2005－HPC－103（20）
2005／8／5

Performance Enhancement for Matrix Multiplication on an SMP PC Cluster

TA QUOC VIET,† TSUTOMU YOSHINAGA † and BEN A. ABDERAZEK†

Our study proposes a Reducing-size Task Assignation technique (RTA), which is a novel approach to solve the grain-size problem for the hybrid MPI-OpenMP thread-to-thread (hybrid TC) programming model in performing distributed matrix mulitplication on SMP PC clusters. Applying RTA, hybrid TC achieves an acceptable computation performance while retaining the dynamic task scheduling capability, thereby it can yield a 22% performance improvement for a 16-node cluster of Xeon dual-processor SMPs in comparison with the pure MPI model. Moreover, we provide formulas to predict hybrid TC performance in different circumstances.

1. Introduction

The Hybrid TC model[1)~3)] has proved its advantages over a pure MPI model for SMP clusters. In Hybrid TC, an SMP node runs only a single MPI process while innernode parallelization is achieved with OpenMP. All communication tasks of a node are performed by the master thread inside the OpenMP fork-and-Join section. During the communication, other threads execute computation tasks, thereby resulting in the appearance of the overlap between computation and communication inside a node, which improves overall performance for the cluster. This phenomenon can be explained based on the limitation of shared resources for computation and/or communication. When all processors together perform computation or communication, they require the same kind of resources. This may lead to a scarcity of resources, thereby forming an execution bottleneck. In our SMP PC system, network and memory bus bandwidth limitations are the root causes of communication and computation bottlenecks, respectively.

The hybrid TC model requires a dynamic task schedule, by which the master thread can join the computation group anytime when it is free of communication. Hence, the MPI coarse-grain size approach is not available and a fine-grain size approach costs a performance degradation. This study proposes RTA that allows computation threads to work with coarse grain size most the execution time while retaining the dynamic task schedule capability. Applying RTA, the cluster can obtain 99.3% of computation performance and 122% of overall performance in comparison with pure MPI. Besides, we provide formulas to predict performance of hybrid model according to different circumstances, accurrancy of which is proved by the experimental results.

As an example, RTA is applied to perform a distributed matrix multiplication. The experimental environment consists of a cluster of 16 Intel Xeon 2.8 GHz dual-processor nodes connected via a Gigabit Ethernet network. Each node has 1.5 GB of memory, Red Hat Linux 9.0 as the operating system, and MPICH 1.2.6[4)] as the MPI library. Local matrix multiplication ($dgemm$) is performed by the goto-blas library[5)].

The remaining paper is organized as follows. Section 2 introduces related studies that also examine programming models for SMP clusters. Section 3 describes a distributed matrix multiplication solution, the SUMMA algorithm and the developtment of the hybrid TC solution. Section 4 not only presents RTA with its implementation and performance but also shows formulas determining the outperformance of hybrid TC with RTA over MPI. Section 5 shows the experimental results. Section 6 explains initial ideas of an asynchronous MPI model, which is another approach expected to achieve a similar performance as hybrid TC. Finally, section 7 concludes the paper.

2. Related Studies

With regard to specific hierarchical memory systems, several studies have proposed hybrid MPI-OpenMP programming models in which each SMP node runs only a single MPI process and computation parallelization inside a node is achieved by OpenMP. In comparison with pure MPI, hybrid models not only replace internode communication with the usage of shared variables located in share memory areas but also reduce the number of MPI processes.

Hybrid models are classified as a hybrid model with process-to-process communication (hybrid PC) and a hybrid model with thread-to-thread communication (hybrid TC). The hybrid PC model has been examined earlier but it could not show a positive result. F. Cappello et. al have shown a common path to develop a fine-grain hybrid PC code from an existing MPI model[6)]. Based on this path, they derived a fine-grain hybrid PC solution for the NAS benchmarks and compared its performance with that of a pure MPI model for a cluster of IBM SP nodes[7),8)]. Using COSMO–a cluster of Intel dual processor nodes–T. Boku et. al compared hybrid PC with pure MPI through solving the smooth particle applied mechanics (SPAM) problem[9)]. The above-mentioned studies revealed that in most cases, hybrid PC is inferior to pure MPI despite its three main advantages: (1) low communica-

