ハイパフォーマンス
コンピューティング 91－19
（２００２．８．２２）

SMPクラスタにおけるハイブリッドMPI-OpenMPプログラミングのためのマスタースレーブアルゴリズム

タ クオク ヴィエト，吉永努，曽和将容
電気通信大学　大学院情報システム学研究科

SMPクラスタ上のMPIとOpenMPのハイブリッドプログラミングモデルのためのスレッド間通信モデルを提案する．本モデルでは，１つのマスタースレッドを設け，それがスレーブスレッドとの通信と計算の制御を行う．スレッド間でMPI通信することで，SMPノード間の通信と同期のオーバヘッドを削減する．実験の結果，マスタースレーブ並列モデルに適合する問題では，MPIのみを用いたプログラムに対して優位な性能を示すことがわかった．

A Master-Slave Algorithm for Hybrid MPI-OpenMP Programming on a Cluster of SMPs

Ta Quoc Viet, Tsutomu Yoshinaga and Masahiro Sowa
The Graduate School of Information Systems
University of Electro-Communications, Tokyo Japan
E-mail: viet@sowa.is.uec.ac.jp

In this paper, we evaluate a hybrid MPI and OpenMP programming model for a cluster of SMPs, and compare it with the pure MPI model. We suggest a "thread-to-thread" Master-slave communication method between SMP nodes. A thread is chosen to be the master, which controls both communication and computation tasks for all the others. This communication method can decrease communication time in comparing with the "process-to-process" one.

1. Introduction

MPI is a distributed memory programming model with explicit control of parallelism. Each process has its own memory space and run the same procedure (SPMD). Communication between any two processes is to be done by the explicit participation from both processes. In general, MPI is relatively scalable. We can apply MPI on both shared and distributed memory systems [8].

In contrast, OpenMP is a shared memory programming model. Parallel execution is performed and controlled by a combination of compiler directives, library routines and environment variables. Within OpenMP, "communication" can be performed via the shared memory variables, and nearly with no cost. However, unlike MPI, the OpenMP paradigm can be used only on shared memory systems, and is not scalable to the distributed memory systems. We do not take into account here "software shared memory systems", which have relatively poor performance with comparing to pure MPI [7].

Thus, to parallelize a solution on a cluster of SMPs, we need to use MPI. In this case, we can divide messages into two types: among the processes on the same SMP node, and among those on different SMP nodes. With regard to communication

between the processes on the same node, we expect that we can reduce the cost by using OpenMP, instead of MPI send and receive functions.

So we decided to explore the hybrid MPI-OpenMP model, and compare its performance to that of the pure MPI one. In the hybrid model, MPI is used to perform communication between nodes, and OpenMP is in charge with communication occurred within the same node. To confirm the idea, we build some hybrid algorithms to solve two example problems: "finding the n^{th} prime number", and "solving a dense linear equation system by LU decomposition". The results of the comparison between hybrid and pure MPI solutions allow us to make the conclusion that hybrid model has clear advantages over MPI model in certain circumstances.

In general, this paper discusses the benefits of a hybrid MPI-OpenMP solution on a cluster of SMPs. Section 2 describes related work on hybrid programming. Section 3 discusses principles of hybrid programming and suggests a new method for communication between the nodes, comparing it with the previous one. Section 4 describes the implementation of the hybrid model on solving the "n^{th} prime number" as well as the "dense linear equation system" problems. We present experimental results and discussions in section 5. Finally we conclude the paper with section 6, writing about our conclusions and future work.

2. Related work

Lorna Smith et al. showed a schematic representation of the hybrid programming model. Then they compared the two models by using them to write the code of "Game of Life" program. Their conclusion is that hybrid model is better than pure MPI only in some situations such as too fine grain size or memory limitation [1].

Frank Cappello et al. suggested the process-to-process communication method between SMP nodes for hybrid programming. Using this method, they rewrote several NAS benchmarks on hybrid mode. Their work showed that pure MPI in most cases gives a better result than hybrid one [3].

Jahed Djomehri et al. used the same communication method to solve "Overflow" problem by hybrid model. They also showed some cases that hybrid model could get better performance than pure MPI [4].

P.E.Strazdins et al. explained the parallel algorithm for solving a linear equation system [5]. We consulted their algorithm to solve the linear equation system.

T.Boku et al. explored another kind of hybrid parallel programming on PC-CLUMP (Cluster of Multiprocessors): MPI-Pthread. Their result shows that the MPI-Pthread hybrid model is even better than both pure MPI and MPI-OpenMP hybrid models [6].

