# MPIPP: An Automatic Profile-guided Parallel Process Placement Toolset for SMP Clusters and Multiclusters

Hu Chen
Intel China Research Center
hu.tiger.chen@intel.com

Wenguang Chen
Tsinghua University
cwg@tsinghua.edu.cn

Jian Huang, Bob Robert H. Kuhn
Advanced Parallel Software Platforms, Intel Corp.
{eric.huang,bob.kuhn}@intel.com

## ABSTRACT

SMP clusters and multiclusters are widely used to execute message-passing parallel applications. The ways to map parallel processes to processors (or cores) could affect the application performance significantly due to the non-uniform communicating cost in such systems. It is desired to have a tool to map parallel processes to processors (or cores) automatically.

Although there have been various efforts to address this issue, the existing solutions either require intensive user intervention, or can not be able to handle the situation of multiclusters well.

In this paper, we propose a profile-guided approach to find the optimized mapping automatically to minimize the cost of point-to-point communications for arbitrary message passing applications. The implemented toolset is called MPIPP (MPI Process Placement toolset), and it includes several components:
1) A tool to get the communication profile of MPI applications
2) A tool to get the network topology of target clusters
3) An algorithm to find optimized mapping, which is especially more effective than existing graph partition algorithms for multiclusters.

We evaluated the performance of our tool with the NPB benchmarks and three other applications in several clusters. Experimental results show that the optimized process placement generated by our tools can achieve significant speedup.

## Categories and Subject Descriptors

D.4.4 [**Communications Management**]: Network communication

## General Terms

Experimentation, Performance, Algorithms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICS06, June 28-30, Cairns, Queensland, Australia.
Copyright (c) 2006 ACM 1-59593-282-8/06/0006 ...$5.00.

## Keywords

process placement, graph mapping, graph partitioning, cluster, parallel computing

## 1. INTRODUCTION

SMP(Symmetric Multi-Processor) clusters and multiclusters are widely used to execute message-passing parallel applications. The ways to map parallel processes to processors (or cores) could affect the application performance significantly due to the non-uniform communicate cost in such systems. For example, in an SMP cluster, intra-node communication is usually much faster than inter-node communication. In multiclusters, the bandwidth among nodes inside a single cluster is normally much higher than the bandwidth between two clusters. Hence, it is very important to find optimized mapping for parallel processes.

Various approaches have been proposed to optimize the communication performance of MPI (Message Passing Interface) in SMP clusters and multiclusters. MagPIe[13] optimizes the collective communication primitives of MPICH [7] for clustered wide area systems. Other implementations [25, 22] of MPI use graph partitioning algorithms to optimize the point-to-point communication performance. MPI/ SX [25] uses a graph partitioning algorithm to optimize the implementation of MPI_Cart_create() and MPI_Graph_create() with the user-supplied communication topology information of applications. MPICH VMI[22] proposed the profile-guided approach to get the application communication topology, and also proposed to use general graph partitioning algorithms to find optimized mapping from parallel processes to processors. However, MPICH VMI requires users to provide the topology information of the target machine.

The contribution of this paper can be summarized as follows:

- We propose a fully automatic scheme for optimized parallel process placement in SMP clusters and multiclusters without users' knowledge on either applications or target systems. The proposed scheme includes three major components:
  1. A tool to get communication profile of MPI applications
  2. A tool to get the network topology of target clusters
  3. An algorithm to find optimized mapping

- We propose a mapping algorithm, which outperforms the existing graph-partition algorithms for multiclusters.
- We performed extensive experiments with the NPB benchmarks and three other applications in several clusters and proved the effectiveness of our scheme.

The rest of the paper is organized as follows: Section 2 explains the problem that we are addressing. Section 3 describes our new graph mapping algorithm. The pseudo-code of the algorithm is given in the appendix. Section 4 covers the implementation of the profile-guided process placement mechanism. The experimental environments, test results, and data analysis are presented in Section 5. Section 6 summarizes and describes future work.

# 2. BACKGROUND AND RELATED WORKS

Much research effort was spent to improve the MPI communication performance by taking advantage of the different communication costs in a multiple-hierarchy environment. The related works comprise:

1. Approaches to optimize the MPI communication performance
2. Algorithms to map parallel processes to processors.

## 2.1 Approaches to Optimize the MPI Communication Performance

A) Optimizing communications in transport layer
The MPI implementation has used different devices to optimize the communication performance over different transport protocols. For example, LAMMPI or Intel MPI Library can use shared memory for the intra-node communication, while using other protocols for inter-node communication.

B) Optimizing the collective communication primitives
Most of collective communications are implemented by a bunch of point-to-point messages. MagPIe[13] optimizes the collective communication primitives of MPICH for clustered wide area systems by minimizing the amount of data communicated over the slow wide area links.

C) Optimizing the point-to-point communications
Unlike the collective communications, optimizing the point to point communications requires the communication topology information of applications as well as the topology information of target cluster systems.

