A preprocessing system of the EULASH: an environment for efficient use of multiprocessors with local memory

J. Yamamoto, D. Hattori‡, J. Yamato*, T. Tokuyoshi†,
Y. Yamaguchi, H. Amano‡

‡Dept. of Computer Science
Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama 223 JAPAN
E-mail: {junji,hattori,yamaguti,hunga}@aa.cs.keio.ac.jp

Keywords: Software, Processor and memory architecture, preprocessing system

Abstract

The EULASH environment is proposed for making the best use of a multiprocessor with a high speed local memory and slow shared memory. It consists of the preprocessing system, the kernel, and the library for management of variables and synchronization.

On the EULASH, a program is written with light weight threads using only shared variables. The preprocessing system restructures the program in order to use both memory system efficiently without users' optimization. By the optimization using the *trial run*, between 50% and 90% of shared variables are converted so as not to use the shared memory. From the result of evaluations, the execution time with optimization is about 13%–20% better than that without optimization on a switch connected multiprocessor.

1 Introduction

The EULASH (Environment for Using Local and SHared memory) is a software environment for a multiprocessor with two types of memory system: a high speed local memory and a remote memory with a large latency. A simple NUMA system without complicated directory cache mechanism, and switch connected multiprocessors like NYU Ultracomputer [1] and BBN TC2000 [2] fall into this classification. For high speed execution on these machines, the high speed local memory must be used efficiently as possible. If the programmer carefully considers the assignment of the processes and variables, the use of the shared memory can be minimized. However, it is difficult job for a programmer. A software environment EULASH is developed for easy programming and high speed execution of a single job on such multiprocessors.

On the EULASH, a program is written with light weight threads using only shared variables. The EULASH preprocessing system analyzes relation between threads and variables, using dependency graph and the tentative execution of the target program. Using this analysis result, threads are grouped and assigned to the virtual processor so as to minimize the loss caused by the thread switching and shared memory access. Variables in the program are classified into several types and located on the local memory or shared memory appropriately. At the run-time, the EULASH kernel with a quick thread context switching and virtual memory system manages the execution of the program cooperated with run-time library programs.

By these mechanisms, parallel application programs can use both local and shared memory systems efficiently without users' optimization. In this paper, the preprocessing system of the EULASH is introduced and evaluated on real parallel machines.

2 The EULASH preprocessing system

2.1 Overview of the preprocessing system

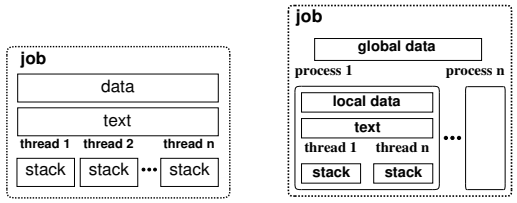
In the EULASH environment, a programmer describes an application program in the similar manner to the multi-threads library of Mach or C-threads library of the UNIX operating system. The programmer is not required to care the data allocation nor the structure of the target multiprocessor. In order to minimize the loss of thread switching and reference of the shared memory with a large latency, the EULASH preprocessing system uses two techniques: the static analysis and trial run.

The relation of the program between threads and variables are statically analyzed. Threads described

*Joined to C&C Research Lab., NEC Co.Ltd.

†Joined to Microelectronics Engineering Lab., Toshiba Co.Ltd.

by the programmer are gathered to make *processes*. In the EULASH, a *process* is a virtual processor which usually corresponds to a real processor and mapped statically. The number of process is not equal to that of processors only when a number of processors in the target multiprocessor is changed with partitioning of the system or system faults. As a result, the simple user programming model shown in Figure 1(a) is restructured as shown in the Figure 1(b).



(a) model for user

(b) restructured model

Figure 1: programming models

In order to hide the latency of reference of variables, five types of variables are supported: local, shared, write-invalidate, write-update, and mobile. However, it is difficult to analyze statically how variables are referred from processors. A technique called *trial run* is introduced. A core of the program is executed for collecting information about references of variables. Based on this information, the compiler decides the type of variables.

The optimized and restructured program is executed on the EULASH kernel which supports quick thread/process management and virtual memory system. Based on the types of variables, the appropriate run-time library is called when the variables accessed.