3. Principles of MPI-OpenMP hybrid programming model

3.1 The way of creating parallelism

Figure 1 shows the schematic representation of the hybrid MPI-OpenMP programming model.

The idea of hybrid programming can be implemented by using a hierarchical model. For the best performance, this programming model should be physically correlative to the hardware. At the top level, parallelism is exploited by MPI. Each MPI process should be mapped on a SMP node of the cluster. These processes exchange information between each other by passing the messages via MPI functions. In the next level, each MPI process uses the OpenMP components (compiler directives, library routines or environment variables) to fork and join a set of OpenMP threads. Each thread is run on a single processor.

3.2 Communication between SMP nodes, process-to-process method

In some related works, authors suggested a simple method of communication between nodes [2, 3, 4].

Figure 1 also shows this method of communication. Each MPI process performs a task by "fork" into many OpenMP threads running on different processors. After finishing an assigned task, these threads will "join" to a single one, now is a process. Only processes (but not threads) exchange information with their comrade processes by passing messages. Communication phase is put outside of the fork and join phase. The "fork and join then communicate" structure can be duplicated as many times as needed. From now on, we call it "process-to-process communication method".

The main advantage of this method is simplicity. It is easy to build the algorithm, easy to manage the tasks and easy to debug.

However, this method cannot give us the best performance. The reason is that, during the time of MPI communication, all the processors in a node, excluding the one that actually perform the communication task, have nothing to do. In addition, all the OpenMP threads may not finish their tasks in "fork and join" period at the same time, and the fastest one has to waste time to wait slower ones. Due to these reasons, hybrid solutions in many cases give the worse performance than the model of pure MPI.

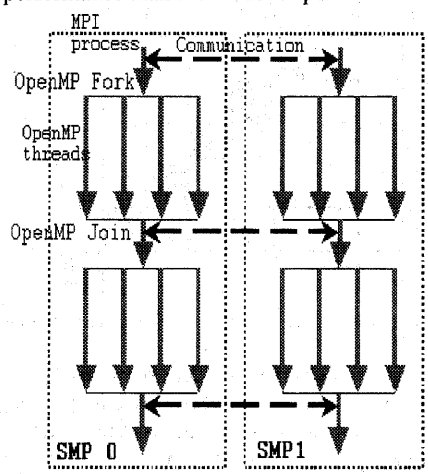

Figure 1. Schematic representation of the hybrid MPI-OpenMP programming model, using a process-to-process communication method.

3.3 Communication between SMP nodes, thread-to-thread method

To avoid the wasted time during communication, we suggest a new method, which is shown in figure 2. In our schema, SMP nodes communicate with each other by threads but not processes. When a thread, being run on a processor, carries out communication, other processors of the same node are free to execute their own duties. In other words, communication task are carried out between OpenMP "fork" and "join" points of time. In this paper, we call this as "thread-to-thread communication method".

To make the algorithm simple and easy to manage, we use a thread on a node as the master thread. All the threads on other nodes will communicate through that master one.

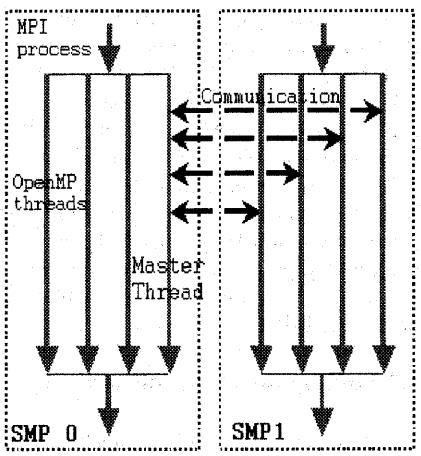

Figure 2. Communication between a master thread and slave threads

This approach is especially suitable for the computation of a Master–slave parallel model. The "communication master thread" is also the "computation master". It controls and schedules not only the communication but also the computation tasks.

To solve the problem what the threads will do during the communication time, we use the following method. Using the Master-slave model, the master thread sends tasks to the SMP nodes and receives results. In the first stage, it sends to each

slave SMP node several tasks. In the slave SMP nodes, there is a task-list to store unsolved tasks. OpenMP threads solve the tasks in parallel. When a task is finished, any thread can send the result back to the master, then receive a new task, and put it to the task list. During the communication time, other threads are still solving the next tasks, which are taken from the task list. Thus, we do not waste time for waiting the communication operation to be completed. After receiving enough results, the master thread sends a stop command to all the slaves.