The MPI standard provides primitives such as MPI Cart create and MPI Graph create for users to provide the communication topology information. MPI/ SX [25] optimizes these primitives by virtually ranking the MPI process so that more communications in the topology are conducted in one SMP node.

The other way to get the communication topology information is by profiling. The Intel Trace Collector [9] and some MPI implementations can provide trace data for MPI messages. From the statistical information, we can analyze the communication pattern of the application, and optimize it. MPICH VMI [22] and ScaliMPI [24] fall into this category.

Our work is profile-guided as well. The major differences are that our solution is fully automatic and that our algorithm addresses the multi-hierarchy interconnect effectively.

## 2.2 Algorithms to Map Parallel Processes to Processors

The goal is to improve the communication performance of the applications by aligning their communication with the clusters interconnect hierarchy. Hence, two models are needed: one that describes the communication behavior and the other that describes the interconnect hierarchy. We describe the MPI communication of an application with a communication graph (CG). The interconnect hierarchy that connects the processors in a cluster is modeled with a system topology graph (TG).

A) Working with system topology graph only
MagPIe [13] is a network performance-aware collective communication optimization for clustered wide area systems. It discovers the system topology with LogP model [3], only the broadcast trees for the collective communications were be optimized.

B) Working with communication graph only
In general, this problem could be solved by graph partitioning: Given a graph G with costs as edge weights and sizes as vertex weights, partition the vertices of G into k sub-graphs of equal sizes such that the sum of the edge weights crossing sub-graph boundaries is minimized.

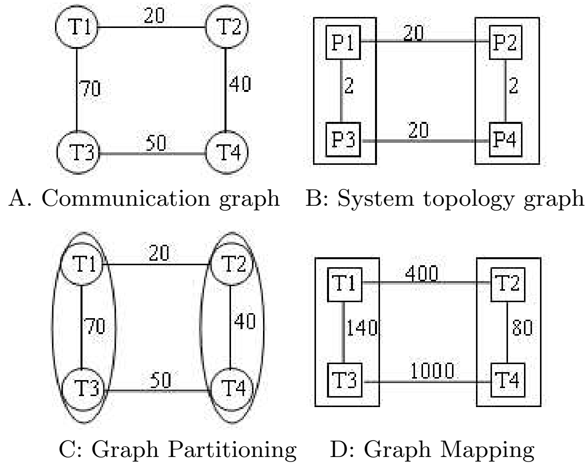
The graph partitioning algorithm [16, 11, 12, 27] is widely used in the MPI performance optimization. It requires the communication graph to describe the communication behavior of the program, which could be derived from trace or profile or from user input. Many MPI implementations, such as MPICH VMI[22] and ScaliMPI [24] provide the interface for this kind of algorithms. MPI/SX [25] uses a graph partitioning algorithm to optimize the MPI implementation of MPI Cart create and MPI Graph create with the user-supplied communication topology information.

There are a few public suites available for the implementation of graph partitioning algorithms, for instance, METIS[11], Chaco[8], Jostle[26] and Party[21].

C) Working with both the communication graph and the system topology graph
The problem is modeled as graph mapping problem: Given a graph G with costs as edge weights and another graph T also with costs as edge weights, map the vertices of G to the graph T with equal size such that the value of a cost function F is minimized. Here F calculates the new cost of the edges when G is mapped to T.

In [17], a basic graph mapping scheme was described. More sophisticated algorithms [15, 18, 14, 23] were proposed for the task assignment problem afterwards. These works help to dispatch workloads to distributed systems and resolve the dependences among them.

In this paper we implement a new graph mapping algorithm. The communication graph is obtained from profile collected by ITC and we assume that workload balance is not an issue. In the process of mapping a CG to a TG, we




C: Graph Partitioning D: Graph Mapping

Figure 1: Profile-guided parallel processes placement, where the weight of the edge is communication cost between two vertices and the smaller the better. ( The communication graph $A$ describes the communication behavior of the application; the system topology graph $B$ describes connection overhead/bandwidth between processors in the SMP nodes; $C$ shows sub-groups after min-cut graph partitioning, which can direct the placement; $D$ is the optimal mapping result from the communication graph $A$ to the system topology graph $B$. Here the cost function is a simple multiplication .)

hope to achieve the result that min-sums the total communication cost. For comparison purposes, a graph partitioning algorithm is included in our study as well.

An example of graph $G$ is in Figure 1.A, and the graph $T$ is in Figure 1.B. The result of partitioning the graph into 2 sub-graphs is shown in Figure 1.C. Moreover, the mapping result from graph $G$ to graph $T$ is in Figure 1.D.

## 3. THE GRAPH MAPPING ALGORITHM

In most MPI applications, the process number corresponds to the CPU number. We omit the workload balance issue in our algorithm to simplify the model.