† Graduate School of Information Systems, University of Electro-Communications

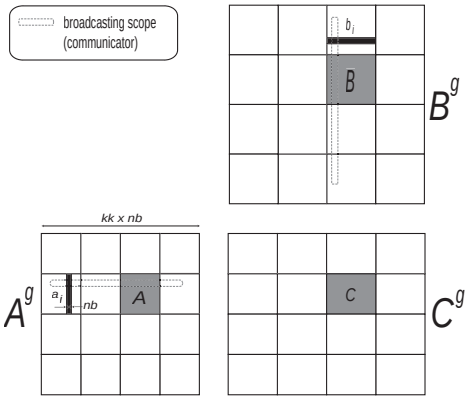


Fig. 1 The SUMMA algorithm

tion cost, (2) dynamic load balancing capability, and (3) coarse-grain communication capability[9]. The poor performance of the fine-grain hybrid PC model is primarily due to its poor intranode OpenMP parallelization efficiency[8] resulting from an extremely low cache hit ratio[9].

Then, we proposed an enhanced hybrid version, the hybrid TC model[1),2)], which was also discussed by G. Wellein et. al[10] and R. Rabenseifner et. al[11),12)]. Finally, we proposed a middle grain size approach that allows hybrid TC to achieve an impressive performance on different platforms in various types of experiments[3].

This paper improves the hybrid TC model by proposing RTA that is more effective than the middle grain size approach. Moreover, the paper also provides formulas to prove the outperformance of the hybrid TC model and to predict its performance according to different circumstances.

3. Distributed Matrix Multiplication

3.1 SUMMA

In this study, we focus on performing a distributed matrix multiplication operation:

$$C^g \Leftarrow \alpha \quad op(A^g) \cdot op(B^g) + \beta C^g, \qquad (1)$$

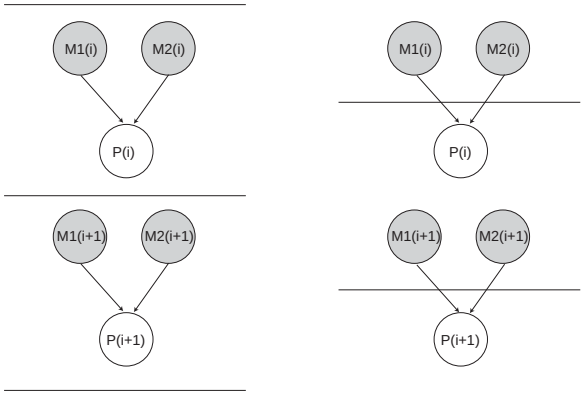
where $op(X) = X$, X^T or X^H; A^g, B^g, and C^g are global matrices of sizes $m \times k$, $k \times n$, and $m \times n$, respectively; α and β are scalars. In this paper, we focus on the case $op(X) = X$, but the idea can easily be extended to other cases.

The scalable universal matrix multiplication algorithm (SUMMA)[13] is one of the most effective algorithms for this problem. In comparison with other strong candidates like PUMMA[14] or DIMMA[15], it exhibits a comparable performance and high scalability. Moreover, SUMMA is simple and straightforward. Due to these advantages, SUMMA is used as the basic for developing a hybrid TC solution with RTA.

In SUMMA, processes are mapped onto an $nprows \times npcols$ process grid. Global matrices A^g, B^g, and C^g are split into $nprows \times npcols$ equal submatrices. A process stores the local submatrices A, B and C that are coressponding to its location in the process grid. Then, the local matrices A and B are divided into kk blocks of columns and kk blocks of rows, each of size nb. Figure 1 shows the data distribution corresponding to a 4×4 process grid. Shaded cells A, B, and C are the submatrices distributed to a process that is located in the second row, third column of the process grid.

```
C = βC
for (i = 0; i < kk; i++) {
  bcast(a_i, row_comm);
  bcast(b_i, col_comm);
  C += α(a_i · b_i);
}
```

Fig. 2 Pure MPI SUMMA pseudo-code.