4. Implementations of hybrid MPI-OpenMP model

4.1 Experimental environment

Our experiments are carried out on a cluster of two Sun Enterprise 3500 systems; each of them has eight UltraSPARC-II 336 MHz processors. The systems are connected by 100Mbps Ethernet cable. Each SMP node has 2 Gbs of memory.

Operating system running on the SMPs is Solaris 8. MPI library and environment are provided by Sun HPC Cluster Tools 4. To compile the programs, we use Sun Forte 6 Update 2 C compiler, which already supports OpenMP.

4.2 Finding the n^{th} prime number

_Algorithm: Sieve of Eratosthenes has been used with some enhancements. In detail, we firstly build a checklist, which is a list of the first prime numbers. Using this list, we can check primality of numbers not larger than square of the biggest member of the list. The set of positive integers is divided into many subsets of *size* numbers. These subsets are checked primality by using the checklist until the quantity of prime numbers received exceeds n. Finally we make a back-step to find the exact n^{th} prime number.

_Parallelism: We have used the Master-Slave algorithm. The master sends the beginning number of the subsets to the slaves. Then it receives the quantity of prime numbers as well as the largest one that the slaves found from the subset.

_Major functions built and used:

int *ListInitial(int *m*). Returns the list of the entire prime numbers not exceeding m This function is used to build the checklist.

int *CheckSumPrimeRange (int* *checklist*, int *bottom*, int *range*). Returns the quantity of prime numbers and the biggest among them.

With this problem, we can easily change the problem size by changing n. The thread-to-thread communication method is also very suitable in this case so that we expect good efficiency here.

4.3 Solving a linear equation system Ax=b by LU decomposing

Using the LU decomposing algorithm to solve the system, LU decomposing takes most of the execution time. By this reason, we take into account only the LU decomposing step here, the forward and backward substitution time is not included to the discussion.

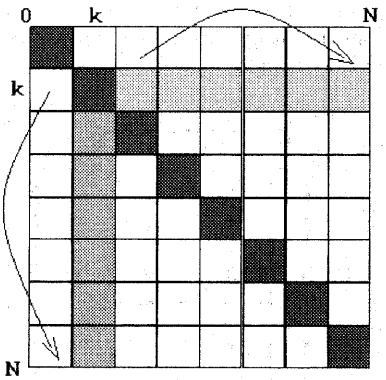

Figure 3. Algorithm for parallel LU decomposition

_Algorithm for LU decomposition: we use the Crout's algorithm with row pivoting [7, 10]. We have changed the order of finding the elements of L and the U triangular to parallelize the algorithm. Figure 3 shows the order that we use. At each step k, we firstly find all the members of column k of the lower triangular, then choose the pivot element, and finally find all the members of row k of the upper

triangular. In the figure, the elements to be found in step k are presented by shaded cells.

_Parallelism: Each MPI process stores the whole data of matrix A. At each step, the mission of finding column's elements or row's elements is divided into MPI processes, each of which is located on a single SMP node. At nodes, this piece of the mission is divided once more to the OpenMP threads. After finish the piece of the mission, MPI processes broadcast the result to all the nodes and update their own data.

In solution of the problem, we have not used the thread-to-thread method of information exchange yet. The nodes exchange information via the MPI processes, so that we do not expect here the best performance. We will optimize the solution by using thread-to-thread communication method in the future work.

5. Results and discussions

5.1 The nth prime number problem

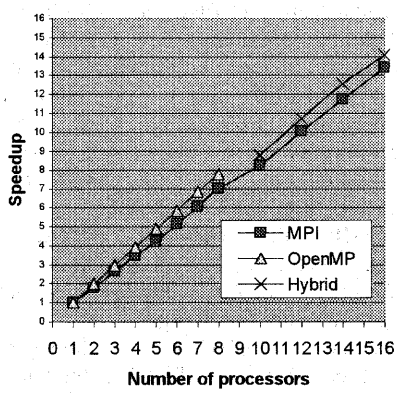

Figure 4. Speedups of pure MPI, OpenMP and hybrid models on solving the n^{th} prime number problem, n=30.000.000

Figure 4 shows speedups which are given by different models when we change the number of processors. Because of weak scalability, we can use OpenMP on a single node, up to eight processors only. With pure MPI, we are able to explore the performance up to sixteen processors. The "n^{th} prime number" problem can be parallelized very effectively, so that within a single SMP node, we can get a fairy good speedup for both OpenMP and MPI, very near to the ideal one. However, OpenMP is a little better than MPI. This is caused by the extra MPI sending and receiving cost. To compare hybrid model to pure MPI, we use the same number of processors on each SMP node. As we expected, using the thread-to-thread method of communication between nodes, we manage to avoid a lot of communication time. The hybrid model gives a result clearly better than the pure MPI one.

