 Digital Object Identifier : 10.1109/CISIS.2009.108

Estimating the Optimal Configuration of a Multi-Core Cluster: A Preliminary Study

Shuichi Ichikawa and Shoichiro Takagi
Department of Knowledge-based Information Engineering
Toyohashi University of Technology
1-1 Hibarigaoka, Tempaku, Toyohashi 441-8580, Japan
E-mail: ichikawa@ieee.org

Abstract

While multi-core processors became crucial elements of high-performance computing, the clusters of multi-core processors are still difficult target for optimization, since the communication time differs significantly in intra-node communication and inter-node communication. It is thus very important to select the optimal configuration of multi-core clusters; i.e., to find the optimal number of processes and to find the optimal allocation of processes to nodes. This study first elucidates the multi-core specific issues by examining the effect of process allocation in a multi-core cluster, using divergent and convergent allocations of four benchmark programs. Then, the estimation of optimal configuration is attempted for divergent and convergent allocations, using execution-time estimation models for single-core clusters. Though the estimation errors were less than 20% in most cases, further improvement is expected by incorporating the enhancements for multi-core systems into estimation models.

1 Introduction

Recently, multi-core processors are very popular for their performance and power efficiency. They have already become the mainstream in servers and desktop computers, and the clusters of multi-core processors (*multi-core clusters*) are becoming more and more important. The effective use of multi-core clusters is an urgent matter in scientific computing.

A multi-core cluster is categorized as an SMP cluster, which consists of symmetrical multiprocessors (SMP). Such systems are known to be difficult targets for optimization, because the communication time differs largely in intra-node communication and inter-node communication. Consequently, the performance of multi-core cluster is largely affected by process allocation, which determines the communication paths between processes. It is a very difficult task to select the best process allocation, since it is dependent on the nature of application program. Section 3 of this work describes the evaluation results of two policies with four applications, and highlights some multi-core specific issues.

Another purpose of this study is to explore the means to find the best configuration of multi-core clusters. Here, the best configuration designates the configuration that minimizes the execution time for a given problem size, where a configuration of cluster designates the number of processes and the allocation of processes. For conventional single-core clusters, a scheme to estimate the best configuration is already presented [8][9][6]. Section 4 of this study describes the attempt to estimate the optimal configuration for our multi-core cluster.

2 Related Studies

The performance evaluation and analysis of multi-core clusters have been presented by a number of groups recently. Chai, Gao, and Panda [2] studied the impact of multi-core architecture on cluster computing, evaluating parallel benchmark programs with dual-core Xeon processors on Intel Bensley system. They pointed out that potential bottlenecks might include intra-node communication, inter-node communication, cache, and memory contention. Pourreza and Graham [14] explored three issues to utilize multi-core clusters: communication efficiency, process affinity, and initial process distribution. They examined NAS parallel benchmarks (NPB), and suggested that some programs (e.g., IS kernel) benefit from process affinity. They also pointed out that processes should be distributed to balance the communications among nodes.

These studies analyzed the factors that affect the performance of multi-core clusters, which are suggestive for per-

formance optimization of multi-core clusters. Meanwhile, they did not provide a systematic method to derive maximal performance from a multi-core cluster. This study discusses a method to estimate the optimal configuration of a multi-core cluster.

Alam et al. [1] reported the various aspects of a parallel computer with AMD Opteron dual-core processors, and pointed out that an appropriate use of processor affinity can result in 25% performance improvement for scientific applications. Particularly in STREAM benchmark, better performance was derived by leaving one of cores unused compared to making full use of cores. Luecke and Li [10] evaluated NPB on Intel Xeon and AMD Opteron dual-core Linux clusters, and found that better performance can sometimes be achieved with a single MPI process per node. Pinto, Tomazella, and Dantas [13] investigated intra-node and inter-node communication behavior of four cluster setups, and proposed to leave one core per host idle in order to process communication overhead of a running application.

These studies are interesting in that they pointed out that using all cores is not always the best policy. However, they did not provide a method to find the optimal configuration, since they lack the viewpoint that the optimal configuration is dependent on both the application and problem size. Meanwhile, this study focuses on this very point.

