Performance Analysis of PC-CLUMP based on SMP-Bus Utilization

Taisuke Boku* Ken'ichi Itakura** Shigehiro Yoshikawa*

Masaaki Kondo† Mitsuhisa Sato‡

*Institute of Information Sciences and Electronics, University of Tsukuba
*Center for Computational Physics, University of Tsukuba
†Research Center for Advanced Science and Technology, University of Tokyo
‡Real World Computing Partnership
{taisuke,itakura,yoshi,kondo,msato}@arch.is.tsukuba.ac.jp

Abstract

PC-CLUMP (Cluster of Multiprocessor) is one of the most cost-effective commodity-based platforms for HPC applications. The increasing number of CPUs per SMP node realizes very compact system size and very low price on the network interface per processor keeping the number of CPUs in the system. However, the performance of SMP-bus on such an SMP-PC node is relatively poor compared with that of SMP-Workstations. Therefore, the application program must achieve a high cache-hit ratio to reduce SMP-bus access for higher performance.

In this study, we analyze the performance of PC-CLUMP based on the SMP-bus access ratio on several benchmarks as representatives of HPC applications. We measure the ratio of SMP-bus access per total memory access instructions utilizing the performance counters equipped with modern CPUs. From the result of analysis, we can estimate the best numbers of CPUs/node and nodes/system.

Moreover, we also confirm the combination of parallel descriptions with OpenMP and MPI is very suitable for programming on PC-CLUMP while there exist several problems on data-process mapping which are related to cache-hit ratio of the applications.

1 Introduction

PC-CLUMP is one of the most cost-effective commodity-based platforms for HPC applications. Pentium-Xeon series make increase the number of CPUs on an SMP-bus up to 16. It realizes a very compact system size and causes less price/performance in such a system for applications with high cache-hit ratio (or low SMP-bus access ratio). For typical HPC applications, however, the cache-hit ratio is not always high and the high SMP-bus access ratio degrades the performance per node dramatically when the number of CPUs increases. On the other hand, the number of nodes which can be connected to a switch has been constantly increased for 100-base Ethernet with keeping the price/port very low. However, the price of interface cards for faster network like Myrinet or Gigabit Ethernet is still relatively expensive compared with the price of single CPU. In this sense, it makes the total price of the system low to enlarge the number of CPUs/node.

Under the above situations, all three major factors to determine the total price/performance of PC-CLUMP; the price of nodes, the price of networks and the performance of nodes are the function of the number of total CPUs and nodes. We analyze how the number of CPUs/node effects on the total system performance as well as the number of nodes does. For this purpose, system performance counters of Pentium-II Xeon are utilized to measure accurate time consumption and SMP-bus access ratio for detailed program sections including inter-node communication parts. We have selected two major benchmarks, Linpack and NAS Parallel Benchmark Kernel-CG as typical applications with opposite characteristics on cache-hit ratio.

As a programming model for PC-CLUMP, we tried three types: MPI-ONLY, Pthread+MPI and OpenMP+MPI. We mainly compare the performance by MPI-ONLY and Pthread+MPI. In OpenMP+MPI, the behavior of threads on a node is almost the same as Pthread+MPI. In several cases, however, the weak thread-control feature of OpenMP makes the performance degraded compared with well-controlled explicit Pthread programming. We consider the condition and methodology to solve this problem.

2 COSMO - PC-CLUMP test bed

COSMO (Cluster Of Symmetric MultiprOcessor) is a test bed for our research on PC-CLUMP performance analysis. COSMO consists of four SMP-PC nodes each of them consists of four Pentium-II Xeon processors. [1] Each node of COSMO is DELL PowerEdge 6300 PC-Server with 4-way SMP architecture. The CPU frequency is 450MHz and L2 Cache size for each CPU is 512KB. Main memory size per node is 2GB (EDO-DRAM). The inter-node network is 100base-TX Ethernet with switch.[2]

Operating System of COSMO is Linux 2.2.10 for SMP architecture. Libraries for thread and MPI programming are LinuxThreads (Pthreads)[3] and MPICH-1.1.2[2], respectively. RWCP Omni-1.1 OpenMP compiler[6] is also available for C and Fortran77.