(a) Before loop reconstruction. (b) After loop reconstruction.

Fig. 3 Elimination of dependency by loop reconstruction.

Figure 2 shows the pseudo-code of the algorithm, in which function bcast($data, comm$) broadcasts $data$ over the communicator $comm$.

3.2 Elimination of Dependency

Hybrid TC requires independent communication and computation tasks, which can be executed simultaneously by different threads of a node. The pure MPI task schedule, two iterations of which is shown in Figure 3(a), does not satisfy this requirement. In the figure, shaded and non-shaded circles imply communication and computation tasks, respectively, descriptions of which are shown in Table 1. The computation task P(i) cannot be executed until the completion of communication tasks M1(i) and M2(i).

To eliminate the dependency, we reconstruct the loop such that a new iteration includes the computation part of the original i^{th} iteration and the communication part of the $(i+1)^{th}$ iteration. By this reconstruction, we obtain a new iteration with no dependency between the communication and computation parts. The result of the reconstruction is shown in Figure 3(b).

Table 1 SUMMA task list.

No.	Description	Type
M1(i)	bcast(a_i, row_comm)	communication
M2(i)	bcast(b_i, col_comm)	communication
P(i)	C += $\alpha(a_i \cdot b_i)$	computation

```
#pragma omp parallel default(shared) private(i)
{
  #pragma omp master
  {
    M1(i + 1);
    M2(i + 1);
  }
  #pragma omp for schedule(dynamic) nowait
  for (j = 0; j < ngrains; j + +)    P(i)(j);
}
```

Fig. 4 Hybrid TC SUMMA pseudo-code for an iteration.

Table 2 Task assignation efficiency.

Variation	MFlops	Percentage of MPI
MPI	4799	100%
Cell based	3653	76.12%
Colunm based	4272	89.02%
RTA	4762	99.23%

3.3 Hybrid TC Solution for SUMMA

Pseudo-code for a single iteration of the hybrid TC SUMMA solution is shown in Figure 4. The communication tasks of a node are executed by the master thread. The computation task P(i) of an iteration is split into $ngrains$ grains, which are executed in parallel by the *#openmp for* pragma. The *schedule(dynamic)* option allows the master thread to join the computation group after the communication completion.

4. RTA Development and Efficiency

4.1 Task Assignation Variations

During the computation-communication overlap, hybrid TC exceeds pure MPI in performance. However, the model requires the dynamic task scheduling capability, which may decreases performance of hybrid TC outside of the overlap. This section discusses a method to avoid this performance degradation.

In each SUMMA loop iteration, a node has to perform a rank-kb update to its local matrix C:

$$C += \alpha\ (a_i \cdot b_i), \tag{2}$$

which should be split into $ngrains$ grains. Each grain updates a part of C by calling a well-tuned function dgemm() supplied by the goto-blas library[5]. Performance of dgemm() depends on the the level of efficiency it uses the cache. In general, with a large grain size, dgemm() is more capable to get a high cache hit ratio. However, the large grain size costs a large synchronization time, during which a processor has to wait for completion of the remaining.

We have examined several variations of task assignation that are shown in Figure 5, in which shaded areas present data to be accessed during a grain execution. The performance of the variations is shown in Table 2 in comparison with the MPI coarse grain model. Experiments are based on a matrix C of size 8000×8000 and kb is fixed to 112.