Other researchers aimed at optimizing communication for multi-core clusters. Specifically, MPI implementations for multi-core or SMP clusters have been discussed by the following groups. Chai, Hartono, and Panda [3] presented a new design for MPI intra-node communication that is suited for SMP or multi-core nodes. Chen et al. [4] proposed a profile-guided approach to find the optimized process mapping to minimize communication cost for arbitrary MPI applications.

Although minimizing the communication cost may reduce the execution time, it does not solely guarantee the optimal execution time. As shown in the following sections, selecting the optimal number of processes is a prerequisite. This study focuses on a method to find the optimal process allocation considering the balance of computation, communication, and memory bandwidth.

3 Effect of Process Allocation

3.1 Evaluation Platform

The evaluation platform of this study is a quad-core cluster comprised of four nodes, each of which consists of an Intel Core 2 Quad Q6600 processor (2.4 GHz) [7] with 4 GB of memory. A Q6600 processor contains two 4 MB L2 caches, each of which is shared by two cores; i.e., each Q6600 consists of two dual-core processors, which are connected by FSB (front side bus) each other. Then, four nodes of Q6600 are connected by a Giga-bit Ethernet switch.

The following four benchmarks were chosen for performance evaluation, which are the same as in the previous study [6]. Intel C/Fortran compilers on FedoraCore 6 were used for compilation with mpich-1.2.7p1 and Atlas 3.60 libraries.

- **Himeno benchmark** [5] solves a pressure Poisson equation with Jacobi iteration.
- **HPCMW** [11] measures the performance of three-dimensional linear elastic finite-element application.
- **HPL** (high-performance Linpack) [12] solves a random dense linear system of equations.
- **FFTE** [15] measures the double precision complex one-dimensional DFT (one-dimensional discrete Fourier transforms).

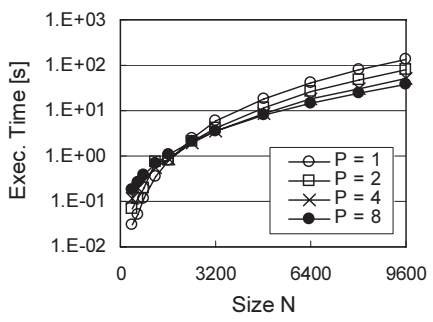
3.2 Process Allocation Policies

Let us start with investigating the optimal configuration of a *single-core* cluster, before discussing a multi-core cluster. Figure 1 summarizes the execution time of HPL benchmark, measured with a cluster of eight Pentium 4 (3.6 GHz) processors with Gigabit Ethernet. In Fig. 1(a), P represents the number of processes, or the number of nodes; e.g., $P = 4$ means that HPL was executed with four nodes, each of which accommodates one MPI process of HPL.

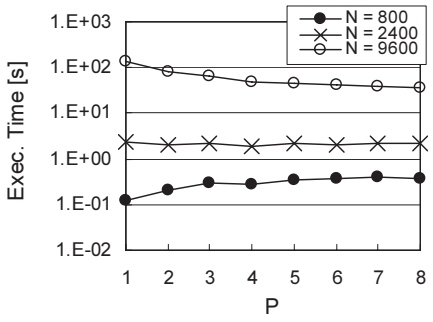
It is readily seen from Fig. 1(a) that the optimal P depends on the problem size N. For small sizes ($N = 400 - 1600$), the execution time is minimized with $P = 1$; for middle sizes ($N = 2400, 3200$), $P = 4$ is best; for large sizes ($N = 4800 - 9600$), using all nodes ($P = 8$) is the best choice. This phenomenon is interpreted as follows.

The execution time consists of computation time and communication time. Using more nodes, the computation time of each node is reduced, since the computational load is distributed among nodes; meanwhile, the communication time among nodes increases. In a small-sized problem, the computation time is smaller than the communication time; it is thus better to use fewer nodes. In a large problem, the execution time is reduced by using more nodes, because the computation time is a dominant factor. Figure 1(b) definitely supports this working hypothesis. It is hence very important to select the best P in accordance with the problem size N for both performance optimization and effective resource utilization.

Though the best configuration solely depends on P in a single-core cluster, things are more complicated in a multi-core cluster. Since communication time differs largely for



(a) Execution time (T) vs. various problem sizes (N).



