EWOMP03 OMPlab proposal:

Experiment with Cluster-enabled OpenMP: OpenMP for Software DSM System on PC Clusters

Mitsuhisa Sato, Yoshinori Ojima
University of Tsukuba, Japan
msato@is.tsukuba.ac.jp

1. Introduction

We are developing a research OpenMP compiler for various parallel platforms including shared memory multiprocessors and a cluster of PC. Our objectives are providing a portable implementation for various native and advanced thread libraries, and enabling OpenMP to be used for parallel programming on a cluster. We use a software distributed shared memory system, SCASH, as an underlying runtime system of OpenMP on a cluster. This compiler, called Omni/SCASH, allows the programmers to execute OpenMP programs transparently on a cluster environment, and provides seamless programming environment from SMPs to clusters.

In the OMPlab, we would propose to run several codes from application developers on the cluster as well as on SMP machines using Omni OpenMP compiler.

2. Cluster-enabled OpenMP: Omni/SCASH

For programming distributed memory multiprocessors such as clusters of PC/WS and MPP, message passing programming is usually used. The message passing requires programmers to explicitly code the communication and makes writing parallel programs cumbersome.

The concept of "cluster-enabled OpenMP" is a system of OpenMP compiler and its runtime system which enables transparent execution of OpenMP programs on a cluster. For cluster-enabled OpenMP, we are using a page-based software distributed shared memory system, SCASH, as an underlying runtime system on a cluster. The Omni OpenMP compiler is a translator which takes OpenMP programs as input to generate the multithreaded C program with runtime library calls. It translates an OpenMP program into a code which runs on the SCASH.

SCASH is a page-based software distributed shared memory system provided as a part of SCore Cluster software. It use PM communication library for Myrinet and Ethernet, and various memory management functions, such as memory protection, supported by an operating system kernel. The consistency of shared memory is maintained on a per-page basis. We use the invalidate protocol of SCASH. The home node of a page is a node that keeps the latest data of the page. In the invalidate protocol, the home node sends an invalidation message to nodes which share the page so that the page is invalidated on these nodes. SCASH is based on the Release Consistency (RC) memory model with

the multiple writers protocol. The consistency of a shared memory area is maintained at a synchronization point called the "memory barrier synchronization" point. At this point, only modified parts are transferred to update pages. Explicit consistency management primitives are also supported for the lock operations.

In most SDSMs, only part of the address space is shared. In SCASH, the address space allocated by a shared memory allocation primitive can be shared among the processors. Variables declared in global scope are private in the processor. We define this memory model the ``shmem memory model". Parallel programs using the Unix ``shmem" system calls belongs to this memory model. In this model, all shared variables must be allocated at run-time at the beginning of execution. In the OpenMP programming model, global variables are shared as the default. To compile an OpenMP program into the "shmem memory model" of SCASH, the compiler transforms code to allocate a global variable in shared address space at run time. And it detects references to a shared data object, and rewrites them into the object which is to be allocated in shared memory area at run time.

The OpenMP directives are transformed into a set of runtime functions which use SCASH primitives to synchronize and communicate between processors. In SCASH, the consistency of all shared memory areas is maintained at a barrier operation. This matches the OpenMP memory model. The lock and synchronization operations in OpenMP use the explicit consistency management primitives of SCASH on a specific object.

3. OpenMP extension for Omni/SCASH

In SDSMs, the home node allocation of pages affects the performance because the cost of consistency management is large compared to that of hardware NUMA systems. In SCASH, a reference to a page in a remote home node causes page transfer through the network. When the home node of a page is different from the current node, the modified memory must be computed and transferred at barrier points to update the page in remote nodes. SCASH can deliver high performance for an OpenMP program if the placement of data and computation is such that the data needed by each thread is local to the processor on which that thread is running. In OpenMP, a programmer can specify thread-parallel computation, but its memory model assumes a single uniform memory and provides no facilities for laying out data onto specific distinct memory space. In addition, no loop scheduling method is provided to schedule in a way that recognizes the data access made by that iteration.

We have extended OpenMP directives with a set of directives to allow the programmer to specify the placement of data and computation on the shared address space. The data mapping directive specifies a mapping pattern of array objects in the address space. It is borrowed from High Performance Fortran (HPF). For example, the following directive specifies block mapping with the second dimension of a two-dimensional array A:

In Fortran:

```
  dimension A(100,200)
!$omn mapping(A(*,block))
```

In C:

```
 double A[200][100];
#pragma omni mapping(A[block][*])
```

The asterisk (*) for the first dimension means that the elements in any given column should be mapped in the same node. The block keyword for the second dimension means that for any given row, the array elements are mapped on each node in large blocks of approximately equal size. As a result, the array is divided into contiguous groups of columns, with home nodes for each group assigned to the same node. The keyword cyclic(n) can be used to specify cyclic mapping.

Since the consistency is maintained on a page-basis in SCASH, only page-granularity consistency is supported. If mapping granularity is finer than the size of the page, the mapping specification may not be effective. Different from HPF, each processor may have the entire copy of the array in the same shared address space. In this sense, this directive specifies ``mapping'' in the memory space, not ``distribution'' in HPF.

In addition to the data mapping directive, we have added a new loop scheduling clause, ``affinity'', to schedule the iterations of a loop onto threads associated with the data mapping. For example, the iterations are assigned to the processor having the array element a[i][*] in the following code:

```
#pragma omp for schedule(affinity,a[i][*])
 for(i = 1; i < 99; i++)
  for(j = 0; j < 200; j++)
   a[i][j] = ...;
```

Note that, in the current implementation, mapping and loop scheduling for only one of the dimensions can be specified because our current OpenMP compiler supports single level parallelism.

4. Proposal for EWOMP03 OMPlab

In the EWOMP03 OMPlab, we propose an experiment in which we take codes from application programmers to execute on the PC cluster by Omni/SCASH. Basically, Omni/SCASH is compatible with a conventional OpenMP, so that the application can be executed without any modification. To order to get reasonable performance, however, we should do turning by adding the Omni/SCASH extension for data placement. We would like to see how much performance we can get for real applications, and receive feed back from application users.

Unfortunately, the current version of Omni/SCASH can run only on the SCore cluster system. We want to use our PC cluster in our university, Japan, through the internet access. We will require the internet access from the workshop place.

We are currently working for the new version of Omni OpenMP compiler, which will include the new

version of SCASH system. The new SCASH system will run on conventional TCP/IP without SCore cluster software. If the new version is available until the workshop, we will hopefully be able to demonstrate our software on the machine prepared by the workshop organizer.

As well as the cooperation with application developers, we have a plan to collaborate with compiler-developers in the OMPlab. The group of University of Huston, leaded by Prof. Barbara Chapman, is working on the optimization of OpenMP programs by array privatization technique. We expect this optimization to help the performance improvement especially for Omni/SCASH. We will demonstrate this collaborative work with Barbara's group in my OMPlab.