In the cell-based task assignation, which is shown in

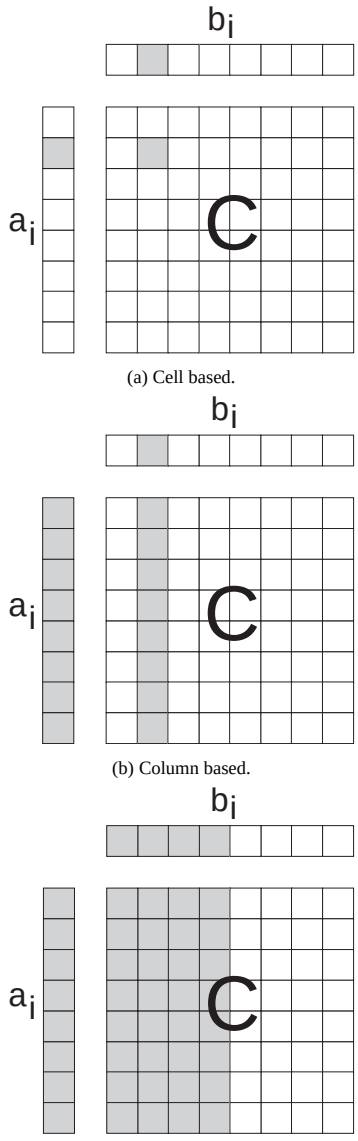


Fig. 5 Task Assignation Variations.

```
ngrains = 0;
remainsize = originalsize;
for (; remainsize > threshold; ngrains++) {
  grain[ngrains].size = remainsize/2;
  remainsize -= grain[ngrains].size;
}
grain[ngrains++]=remainsize;
```

Fig. 6 Process of RTA pseudo-code.

Figure 5(a), each grain updates a cell of size $nb \times nb$. This can be considered as a fine grain size model. The synchronization time is minimal but the overall performance is fairly poor (76.12% of pure MPI). Then, we expand the grain size to improve the performance. Figure 5(b) shows the column-based task assignation, in which each grain updates nb columns of C. As expected, a performance improvement appears with 89.02% of MPI performance. The synchronization time is acceptable with a value of less than 1% of ex-

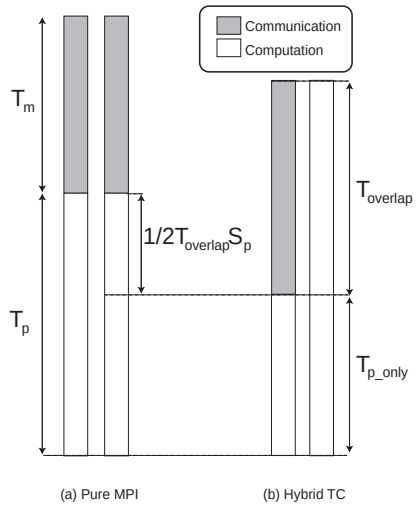


Fig. 7 Pure MPI and Hybrid TC comparision for a dual-processor SMP node

ecution time. However, this result is still far from the MPI performance.

Finally, Figure 5(c) shows the *reducing-size task assignation* technique (RTA). With RTA, a grain size is defined by half of the remaining task size. However, if the remaining task size is smaller than a predefined threshold, it will not be split any more. By this assignation, at any point of time, a free thread is assigned with a maximum possible grain size, thereby achieving an acceptable performance. On the other hand, the size of the remaining task is large enough to limit the synchronization time. The pseudo-code of the RTA process is shown in Figure 6. The new task assignation technique results in a 99.23% of pure MPI performance for the hybrid TC model.

4.2 Hybrid TC and Pure MPI Comparision

Figure 7 shows activities of a dual-processor SMP node for the pure MPI and hybrid TC solutions. In the pure MPI solution, all the processors of a node work together. They execute communication or computation simultaneously. The execution time T_{MPI} is the sum of the communication time T_m and the computation time T_p, which usually depend on the problem size:

$$T_{MPI} = T_m + T_p \tag{3}$$

In hybrid TC, the master thread executes communication tasks first. Meanwhile, the remaining thread performs computation by executing a loop. After the communication completion, the master thread enter the computation loop. In matrix multiplication, the growth rate of T_p is O(n^3) while that of T_m is only O(n^2). In other words, T_m grows faster than T_p. In this study, we focus on the case that problem size is sufficiently large such that at the time of communication completion, there remains computation task. Consequently, the communication time is just the overlap time and is denoted by $T_{overlap}$. After $T_{overlap}$, both the processors execute computation task. The time that they perform computation together is denoted by T_{p_only}. The sum of $T_{overlap}$ and T_{p_only} forms the hybrid TC model execution time T_{TC}:

$$T_{TC} = T_{overlap} + T_{p\_only}. \tag{4}$$

Applying RTA during T_{p_only}, hybrid TC achieves an approximately similar performance as that of pure MPI. In constrast, hybrid TC can omits intranode communication, thereby somewhat gaining performance advantage. Further discussions are based on the assumption of the elimination of these two disadvantage and advantage of the hybrid model.

