

Short Paper

Design and Implementation of the OpenMP Programming Interface on Linux-based SMP Clusters^{*}

TYNG-YEU LIANG, SHIH-HSIEN WANG[†], CE-KUEN SHIEH[†],
CHING-MIN HUANG AND LIANG-I CHANG

*Department of Electrical Engineering
National Kaohsiung University of Applied Sciences
Kaohsiung, 807 Taiwan*

[†]*Department of Electrical Engineering
National Cheng Kung University
Tainan, 701 Taiwan*

Recently, cluster computing has successfully provided a cost-effective solution for data-intensive applications. In order to make the programming on clusters easy, many programming toolkits such as MPICH, PVM, and DSM have been proposed in past researches. However, these programming toolkits are not easy enough for common users to develop parallel applications. To address this problem, we have successfully implemented the OpenMP programming interface on a software distributed shared memory system called Teamster. On the other hand, we add a scheduling option called Profiled Multiprocessor Scheduling (PMS) into the OpenMP directive. When this option is selected in user programs, a load balance mechanism based on the PMS algorithm will be performed during the execution of the programs. This mechanism can automatically balance the workload among the execution nodes even when the CPU speed and the processor number of each node are not identical. In this paper, we will present the design and implementation of the OpenMP API on Teamster, and discuss the results of performance evaluation in the test bed.

Keywords: Linux-based SMP, OpenMP, software distributed shared memory, Teamster, profiled multiprocessor scheduling

1. INTRODUCTION

OpenMP [1] is a standard shared memory programming interface which provides a set of directives such as parallel region, work sharing, and synchronization. Additionally, OpenMP applies several data scope attribute clauses, e.g. private or shared, in conjunction with these directives to explicitly direct the shared memory parallelism. When users wish to parallelize their problems, they need only add the proper directives and clauses at the front of the program blocks which are to be parallelized. Using the OpenMP compiler,

Received July 4, 2005; accepted April 11, 2006.

Communicated by Yau-Hwang Kuo.

^{*} This work was supported by the freeware project of the National Science Council of Taiwan, R.O.C., under project No. 93-2218-E-151-004.

the sequential programs can be automatically transformed into corresponding C source codes embedded with the OpenMP run time functions. These source codes can then be compiled by the GCC compiler to form multithreaded execution files by linking with the OpenMP run time library. Consequently, the OpenMP interface provides users with a straightforward interface for developing multithreaded programs on SMP machines.

In recent years, computational clusters [2] have emerged as a cost-effective approach for high performance supercomputing. Such systems unify individual resources located on LAN to form a single computational resource. From the point of user view, a computational cluster is regarded as a virtual main frame with a numerous number of processors while it is cheaper and more expandable and reliable than real main frames. Consequently, computational clusters have been exploited in many data-intensive researches such as bioinformatics, high physical energy, weather forecast, brain analysis and astronomy physics etc.

However, even though computational clusters offer many advantages, the growth of cluster-computing applications is still relatively slow. The principal reason for this appears to be that the existing programming toolkits such as MPICH [3] and PVM [4] are rather complicated since they require programmers to use explicit function invocations to perform data communication or work distribution functions. In contrast, software distributed shared memory (DSM) [5-9] systems allow users to make use of shared variables rather than message passing to write parallel programs in a distributed environment. When processes or threads access the same shared variables on different nodes, data consistency is automatically maintained by the DSM library. As a result, users can concentrate their efforts exclusively on developing their program algorithms, free of the need to consider data communication aspects. However, due to performance considerations, most modern DSM systems adopt weaker consistency protocols such as released [10], entry [11] or scope [12]. As a consequence, users must be aware of the particular consistency protocol used in their chosen DSM system and must establish appropriate data synchronization points within their programs if the application is to be executed as intended. For overcoming this consistency protocol problem, enabling the OpenMP programming interface on software DSM systems has emerged as a promising means. OpenMP programs not only minimize the programming complexity on computational clusters, but also extend the range of feasible cluster-computing applications since OpenMP programs developed on SMPs can be seamlessly applied to computer clusters.

