OpenMP compiler for a Software Distributed Shared Memory System SCASH — Extended Abstract —

Mitsuhisa Sato, Hiroshi Harada and Yutaka Ishikawa

Real World Computing Partnership, Tsukuba, Ibaraki 305-0032, Japan
E-mail:{msato,h-harada,ishikawa}@trc.rwcp.or.jp

1 Introduction

In this paper, we present an implementation of OpenMP compiler for a page-based software distributed shared memory system, SCASH on a cluster of PCs.

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

OpenMP is attracting wide-spread interests because of its easy-to-use parallel programming model. While OpenMP is designed as a programming model for shared memory hardware, one way to support OpenMP in a distributed memory environment is to use a software distributed shared memory system (SDSM) as an underlying runtime system for OpenMP.

Our target SDSM is a page-based software distributed shared memory system, SCASH[2], which is running on a cluster of PCs connected by a high speed network Myrinet.

In the 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 call this memory model, "shmem memory model". Parallel programs using the Unix "shmem" system calls use this memory model. In this model, all shared variables must be allocated at run-time at the beginning of execution. We have implemented the OpenMP compiler for "shmem memory model" using Omni OpenMP compiler system [4]. The compiler detects references to a shared data object, and rewrite it to the reference into objects re-allocated in shared memory area.

H. Lu, et.al. [3] present an OpenMP implementation for TreadMarks[1] software DSM. Their compiler only supports a subset of OpenMP. Instead, they propose some modifications of OpenMP standard to make OpenMP programs run easier and more efficient. This approach may loose portability of OpenMP programs. Our compiler supports a full set of OpenMP so that OpenMP compliant programs runs on SDSMs without any modifications.

In the next section, we present overview of the Omni OpenMP compiler system and SCASH as background. Section 3 describes how to translate the OpenMP programs for "shmem memory model" and some preliminary results. Section 4 presents our concluding remarks.

2 Background

2.1 Omni OpenMP Compiler System

Our Omni compiler is a translator which takes OpenMP programs as input to generate the multi-threaded C program with runtime library calls. The compiler system consists of Omni Exc toolkit and language front-ends, as shown in Figure 1. C-front and F-front are front-end programs that parse C and Fortran source codes into intermediate codes, called Xobject code. Exc Java tool is a Java class library that provides classes and methods to analyze and modify the program easily with a high level representation, and to unparse the Xobject code into a C program. The representation of Xobject code is a kind of AST

(Abstract Syntax Tree) with data type information, which each node is a Java object that represent a syntactical element of the source code, and that can be easily transformed.

The Omni OpenMP compiler is implemented as a part of the Omni Exc toolkit. The translation pass from an OpenMP program to the target multi-threaded code is written by Java in the Exc Java tool.The generated program is compiled by the native back-end compiler linked with the runtime library. Currently, the OpenMP compiler for SMP platforms is freely available on our web site (`http://pdplab.trc.rwcp.or.jp/Omni/`).

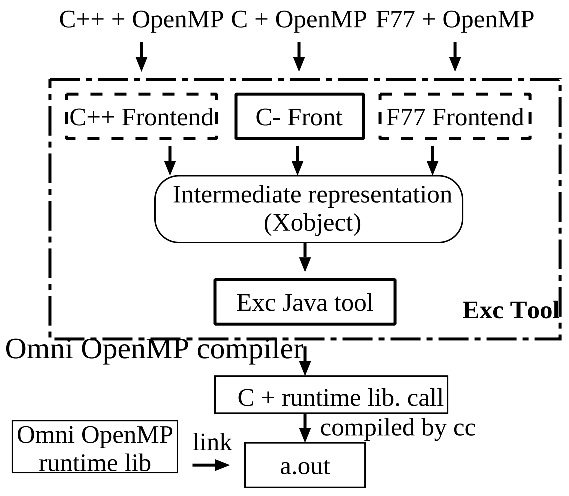

Figure 1: Overview of Omni OpenMP compiler

2.2 Software Distributed Shared Memory System SCASH

SCASH is a page-based software distributed shared memory system using the PM low-latency and high bandwidth communication library [5] for a Myrinet gigabit network and 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. SCASH supports two page consistency protocols, *invalidate* and *update*. 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 shares the page so that the page is invalidated on these node. The invalidate protocol is used as default.

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 called the "memory barrier synchronization" point such as a barrier. At this point, only modified part is transferred to update pages. Explicit consistency management primitives are also supported for the lock operations.

3 Translation of OpenMP Programs to SCASH

3.1 Transformation for "shmem memory model"

In OpenMP programming model, global variables are shared as default. To compile OpenMP program into "shmem memory model" of SCASH, the compiler transforms code to allocate a global variable in shared address space at run time. All declarations of global variables are converted into the pointers which contain the address of the data in shared address space. The compiler rewrites all references to the global variables to the indirect references through the corresponding pointers.