(b) Execution time (T) vs. the number of processes (P).

Figure 1. The execution time of HPL benchmark, measured by a PC cluster comprised of eight Pentium 4 (3.6 GHz) nodes.

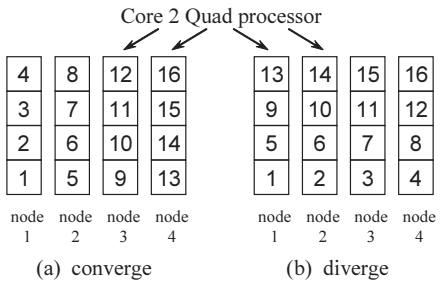


Figure 2. Process allocation policies for multi-core clusters.

on-chip, intra-node, and inter-node communications, the communication time is strongly influenced by the allocation of processes.

Though there are many possibilities to allocate processes to cores, it is impossible to examine every allocation policies in this current study. Therefore, two distinctive policies are examined in the following discussion (Fig. 2). The first allocation policy is "converge", where the processes are allocated to use as few nodes as possible. Convergent allocation naturally reduces the inter-node communication between processes. The second policy is "diverge", where the processes are allocated to use as many nodes as possible, while balancing the number of processes in each node. With divergent allocation, the cache capacity and memory bandwidth for each process are maximized in exchange for increased inter-node communication.

Figure 2 illustrates two allocation policies in case of our multi-core cluster. Since our cluster consists of four quad-core processors, it is natural to assume $1 \leq P \leq 16$, where P is the total number of processes. In Fig. 2, each rectangle represents a core, and four adjacent cores correspond to a Core 2 Quad processor. The numbers shown in rectangles designate the order of process allocation; i.e., P processes are allocated to the cores numbered from 1 to P in order.

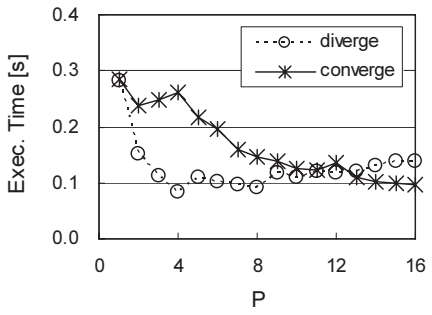
3.3 Measurement Results

Figure 3 summarizes the execution times of four benchmarks with two allocation policies.

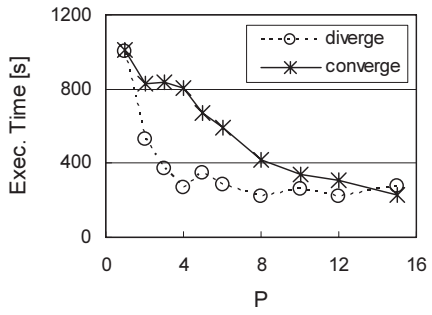
In Himeno benchmark ($N = 256$), divergent allocation is superior to convergent allocation overall (Fig. 3(a)). With divergent allocation, the execution time decreases quickly in incrementing P from 1 to 4; meanwhile, with convergent allocation, the execution time does not decrease substantially between $P = 1$ and $P = 4$. These facts suggest that the bottleneck of Himeno benchmark exists in memory bandwidth for this size ($N = 256$). The identical behavior is observed in even smaller sizes.

In case of convergent allocation, discontinuities are found at $P = 4$ and $P = 12$. It is natural to find the discontinuities there, because the total memory bandwidth depends on the number of nodes. In divergent allocation, the execution time exhibits an obvious discontinuity at $P = 5$, which is caused by memory access imbalance. In case of $P = 5$, node 1 accommodates two processes, whose performances are degraded by sharing memory bandwidth; thus the processes on node 1 become the bottleneck of the overall execution time. Similar behavior is observed at $P = 9$, as expected.