As discussed above, we have previously developed a user-friendly software DSM system known as Teamster [13] for Linux-based SMP clusters. To further minimize the programming load of cluster users, the objective of this study is to enable the OpenMP programming interface on Teamster. Accordingly, this study develops an OpenMP compiler and a distributed OpenMP run time library for Teamster based on the Omni compiler and its run time library, respectively. In addition, a user-level thread library referred to as distributed Portable thread is introduced to minimize the parallelization overhead and to support thread migration for application adaptation and resource reallocation. Additionally, this study proposes a loop scheduling algorithm designated Profiled Multi-processor Scheduling (PMS) to resolve the load balancing problem for OpenMP programs. The efficiencies of the implemented OpenMP compiler and run time library are investigated, and the effectiveness of the PMS algorithm has been evaluated in this study. The experimental results reveal that the overhead of the run time library is very small and

demonstrate that the PMS algorithm is more effective than others in attaining a load balance.

The remainder of this paper is organized as follows. Section 2 provides an overview of Teamster and the Omni compiler. Sections 3 and 4 discuss the design considerations and implementation, respectively, of the OpenMP interface on Teamster. Section 5 presents the results of performance evaluation. Section 6 provides an overview of related studies. Finally, section 7 presents the conclusions of this study and our future work.

2. BACKGROUND

The aim of this study is to enable the OpenMP programming interface on Teamster for cluster computing. In order to simplify our work, the Omni compiler and the OMP run time library are adopted as the basis for the current implementation.

2.1 Teamster

Teamster [14] is a user-level DSM system built on Linux-Red hat 9.0 operating system. Its hardware platform is a cluster grouped by a set of Intel 80 × 86 PCs or SMP machines connected with Fast Ethernet network. This system supports transparency to users in data-distribution. Teamster constructs a single and global address space. With this global shared space, programmers can declare and allocate shared data by using the same user interface as using in shared-memory systems while they need to use additional annotations in the other DSM systems such as `tmk_alloc` and `tmk_distribute` calls in TreadMarks [15]. Moreover, Teamster supports multiple memory consistency protocols, i.e. sequential and eager-update released in the shared address space to make a compromise between transparency and minimization in data-consistency communication.

2.2 Omni Compiler

The Omni compiler [16] is one part of the RWCP Omni compiler developed to allow researchers to build a code transformer. In this study, the Omni compiler is used for transforming of OpenMP programs into multi-threaded programs. Basically speaking, the OpenMP programs are initially translated into X-object codes. These codes are then transformed into corresponding C source codes by means of the Exc Java toolkit. The C source codes are then compiled into executable binary code by using the GCC compiler and linking to the OMP runtime library, which is implemented using a kernel-level POSIX thread.

3. CONSIDERATIONS

Source code compatibility and program performance are two main considerations while implementing the OpenMP interface on Teamster. When source compatibility is maintained, users can execute their applications directly on a computer cluster once the source codes have been recompiled. However, to prevent performance degradation, the

program parallelization costs must be minimized. Furthermore, it is impossible to guarantee that the computational power of the processors in a cluster is identical. Therefore, achieving a dynamic load balance is essential if an acceptable program performance is to be obtained from the cluster.

3.1 Source Compatibility

To maintain source compatibility, three problems must be addressed. The first problem is that of the memory allocation of the global variables. In SMPs, the global variables declared in the user programs are shared between threads even when they are not assigned with an initial value by the programmer. However, Teamster allocates any global variables without an initial value into the private space. Consequently, these variables can not be shared among threads located on different nodes. The second problem is that of the memory allocation function. In SMPs, users can call the `malloc()` function to allocate a block of shared memory for data communication between threads. However, in Teamster, this function allocates memory at the private space. Therefore, to allow users to allocate memory at the shared memory address, the memory allocation function must be modified or replaced. The third problem is that of the OMP library functions. For example, the function `omp_get_num_threads` is used to return the total number of threads in a program. In Teamster, this function must be modified to sum the numbers of threads allocated at all of the execution nodes and to return this summed value. This implies that it is necessary to develop an OMP library specifically for Teamster.