No. of nodes	seq	2	4	8	16	32
lap/BLK	17.74(1)	14.30(1.24)	7.84(2.26)	4.69(3.78)	2.88(6.16)	1.79(9.91)
lap/RR	17.74(1)	49.39(—-)	33.88(—-)	20.15(—-)	12.87(1.38)	10.15(1.75)
cg/BLK	83.79(1)	48.90(1.73)	29.49(2.84)	20.86(4.02)	18.08(4.63)	19.65(4.26)
cg/RR	83.79(1)	55.20(1.52)	33.88(2.47)	23.33(3.59)	18.83(4.44)	19.73(4.24)

Table 1: Execution time (in sec) and Speedup on Omni/SCASH

For example, a code:

```
double x;  /* global variable declaration */
double a[100];  /* global array declaration */
...
a[10] = x;
```

is transformed as follows:

```
double *_G_x;   /* indirect pointer to "x" */
double *_G_a;   /* indirect pointer to "a" */
...
(_G_a)[10] = (*_G_x); /* reference through the pointers */
```

The compiler generates the global data initialization function for each compilation unit. The initialization functions are linked and called at the beginning of execution to allocate the object in shared address space and store the address into the indirect pointer.

3.2 OpenMP directive translation and Runtime library

The OpenMP directives are transformed into a set of runtime functions which uses SCASH primitives to synchronize and communicate between processors. To translate a sequential program annotated with parallel directives into a fork-join parallel program, the compiler encapsulates each parallel region into a separate function. The master node calls the runtime function to invoke the slave threads which execute this function in parallel. All threads in each node are created at the beginning of execution, and waits for the fork operation on slave nodes. At the fork, pointers to shared variables with auto storage class are copied into shared memory heap and passed to slaves. No nested parallelism is supported.

In SCASH, the consistency of all shared memory area 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.

3.3 Preliminary performance

We take two kinds of benchmarks for performance evaluation: The benchmark ”lap” is a simple Laplace equation solver with 4-point stencil operation (1024*1024 and 50 iterations). The benchmark ”cg” is NAS parallel benchmark CG of class A, written in C. Table 1 shows the preliminary performance on the RWC PC cluster II (Pentium Pro 200MHz, 256MB memory, Myrinet network, Linux).

We have implemented two kinds of home distributions, block distribution BLK and round-robin distribution RR. In SCASH, home nodes are assigned to the pages in a round-robin manner as default. We extended our OpenMP compiler to allow the user to specify the block distribution on an array object by pragma. In BLK, the large shared objects are equally divided into successive blocks for nodes. In ”lap” benchmark, we found that the home distribution gives great impact on the performance. In ”cg”, the distribution affects less that in ”lap”, and its performance does not scale on more than 8 nodes.

It should be noted that the overhead of rewriting global variable references by indirect pointers is very small in these benchmarks.

4 Concluding Remarks

We have presented an OpenMP compiler for a full set of OpenMP on the software distributed shared memory system, SCASH, and its performance on the RWC PC cluster. We found that the page home distribution is a key to achieve the good performance on the SDSM.

Currently, we are introducing OpenMP extensions to specify data mapping in a flexible way, which gives application specific knowledge to the compiler. The loop scheduling to exploit locality for data mapping is also an important issue to investigate. Recently, SCASH supports the *dynamic home node reallocation mechanism*[2]. We expect that this mechanism will improve the performance of OpenMP programs on SCASH.

References

[1] C. Amza, A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, W. Zwaenepoel. “Treadmarks: Shared memory computing on networks of workstation”, IEEE Computer, 29(2):18-28, Feb. 1996.

[2] H. Harada, Y. Ishikawa, A. Hori, H. Tezuka, S. Sumimoto, T. Takahashi, “Dynamic Home Node Reallocation on Software Distributed Shared Memory”, In Proc. of HPC Asia 2000, pp. 158–163, Beijing, China, May, 2000.

[3] H. Lu, Y. C. Hu, W. Zwaenepel. “OpenMP on Network of Workstations”, In Proc. of Supercomputing’98, Nov. 1998.

[4] M. Sato, S. Satoh, K. Kusano, Y. Tanaka. “ Design of OpenMP Compiler for an SMP Cluster”, In Proc. of Proc. of 1st European Workshop on OpenMP (EWOMP’99), pp. 32-39, Lund, Sweden, Sep. 1999.

[5] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An Operating System Coordinated High Performance Communication Library. *Lecture Note in Computer Science, High-Performance Computing and Networking*, pages 708–717, 1997.