The Design and Implementation of User-Level Software Distributed Shared Memory System: SMS
― Implicit Binding Entry Consistency Model ―

Seikei University
Hiroko Midorikawa, Yusuke Ohashi, Hajime Iizuka

1. Introduction

A distributed shared memory system, named SMS, is a user-level software system. It provides a virtual shared memory environment on a computer cluster consisting of computers connected by a communication network. Although the SMS requires only commodity hardware and software, it enables users to write parallel programs under a shared memory programming model.

2. Distributed Shared Memory System SMS

The SMS has several features.

- User-level software implementation

  The SMS requires no specific functions or privileged level processing for an OS. It can be easily installed on any PC or workstation cluster system with a Unix-based OS, like Linux or FreeBSD. It realizes a shared-memory programming environment on systems with commodity hardware and free software without root privilege.
- General socket communication with TCP or UDP

  The SMS uses a widely used communication protocol, TCP or UDP, and no special communication functions dependent on underlying communication devices. Any communication devices that support TCP or UDP are possible to use with SMS, a fast-Ether, a giga-bit Ether or a Myrinet for your budge and speed requirements.
- New relaxed memory consistency model

  The SMS employs a newly proposed IBEC (Implicit Binding Entry Consistency) as a memory consistency model. It is a variant of the entry consistency model, but implicitly binds shared data with a lock variable.
- Condition variables

  To manage exclusive access to shared data more efficiently, the SMS supports functions for condition variables, condition_wait(), condition_signal() and condition_broadcast(), besides ordinary shared data access control functions, lock(), unlock() and barrier(). These functions were not supported in most of the other distributed shared memory systems.
- Virtual processor independent from underlying computer configuration

  An arbitrary number of virtual processors (processes) can be forked independently from the number of available computers or CPUs. The SMS can be installed on not only a single-CPU computer cluster but also a multiple-CPU computer (SMP) cluster.

3. Design

3.1 Implicit binding entry consistency (IBEC)

The IBEC is one of the relaxed memory consistency models, which makes communication traffic for memory update lighter, as an entry consistency (EC) and a lazy release consistency (LRC). When compared to the LRC or a release consistency (RC), the IBEC is characterized by its multiple time clocks, which are related to lock variables. The LRC has a unique global time clock and its shared data is updated in the global time stamp order. On the other hand, in IBEC, shared data updates are done independently unless these data are not associated with the same lock variable. Moreover the IBEC has no explicit description in programs to associate a lock variable with a shared data variable unlike the programs under the EC model. Binding a lock variable with a shared data variable is done automatically on program execution.

3.2 User Interface

Table1 shows C functions and constants provided by SMS. Figure1 shows a simple program example for SMS.

```
sms_nproc  :Number of processes
sms_proc_id  :Process ID( id=0,1,2,...)
void sms_startup(int argc,char *argv)
void sms_shutdown()
void sms_error(char *errmsg)
void *sms_alloc(int size,int PageM_pid)
void *sms_calloc(int num, int itemsize,int PageM_pid)
void  sms_change_page_manager(void *adr,int size,int PageM_pid)
void sms_barrier(int BarrierM_pid)
void sms_lock(int Lock_id)
void sms_unlock(int Lock_id)
void sms_cond_wait(int cond_id, int Lock_id)
void sms_cond_signal(int cond_id)
void sms_cond_broadcast(int cond_id)
```