3.2 Program Performance

The original OMP run time library is implemented using the kernel-level Pthread. However, to minimize the cost of program parallelization and to support thread migration and resource reallocation, this study uses a user-level thread to implement the OMP library. For the sake of compatibility, this study chooses GNU Pth [17], which is a portable thread package supporting UNIX-compatible systems. However, this thread package does not support distributed systems and therefore a distributed Pth library must be developed specifically for Teamster.

Many solutions have been proposed for achieving a load balance in SMPs and DSMs, respectively. Basically, the methods proposed for SMPs are based on loop scheduling [18], while those for DSM systems are based on thread migration [19-21]. Recently, Sakae [22] proposed a loop partition algorithm known as profiled scheduling to address the load balance problem in loop applications. The basic principal of this algorithm is to assign the same number of iterations to program threads for execution and to then profile the execution times of the individual threads. Based on the profiled thread execution times, the execution time of each node is estimated using Eq. (1) and the number of iterations assigned to each node determined by Eq. (2). After loop re-partitioning, the threads located at the same node evenly share the iterations distributed to their execution node.

$$T_x = \frac{\sum_{y \in S_x} T_{yx}}{N_x}, \quad (1)$$

where T_x is the execution time of node x , S_x is the set of threads located at node x , T_{yx} is the execution time of thread y at node x , and N_x is the number of threads located at node x .

$$W_x = \frac{1/T_x}{\sum_{x=1}^n 1/T_x} \times I, \quad (2)$$

where W_x is the number of iterations distributed to node x , n is the number of execution nodes, and I is the total number of iterations in the loop structure.

Compared to the methods proposed for DSM systems, the cost of the profiled scheduling algorithm described above is reduced since thread migration is not necessary. However, the profiled scheduling algorithm may misestimate the execution time of an execution node if the number of threads assigned to that node exceeds the number of processors at that node. Therefore, Eq. (1) should ideally be modified to consider both the number of threads and the number of processors at the node if a profiled scheduling algorithm is to be employed.

4. IMPLEMENTATION

In accordance with the considerations discussed above, the present implementation task involves modifying the Omni compiler, developing a distributed OMP library (OMPLIB), constructing a user-level distributed Pth thread library (PTHLIB), and designing a load balance mechanism.

4.1 Modification of Omni Compiler

By tracking the compiling process, it is found that the Omni compiler uses Ident objects and XobjectsDef objects as descriptors of the data variables in the user programs. The Ident object is used to describe the type, name and address of a variable. If a variable is not initialized, the XobjectsDef field will be filled with a NULL value; otherwise the field will be filled with an initial value. Accordingly, this study modifies the Omni compiler to automatically assign an initial value to the XobjectsDef fields of any variables whose initial value are NULL. As a result, all of the variables in the transformed C source codes will be initialized and then be allocated by Teamster at the shared memory space.

As described earlier, it is necessary to modify or replace the memory allocation function, i.e. malloc(), when enabling OpenMP on Teamster. Teamster provides the pRelease_new() function for memory allocation. In order to achieve source compatibility, the macro #define malloc pRelease_new is designed in the head file, i.e. omp.h. Since all original OpenMP programs must include this head file, the malloc() function can be automatically replaced by the pRelease_new() function without the need to modify the source code.

4.2 Distributed OMPLIB

This study develops a distributed version of the OMP library for Teamster. For example, the parallel directive is mapped to the `_ompc_do_parallel()` function. When this parallel directive is placed in front of a program block, the block will be replaced by the `_ompc_do_parallel()` function once the OpenMP program has been transformed by the Omni compiler. The replaced program block is packed into a working function. The name of the working function is the parameter of the `_ompc_do_parallel()` function. When the `_ompc_do_parallel()` function is performed during the execution of the program, the function will fork a number of threads to execute the same working function by assigning a different working data set. To parallelize the OpenMP program on a cluster, the `_ompc_do_parallel()` function is modified to broadcast the name of the working function to the other nodes. Each node then forks a number of threads and binds these threads with the working function to process the work of the parallel region in accordance with the received function name.