3 Hybrid Programming

To combine shared-memory and distributed-memory programming paradigms on PC-CLUMP, we introduce three programming methods.

MPI-ONLY: All processes communicate with each other by MPI functions even they are on the same SMP node.

Pthread+MPI: All threads communicate with each other on a shared address space by Pthread feature, and a representative thread communicates with someones on other nodes by MPI functions.

OpenMP+MPI: On-node parallelism is described with OpenMP pragma, and a representative thread on a node communicates with someones on other nodes by MPI functions.

Here in Pthread+MPI and OpenMP+MPI, only one thread per node should execute MPI functions because general MPI functions may not be *thread-safe*.

4 Performance Evaluation

4.1 SMP-Bus Access Ratio

On recent CPUs including Intel Pentium-Xeon, several counters which can be easily accessed by user-mode are provided to measure the system performance. In this research, we utilize two counters on Pentium-Xeon:

DATA_MEM_REFS: The number of data memory references.

BUS_TRAN_ANY: The number of bus access by any memory access. [3]

From these counters, we define the **SBAR** (SMP-Bus Access Ratio) as the ratio of them. The count of data memory access can be estimated from the source code and algorithm of the application, and it is not influenced by programming paradigms. Therefore, we can evaluate the programming methodologies based on SBAR. Moreover, we also measure the program execution time by clock counter very accurately.

4.2 Benchmark Programs

We selected two benchmarks in this study.

Linpack[4]: Two matrix sizes (1000×1000 and 3000×3000) are examined. Hereafter, they are represented as $M = 1000$ and $M = 3000$ for short.

Kernel-CG[1]: Two problem sizes (Class-A and Class-B) are examined.

In Linpack, we use an outer-product method and cache blocking to utilize the data access locality for high cache-hit ratio. On the other hand, the most time consuming part of Kernel-CG is a multiplication of the sparse matrix and vector, where each data of the matrix is scanned just once per iteration and it is hard to achieve high cache-hit ratio. We selected them as problems with opposite characteristics on cache-hit ratio.

In the evaluation, we vary the number of CPUs per node (P) and the number of nodes (N). In MPI-ONLY programming, the total number of processes is $P \times N$ while the number of processes is N in Pthread+MPI and OpenMP+MPI. In any case, MPI processes are cyclically mapped onto nodes. In Pthread+MPI and OpenMP+MPI, P is controlled by specifying the number of threads. Both Pthread+MPI and OpenMP+MPI programs are written based on MPI-ONLY version, and it is the same as the original sequential program when $N = 1$ and $P = 1$. The basic performance of sequential version is shown in Table 1.

Hereafter, we describe the performance of parallelized versions as the speed-up ratio to the sequential ones. In these figures, "Number of processors"

[1] The last node of these four nodes consists of four Pentium-III Xeon processor instead of Pentium-II Xeon due to lack of P-II Xeon at the installation time.

[2] There is another inter-node network with 800Mbps Myrinet, however it is not used in this study.

[3] It includes not only bus accesses by data reference but also that by instruction cache-miss. There is another counter of bus access only by data reference, however it cannot count the bus access by cache-hit on memory write causing invalidation on SMP protocol. Since we are pointing all bus access and traffic, we use the former one.

Table 1: Processing time by the sequential version

Benchmark	Time (sec)
Linpack $M = 1000$	9.15
Linpack $M = 3000$	289.54
NPB kernel CG Class-A	82.76
NPB kernel CG Class-B	4146.94

shows $P \times N$, and each curved line represents a certain number of N for various P.

The basic behavior of threads are the same both in Pthread+MPI and OpenMP+MPI, then we mainly compare the performance of MPI-ONLY and Pthread+MPI in the following few sections. On the performance differences between Pthread+MPI and OpenMP+MPI, we describe later.

5 Results and Discussions

5.1 Linpack

First we show the poor results by MPI-ONLY in Figures 1 and 2 for matrix sizes $M = 1000$ and $M = 3000$, respectively.