2.2 Restructuring of the parallel program

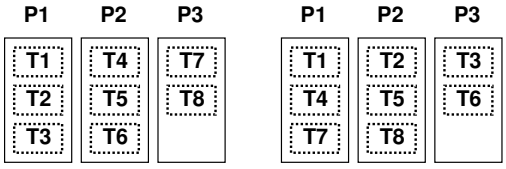
EULASH provides four grouping methods for assigning threads into a process: block assignment, cyclic assignment, data dependency assignment, and tightly communicated thread first.

Block assignment(Block)
Threads are grouped in the order of creation, and are assigned processes. (Fig. 2.2(a))

Cyclic assignment(Cyclic)
Threads are assigned process one by one in the order of creation. (Fig. 2.2(b))

Data dependency assignment(DD)
Make a dependency graph $D(t, s)$ where threads t are connected with an edge s when there are variables shared by them. Mark the first created thread as the master thread, and select m sub-master threads which are located distant from the master thread as possible (where $m + 1$ is the number of processes). First, each of the master thread and sub-master threads selects its own process. Assign threads connected to the master/sub-master thread to the same process to which the connected master/sub-master thread is assigned. Then, assign other threads connected to assigned threads to the same process to which the connected threads are assigned. Iterate these operations until all threads are assigned.

Tightly communicated threads first assignment(TCTF)
In this method, numbers of threads referring variables are counted. Pick up the variable which is referred by the largest number of threads. Here, assign every thread which refers the variable into a process. Then, pick up the variable which is referred by the next largest number of threads, and assign every thread which refers the variable into a process. Iterate this operation until every thread is assigned.



(a) Block assignment

(b) Cyclic assignment

Figure 2: Thread grouping method

2.3 Type of Variables

After threads grouping, the preprocessing system decides the type of each variable. There are five types of variables: two ordinary variable types and three special variable types. The ordinary types are the local type and shared type, which are statically assigned into the local memory and shared memory respectively.

The special types are write update, write invalidate and mobile. The write update type variable is allocated into the shared memory first, and copied to the local memory when it is accessed. When the variable is updated, the information is broadcast and all copied

variables are updated. The write invalidate type variable is also allocated into the shared memory first, and copied to the local memory when it is accessed. Unlike the write update type variables, the copies are invalidated when the variable is updated by a processor. Using write update/invalidate type variables, the local memory can be used as a software cache of which line size is equal to the size of a variable. These variables are efficiently used if the access speed of the local memory is much faster than that of the shared memory.

The mobile type variable is allocated into the local memory first, but never copied even if they are accessed. When it is accessed by other processors, it is moved to the shared memory. During they are accessed only by the initially allocated processor, the communication between shared memory is not required.

In order to decide the types of variables, the information about the references to variables is required. This information is collected in the trial run stage described later.

The type is decided by applying the following rules in order:

1. Variables accessed by only one process are classified as the *local* type.
2. Read-only variables are classified as the *local* type. For initialization, those variables are accessed as *shared* type before processes are forked.
3. If the access ratio of the variable from a particular process is beyond a threshold (`Th 1`), the variables is classified as the *write invalidate*. As described later, `Th 1` is empirically chosen.
4. If the ratio of read access to the total number of access is larger than a threshold(`Th 2`), the variable is classified as the *write update* type. `Th 2` is also empirically chosen.
5. If the variable is accessed only by a thread until the thread is terminated, it is classified as the *mobile* type. Another thread can access the *mobile* after the previous thread is terminated.
6. Other variables are classified as the *shared* type.

2.4 Trial Run

For the optimization in the compiler, the program is tentatively executed to collect the information about references of variables. In order to reduce the time of the tentative execution, the programmer can use a program core in which a number of the iteration or the size of the data is reduced and the output of the result is omitted. In this step, which is called the trial run, the program is executed as follows:

- All variables are located on the shared memory.
- Threads are tentatively assigned to processes so as not to cause a severe load imbalance.
- When variables are accessed, the library program is called and the number and type of the access are recorded.

After the trial run, the compiler decides the thread assignment and the type of variables by using the collected information. By the overhead of collecting the information, the execution time of the trial runs is about 1.5 times as that of the program without information collecting.

3 Performance Evaluation

3.1 Target machines for evaluation

The target machine of the EULASH is a large scale NUMA or switch connected UMA which does not provide coherent cache mechanism. Here, a switch connected multiprocessor called SNAIL [3] is used for evaluation.