Table1 SMS Constant and Functions

4. Implementation

4.1 Page based distributed shared memory system

The SMS uses a memory protection mechanism in a memory management system for a detection of access to shared data, so shared data in the SMS is based on a memory page with a page manager specified when the data is allocated. Figure2 shows a page status

diagram. Page managers are responsible for transmitting required pages to requesting processors when shared data is first accessed. All pages are updated just after barrier() to maintain memory consistency. If a page cached in a processor is not accessed for a long time, the page is thrown out and its page status is changed to *unmapped*. This mechanism is effective to reduce unnecessary communication traffic for updates.

```
#include<stdio.h>
#include<sms.h>    /* sms header file */
#define NUM 1024
void main(int argc,    char *argv[ ])
{
   int *data, *max, n, i;
   FILE *frp;
   frp = fopen (argv[1],"r");      /*data file*/
   sms_startup(argc, argv);      /* sms startup */
   n=NUM / sms_nproc;
   /* allocate shared data */
   data=(int *)sms_calloc(NUM, sizeof(int),0);
   max =(int *)sms_alloc(sizeof(int),0);
   /* Data initialization by master process */
   if(sms_proc_id==0){
        for(i=0; i<NUM; i++) fscanf(frp,"%d",&data[i]);
      *max=0;
   }
   /* search max by all processes */
   sms_barrier(0);            /* barrier synchronization */
   for(i=sms_proc_id*n;    i < (sms_proc_id+1)*n;    i++)
   { sms_lock(0);          /* lock acquire */
         if(*max < data[i])    *max = data[i];
      sms_unlock(0);      /* lock release */
   }
   sms_barrier(0);            /* barrier synchronization */
                        /* result output by master process */
   if(sms_proc_id==0) printf("max =%d¥n",*max);
   sms_shutdown();          /* sms shutdown    */
}
```

Figure 1 SMS Program example

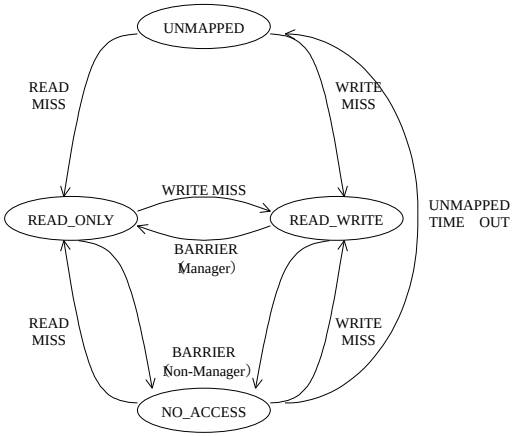


Figure 2 Page State Diagram

4.2 Multiple granularity for shared data update

Share data updates for memory consistency maintenance are done in different levels based on the granularity specified when the data was allocated. When a barrier() or a lock() is called, data update information is made by comparing the page where the data exists to its twin page, which was copied when the data was first written after previous barrier(). Its comparison is basically made byte by byte. Though, if a granularity is specified, both a comparison to make update information and an application of this update information to a page in other processor are done in this granularity, as in integer, short, double, float, or an arbitrary number byte size. It is very effective in reducing the total amount of communication traffic for updates and processing time to apply an update to a page.

4.3 Barrier manager and Lock manager

A barrier manager can be specified in barrier() by users and it is responsible for the barrier execution. It manages a synchronization of multiple process executions and an update of all shared data in processors. When lock() and unlock() are called, lock managers are responsible for the related data updating and an exclusive data access control. Lock managers are allocated to a processor automatically in round-robin fashion or statically. To manage condition variables, condition managers are usually allocated to the lock manager process that is responsible for the same lock variable associated with the condition variable.

4.4 Socket communication

Communication between processors is done in interrupted communication using a signal. Both TCP and UDP are options for users when the SMS system is configured. Two sockets are used between processes for data transmission and control signals.