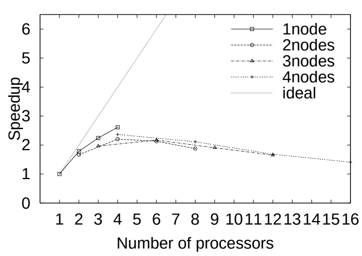

Figure 1: Linpack ($M = 1000$, MPI-ONLY)

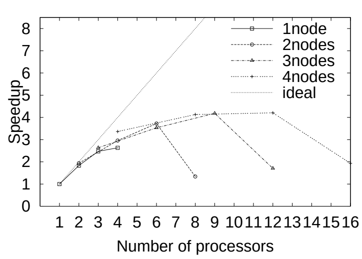

Figure 2: Linpack ($M = 3000$, MPI-ONLY)

It is shown that the scalability of MPI-ONLY version of Linpack is very low, especially for small problem size. To analyze the drastic performance degradation when $P = 4$ on $M = 3000$ case, we examine SBAR for MPI-ONLY and Pthread+MPI on Linpack. The results are shown in Table 2 (**MO** means MPI-ONLY and **PM** means Pthread+MPI).

It is shown in the table that SBAR for $P = 4$ is drastically increased for $M = 3000$. It can be estimated due to an additional process for MPICH on each node to manage the inter-node communication. We also checked the CPU-id for each process during execution, and confirmed the process migrations frequently occur when $P = 4$ by lack of free CPU. As a result, the primitive high cache-hit ratio of this benchmark is interfered by process migration, and the performance is degraded.

Table 2: SBAR for Linpack (%)

#N	#P	$M = 1000$		$M = 3000$	
		MO	PM	MO	PM
1	1	1.24	1.24	1.56	2.64
1	2	1.22	1.17	1.68	2.51
1	3	1.23	1.07	1.80	2.57
1	4	1.22	0.99	2.04	2.43
2	1	1.21	1.21	1.59	2.10
2	2	1.29	0.87	2.03	1.72
2	3	1.12	0.64	1.68	1.51
2	4	1.29	0.49	5.18	1.31
3	1	1.18	1.18	1.59	2.32
3	2	1.07	0.66	1.92	1.65
3	3	1.01	0.43	1.59	1.20
3	4	1.31	0.28	5.81	1.13
4	1	1.15	1.15	1.59	2.09
4	2	1.04	0.58	1.91	1.44
4	3	0.96	0.33	1.57	1.06
4	4	1.27	0.22	7.32	0.88

Figures 3 and 4 show the performance of Pthread+MPI on $M = 1000$ and $M = 3000$, respectively. From the result, they achieves higher performance than MPI-ONLY. Especially, curves for $M = 3000$ seem very smooth where the increments of P and N are almost contiguous. It is because the coarse granularity of process reduces the relative overhead. From Table 2, SBAR of Pthread+MPI is much lower than that of MPI-ONLY, regardlessly to P. Moreover, SBAR seems to be decreased when P increases for certain N. We estimate the reason as follows. After the elimination for a block of columns, all threads wait in a barrier for the master thread finishing the communication. In this situation, most of threads are in a polling loop which increases the number of data memory read. Here, most of data access hits on the cache and the average SBAR may be decreased. To confirm this scenario, more detailed partial SBAR examination is required.

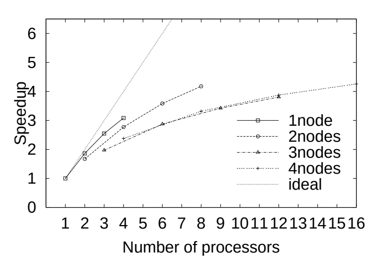

Figure 3: Linpack ($M = 1000$, Pthread+MPI)

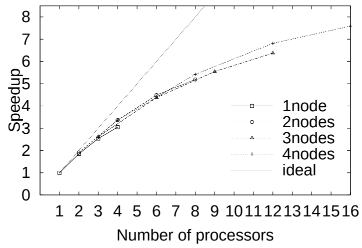

Figure 4: Linpack ($M = 3000$, Pthread+MPI)

5.2 Kernel-CG

Figures 5 and 6 show the performance results of Kernel-CG on MPI-ONLY for Class-A and Class-B, respectively. They show the scalability for N is very well while that for P is very poor. Especially, we cannot get any gain for $N = 3$ and $N = 4$. To analyze it, we show SBAR for Kernel-CG in Table 3. Comparing them to those of Linpack, there exist about 2 to 6 times bus accesses. It causes the miserable bus bottleneck in a node.