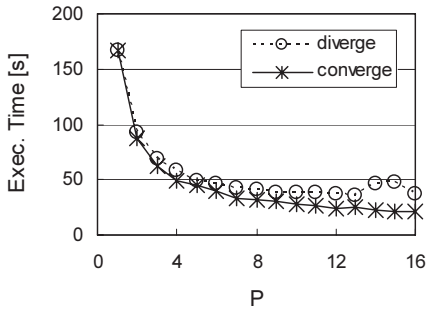
The results of HPCMW benchmark (Fig. 3(b)) are very similar to that of Himeno benchmark. It should be noted that it is mandatory in HPCMW to use P that is a factor of N; hence, there are no data for $P = 7, 9, ..., 16$ in Fig. 3(b) where $N = 600$. Divergent allocation is superior to



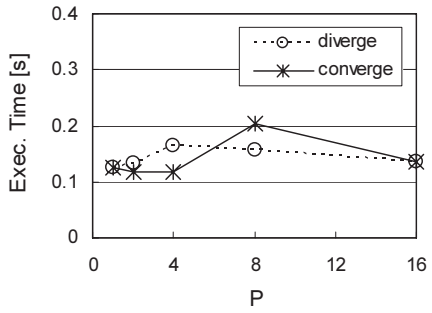
(a) Himeno benchmark (N = 256).



(b) HPCMW benchmark (N = 600).



(c) HPL benchmark (N = 9600).



(d) FFTE benchmark (N = 2^{20}).

Figure 3. Comparison of two allocation policies for four benchmark programs.

convergent allocation for most P in HPCMW, and discontinuities are found where P is a multiple of 4. Although Fig. 3(b) summarizes the case of $N = 600$, similar characteristics are observed in even smaller sizes. Obviously, memory bandwidth is the bottleneck in HPCMW benchmark.

HPL benchmark (Fig. 3(c)) behaves much differently from Himeno and HPCMW benchmarks. Although convergent allocation is superior to divergent allocation, the difference is not large for $N = 9600$. The execution time of HPL decreases almost monotonically in both allocations, which implies that the computation time is a dominant factor in both allocations. On the other hand, the difference between divergent and convergent allocations becomes larger in smaller sizes, which is naturally interpreted as the effect of communication time. Since the respective computation and communication times of HPL are $O(N^3)$ and $O(N^2)$, the ratio of communication to total execution time becomes larger in smaller sizes of problems.

Figure 3(d) summarizes the results of FFTE benchmark of $N = 2^{20}$. Since it is mandatory in FFTE to use N of a power of two and a multiple of P^2, only $P = 2^i$ $(0 \leq i \leq 4)$ are plotted in Fig. 3(d). The results of FFTE benchmark exhibit peculiar behavior, which can be interpreted as follows.

As Ichikawa et al. [6] reported, all-to-all communication is a dominant factor in the execution time of FFTE benchmark. In divergent allocation, the execution time increases as P increases from 1 to 4, which is caused by the increase of inter-node communication. For $P = 4, 8$, and 16, all four nodes are used, while more cores are allocated in each node. Thus, there is a difference of nature between $1 \leq P \leq 4$ and $4 \leq P \leq 16$ of divergent allocation.

In convergent allocation, only one node is used for $1 \leq P \leq 4$, where all communication is performed by intra-node communication; meanwhile, two or four nodes are used for $P = 8$ and $P = 16$, where inter-node communications are performed. Thus, it is natural to find a discontinuity between $1 \leq P \leq 4$ and $8 \leq P \leq 16$.

4 Estimation of Optimal Configuration

As shown in Section 3, it is generally difficult to find the optimal configuration of a cluster. The P that minimize T depends on both problem size N and process allocation. The best process allocation is also not obvious, since it heavily depends on the nature of the target applica-

tion. Things become even more complicated in multi-core clusters, because of discontinuous behavior of T against P. Thus, a systematic method is desired to estimate the optimal configuration of a multi-core cluster.

Kishimoto and Ichikawa [8][9] presented a scheme to estimate the optimal configuration of a heterogeneous cluster, i.e. the optimal subset of PEs and the optimal process allocation. They constructed the execution-time estimation models from the measurement results of HPL [12], and showed that the optimal or sub-optimal configurations were actually estimated for various sizes. Ichikawa, Takahashi, and Kawai [6] further enhanced this scheme, and showed that their new models (NP-T models) can estimate sub-optimal configurations for HPL, HPCMW, FFTE, and Himeno benchmarks.