### 3.1 Problem Definition

A formal description of the application communication graph $G_p$ is given:

$G_P = \langle V_P, E_P, W_E \rangle$, where:

- $V_P$ is a set of application processes, $V_P = \{v_i | v_i \in G_P\}$, $v_i$ is a vertex in the graph;
- $E_P$ is a set of communications between $v_i$ and $v_j$, $E_P = \{e_{i,j} | v_i, v_j \in V_P\}$, $e_{i,j}$ is the edge between two vertices;
- $W_E$ is the communication cost of the $E_P$, $W_E = \{w_{i,j} | e_{i,j} \in E_P\}$, $w_{i,j}$ is the weight of the edge.

Besides the definition of the application communication graph, we model the system topology with the graph $T$:

$T_N = \langle M_N, D_N \rangle$, where:

- $M_N$ is a set of machine nodes or processors, $M_N = \{m_i | m_i \in T_N\}$, $m_i$ is vertex in the graph;
- $D_N$ is a set of communication cost between $m_i$ and $m_j$, $D_N = \{d_{i,j} | m_i, m_j \in T_N\}$, $d_{i,j}$ is the distance between two vertices.

The definition of graph mapping from $G$ to $T$ is:

$G'_{PN} = \langle V'_{PN}, E'_{PN}, W'_{E'} \rangle$, where:

- $V'_{PN}$ is a set of application processes after being mapped to the set of machine nodes or processors, $V'_{PN} = \{v'_{ik} = v_i \to m_k | v_i \in V_P, m_k \in M_N\}$, $v'_{jk}$ is the vertex after mapping;
- $E'_{PN}$ is a set of communications between $v'_{ik}$ and $v'_{jl}$, $E'_{PN} = \{e'_{ik,jl} | v_i, v_j \in V_P, m_k, m_l \in M_N\}$, $e'_{ik,jl}$ is the edge between two vertices after mapping;
- $W'_{E'}$ is the communication cost after mapping, $W'_{E'} = \{f(w_{i,j}, d_{k,l} | e'_{ik,jl} \in E'_{PN}\}$, $f(w,d)$ is the mapping function that calculates the cost when the communication edge is mapped to the topology edge. It could be different equations for different applications.

The objective function of graph mapping is minimizing the real communication cost via intelligent mapping the communication graph to the system topology graph.

$$MAP_{G,T} = \{G'_{PN} | \min(\sum W'_E)\}$$

### 3.2 New Heuristic in Graph Mapping

As described earlier, two graphs are generated from user-input or trace analysis: one describes the application communication pattern (CG); the other describes the system communication topology (TG). We assume the numbers of vertices in these two graphs are equal, which means we can ignore the problem of load-balancing. i.e., the mapping of vertices between two graphs is one to one:

$$V'_{PN} = \{v'_{ik} = v_i \leftrightarrow m_k | v_i \in V_P, m_k \in M_N\}$$

Then our new mapping algorithm could map the communication graph to the system topology graph.

The graph partitioning and graph mapping problems have been proved to be NP-hard[5]. We adopt the heuristic k-way graph partitioning algorithm proposed in[16] as the basis to solve the graph mapping problem with a new objective function:

$$\min \sum W'_E = \min \left( \sum_{v'_{ik}, v'_{jl} \in V'_{PN}} f(w_{i,j}, d_{k,l}) \right)$$

A basic operation in the proposed heuristic is to exchange a pair of process-node mappings and calculate the change of the value of the objective function, which is defined as the gain function $g$:

$$g(v'_{ik}, v'_{jl}) = \sum_{\substack{m \neq i,j,\\ v'_{mn} \in V'_{PN}}} f(w_{i,m}, d_{k,n}) + f(w_{j,m} + d_{l,n}) - \sum_{\substack{m \neq i,j,\\ v'_{mn} \in V'_{PN}}} f(w_{i,m}, d_{l,n}) + f(w_{j,m} + d_{k,n})$$

where the second sum is the communication cost between other vertices and the two vertices $v'_{ik}$ and $v'_{jl}$ after exchanging the mapping of $v_i$ and $v_j$ . The first sum is the original communication cost before the exchanging.

The algorithm starts with a random mapping scheme to avoid being stuck in a poor local maximum. Each pass improves the quality of the mapping scheme. If no positive gain is available, the algorithm stops.

In each pass, the algorithm first calculates the matrix $g(v_{ik}^{'}, v_{jl}^{'})$ for each pair of vertices in the communication graph. It then selects the pair with max gain. Recalculate the matrix $g(v_{ik}^{'}, v_{jl}^{'})$ for the rest of vertices until all of the vertices are selected. Choose the pairs of vertices in the selection to exchange, if the sum of the gains is positive and maximum. Exchange those chosen pairs, loop to next pass. The detail pseudo-code is in the appendix.

The following points are worth noting:

1. The proposed algorithm is more efficient than those simple pair-wised exchange algorithms. In each pass, the algorithm exchanges the maximal number of the pairs of which the sum of the gains is positive and maximum.