5. Performance

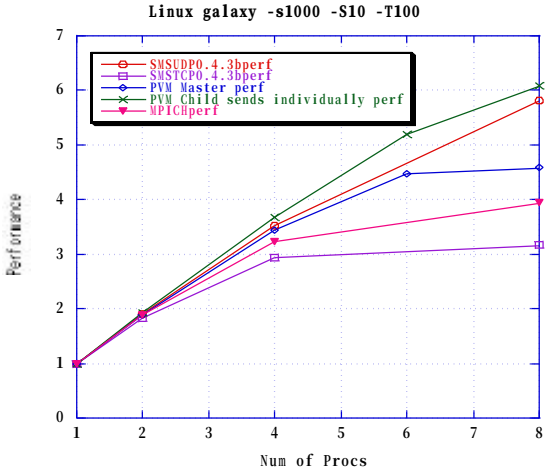
The performance of SMS is evaluated in two environments shown in Table2. Figure3 and Figure 4 show comparisons with PVM and MPICH for an n-body problem and MandelBrot set calculation respectively. Figure3 includes two PVM programs with different communication patterns, a master administrative and a slave distributed. The performance differences among PVM, MPICH and SMS are mainly caused by the communication pattern difference in each programs. Even using message-passing systems, such as MPICH and PVM, a communication traffic congestion causes a degradation in their performance, such as the PVM master-concentrated program in Figure3(a). The barrier synchronization in the SMS makes a moderate level of traffic concentration because the each process's arrival time to the same barrier is different, so the time of an update information transmission disperses. The more the number of iterations in n-body problem

becomes, the less a communication pattern difference affects the performance as in Figure3 (b).

Figure5 and Figure6 show comparisons with

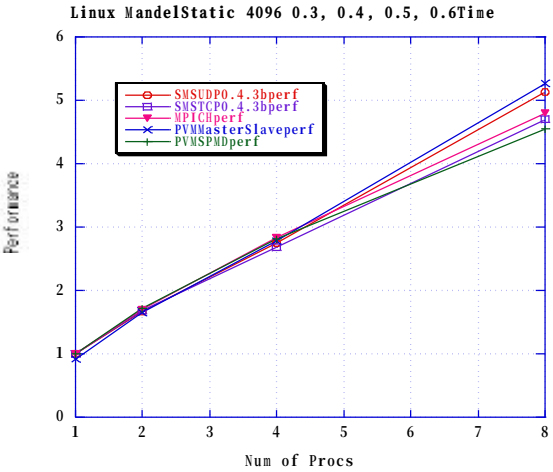
Table 2 Performance Evaluation Environment

Environment1 8PCs (8CPUs)
● CPU Intel MMX Pentium 166MHz
● Memory 64MB
● OS FreeBSD2.2.2R
● Network 100Mbps Ether Hub/Switch
Environment2 8PCs (8CPUs)
● CPU Intel Celeron 400MHz
● Memory 128MB
● OS RedHatLinux6.0(Kernel2.2.5-15)
● Network 100Mbps Ether Hub/Switch



(a) N-body problem, 1000bodies 10 ite. (Env.2)

Figure 3 SMS v.s. message passing system



(a) MandelBrot Static problem (Env.2)

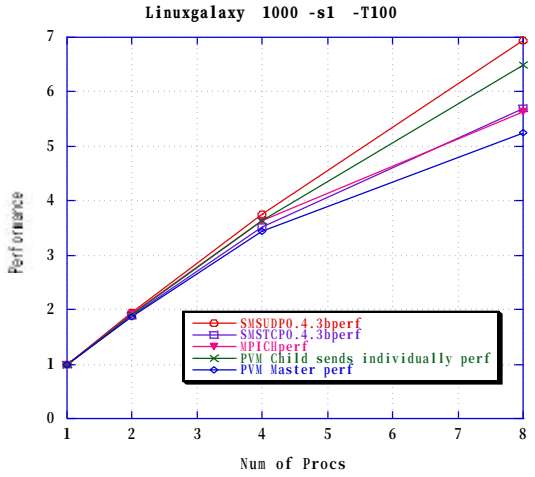
Figure 4 SMS v.s. message passing system

TreadMarks, the widely known software shared memory systems, for the same applications. Figure7 shows message bytes and a message count in master process for each program. In UDP packet level, message traffic in TreadMarks is less than SMS, but the performances especially in Figure 5(a) and Figure 6 are poor than SMS. One of the reasons is that the TreadMarks takes more time to startup processes. As the number of processes increase, the ratio of startup time in total execution time increases, which results in poor performance compared with SMS.