4.3 Distributed PTHLIB

The distributed Pth library consists mainly of the thread management and thread synchronization functions. In the present implementation, five scheduling queues, i.e. new, ready, waiting, suspend and dead, are created for each processor at a node. Threads forked in a parallel region are evenly distributed to the ready queues of the processors and each thread scheduler then fetches the threads from its ready queue for execution. If the ready queue is empty, the scheduler fetches threads from other queues at the same node. In addition, the global thread scheduler, which manages all of the program threads, uses a data structure known as LoadMap to store the information relating to the threads, e.g. the thread state, the returned value, the identifier of the execution node, and the address of thread control block. When the master thread of a user program intends to join a slave thread, the identifier of that slave thread is sent to the global scheduler. If the state of the slave thread is `THREAD_TERMINATED`, the return value of the slave thread is sent back to the main thread, which then continues its work. Otherwise, the main thread is blocked, the state of the slave thread is marked as `THREAD_JOIN`, and the location of the main thread and the TCB address of the main thread are stored. Once the slave thread has finished its work, its identifier is sent to the global scheduler and its return value is sent to the main thread, which then resumes its work. However, if the state of the slave thread is not marked as `THREAD_JOIN`, the global scheduler simply updates its state as `THREAD_TERMINATED` and stores the return value in LoadMap.

The lock and barrier of the distributed Pth library are mapped to the distributed-queue lock and the hierarchical barrier, respectively, in Teamster. Significantly, to minimize the synchronization overhead, when a lock is released, a thread whose node/cluster allocation is the same as that of the lock has a higher priority to get the access right of the lock than a thread whose node/cluster allocation is different from that of the lock.

4.4 Load Balance Mechanism

This study develops a novel loop scheduling algorithm called Profiled Multiproces-

sor Scheduling (PMS) to address the load balance problem for user applications executed in a cluster environment. PMS is similar to the profiled scheduling algorithm proposed by Sakae, but uses Eq. (3) to estimate the execution time of each node. Since this equation considers both the number of processors at each node and the number of threads at each node, it overcomes the potential for estimation error in Sakae's profiled scheduling algorithm.

$$T_x = \frac{\sum_{y \in S_x} T_{yx}}{N_x} \times \left[\frac{N_x}{P_x} \right], \quad (3)$$

where P_x is the number of processors at node x .

This study implements a load balance mechanism based on the PMS algorithm into Teamster. When a thread commences work on an iteration, the thread scheduler records the start time of the thread. When the thread arrives at the end of the iteration, e.g. at a barrier, the thread scheduler records the arrival time and calculates the elapsed time between the start time and the arrival time. The calculation results from each node are then sent to the root node, which estimates the execution time of each node using Eq. (3) and then calculates and broadcasts a new loop partition pattern. Through this broadcast, each node is informed of the number of iterations it must work for and evenly distributes these iterations across its local threads for parallel execution by adjusting the start and end iteration variables of the working function bound to each thread.

5. PERFORMANCE

This study implements the SOR, N -body and EP applications to evaluate the performance of the modified Omni compiler, the distributed OMP library, the distributed Pth thread library, and the PMS loop scheduling algorithm. The parameters of these test applications are shown in Table 1 and the experimental environment is shown in Table 2.

Table 1. Program parameters.

Application	Problem size	CPU demand	Memory demand
SOR	7168 × 7168 250 iterations	811.450 sec	392 MB
N -body	8192 particles 200 loops	1084.904 sec	320 KB
EP	Class C	1759.850 sec	1 MB

5.1 System Overhead

Table 3 summarizes the estimated overheads of the distributed Pth library and the distributed OMP library. It can be seen that the costs of thread management and synchronization, and those of loop parallelization, are very small. In other words, the parallelization cost of the user applications is effectively reduced.