2. To avoid getting the local optimum for the heuristic, the algorithm chooses random mapping as the start. To improve the quality of the heuristic, we can start the mapping algorithm with several random initializations. In addition, multiple execution with different initial start mapping could be performed in parallel.

3. The complexity of the algorithm is dominated by the calculation of the gain, which takes $O(N^2)$ time. Each pass calculated the gain function for $N$ times, so the complexity for each pass is $O(N^3)$. The execution time of the mapping algorithm depends on the complexity of the graph CG and TG, i.e., how many passes to get the optimized solution. In section 5.3, we give the execution time of the mapping algorithm and analyze its scalability.

# 4. IMPLEMENTATION OF THE PROFILE GUIDED PROCESSES PLACEMENT MECHANISM

The processes placement mechanism was implemented as an experimental component of the Intel Cluster Tool Kit[9]. In Figure 2, the communication graph can be parsed and abstracted from the trace file of the application after its execution with Intel Trace Collector[9] or Intel MPI library[10]. The system topology graph can be obtained using our MPI parallel ping-pong tool or other topology discovery mechanisms, in addition to using the administrator's input.

## 4.1 Application Communication Graph Generator

Message count and message volume are used to represent the communication cost between MPI processes to form the application communication graph. Message count is a good metric for short message dominant applications because short messages are more sensitive to the communication overhead occurred each time regardless the message size. Message volume is more proper to represent the communication cost of large message dominated applications because it is more sensitive to the total size of messages.

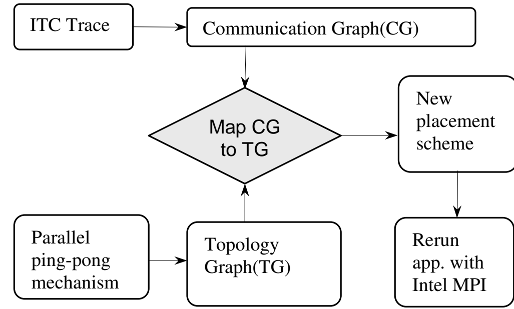



Figure 2: Workflow of the profile-guided parallel processes placement.

In our implementation, we first calculate the median message size of the application and compare it with a threshold to determine whether it is dominated by large messages or small ones. Then choose either message count or message volume accordingly.

It should be noted that the threshold to distinguish small and large messages varies on different MPI implementations. In our implementation, it is obtained by measuring the latency and bandwidth of ping-pong messages with various sizes.

## 4.2 Exploration of Interconnect Topology

The topology graph describes the interconnect hierarchy of a cluster. This hierarchy has a large impact on the applications' communication behavior. We use parameters, such as gap and overhead, from the LogP model [20] to weight the connections in the cluster topology graph. *Overhead* dominates the small message transmission time, while gap, the reciprocal of bandwidth, determines the transmission time of bulk messages. These parameters can be obtained from

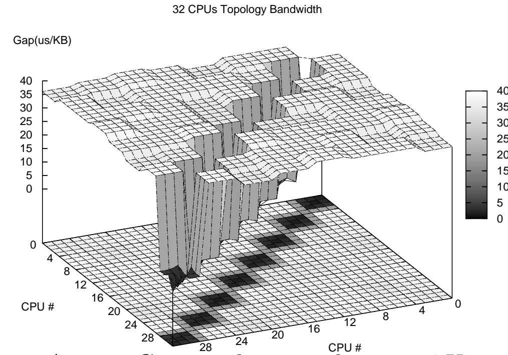



Figure 3: A 2-tier Gap topology graph in a 8x4 Xeon cluster with gigabit Ethernet connections.

the technical specification of a cluster or from conducting a simple ping-pong test. We implemented a MPI parallel ping-pong tool to measure these parameters that adopts single-circle-match algorithm, which requires N?1 iterations of ping-pong tests. In each iteration N/2 pair-processors are exchanging MPI ping-pong message simultaneously. This approach mimics the environment of real applications where simultaneous data transfer is a norm. In addition, to reduce the noise, the 10% data with the highest overhead/gap ping-pong results are ignored, because these data are likely to suffer from extreme congestion. Another method we use to enhance the data is the average-median statistic. For

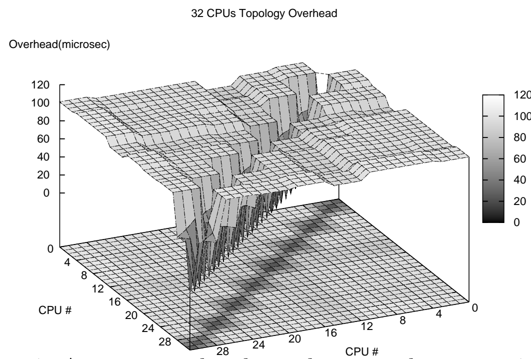


Figure 4: A 2-tier overhead topology graph on a 8x4 Xeon cluster with gigabit Ethernet connections.

the examples of gap and overhead topology graphs shown in Figures 3 and 4, we can see that inter-node communication is slower than the intra-node communication.

