RESEARCH ARTICLE OPEN ACCESS

Distributed Shared Memory – A Survey and Implementation Using Openshmem

Ryan Saptarshi Ray, Utpal Kumar Ray, Ashish Anand, Dr. Parama Bhaumik
Junior Research Fellow Department of Information Technology, Jadavpur University Kolkata, India
Assistant Professor Department of Information Technology, Jadavpur University Kolkata, India
M. E. Software Engineering Student Department of Information Technology, Jadavpur University Kolkata, India
Assistant Professor Department of Information Technology, Jadavpur University Kolkata, India

Abstract
Parallel programs nowadays are written either in multiprocessor or multicomputer environment. Both these concepts suffer from some problems. Distributed Shared Memory (DSM) systems is a new and attractive area of research recently, which combines the advantages of both shared-memory parallel processors (multiprocessors) and distributed systems (multi-computers). An overview of DSM is given in the first part of the paper. Later we have shown how parallel programs can be implemented in DSM environment using Open SHMEM.

I. Introduction

Parallel Processing

The past few years have marked the start of a historic transition from sequential to parallel computation. The necessity to write parallel programs is increasing as systems are getting more complex while processor speed increases are slowing down. Generally one has the idea that a program will run faster if one buys a next-generation processor. But currently that is not the case. While the next-generation chip will have more CPUs, each individual CPU will be no faster than the previous year's model. If one wants programs to run faster, one must learn to write parallel programs as currently multi-core processors are becoming more and more popular. Parallel Programming means using multiple computing resources like processors for programming so that the time required to perform computations is reduced. Parallel Processing Systems are designed to speed up the execution of programs by dividing the program into multiple fragments and processing these fragments simultaneously. Parallel systems deal with the simultaneous use of multiple computer resources. Parallel systems can be - a single computer with multiple processors, or a number of computers connected by a network to form a parallel processing cluster or a combination of both.
Cluster computing has become very common for applications that exhibit large amount of control parallelism. Concurrent execution of batch jobs and parallel servicing of web and other requests **[1]** as in Condor **[2]**, which achieve very high throughput rates have become very popular. Some workloads can benefit from concurrently running processes on separate machines and can achieve speedup on networks of workstation using cluster technologies such as the MPI programming interface **[3]**. Under MPI, machines may explicitly pass messages, but do not share variables or memory regions directly.

Parallel computing systems usually fall into two large classifications, according to their memory system organization: shared and distributed-memory systems.

Multiprocessor Environment

A shared-memory system **[4]** (often called a tightly coupled multiprocessor) makes a global physical memory equally accessible to all processors. These systems enable simple data sharing through a uniform mechanism of reading and writing shared structures in the common memory. This system has advantages of ease of programming and portability. However, shared-memory multiprocessors typically suffer from increased contention and longer latencies in accessing the shared memory, which degrades peak performance and limits scalability compared to distributed systems. Memory system design also tends to be complex.

Multicomputer Environment

In contrast, a distributed-memory system (often called a multicomputer) consists of multiple independent processing nodes with local memory modules, connected by a general interconnection network. The scalable nature of distributed-memory systems makes systems with very high computing power possible. However, communication between processes residing on different nodes involves a message-passing model that requires explicit use of send/receive primitives. Also, process migration imposes problems because of different address

spaces. Therefore, compared to shared-memory systems, hardware problems are easier and software problems more complex in distributed-memory systems. **[5]**

Distributed shared memory (DSM) is an alternative to the above mentioned approaches that operates over networks of workstations. DSM combines the advantages of shared memory parallel computer and distributed systems. **[5],[6]**

II. DSM – An Overview

In early days of distributed computing, it was implicitly assumed that programs on machines with no physically shared memory obviously ran in different address spaces. In 1986, Kai Li proposed a different scheme in his PhD dissertation entitled, "Shared Virtual Memory on loosely Coupled Microprocessors", it opened up a new area of research that is known as Distributed Shared Memory (DSM) systems. **[7]**

A DSM system logically implements the shared-memory model on a physically distributed-memory system. DSM is a model of inter-process communications in distributed system. In DSM, processes running on separate hosts can access a shared address space. The underlying DSM system provides its clients with a shared, coherent memory address space. Each client can access any memory location in the shared address space at any time and see the value last written by any client. The primary advantage of DSM is the simpler abstraction it provides to the application programmer. The communication mechanism is entirely hidden from the application writer so that the programmer does not have to be conscious of data movements between processes and complex data structures can be passed by reference. **[8]**

DSM can be implemented in hardware (Hardware DSM) as well as software (Software DSM). Hardware implementation requires addition of special network interfaces and cache coherence circuits to the system to make remote memory access look like local memory access. So, Hardware DSM is very expensive. Software implementation is advantageous as in this case only software has to be installed. In Software DSM a software layer is added between the OS and application layers and kernel of OS may or may not be modified. Software DSM is more widely used as it is cheaper and easier to implement than Hardware DSM.