Table 2. Experimental environment.

	Cluster I	Cluster II
Node id	Node(0, 1, 2, 3)	Node(4, 5)
CPU	Pentium III Xeon 500Mhz * 4	Pentium III Xeon 700Mhz * 4
Memory	512 MB SDRAM	
Network	Fast Ethernet (100Mps)	
Software	Fedora Core(2.6.8-1.52 smp) and gcc 3.3.3	

Table 3. System overheads.

Thread creation	41.12 us	Lock acquire	0.02393 ms
Thread joining	0.01101 ms	Lock release	0.03093 ms
Thread migration	1.55043 ms	Barrier arrive	0.03670 ms
Thread context switch	0.01163 ms		

5.2 Application Performance

The Pthread and OpenMP interfaces are both used to implement the test applications in order to evaluate the effectiveness of the modified Omni compiler. The applications were all run in Cluster I. Table 4 shows that the performances of the test applications implemented by OpenMP are very similar to those of the same applications implemented by Pthread. This implies that the modified OpenMP compiler provides an effective translation of the OpenMP programs. Furthermore, the results confirm the efficiency of both the OMP library and the Pth library.

Table 4. Comparison between OpenMP and D-Pth.

	D-Pth			OpenMP		
	$N = 1$	$N = 2$	$N = 4$	$N = 1$	$N = 2$	$N = 4$
SOR						
Exec. Time (sec)	811.450	428.744	253.385	811.393	428.519	253.506
Speed up	1	1.893	3.202	1	1.893	3.201
N -Body						
Exec. Time (sec)	1084.904	545.320	280.611	1082.898	544.918	281.926
Speed up	1	1.989	3.866	1	1.987	3.841
EP						
Exec. Time (sec)	1759.850	881.439	442.048	1760.953	882.428	443.563
Speed up	1	1.997	3.970	1	1.996	3.970

Table 5. Comparison between SMP and computer cluster.

	N -Body (Exec. Time)	SOR (Exec. Time)
$n = 1; p = 4$	1118.463 sec	828.255 sec
$n = 4; p = 1$	1158.513 sec	775.027 sec

The test applications were also executed on a four-processor node and a cluster of four single-processor nodes, respectively. As shown in Table 5, the performance of the *N*-body application running on the four-processor cluster is poorer than when it is run on the shared memory multiprocessor. The reason for this is that the *N*-body application involves heavy data sharing and the cost of maintaining data consistency over a computer network is far greater than maintaining data consistency via a system bus. However, the performance degradation is less than 1% and is therefore acceptable. By contrast, in the SOR application, data sharing occurs only at the boundaries of the data matrixes. However, the memory demands are very high. Accordingly, when this application is executed on an SMP machine, a large number of page swapping operations must be performed since the memory resources of the machine are insufficient to meet the memory demands of the application. Consequently, the cost of memory accesses is dramatically increased and the performance of the SOR application is seriously degraded. However, this situation is improved when the SOR application is executed on the computer cluster. Since the work of the application is evenly distributed across the four different nodes in the cluster, each node must satisfy only one quarter of the total memory demands of the SOR application. Consequently, the number of page swapping operations is significantly reduced and hence the performance of the SOR application is greatly improved. This advantage of DSM systems is attractive for users to port their applications on a computer cluster.

5.3 Load Balance

To evaluate the performance of the loop scheduling algorithm in achieving a load balance, two different thread mapping patterns were used to run the test applications. In the first mapping pattern, a single thread was assigned to each processor, while in the second, two threads were assigned to the first processor of the first node and a single thread was assigned to each of the other processors. The applications were executed using two nodes at Cluster I and two nodes at Cluster II. In the performance evaluation experiments, the work of the test applications was initially distributed evenly to the program threads using a static scheduling approach. After profiling the information necessary for load balancing, two different scheduling algorithms, i.e. profiled and PMS, were applied to re-partition the work of the test applications to the program threads.