We define the communication speed-up S_m as the ratio between communication speeds per processor during $T_{overlap}$ of hybrid TC and T_m of pure MPI. The computation speed-up S_p is defined similarly. During $T_{overlap}$, an SMP node has better communication and computation speeds than the pure MPI model has. In other words,

$$S_m > 1 \quad and \quad S_p > 1. \tag{5}$$

In addition,

$$T_m = 1/2 \;\; T_{overlap} \times S_m, \tag{6}$$

and the time that pure MPI needs to perform the computation volume that hybrid TC performs during Tp_only is $1/2 \;\; T_{overlap} \times S_p$. Consequently, the difference between execution time of pure MPI and hybrid TC dT can be found by:

$$dT = \frac{1}{2} \times T_{overlap}(S_m + S_p - 2). \tag{7}$$

The performance speed-up S of hybrid TC is evaluated by

$$S = \frac{T_{MPI}}{T_{TC}} = 1 + \frac{dT}{T_{TC}}, \tag{8}$$

or

$$S = 1 + \frac{1}{2} \times \frac{S_m + S_p - 2}{1 + \frac{T_{p\_only}}{T_{overlap}}}. \tag{9}$$

Equations (6), (7), and (9) define the time saved and the speed-up of hybrid TC through T_m, S_m, and S_p. In general, T_m increases with the problem size, which also increases $T_{overlap}$ and dT. This implies that as the problem size increases, the time saved by hybrid TC also increases. However, the ratio between T_{p_only} and $T_{overlap}$ also increases then, which results in a degra-

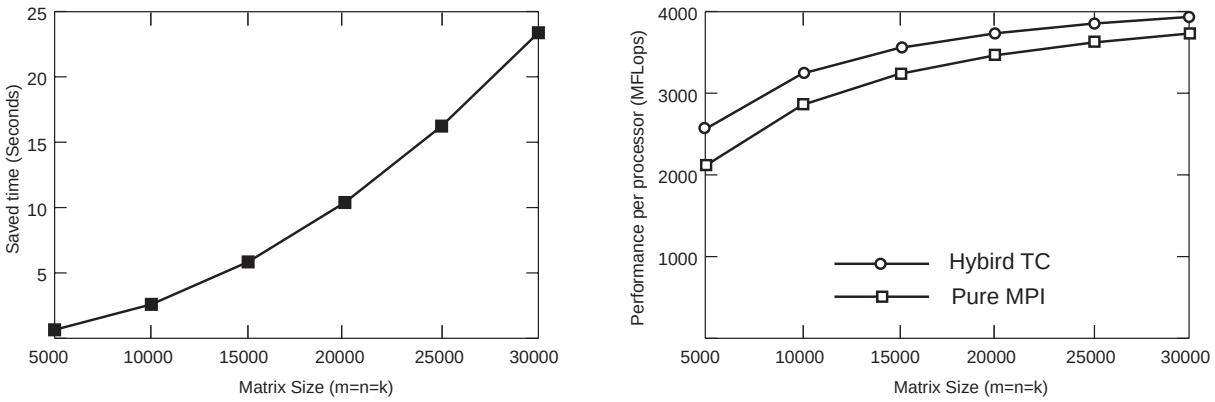


Fig. 8 Experimental results.

Table 3 Predicted saved time dT with different problem sizes.

m=n=k	T_m (sec.)	Expected dT (sec.)
5000	1.72	0.65
10000	6.85	2.59
15000	15.69	5.94
20000	27.80	10.53
25000	43.56	16.50
30000	62.35	23.62

dation of hybrid TC speedup S.