The performance of SMS is comparable or better than PVM, MPI, and TreadMarks for these programs.

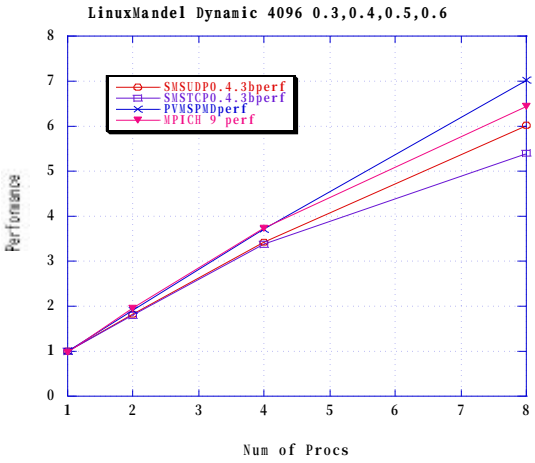
6. Conclusion

The SMS was designed with IBEC model, to realize a shared memory programming environment on a computer cluster. Not only dose it enable us to write a parallel program easier, but it achieves a comparable performance to MPI, PVM and TreadMarks.



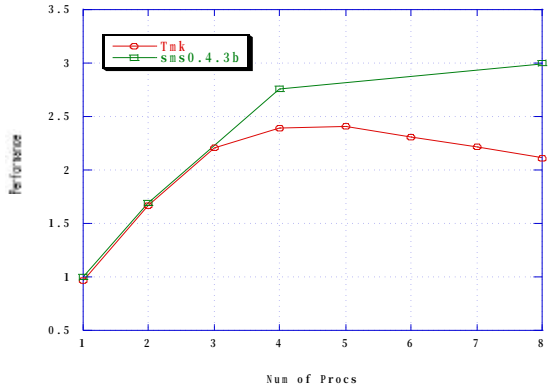
(b) N-body problem, 1000bodies 100 ite (Env.2)

Figure 3 SMS v.s. message passing system



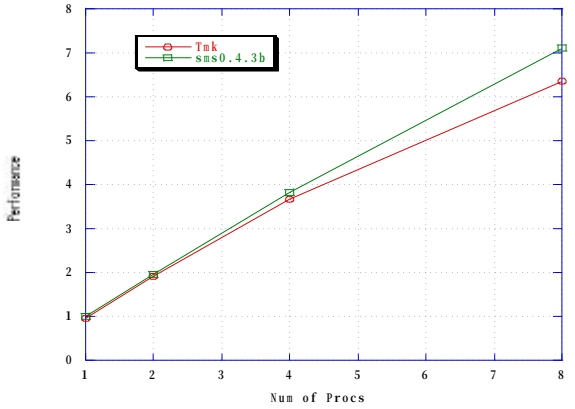
(b) MandelBrot Dynamic problem (Env.2)

Figure 4 SMS v.s. message passing system



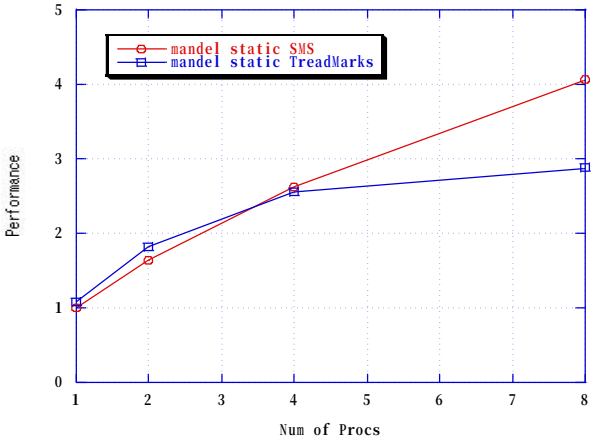
(a) N-body problem 100bodies 100 ite (Env.2)