Fig. 1 shows that the profiled scheduling algorithm and the PMS algorithm both successfully minimize the load imbalance overheads of the two test applications irrespective of the thread mapping pattern applied. It is also observed that the proposed PMS algorithm is as effective as the profiled algorithm when the first thread mapping pattern is employed. This is because the program threads are mapped to the processors with a one-to-one pattern and therefore the profiled algorithm can estimate the execution time of a node as precisely as the PMS scheduling algorithm. However, it can be seen that the PMS algorithm is more effective than the profiled algorithm when the second thread mapping pattern is applied. The reason for this is that the PMS algorithm considers both the number of threads and the number of processors when estimating the node execution time. Therefore, it obtains more precise estimates of the node execution time and achieves a better load balance and hence an improved application performance as a result.

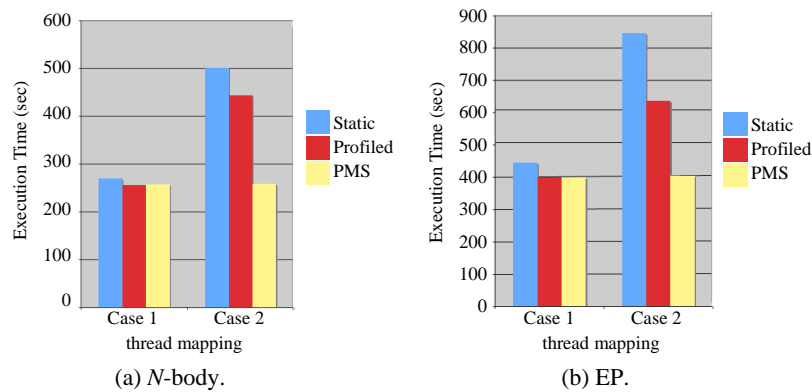


Fig. 1. Effectiveness of load balancing.

6. RELATED WORK

Several past studies have also investigated the implementation of the OpenMP interface on software DSM systems such as TreadMarks [23], SCASH [24], JIAJIA [25], and COMPaS [26]. The implementation of TreadMarks is similar to that of the present study. However, TreadMarks does not provide complete source compatibility for OpenMP programs written on the SMP platform. SCASH uses the Exec Java toolkit to translate OpenMP programs into multithreading programs. In addition to its support of the OpenMP interface, SCASH also provides a set of extended OpenMP directives for co-allocating shared data in the same data pages to minimize data consistency maintenance costs. However, that makes the programs incompatible since these directives are not standards of OpenMP. JIAJIA implements a compiler known as AutoPar to analyze the correctness of parallel programs and automatically adjusts the computation granularity to achieve a compromise between the performance benefits of parallelism and the parallelization cost. Finally, COMPaS maintains data consistency by using the Omni compiler to insert message passing calls into the source programs rather than by using the DSM library as in other DSM schemes.

Compared with the approaches discussed above, the approach developed in this study provides complete source compatibility for applications developed on SMPs. In addition, previous studies generally implemented the OMP library based on the kernel-level Pthread, whereas in this study, the OMP library is implemented using a user-level distributed POSIX thread library. Finally, load balancing in DSM schemes is traditionally achieved by using a thread migration approach. However, this study employs a loop re-partitioning approach to achieve load balance in order to minimize the overheads of load balancing.

7. CONCLUSIONS AND FUTURE WORK

This study has successfully implemented the OpenMP programming interface on Teamster for Linux-based SMP clusters. The proposed approach significantly reduces the complexity of programming on a cluster environment. Since source compatibility is

maintained in the current implementation, users can seamlessly apply their OpenMP programs developed on SMPs to a cluster environment. As a result, the range of feasible cluster-computing applications is significantly increased. This study has also proposed a novel loop scheduling algorithm, referred to as PMS, to address the problem of load balancing for user applications executed on a cluster environment. The experimental results have shown that the proposed scheduling algorithm provides a greater application performance improvement than other algorithms such as static and profiled scheduling. The experimental results have also shown that DSM is beneficial in minimizing the number of page replacements, particularly when the user applications require sizeable memory resources to cache the necessary data. In other words, DSM provides an attractive choice for users seeking to use clusters rather than SMPs to reduce the execution time of their applications.