In SNAIL (Figure 3), 16 processors are connected with 16 shared memory modules through a high speed MIN (Multistage Interconnection Network) called the SSS (Simple Serial Synchronized)-MIN[4]. Each processor board consists of 4 CPU (MC68040) and local memory systems. Shared memory system supports a synchronization mechanism such as Fetch & Decrement and Test & Set operation. Table 1 shows the access latency of the memory system. As shown in the table, the access latency of the shared memory is about 5 times larger than that of the local memory at least. Because of the congestion in the MIN, the access latency may be slightly stretched.

Table 1: access time of memory systems(SNAIL)

memory system	access type	time(ns)
local memory	read/write	250
shared memory	read	1375
	write	250

3.2 Efficiency of the optimization

Efficiency of optimization by the preprocessing system is evaluated with two sample programs:

- **Jacobi** solves simultaneous linear equations with Jacobi's iterative method.
- **Heat** solves a thermal conduction equation with the difference method.

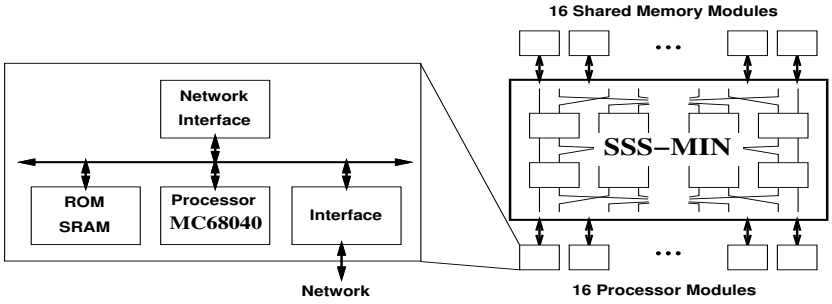


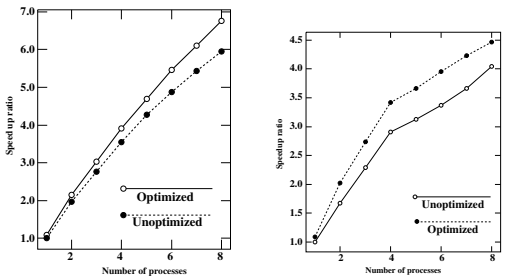
Figure 3: Block diagram of SNAIL

Threshold for type selection of variables is as follows:

- **Th 1:** The write invalidate type is used when access of a thread is 90% or more of all accesses.
- **Th 2:** The write update type is used when read access is 70% or more of all accesses.

In **Jacobi**, regardless of the way to group, 60% of data is selected to be "local" and the rest is "write update". Variables selected to be "local" are elements of the coefficient matrix and are written only once by a thread which initializes data at the beginning. Since other variables used in **Jacobi** are read out frequently compared with writing, "write update" type is automatically selected by the preprocessing system.

Figure 4 shows the effects of optimization for **Jacobi** and **Heat** respectively. The optimized program is 13% (**Jacobi**) and 22 % (**Heat**) faster than that of unoptimized one.



(a) Jacobi (b) Heat

Figure 4: ratio of speed up

4 Conclusion

A software environment called the EULASH is proposed for efficient use of two memory systems: a high speed local memory and a shared memory with a large latency. The EULASH consists of the compiler, kernel and library for management of variables and supporting synchronization.

Optimization in the compiler with trial run, 10% – 20% of variables can be located on the local memory, and most of other variables can be classified into write invalidate or write update type. Using this optimization, the performance is improved 13% – 20% on a real switch connected multiprocessor.

References

[1] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. Mcauliffe, L. Rudolf, and M. Snir, "The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer," IEEE Transaction on Computer Systems, Vol. c-32, No.2, 1983.

[2] BBN Laboratories, "Butterfly (TM) Parallel Processor Overview," BBN Computer Company, Cambridge, MA, 1st edition, June 1985.

[3] M. Sasahara et al. "SNAIL: A Multiprocessor Based on the Simple Serial Synchronized Multistage Interconnection Network Architecture," Proceedings of the 1994 International Conference on Parallel Processing, Vol.I, pp.I-117–I-120, Aug. 1994.

[4] H. Amano, L. Zhou, and K. Gaye, "SSS(Simple Serial Synchronized)-MIN: a novel multi stage interconnection architecture for multiprocessors," Proceedings of the IFIP 12th World Computer Congress, Vol. I, pp.571–577, Sep. 1992.