The purpose of this section is to verify the above models with our multi-core cluster. Since their models were originally developed for single-core clusters, it is not clear whether these models are practically applicable to multi-core clusters (and how good their estimations are). As stated in Sect. 3, the behavior that is specific to multi-core clusters may have a negative impact on the estimation quality. Thus, we start with evaluating the original models as they are, and then proceed to the discussion of possible improvements.

In the evaluation of this section, we constructed N-T and NP-T models with non-negative least squares method [9][6]. Since the target applications are all the same, model equations were also taken from the previous study [6]. Following the previous studies, NP-T models were adopted for $P \neq 1$, while N-T models were adopted for $P = 1$. More details of model construction are found in the previous studies [9][6].

Once the models are constructed, the execution time T can be estimated for arbitrary P and N. That is, the optimal P can be estimated by selecting the P $(1 \leq P \leq 16)$ that minimize T for a given N. It should be noted that the estimated optimal P (estimated P, in short) is not necessarily equal to the actual optimal P (optimal P, in short). The models are no more than the approximations of reality, and inevitably involve approximation errors.

The overall estimation error (ε) is defined by the following equation, where $\hat{\tau}$ and $\hat{T}$ are the respective actual execution times of the estimated optimal P and of the actual optimal P. In other words, ε represents the overhead of the estimated P over the optimal P.

$$\varepsilon = \frac{\hat{\tau}}{\hat{T}} - 1 \tag{1}$$

As shown in Sect. 3, the profiles of divergent and convergent allocations are quite different. Therefore, we build and evaluate the models for divergent and convergent allocations separately, in this current study. Although it would be desirable to formulate a general scheme to include divergent and convergent allocations as special cases, it is left for future studies.

Figure 4 summarizes the estimation results of four benchmark programs by two allocation policies. Each graph presents (1) the estimated optimal P and the actual optimal P with bars, and (2) the estimation error (ε or epsilon) with lines.

In convergent allocation of Himeno benchmark (Fig. 4(a)), the optimal P increases from 4 to 16 in accordance with size N, while the estimated P ranges from 14 to 15. The errors are quite small in most sizes, except in small sizes. Though the relative errors (ε) of small sizes look large, it is not serious because the absolute difference in execution time is small.

In divergent allocation (Fig. 4(b)), the optimal P remains small and the estimated P matches to the optimal P quite well. It should also be noted that the optimal execution time of divergent allocation is smaller than that of convergent allocation for most sizes (e.g, Fig. 3(a)), while using a smaller number of processes. Our models successfully estimated these facts.

Figure 4(c) summarizes the convergent allocation of HPCMW. The optimal P ranges from 12 to 16, while the estimated P ranges from 10 to 15. Though the errors are small for most sizes, there are some exceptions. Since there is a constraint in HPCMW that P must be a factor of N, P cannot take contiguous values, which makes HPCMW a difficult target for optimization. For example, in case $N = 510$, the optimal P was 15, while the estimated P was 10. Although $P = 10$ is the second-best choice, the resulting error becomes approximately 30%.

The optimal P of divergent allocation for HPCMW is smaller than that of convergent allocation (Fig. 4(d)); the optimal execution times are also smaller, as in Himeno benchmark. Our models succeeded to estimate the optimal or sub-optimal P with modest errors.

HPL is a relatively easy target for estimation (Figures 4(e) and 4(f)). In both allocations, the optimal P increases according to the size N, while the estimated P reproduces this behavior quite well. The estimation errors remain very small in both allocations.

FFTE exhibits peculiar behavior (cf. Sect. 3), which makes the estimations difficult. As readily seen from Figures 4(g) and 4(h), the estimated P monotonically increases as N increases, while the optimal P does not. Although the overall error (ε) remains modest, the estimations are not satisfactory. The considerations of multi-core specific issues seem essential to improve the estimation of FFTE. These are left for future studies.

5 Conclusion

This study presented two process allocation policies for multi-core clusters, which were examined with the execution times of four benchmark programs. The estimation functions of execution time were constructed, and the estimation of the optimal configuration for our multi-core cluster was attempted. Though the optimal or sub-optimal configurations were found in most cases with conventional models, there are still some exceptions where the errors become large. More consideration of multi-core specific issues is required for more accurate estimations.

In the following studies, the evaluation with larger number of nodes should be conducted. Since the quality of models largely depends on the number of measurements, more nodes are necessary for more precise discussions.