III. DSM – Pros and Cons Pros

Because of the combined advantages of the shared-memory and distributed systems, DSM approach is a viable solution for large-scale, high-performance systems with a reduced cost of parallel software development. **[5]**

In multiprocessor systems there is an upper limit to the number of processors which can be added to a single system. But in DSM according to requirement any number of systems can be added. DSM systems are also cheaper and more scalable than both multiprocessors and multi-computer systems. In DSM message passing overhead is much less than multi-computer systems.

Cons

Consistency can be an important issue in DSM as different processors access, cache and update a shared single memory space. Partial failures or/and lack of global state view can also lead to inconsistency.

IV. Implementation of DSM using OpenSHMEM

An Overview – OpenSHMEM

OpenSHMEM is a standard for SHMEM library implementations which can be used to write parallel programs in DSM environment. SHMEM is a communications library that is used for Partitioned Global Address Space (PGAS) **[9]** style programming. The key features of SHMEM include one-sided point-to-point and collective communication, a shared memory view, and atomic operations that operate on globally visible or "symmetric" variables in the program. **[10]**

Code Example

The code below shows implementation of parallel programs in DSM environment using OpenSHMEM.

```
#include <stdio.h>
#include <shmem.h>    //SHMEM library is included
#define LIMIT 7
long pSync[SHMEM_BARRIER_SYNC_SIZE];

int
pWrk[SHMEM_REDUCE_MIN_WRKDATA_SIZE
];
int global_data[LIMIT] = {1,2,3,4,5,6,7};
int result[LIMIT];
int main(int argc, char **argv)
{
   int rank, size, number, i, j;
   int  local_data[LIMIT];
   start_pes(0);
   size = num_pes();
   rank = my_pe();
   shmem_barrier(0,0,3,pSync);
   if (rank == 0)
   {
          for(i=0; i<LIMIT; i++)
                 local_data[i] = 0;
   //Local array is initialized
   }
```

```
  else
  {
          if (rank%2 == 1)
          {
              for(i=0; i<LIMIT; i++)
              {
                    local_data[i] = global_data[i] + 1;
              }
          }shmem_quiet();
          if(rank%2 == 0)
          {
              for(i=0; i<LIMIT; i++)
              {
                    local_data[i] = global_data[i] - 1;
              }
          }shmem_quiet();
   }
   shmem_int_sum_to_all(result,
local_data,LIMIT,0,0,size, pWrk,pSync);
   shmem_quiet();
   if (rank == 0)
   {
          printf("Updated Data\n");
          for(i=0; i<LIMIT; i++)
            printf("%3d", result[i]);
          printf("\n");
   }
   shmem_barrier_all();
   return 0;
}
```

In the above program, an array of integers is taken as input. Increment operation and decrement operation are performed on the array by multiple Processing Elements (PEs) in the network. PEs with odd *rank* perform increment and those with even *rank* perform decrement on the array. Finally sum of these values is shown as output.

Various functions of SHMEM library are used here. Below we are giving a brief overview of these functions.

start_pes() – This routine should be the first statement in a SHMEM parallel program. It allocates a block of memory from the symmetric heap.

num_pes() – This routine returns the total number of PEs running in an application.

my_pe() – This routine returns the processing element (PE) number of the calling PE. It accepts no arguments. The result is an integer between 0 and npes - 1, where npes is the total number of PEs executing the current program.

shmem_barrier(PE_start, ***logPE_stride***, ***PE_size, pSync)*** – This routine does not return until the subset of PEs specified by PE_start**,** logPE_stride and PE_size, has entered this routine at the same point of the execution path. The arguments are as follows:

PE_start – It is the lowest virtual PE number of the active set of PEs. PE_start must be of type integer.

logPE_stride - The log (base 2) of the stride between consecutive virtual PE numbers in the active set. *logPE_stride* must be of type integer.