5. Experimental Results

Preliminary examination results show that when the sizes of local matrices A, B, and C are sufficiently large to guarantee stability of the local matrix multiplication function $dgemm$, the communication and computation speed-ups of the hybrid TC model during $T_{overlap}$ are relatively stable for different problem sizes.

$$S_m \approx 1.32; \quad S_p \approx 1.18$$

Experimental results for different problem sizes are shown in Figure 8. Square and equal size matrices A^g, B^g, and C^g are considered. The block size nb is fixed to 112. The values of m=n=k vary between 5000 and 30000, which are sufficiently large to stabilize the local goto-blas $dgemm$ and small enough to fit the memory limit. The results are in good agreement with those obtained by using the formulas given by Equation (7) and shown in Table 3, although there is a small disparity due to measurement error.

The left chart of Figure 8 shows the time saved by the hybrid solution in comparision with the pure MPI one. As expected, the saved time increases together with the problem size.

The right chart shows the absolute performance per processor. The hybrid TC model always exhibits a better result. However, the distance between the two performance lines reduces gradually. At m=n=k=5000, the difference is approximately 21.7% (458 MFlops). At m=n=k=30000, it decreases by 5.8% (216.3 MFlops). This result can be exlained by Equations (7) and (9) together with the nature of the matrix multiplication. The growth rate of T_p is O(n^3) while that of T_m is only O(n^2). Consequently, the growth rates of dT and T_{TC} are O(n^2) and O(n^3), respectively. This implies that the increase in dT is slower than that in T_{TC}, and S becomes smaller with an increase in n.

6. Discussions: the Asynchrnonous MPI Model

Performance enhancement of hybrid TC over pure MPI results by the computation-communication overlap. In this section, we propose initial ideas of another programming approach, the asynchronous MPI model, which is also capable of generating these overlaps. Figure 9 shows activities of the two processors of a dual-processor SMP node running an asynchoronous MPI solution. Like pure MPI, each processor of a node runs an MPI process. OpenMP is not invoked. Tasks performed by the two processes of a node need to be rescheduled such that the even processor performs its internode communication first, then computation; the odd processor executes in a reverse order: computation followed by internode communication. Intranode communication should be performed simultaneously by both processes.

Asynchronous MPI has some strong points over the hybrid TC. The most important point is the retaining of the MPI computation pattern. The two processors of a node have separated computation tasks and thereby, the effective coarse-grain size approach becomes available. The problem of grain-size does not exist anymore.

However, the model also has obstacles. For example, it requires a global communication arrangement. In the model, an even process can perform communication operations only with other even processes from other nodes. Therefore, every internode communication operation needs to be reorganized to satisfy this requirement. In addition, the asynchronous model retains the intranode communication, which is omitted with hybrid TC.

By the above analyses, when the communincation rearrangement problem is solved, asynchronous MPI

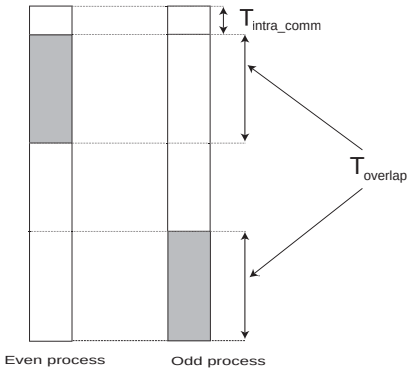


Fig. 9 Asynchronous MPI for a dual-processor SMP node

is expected to achieve a similar performance as that of hybrid TC.

7. Conclusions

This paper proposed a Reducing-size Task Assignation technique (RTA), an effective method to assign tasks dynamically to OpenMP threads during a hybrid TC solution. Using the method, hybrid TC can overcome the poor intranode computation parallelization performance, which is the main obstacle of hybrid models. Thereby, RTA allows hybrid TC to overcome pure MPI in overall performance. The formulas for evaluating the increase of performance are also remarkable. Beside proving performance advangtage of hybrid TC over pure MPI, they can predict the behavior of a cluster in various circumstances.