In a future study, the current authors intend to develop an grid-enabled software DSM systems by integrating the Globus toolkits [27] in order for enabling the OpenMP programming interface on computational grids [28-30]. Additionally, the authors will investigate the feasibility of developing a web-based OpenMP program development environment to allow users to write their applications, and execute and monitor their programs on a grid environment, from any location and using any device.

ACKNOWLEDGMENT

We appreciate the support of National Science Council, and High Performance Parallel and Distributed System Lab. at the Department of Electrical Engineering of National Cheng Kung University in Taiwan.

REFERENCES

1. M. Sato, "OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors," in *Proceedings of 15th International Symposium on System Synthesis*, 2002, pp. 109-111.
2. R. Buyya, *High Performance Cluster Computing: Architectures and Systems*, Prentice Hall PTR, New Jersey, 1999.
3. W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, Vol. 22, 1996, pp. 789-828.
4. V. Sunderam, "PVM: a framework for parallel distributed computing," *Concurrency: Practice and Experience*, Vol. 2, 1990, pp. 315-339.
5. K. Li, "IVY: a shared virtual memory system for parallel computing," in *Proceedings of the International Conference on Parallel Processing*, Vol. 2, 1988, pp. 94-101.
6. J. B. Carter, J. K. Bennett, and W. Zwaenepoel, "Implementation and performance of Munin," in *Proceedings of 13th ACM Symposium on Operating System Principles*, 1991, pp. 152-164.
7. E. Speight and J. K. Bennett, "Brazos: a third generation DSM system," in *Proceedings of the USENIX Windows NT Workshop*, 1997, pp. 95-106.

8. R. Friedman, M. Goldin, A. Itzkovitz, and A. Schuster, "Millipede: easy parallel programming in available distributed environments," *Software – Practice and Experience*, Vol. 27, 1997, pp. 929-965.
9. W. Hu, W. Shi, and Z. Tang, "JIAJIA: an SVM system based on a new cache coherence, protocol," in *Proceedings of the High Performance Computing and Networking*, 1999, pp. 463-472.
10. P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy release consistency for software distributed shared memory," in *Proceedings of 19th Symposium on Computer Architecture*, 1992, pp. 13-21.
11. B. N. Bershad and M. J. Zekauskas, "Midway: shared memory parallel programming with entry consistency for distributed memory multiprocessors," Technical Report, No. CMU-CS-91-170, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1991.
12. L. Iftode, J. Singh, and L. Li, "Scope consistency: a bridge between release consistency and entry consistency," in *Proceedings of 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996, pp. 277-287.
13. J. B. Chang and C. K. Shieh, "Teamster: a transparent distributed shared memory for cluster symmetric multiprocessors," in *Proceedings of 1st International Symposium on Cluster Computing and the Grid*, 2001, pp. 508-513.
14. J. B. Chang, T. Y. Liang, and C. K. Shieh, "Teamster: a transparent distributed shared memory for clustered symmetric multiprocessors," accepted for publication in the *Special Issue of the Journal of Supercomputing*, September 6, 2003.
15. C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "TreadMarks: shared memory computing on networks of workstations," *IEEE Computer*, Vol. 29, 1996, pp. 18-28.
16. K. Kusano, S. Satoh, and M. Sato, "Performance evaluation of the omni OpenMP compiler," in *Proceedings of 3rd International Symposium on High Performance Computing*, 2000, pp. 403-414.
17. R. S. Engelschall, GNU Pth – The GNU Portable Threads, <http://www.gnu.org/software/pth>, 2005.
18. E. P. Markatos and T. J. LeBlanc, "Using processor affinity in loop scheduling on shared-memory multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, 1994, pp. 379-400.
19. A. Schuster and L. Shalev, "Using remote access histories for thread scheduling in distributed shared memory systems," in *Proceedings of 12th International Symposium on Distributed Computing*, 1998, pp. 347-362.
20. T. Y. Liang, C. K. Shieh, and D. C. Liu, "Scheduling loop applications in software distributed shared memory systems," *IEICE Transactions on Information and Systems*, Vol. E83-D, 2000, pp. 1721-1730.
21. K. Thitikamol and P. Keleher, "Thread migration and communication minimization in DSM systems," in *Proceedings of the IEEE*, Vol. 87, 1999, pp. 487-497.
22. Y. Sakae, S. Matsuoka, M. Sato, and H. Harada, "Preliminary evaluation of dynamic load balancing using loop re-partitioning on omni/SCASH," in *Proceedings of 3rd IEE/ACM International Symposium on Cluster Computing and the Grid, Distributed Shared Memory on Clusters Workshop*, 2003, pp. 463-470.
23. C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W.