The advantage of this tool is twofold. Firstly it measures the effective bandwidth and overhead for the interconnect by simulating the application run time, while the technical specifications only describes the theoretic values and could not reflect situations such as congestion. Secondly, it scales to clusters with tens of thousands of nodes in that the measurement is performed in parallel so the time spent in topology discovery is kept reasonable.

## 4.3 Generating the New Process Placement

With a communication graph and a system topology graph, we can use the mapping algorithm described in the last section to generate the new process placement. The mapping function is defined as:

$$f(w_{i,j}, d_{k,l}) = \begin{cases} Frequency \times Overhead & \text{if small message} \\ Volume \times Latency & \text{if bulk message} \end{cases}$$

The new process placement can direct the scheduling of next run.

# 5. EXPERIMENTS AND ANALYSIS

We have tested the performance of profile-guided placement with the following MPI benchmarks and applications:

- NPB[1]: Five applications from NASA Parallel Benchmark:
  - LU: a version of SSOR algorithm by splitting of the operator of the Navier-Stokes equation into a product of lower triangular matrix and upper triangular matrix. It sends large numbers of very small (40byte) messages and is very sensitive to the small message communication performance.
  - BT: use Alternating Direction Implicit(ADI) approximate factorization of the operator of Navier-Stokes equation.
  - SP: uses the Beam-Warming approximate factorization and Pulliam-Chaussee diagonalization of the operator of the Navier-Stokes equation.
  - MG: iterations of V-cycle multi-grid algorithm for solving a discrete Poisson problem on a 3D grid with periodic boundary conditions.
  - CG: a solution of a sparse system of linear equations by iterations of the conjugate gradient method.
- WRF[28]: a next-generation mesoscale numerical weather prediction model designed to serve both operational forecasting and atmospheric research need. The WRF MPI version uses MPI messages to exchange the data at the division boundary.
- ZEUSMP[6]: a message passing implementation of the ZEUS algorithm, which solves the ideal MHD equations governing the evolution of a wide variety of astrophysical systems. It divide the computational domain into 3-dimensional tiles, each exchanging the boundary data with neighboring tiles by MPI non-blocking message passing.
- COMBUSTION[4]: an application which implements the direct numerical simulations (DNS) with a time dependent turbulence structure and a finite rate reaction model. The scope is to study auto ignition processes of synthetic turbulent diffusion flames.

The test-beds include clusters based on Itanium®-based and IA-32 systems. Each cluster has eight 2-way SMP nodes (Itanium® 2 or Intel® Xeon® processor) connected by a gigabit Ethernet.

- Itanium® 2 processor with 1.5GHz and 6MB L3 cache and 2GB memory, and RHEL3.0.
- Intel® Xeon® processor with 3.60GHz and 2MB L2 cache and 2GB memory, and RHEL3.0. It should be noted that we enabled the hyperthreading feature in the Xeon processor, so we put four MPI processes in each dual Xeon machine in our experiments.
- Intel® MPI 1.0 (icc 8.1) and ITC5.0 are used to trace the application.

To simulate clusters with different scales and interconnect hardware, we construct two kinds of cluster topology:

- Two-tier topology: We setup two clusters. One of them consists of 8 Itanium 2 nodes. The other one consists of 8 Xeon nodes. All the nodes are inside the clusters are connected with one switch while each node of the cluster is a SMP machine. This configuration is common in current clusters.
- Three-tier topology: Divide the 8 nodes into 2 groups connected by two unique switches. The connections between the 2 switches are:
(a) A Gigabit Ethernet for the Itanium 2 cluster;
(b) A 100 Mbps Fast Ethernet for the Xeon cluster.
This configuration is to simulate the situation of clusters of clusters where clusters are connected with shared links with limited bandwidth.

## 5.1 Comparison of MPI Placement Schemes

We have compared three kinds of placement in the experiments:

- MPI default placement: Select processors from the node list in the round-robin order.

Table 1: Two-tier topology for the 8 × 2 Itanium cluster.

| Application | Metric | MPI default | Graph Partition | Graph Mapping | Speedup |
|---|---|---|---|---|---|
| Zeusmp | sec | 2645 | 1579 | 1579 | 1.68 |
| Combustion | sec | 297 | 277 | 278 | 1.07 |
| WRF | sec | 583.71 | 582.99 | 582.99 | 1.00 |
| LU.A.16 | Mflops | 148.70 | 189.59 | 180.83 | 1.22 |
| BT.A.16 | Mflops | 140.04 | 146.50 | 149.61 | 1.07 |
| SP.A.16 | Mflops | 44.53 | 43.84 | 47.02 | 1.06 |
| MG.A.16 | Mflops | 72.16 | 108.32 | 108.32 | 1.50 |
| CG.A.16 | Mflops | 12.88 | 16.37 | 19.12 | 1.48 |

Table 2: Three-tier topology for the 8 × 2 Itanium cluster.