Another urgent issue is the enhancement of models for multi-core clusters. Though this study examined the model equations of continuous functions [6], as a basis of evaluation, the actual execution time on a multi-core cluster becomes a discontinuous function as shown in Section 3. Now that we finished the evaluation of the continuous models, we have to proceed to considering discontinuous functions for multi-core clusters.

More variations of process allocation should be examined, also. Although two typical allocations were examined in this current study, there are many other allocation policies, which should be explored. At the end, it is desirable to find the best allocation policy automatically. Generalizing the models that subsumes process allocation policies might be worth considering.

Acknowledgment

This study was partially supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS).

References

[1] S. Alam, R. Barrett, J. Kuehn, P. Roth, and J. Vetter. Characterization of scientific workloads on systems with multi-core processors. In *Proc. IEEE Intl. Symp. Workload Characterization (IISWC 2006)*, pages 225–236. IEEE Computer Society, 2006.

[2] L. Chai, Q. Gao, and D. K. Panda. Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system. In *Proc. Seventh Intl. Symp. Cluster Computing and the Grid (CCGrid '07)*, pages 471–478. IEEE Computer Society, 2007.

[3] L. Chai, A. Hartono, and D. Panda. Designing high performance and scalable MPI intra-node communication support for clusters. In *Proc. 2006 Intl. Conf. Cluster Computing (Cluster 2006)*, pages 1–10. IEEE Computer Society, 2006.

[4] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn. MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. In *Proc. 20th Intl. Conf. Supercomputing (ICS '06)*, pages 353–360. ACM, 2006.

[5] R. Himeno. Himeno benchmark, 2005. `http://accc.riken.jp/E/HPC_e/HimenoBMT_e/index_e.html`.

[6] S. Ichikawa, S. Takahashi, and Y. Kawai. Optimizing process allocation of parallel programs for heterogeneous clusters. *Concurrency and Computation: Practice and Experience*, 2008. (to appear).

[7] Intel. *Intel Core 2 Extreme Quad-Core Processor QX6000 Sequence and Intel Core 2 Quad Processor Q6000 Sequence: Datasheet*, August 2007. 315592-005.

[8] Y. Kishimoto and S. Ichikawa. An execution-time estimation model for heterogeneous clusters. In *Proc. 18th Intl. Parallel and Distributed Processing Symp. (IPDPS 2004)*. IEEE Computer Society, 2004. (CD-ROM).

[9] Y. Kishimoto and S. Ichikawa. Optimizing the configuration of a heterogeneous cluster with multiprocessing and execution-time estimation. *Parallel Computing*, 31(7):691–710, 2005.

[10] G. R. Luecke, Y. Li, and M. Cuma. Using benchmarking to determine efficient usage of nodes in a cluster. *Benchmarking: An International Journal*, 14(6):728–749, 2007.

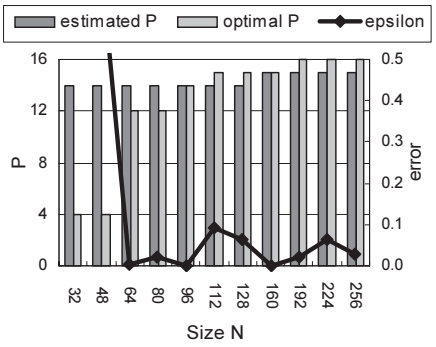
[11] K. Nakajima. Test programs for parallel iterative methods on various types of architectures (hpc-mw-solver-test ver.1.0), 2003. `http://www.fsis.iis.u-tokyo.ac.jp/en/result/software/`.

[12] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL – a portable implementation of the high-performance Linpack benchmark for distributed-memory computers. `http://www.netlib.org/benchmark/hpl/`.

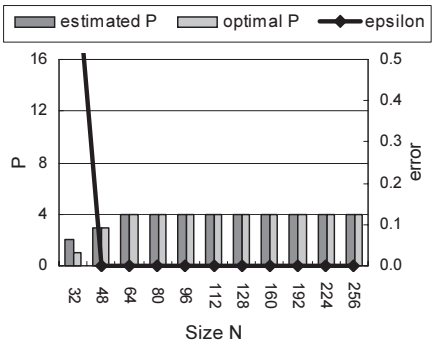
[13] L. C. Pinto, L. H. B. Tomazella, and M. A. R. Dantas. An experimental study on how to build efficient multi-core clusters for high performance computing. In *Proc. 2008 Intl. Conf. Computational Science and Engineering (CSE 2008)*, pages 33–40. IEEE Computer Society, 2008.