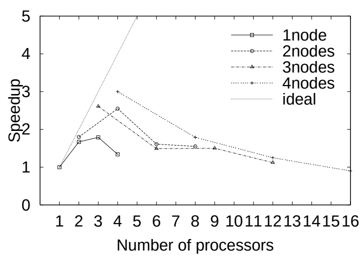

Figure 5: Kernel CG (Class-A, MPI-ONLY)

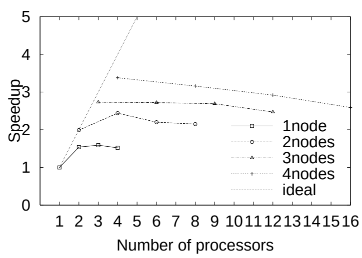

Figure 6: Kernel CG (Class-B, MPI-ONLY)

Figures 7 and 8 show the performance results of Pthread+MPI. The scalability for P is much better than MPI-ONLY, however the performance gain with the increasing P is still poor.

We can find an interesting phenomenon on SBAR for MPI-ONLY and Pthread+MPI in Table 3. There is no large difference between two methods while the performance curves are drastically different. It shows that we cannot evaluate the performance curve based on SBAR only. For detailed analysis, we examined the breakdown of SBAR into two parts; main computation (matrix-vector multiplication) and others (including large part of communication time). As a result, in the former part, there is no

Table 3: SBAR for Kernel-CG (%)

#n	#p	Class-A		Class-B	
		MO	PM	MO	PM
1	1	2.50	2.50	5.49	5.76
1	2	2.58	2.53	6.01	5.58
1	3	2.67	2.55	5.64	5.58
1	4	2.38	2.57	5.79	5.63
2	1	2.50	2.51	5.79	5.67
2	2	2.11	2.53	5.59	5.65
2	3	2.37	2.55	5.60	5.75
2	4	1.92	2.57	5.72	5.67
3	1	2.42	2.42	5.49	5.56
3	2	2.11	2.48	5.41	5.58
3	3	1.75	2.52	5.56	5.71
3	4	1.51	2.54	5.27	5.68
4	1	2.37	2.34	5.40	5.55
4	2	1.68	2.41	5.52	5.44
4	3	1.36	2.50	5.01	5.53
4	4	1.17	2.51	5.05	5.57

difference among two methods both in SBAR and execution time. On the other hand, in latter part, some reversed phenomena were found, that is, SBAR of MPI-ONLY is less than that of Pthread+MPI while the execution time of MPI-ONLY is much larger than that of Pthread+MPI.

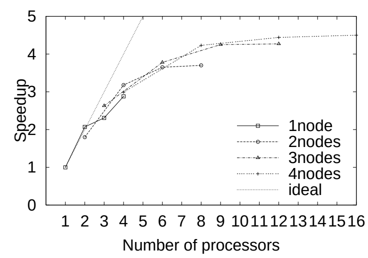

Figure 7: Kernel CG (Class-A, Pthread+MPI)

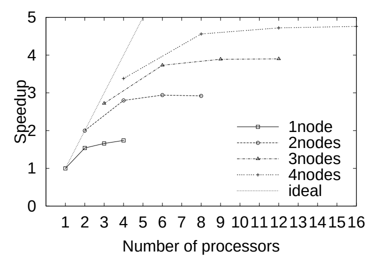

Figure 8: Kernel CG (Class-B, Pthread+MPI)

For communication parts, our estimation based on SBAR does not work well, because there are several complicated factors and some of them confuse us. For instance, as mentioned on Linpack, a polling loop for synchronization or communication ordinarily increases the number of memory reference but bus access, then very low SBAR is observed compared to a long execution time. Therefore, we must examine the detailed breakdown of SBAR in several part of the program and distinguish figures whether or not they are effective to the performance estimation.