| Application | Metric | MPI default | Graph Partition | Graph Mapping | Speedup |
|---|---|---|---|---|---|
| Zeusmp | sec | 5308 | 3223 | 1573 | 3.27 |
| Combustion | sec | 1498 | 307 | 277 | 5.41 |
| WRF | sec | 607.76 | 587.54 | 589.82 | 1.03 |
| LU.A.16 | Mflops | 137.93 | 175.31 | 191.56 | 1.39 |
| BT.A.16 | Mflops | 103.65 | 115.8 | 125.45 | 1.21 |
| SP.A.16 | Mflops | 33.05 | 37.46 | 39.18 | 1.19 |
| MG.A.16 | Mflops | 64.71 | 63.79 | 81.34 | 1.26 |
| CG.A.16 | Mflops | 9.19 | 16.93 | 16.11 | 1.75 |

- Graph partitioning placement: Generated by the existing graph partitioning algorithms which always take the system as a two-tier one. We choose the METIS toolkit because it is one of the most popular toolkit for graph partitioning.
- Graph mapping placement: Generated by our proposed graph mapping algorithm.

Tables 1 shows the results in the Two-tier 8×2 Itanium2 clusters while Table 2 shows the results when the cluster is configured as three-tiered. The speedup in both Table 1 and Table 2 means performance of graph mapping scheme versus performance of the MPi default scheme. **Figure 5** shows the NPB data in Table 1 and Table 2 more intuitively.

The speedup columns in Table 1 and 2 show that both the graph partitioning scheme and the graph partition scheme are effective for the two-tier configuration. However, for the three-tier configuration, although the graph partitioning algorithm could still provide significant speedup over the MPI default placement scheme, it is completed exceeded by the graph mapping schemes.

Table 3: Results for the 8 × 4 Xeon cluster.

| Application | Metric | Two-tier | | Three-tier | |
|---|---|---|---|---|---|
| | | MPI default | Graph Mapping | MPI default | Graph Mapping |
| Zeusmp | sec | 4000 | 1299 | 32640 | 13355 |
| WRF | sec | 80 | 72 | 386 | 158 |
| LU.B.32 | Mflops | 224 | 245 | 44 | 174 |
| BT.B.32 | Mflops | 239 | 250 | 98 | 141 |
| SP.B.32 | Mflops | 108 | 114 | 30.53 | 51.52 |
| MG.C.32 | Mflops | 106 | 106 | 37 | 52 |
| CG.B.32 | Mflops | 7.32 | 16.15 | 1.45 | 3.49 |

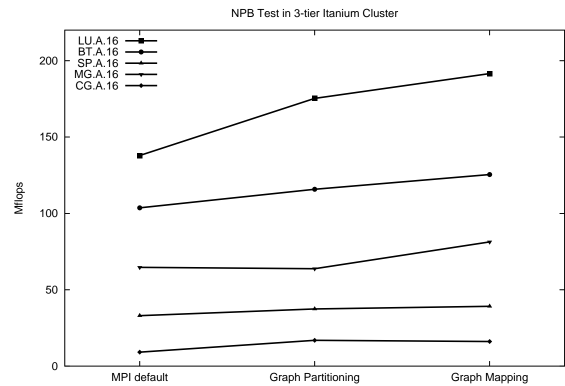



Figure 5: NPB test results in 3-tier Itanium cluster.

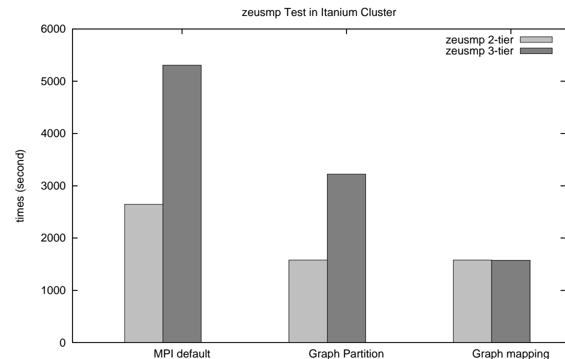



Figure 6: Zeusmp results in Itanium cluster

**Figure 6** shows Zeusmp results by comparing the performance of two-tier and three-tier configuration under different schemes. It is shown that both the MPI default scheme and the graph partition scheme suffered from the three-tier configuration, while the graph mapping scheme almost maintains the performance in the two-tier configuration.

Table 3 demonstrates the results for the 8×4 Xeon cluster with both two-tiered and three-tiered configurations. In the three hierarchy topology of Xeon cluster, the bottleneck is the slow connection between those switches. Comparing Table 2 and Table 3 would show that graph mapping scheme is more effective for multicluster systems when the bandwidth of interconnect between clusters is low. For a example, the WRF shows only marginal speedup in Table 2, but significant speedup in the 3-tier configuration in Table 3. The trend of multicore processors will put many more computing power in a cluster system and the interconnect may be more congested in the future multicluster systems. Our graph mapping algorithms are supposed to be more effective in these systems.