Figure 5 SMS v.s. TreadMarks



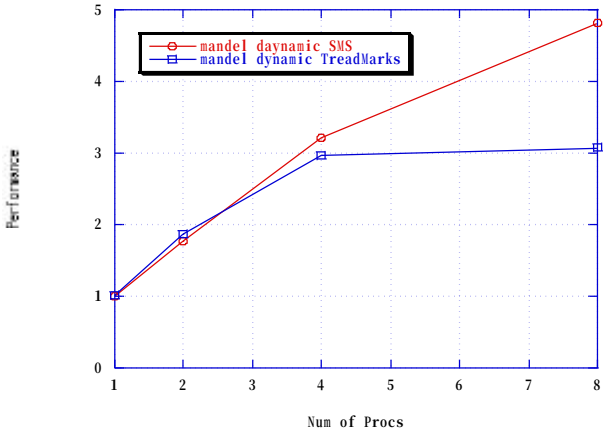
(b) N-body problem 1000bodies 10 ite. (Env.2)

Figure 5 SMS v.s. TreadMarks



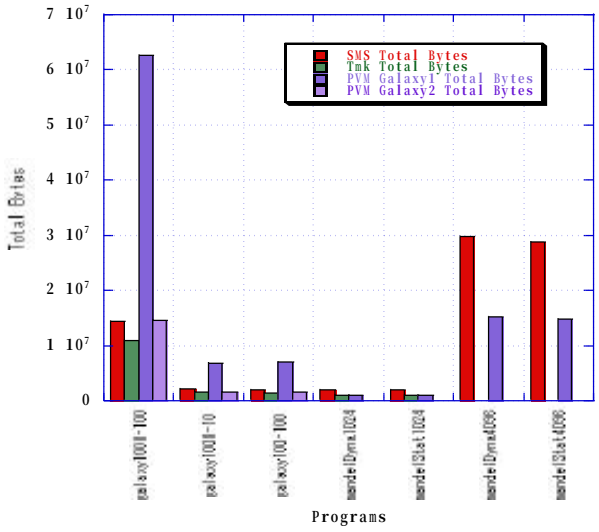
(a) MandelBrot static allocation 1024X1024 (Env.2)

Figure 6 SMS v.s. TreadMarks



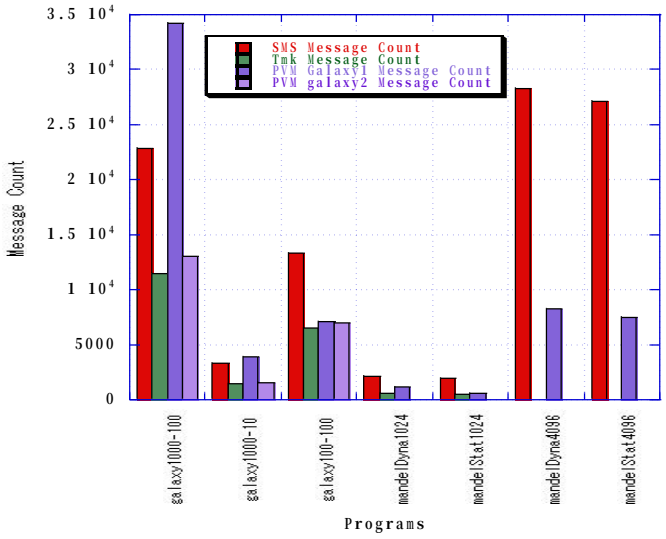
(b) MandelBrot dynamic allocation 1024X1024, 64 blocks (Env.2)

Figure 6 SMS v.s. TreadMarks



(a) Message bytes

Figure 7 Message Comparison in process0



(b) Message count

Figure 7 Message Comparison in process