PE_size – It is the number of PEs in the active set. *PE_size* must be of type integer. *pSync* - It is a symmetric work array.

shmem_quiet() – *It* is one of the most useful routines as it ensures ordering of delivery of several remote operations.

shmem_int_sum_to_all(target, source, nreduce, *PE_start****,*** *logPE_stride****,*** *PE_size****, pWrk, pSync)*** – It is a reduction routine which computes one or more reductions across symmetric arrays on multiple virtual PEs. Some of the arguments are same as mentioned above and the rest are as follows: *target* – It is a symmetric array of length nreduce elements to receive the results of the reduction operations.

source – It is a symmetric array, of length nreduce elements, that contains one element for each separate reduction operation. The *source* argument must have the same data type as *target*.

nreduce – It is the number of elements in the *target* and *source* arrays.

pWrk – It is a symmetric work array. The *pWrk* argument must have the same data type as *target*.

shmem_barrier_all() – This routine does not return until all other PEs have entered this routine at the same point of the execution path.

The code is compiled as following:

$oshcc <filename> -o <object_filename>

The code is executed as following:

$oshrun –np <PE_size> --hostfile <hostfile_name> <object_filename>

Here *hostfile* is a file containing the ip addresses of all PEs in the network. **[13]**

Output of the above code for PE_size = 3 was as shown below:

2 4 6 8 10 12 14

V. STM in DSM Environment

Software Transactional Memory (STM) **[12]** is a promising new approach to programming shared-memory parallel processors. It is an alternative approach to locks for solving the problem of synchronization in parallel programs. It allows portions of a program to execute in isolation, without regard to other, concurrently executing tasks. A programmer can reason about the correctness of code within a transaction and need not worry about complex interactions with other, concurrently executing parts of the program. Up till now STM codes have been executed in multiprocessor environment only. Many works are going on to implement STM in DSM environment (such as Atomic RMI) and it is expected that this will lead to improved performance of STM. **[11]** Atomic RMI is a distributed transactional memory frame-work that supports the control flow model of execution. Atomic

RMI extends Java RMI with distributed transactions that can run on many Java virtual machines located on different network nodes which can hosts a number shared remote objects.

VI. Conclusion

The main objective of this paper was to provide a description of the Distributed Shared Memory systems. A special attempt was made to provide an example of implementation of parallel programs in DSM environment using OpenSHMEM. From our point of view it seems further works in exploring and implementing DSM systems to achieve improved performance is quite promising.

References

[1]. Luiz Andre Barroso, Jeffrey Dean, Urs Holzle. "Web Search For a Planet: The Google Cluster Architecture," In: IEEE Micro,23(2):22-28, March-April 2003.
[2]. M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations", In: Proceedings of the 8th International Conference of Distributed Computing Systems, June, 1988.
[3]. Message Passing Interface (MPI) standard. http://www-unix.mcs.anl.gov/mpi/
[4]. M. J. Flynn, Computer Architecture: Pipelined and Parallel Processor Design, Jones and Barlett, Boston, 1995.
[5]. Jelica Protic, Milo Tomasevic, Veljko Milutinovic, "A Survey of Distributed Shared Memory Systems" Proceedings of the 28th Annual Hawaii International Conference on System Sciences, 1995.
[6]. V. Lo, "Operating Systems Enhancements for Distributed Shared Memory", Advances in Computers, Vol. 39, 1994.
[7]. Kai Li, "Shared Virtual Memory on Loosely Coupled Microprocessors" PhD Thesis, Yale University, September 1986.
[8]. S. Zhou, M. Stumn, Kai Li, D. Wortman, "Heterogeneous Distributed Shared Memory", IEEE Trans. On Parallel and Distributed Systems, 3(5), 1991.
[9]. PGAS Forum. http://www.pgas.org/
[10]. B. Chapman, T. Curtis, S. Pophale, S. Poole, J. Kuehn, C. Koelbel, L. Smith "Introducing OpenSHMEM, SHMEM for the PGAS Community", Partitioned Global Address Space Conference 2010.
[11]. Konrad Siek, Paweł T. Wojciechowski, "Atomic RMI: A Distributed Transactional Memory Framework" Poznan University of Technology, Poland, March 2015.
[12]. Ryan Saptarshi Ray, "Writing Lock-Free Code using Software Transactional Memory", Department of IT, Jadavpur University, 2012.
[13]. http://openshmem.org/site/Documentation/Manpages/Browse