There are some constraints for the profile-guided processes placement. The intelligent placement only improves the performance of applications with heavy point-to-point communications. It works best if these communications are asymmetrical. If an application fails to meet these requirements, slight slow down may occur. This situation occurred when we tested a version of CHARMM [2] application written with point-to-point messages. We have addressed this issue by introducing a threshold for the improvement brought by process placement.

Table 4: Execution Time of the Graph Mapping Algorithm.

| Benchmark | scale(process #) | Mapping time (sec) |
|---|---|---|
| LU.C | 32 | 0.026 |
| LU.C | 64 | 0.326 |
| MG.C | 32 | 0.026 |
| MG.C | 64 | 0.344 |
| CG.C | 32 | 0.023 |
| CG.C | 64 | 0.380 |

## 5.2 Execution Time of the Graph Mapping Algorithm

In this section, we present the execution time of our graph mapping algorithm up to 64 MPI processes with three benchmarks from NPB, LU.C, MG.C and CG.C. We limit the number of process to 64 because it is the largest scale of traces we could get in our test environments. The algorithm is implemented in C++. The experiments were performed on a Desktop PC with Intel P4 2.8Ghz processor, Linux 2.6 and GCC 4.0 at -O3.

From the Table 4, it could be concluded that for small-scale MPI applications whose number of processes is less than 64, the overhead of our graph mapping algorithm is almost ignorable. With these testing data and the $O(N^3)$ complexity, a rough estimation on the algorithm's cost would be around 1000 seconds when the number of processes is 1000.

For an MPI application with 1000 processes, it is expected to execute hours or even days. Besides, this cost is a one time cost for a given application/system pair, which will be amortized with multiple executions. Comparing with the current profile guided compiler optimization for sequential code(which requires hours to have 5%-10% performance improvement), we think that the performance/cost ratio of our approach is acceptable.

# 6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a fully automatic scheme for optimized parallel process placement in SMP clusters and multi-clusters without users' knowledge on either applications or target systems. We implemented this scheme as a toolset named MPIPP. MPIPP is being integrated into the Intel Cluster Toolkit now.

We proposed a graph mapping algorithm which maps the communication graph of parallel applications to the system topology graph. This algorithm is more powerful than the existing graph partitioning algorithms because it takes multi-tier interconnect topology into consideration. With the advent of the multicore processor, future systems are likely to be multiple-tiered instead of just two-tiered where our algorithm should be superior.

We made extensive experiments with several parallel benchmarks and applications on a few cluster systems and multi-cluster systems. Experimental results show that the optimized process placement generated by our tools can achieve significant speedup.

Our current implementation only addresses point-to-point MPI messages. Many applications are written with primarily collective operations, such as LS-DYNA [19]. We are working on the optimization of collective operations.

Besides communication, there are other factors that could affect the application performance, such as memory usage, load balance and application communication pattern. The interaction of these factors and the placement scheme should be investigated.

While this research focus on message passing applications, the approach described in this paper has the potential to be extended to shared memory applications.

# 7. REFERENCES


[1] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, Dec. 1995.

[2] ChARMM Team. CHARMM. `http://www.charmm.org`

[3] Culler, Karp, Patterson, Sahay, Schauser, Santos, Subramonian, and von Eicken. LogP: Towards a realistic model of parallel computation. In *PPOPP: 4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming: PPOPP*, 1993.

[4] M. Deilmann, B. Roberg, M. Baum, and V. Scherer. Numerical simulation of evaporation and ignition of non-premixed $n$-heptane flames. In *5th International Conference on Parallel Processing and Applied Mathematics*, Czestochowa, Poland, 2003.

[5] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *j-COMP-SURV*, 34(3):313–356, Sept. 2003.

[6] R. A. Fiedler. Optimization and scaling of shared-memory and message-passing implementations of the zeus hydrodynamics algorithm. In *Proceedings of Supercomputing'97 (CD-ROM)*, San Jose, CA, Nov. 1997. ACM SIGARCH and IEEE. Hewlett-Packard Company.

[7] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.

[8] B. Hendrickson and R. Leland. *The Chaco User's Guide Version 2*. Sandia National Laboratories, Albuquerque NM, 1995.

[9] Intel Ltd. Intel® Trace Analyzer & Collector. `http://www.intel.com/cd/software/products/asmo-na/eng/cluster/tanalyzer/index.htm`

[10] Intel Ltd. Intel® MPI library. `http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/index.htm`

[11] G. Karypis and V. Kumar. METIS, Unstructured Graph Partitioning and Sparse Matrix Ordering System. Version 2.0. Technical report, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, Aug. 1995.

[12] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, Feb. 1970.

[13] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. In *PPOPP*, pages 131–140, 1999.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[15] G.-J. Lai and C. Chen. A new scheduling strategy for numa multiprocessor systems. In *ICPADS*, pages

222–229. IEEE Computer Society, 1996.