- Zwaenepoel, "TreadMarks: shared memory computing on networks of workstations," *IEEE Computer*, Vol. 29, 1996, pp. 18-28.
24. Y. Ojima, M. Sato, H. Harada, and Y. Ishikawa, "Performance of cluster-enabled OpenMP for the SCASH software distributed shared memory system," in *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2003, pp. 450-456.
 25. F. Zhang, G. Chen, and Z. Zhang, "OpenMP on networks of workstations for software DSMs," *Journal of Computer Science and Technology*, Vol. 17, 2002, pp. 90-100.
 26. Y. Tanaka, M. Matsuda, M. Ando, K. Kazuto, and M. Sato, "COMPaS: a pentium pro PC-based SMP cluster and its experience," *IPPS Workshop on Personal Computer Based Networks of Workstations*, LNCS 1388, 1998, pp. 486-497.
 27. I. Foster and C. Kesselman "Globus: a metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, Vol. 11, 1997, pp. 115-128.
 28. I. Foster and C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann, San Francisco, CA, 1999.
 29. M. Baker, R. Buyya, and D. Laforenza, "Grids and grid technologies for wide-area distributed computing," *International Journal of Software: Practice and Experience*, Vol. 32, 2002, pp. 1437-1466.
 30. F. Berman, G. Fox, and A. J G Hey, *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons, New York, 2003.

Tyng-Yeu Liang (梁廷宇) is currently an assistant professor who teaches and studies at Department of Electrical Engineering, National Kaohsiung University of Applied Sciences in Taiwan. He received his B.S., M.S. and Ph.D. degrees from National Cheng Kung University in 1992, 1994, and 2000. His study is interested in cluster and grid computing, image processing and multimedia.

Shih-Hsien Wang (王釋賢) is a soldier in the army of the R.O.C. He just got his Master degree from the Electrical Engineering Department of National Cheng Kung University in July 2005. His study is focused on cluster and grid computing. He plans to be a software engineering after he finish his duty of being a soldier.

Ce-Kuen Shieh (謝錫堃) currently is a professor at the Electrical Engineering Department of National Cheng Kung University in Taiwan. He is also the chief of computation center at National Cheng Kung University. He received his Ph.D degree from the Department of Electrical Engineering of National Cheng Kung University in 1988. He was the chairman of the Electrical Engineering Department of National Cheng Kung University from 2002 to 2005. His research interest is focused on computer network, and parallel and distributed system.

Ching-Min Huang (黃竟閩) was born in Kaohsiung, Taiwan. He received the B.S. of E.E. degree from Da-Yeh University, Changhua, in 2004. He is currently studying for his Master degree at Departement of Electrical Engieneering of National Kaohsiung University of Applied Sciences. His research interests include parallel processing and image processing.

Liang-I Chang (張良毅) was born in Changhua, Taiwan. He received the B.E. degree from the Department of Computer Science, Shu-Te University, Taiwan, in 2004. He currently study for get his Master degree at the Electrical Engineering Departement of National Kaohsiung University of Applied Sciences. His research interests focus on parallel processing and data mining.