Figure 5 shows the speedups that we achieve when we use pure MPI and hybrid models, too. In this case, we use all 16 processors and change the problem size. The ideal speedup is 16, so that the speedup difference between the two models is significant.

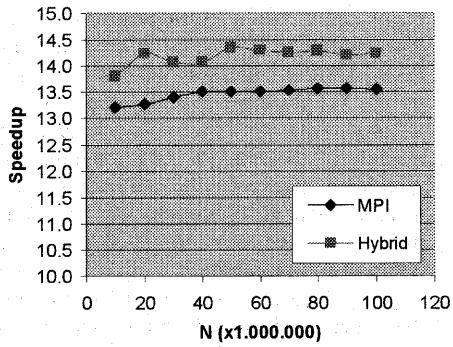

Figure 5. Speedups of hybrid MPI-OpenMP and pure MPI models on solving the n^{th} prime number problem, 16 processors.

5.2 LU decomposition

Figure 6 shows speedup of the hybrid model with different problem sizes. Solving this problem, we do not use the thread-to-thread communication method because of the difficulty on applying effectively the Master-Slave algorithm here. However, the speedup we gained is not so bad when we increase the problem size. We suppose that when we manage to apply the thread-to-thread communication method and use the suitable way of data distribution, we can still improve the performance.

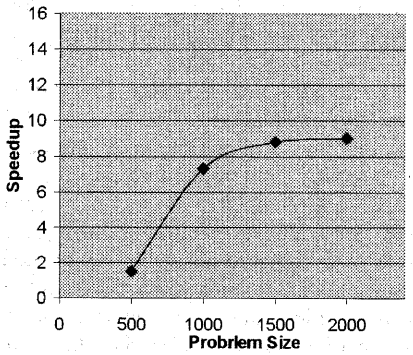

Figure 6. Speedup of hybrid model on LU decomposition, using a process-to-process communication method.

6. Conclusions

Results of the experiments allow us to conclude that, on a cluster of SMPs, in many cases, hybrid MPI-OpenMP programming model can give a better result than the pure MPI one, especially when it is possible to use the thread-to-thread communication method. And we think that this method is useful in solving a large number of problems. However the method is fairly complicated in programming, and some times we have to choose between effectiveness and simplicity.

In the next step, we would like to apply the thread-to-thread communication method on variety of problems. At first we will optimize the LU solution by using the block factorization algorithm, then compare the performance to the one that supplied by High Performance Linpack Benchmark (HPL) [9]. We would like to build a Hybrid Benchmark based on the above-mentioned HPL too.

We will also try to perform our experiments on many different environments.

Acknowledgement

This research is partially funded by Grants-in-Aid for Scientific Research of Japan Society for the Promotion of Science, Encouragement of Young Scientists-A, No.60210738.

References

[1] L.Smith and M.Bull: Development of mixed mode MPI/OpenMP applications. *Scientific Programming 9*, pp. 83-98 (2001).

[2] D.Etiemble: Mixed-mode Programming on Clusters of Multi-processors. *ECE 1755S-Parallel Computer Architecture and Programming*. http://www.eecg.toronto-.edu/de/Pa-06.pdf (2001)

[3] F.Cappello and D.Etiemble: MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. Presented ad *Supercomputing, Dallas* 2000, http://www.sc2000.org/tech-papr/papers/pap.pap214.pdf. (2000)

[4] J.Djomehri: Hybrid MPI+OpenMP Programming of an Overset CFD Solver and Performance Investigations. http://www-.nas.nasa.gov/Research/Reports/techreports.html (May 2002).

[5] P.E.Strazdins: High Performance Dense Linear System Solution on a Beowulf Cluster. http://cs.anu.edu.au/~Peter.-Strazdins/papers/ClustAlgBlk.pdf (2001)

[6] T.Boku, K.Itakura, S.Yoshikawa, M.Kondo, M.Sato: Performance Analysis of PC-CLUMP based on SMP-Bus Utilization. *Proceedings of 2002 International Conference on Parallel Processing in Electrical Engineering, Warsaw*, (Sep. 2002 to appear).

[7] OpenMP, the OpenMP Architecture Review Board, http://www.openmp.org/.

[8] Message Passing Interface Forum. http://www.mpi-forum.org/.

[9] A.Petitet, R.C.Whaley, J.Dongarra, A.Cleary: HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. http://www.netlib-.org/-benchmark/hpl/.

[10] J.Dongarra, I.S.Duff, D.C.Sorensen, H.A.van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. S.I.A.M (1998).