[16] C.-H. Lee, M. Kim, and C.-I. Park. An Efficient K-Way Graph Partioning Algorithm for Task Allocation in Parallel Computing Systems. In P. A. Ng, C. V. Ramamoorthy, L. C. Seifert, and R. T. Yeh, editors, *Proceedings of the First International Conference on Systems Integration, Morristown, NJ, USA, April 1990*, pages 748–751. IEEE Computer Society, 1990.

[17] S.-Y. Lee and J. K. Aggarwal. A mapping strategy for parallel processing. *IEEE Transaction on Computers*, C-36(4):433–442, Apr. 1987.

[18] V. M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans. Computers*, 37(11):1384–1397, 1988.

[19] LS DYNA Team. LS DYNA. `http://www.dynalook.com/`

[20] R. Martin, A. Vahdat, D. Culler, and T. Anderson. Effect of communication latency, overhead, and bandwidth on a cluster architecture. Technical Report CSD-96-925, University of California, Berkeley, June 17, 1998.

[21] B. Monien and S. Schamberger. Graph partitioning with the party library: Helpful-sets in practice. In *SBAC-PAD*, pages 198–205. IEEE Computer Society, 2004.

[22] A. Pant and H. Jafri. Communicating efficiently on cluster based grids with mpich-vmi. In *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 23–33, Washington, DC, USA, 2004. IEEE Computer Society.

[23] S. Sanyal, A. Jain, S. K. Das, and R. Biswas. A hierarchical and distributed approach for mapping large applications to heterogeneous grids using genetic algorithms. In *CLUSTER*, pages 496–499. IEEE Computer Society, 2003.

[24] ScaliMPI Team. ScaliMPI. `http://www.scali.no/download/doc/Scali_MPI_Connect_FF_4_3_6_121104_EXT.pdf`

[25] J. L. Traff. Implementing the MPI process topology mechanism. In IEEE, editor, *SC2002: From Terabytes to Insight. Proceedings of the IEEE ACM SC 2002 Conference, November 16–22, 2002, Baltimore, MD, USA*, pages 1–14, pub-IEEE:adr, 2002. pub-IEEE.

[26] Walshaw and Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIJSSC: SIAM Journal on Scientific and Statistical Computing, apparently renamed SIAM Journal on Scientific Computing*, 22, 2000.

[27] L. Weijian, C. Wenguang, L. Zhiguang, and Z. Weimin. Communication optimization for smp clusters. *Tsinghua Science and Technology*, 6(1):18–23, 2001.

[28] WRF Team. WRF. `http://www.wrf-model.org/`

# APPENDIX

## A. PSEUDO-CODE OF THE GRAPH MAPPING ALGORITHM

**Algorithm**: MAPPING — minimize the sum of communication cost.

**Input**: the communication graph $G_P = W[1..|P|][1..|P|]$ **and** topology graph $G_T = D[1..|N|][1..|N|]$, where define $V = N = P$.

**Output**: mapping scheme /* M[1..|V|] */

**Variables**:
/* M[i] = processor number that task $p_i$ maps. */
$M[1..|P|]$: **integer**;
/* GAIN[i][j] = $g_{p_i,p_j}$ */
$GAIN[1..|P|][1..|P|]$: **real**;
/* 0 = unused, 1 = used state */
$STATE[1..|P|]$: **Boolean**;
/* history information on move operations */
$HISTORY[1..|P|][1..|P|]$: **integer**;
/* temporary gain for move history */
$TEMP[1..|P|]$: **integer**;

```
1) Construct a random initial mapping;
     /* M[i] =random_index(i) */
2) for i := 1 to |P| do
     /* one pass start: calculate gain for all pairs; */
       STATE[i] :=0; /* unused */
         for j := 1 to |P| do
           GAIN[i][j] := g(p_{i,M[i]}, p_{j,M[j]});
         endfor
     endfor
3) for i := 1 to |P|/2 do
3.1) select unused p_l such that
       g(p_{l,M[l]}, p_{m,M[m]}) = max_{i,j}(GAIN[i][j]);
3.2) STATE[l] := 1; STATE[m]:=1 /* p_l switch with p_m */
3.3) M[l] ⇔ M[m]; /* one−switch */
3.4) /* save information of one −switch */
       HISTORY[i][1] := l;
       /* save information of one − switch */
       HISTORY[i][2] := m;
3.5) /*save the gain of v_l ⟺ v_m */
       TEMP[i] := GAIN[l][m];
3.6) for j := 1 to |P| do /* recalculate gains */
         for j := 1 to |P| do
           GAIN[i][j] := g(p_{i,m[i]}, p_{j,M[j]});
         endfor
       endfor /* step 3.6 */
     endfor /* step 3, one pass end */
4) chooe t to maximize Gain = sum_{j=1}^{i} TEMP[j];
5) if Gain > 0 then /* complete one pass */
      for j := t + 1 to |P| do
        /* restore */
        M[HISTORY[j][1]] ⟺ M[HISTORY[j][2]];
      endfor
     goto 2);
   endif
```