For problems in which the computation and communication time have different growth rates (e.g., SUMMA), performance enhancement decreases together with an increase in the problem size. However, the problem size is limited by the memory size of the system, and the degradation of the performance improvement also has a limit. For problems in which the computation and communcation time have the same growth rates (e.g., the NAS CG problem), we expect a constant speedup of hybrid TC over pure MPI.

Though the analyses and experiments in this paper are mainly for a dual-processor cluster, the conclusions are expandable to other clusters with more processors per node. In this case, the formula to determine the grain size should be changed to $1/nppn$ instead of 1/2 of the remaining task size, with $nppn$ is the number of processors per SMP node.

We also shows initial ideas for the asynchronous MPI, which has certain strong points over hybrid TC and is expected to achieve the same performance as hybrid TC with RTA does.

For future study, beside applying the RTA technique to the HPL benchmark, we would like to run the experiments on larger systems to confirm scalability of the technique. Besides, we would like to continue to develop the asynchronous model. Performance and required programming efforts of the two approaches will be analized to find the most effective programming model for SMP PC clusters.

Acknowledgment

This research is partially supported by the Nippon Foundation of the Japan Science Society (JSS) No.17-251.

References

1) T. Q. Viet, T. Yoshinaga, and M. Sowa. A Master-Slaver Algorithm for Hybrid MPI-OpenMP Programming on a Cluster of SMPs. *IPSJ SIG notes 2002-HPC-91-19*, 2002, 107-112.
2) T. Q. Viet, T. Yoshinaga, B. A. Abderazek, and M. Sowa. A Hybrid MPI-OpenMP Solution for a Linear System on a Cluster of SMPs. *Proc. Symposium on Advanced Computing Systems and Infrastructures*, 2003, 299-306.
3) T. Q. Viet, T. Yoshinaga, B. A. Abderazek, and M. Sowa. Construction of Hybrid MPI-OpenMP Solutions for SMP Clusters. *IPSJ Transactions on Advanced Computing Systems*, *4*6(SIG 3), 2005, 25-37.
4) MPICH Team. MPICH, a Portable MPI Implementation. http://www-unix.mcs.anl.gov/mpi/mpich/.
5) K. Goto. High-Performance BLAS by Kazushige Goto. http://www.cs.utexas.edu/users/flame/goto/.
6) F. Cappello and O. Richard. Intra Node Parallelization of MPI Programs with OpenMP. *TR-CAP-9901*, technical report, http://www.lri.fr/~fci/goinfreWWW/1196.-ps.gz, 1998.
7) F. Cappello, O. Richard, and D. Etiemble. Investigating the Performance of Two Programming Models for Clusters of SMP PCs. *Proc. High Performance Computer Architecture*, 2000, 349-359.
8) F. Cappello and D. Etiemble. MPI versus MPI+OpenMP on IBM SP for the NAS Benchmark. *Proc. Supercomputing 2000*, 2000.
9) T. Boku, S. Yoshikawa, and M. Sato. Implementation and Performance Evaluation of SPAM Article Code with OpenMP-MPI Hybrid Programming. *Proc. European Workshop on OpenMP 2001*, 2001.
10) G. Wellein, S. Hager, A. Basermann, and H. Fehske. Fast Sparse Matrix-Vector Muliplication for TeraFlop/s Computers. *Proc. Vector and Parallel Precessing*, 2002.
11) R. Rabenseifner and G. Wellein. Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures. *The International Journal of High Performance Computing Application*, *17*(1), 2003, 49-62.
12) R. Rabenseifner. Hybrid Parallel Programming: Performance Problems and Chances. *Proc. The 45th CUG (Cray User Group) Conference 2003*, 2003.
13) R. A. van de Geijn and J. Watts. SUMMA: Scalable Universal Matrix Multiplication Algorithm. *LAPACK Working Note 99*, technical report, University of Tennessee, 1995.
14) J. Choi, J. J. Dongarra, and D. W. Walker. PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers. *Concurrency: Practice and Experience*, *6*(7), 1994, 543-570.
15) J. Choi. A Fast Scalable Universal Multiplication Algorithm on Distributed-Memory Concurrrent Computers. *Proc. IPPS'97*, 1997.