[14] H. Pourreza and P. Graham. On the programming impact of multi-core, multi-processor nodes in MPI clusters. In *Proc. 21st Intl. Symp. High Performance Computing Systems and Applications (HPCS '07)*, page 1. IEEE Computer Society, 2007.

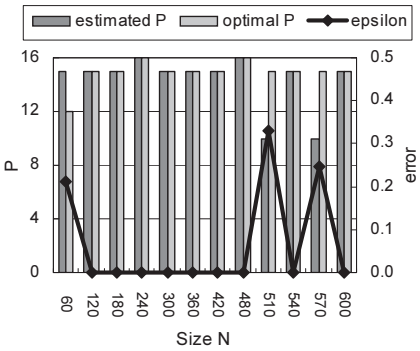
[15] D. Takahashi. FFTE: A fast Fourier transform package, 2005. `http://www.ffte.jp/`.



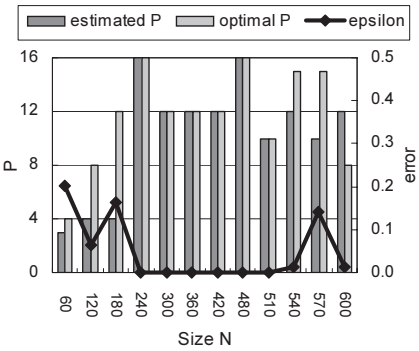
(a) Himeno benchmark (converge).



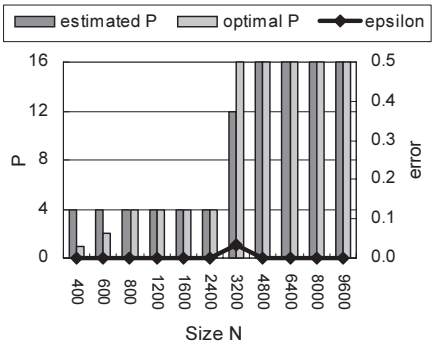
(b) Himeno benchmark (diverge).



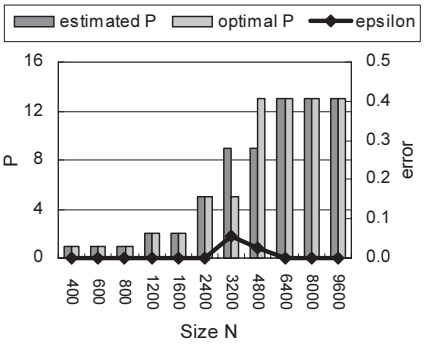
(c) HPCMW benchmark (converge).



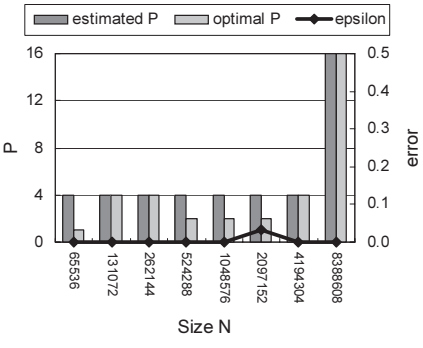
(d) HPCMW benchmark (diverge).



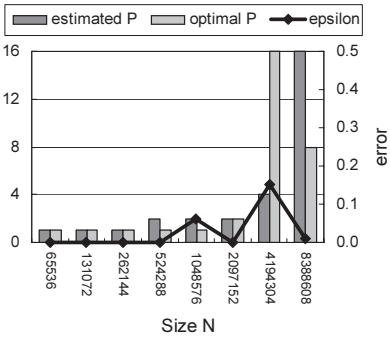
(e) HPL benchmark (converge).



(f) HPL benchmark (diverge).



(g) FFTE benchmark (converge).



(h) FFTE benchmark (diverge).

Figure 4. Estimation results of two allocation policies for four benchmark programs.