE International Parallel and Distributed Processing symposium*, pages 25–29, April 2006.

[79] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems. In *Proc. Eight IEEE Heterogeneous Computing Workshop (HCW'99)*, pages 15–29, 1999.

[80] T. F. Adbelzaher and K. G. Shin. Period-based load partitioning and assignment for large real-time applications. *IEEE Trans. On Computers*, 49(1), 2000.

[81] T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake and C. V. Packer. BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION. In *Proc. of International Conference on Parallel Processing*, pages 11–14, 1995.

[82] The Blue Gene/L Team. An Overview of the BlueGene/L Super Computer. In *Proc. of Super Computing 2002*, 2002.

[83] W. Chemij. Parallel Computer Taxonomy.
URL: http://www.gigaflop.demon.co.uk/comp/chapt7.htm.

[84] X. Martorell, N. Smeds, R. Walkup, J. R. Brunheroto, G. Almási, J. A. Gunnels, L. DeRose, J. Labarta, F. Escalé, J. Giménez, H. Servat and J. E. Moreira. Blue Gene/L performance tools. *IBM Journal of Research and Development*, 49(2/3):407–424, 2005.

[85] Y. Deng and A. Korobka. Performance of a supercomputer Built with Commodity Components. *Parallel Computing*, 27(1-2):97–108, 2001.

[86] Y. Nourani and B. Andresen. A comparison of simulated annealing cooling strategies. *J. Phys. A: Math. Gen.*, 31(41):8373–8385, 1998.

[87] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 1992.