5.3 Memory Bandwidth

The performance evaluation so far, we have not used memory interleaving facility of Intel 450NX chipset equipped on COSMO. When the number of threads is increased in a node, up to 4-way memory interleaving provides four times bandwidth. Using this feature, there is no performance improvement on Linpack since its SBAR is very low. On the other hand, Kernel-CG Class-B achieves about 50% of performance gain from no-interleaving case. For applications with high SBAR, it is one of the primary solutions to enhance the performance even with high-cost in memory.

5.4 Thread-Data Mapping in OpenMP

OpenMP+MPI is very suitable for hybrid programming on PC-CLUMP in the combination with MPI. The programmer is released from bothersome thread programming. We re-programmed all benchmarks so far from MPI-ONLY version with a little work load. MPI functions are mutually-excluded by *master* pragma for safe communication. They achieved about over 90% performance of Pthread+MPI version. SBAR for OpenMP+MPI is also almost the same as that of Pthread+MPI.

In several cases, however, SBAR for computation part is greater than Pthread+MPI. Linpack's basic SBAR is very low, thus this effect is stronger than Kernel-CG. There are two major problems:

Data mapping: In Linpack with Pthread+MPI, we programmed that each data block is always accessed by the same thread. In Pthread+MPI program, there is no redundant thread, and a thread is always executed on the same CPU. Therefore, this data locality care works well to achieve high cache-hit ratio. With OpenMP, it is difficult to describe naturally without explicit thread controlling features like *omp_get_thread_num()*.

Thread migration: In OpenMP, logical threads are mapped onto physical threads, then they are executed on physical CPUs. The mapping of logical to physical threads is not fixed because the free-thread pool is ordinarily managed in certain manner like FIFO. Therefore, the final mapping of logical thread and CPU may vary resulting frequent cache miss-hits.

The first problem is hard to avoid since it depends on the target application and the variation of loop assignment in OpenMP is limited. On the other hand, the second problem will be relatively easily solved to introduce some mapping control feature on current OpenMP specification, for instance, some new pragma like *lastmapped* which enforces a logical thread to be mapped on the same physical thread as last time.

6 Conclusions

In this study, we examined the performance of PC-CLUMP for several HPC benchmarks based on SMP-bus access ratio analysis. This ratio is one of the most important factors on the performance of an SMP node. The performance counter is a strong tool for this analysis, however detailed breakdown on the analysis is required to understand the behavior of a thread and its effect on the execution time. A hybrid programming model with the combination of Pthread and MPI always achieves a higher performance than MPI-only programming. OpenMP is a useful paradigm for hybrid programming, however in several cases, the weakness of thread mapping and control degrades the cache-hit ratio.

PC-CLUMP system which provides well-balanced number of CPUs/node, number of nodes/system and suitable network connections will be a very cost-effective HPC system. To achieve sustained performance, more detailed studies are required.

Acknowledgment

The authors truly appreciate TEA group members for their valuable discussion and suggestions. This study is partly supported by the Grant-in-Aid of Ministry of Education, Science and Culture (C-12680327).

References

[1] D. Bailey, et.al, "The NAS Parallel Benchmarks", Technical report, NAS, March 1994, RNR-94-007.

[2] W. Gropp and E. Lusk, "Installation guide for mpich, a portable implementation of MPI", Technical report, Computer Science Dept. University of Tennessee, 1994, Technical Report CS-94-230.

[3] X. Leroy, "The LinuxThreads library", `http://pauillac.inria.fr/~xleroy/linuxthreads/`.

[4] J. Dongarra, et.al, *LINPACK User's Guide*, SIAM, Philadelphia, PA, 1979.

[5] "OpenMP", `http://www.openmp.org/`.

[6] "RWCP Omni OpenMP Compiler", `http://pdplab.trc.rwcp.or.jp/Omni/home.html`.