OpenMP: Parallel programming API for shared memory multiprocessors and on-chip multiprocessors

(Extended abstract)

Mitsuhisa Sato
University of Tsukuba
1-1-1 Tenou-dai,Tsukuba,
Ibaraki 305-8577 Japan
msato@is.tsukuba.ac.jp

Categories & Subject Descriptors: D.3.3 [Programming Languages]: Language Contructs and Features – *Concurrent programming structures.* D.1.3 [Software]: Concurrent Programming – *Parallel Programming*

General Terms: Languages, Performance

1. INTRODUCTION

The OpenMP Application Programming Interface is an emerging standard for parallel programming on shared-memory multiprocessors. Recently, OpenMP is attracting widespread interest because of its easy-to-use portable parallel programming model.

OpenMP is not a new language. It extends existing languages such as FORTRAN and C/C++ with a set of directives. The OpenMP API [1] defines a set of directives that augment standard C/C++ and Fortran 77/90. In contract to previous API such as POSIX threads and MPI, OpenMP facilitates an incremental approach to the parallelization of sequential program. The programmer may add parallelization directives to loops or statements in the program.

OpenMP is currently used for high performance computing applications running on shared memory multiprocessors. It is also of interest to the cluster computing community, because many recent clusters are built from shared memory nodes. OpenMP API can be used to exploit parallelism on a node while a message passing API is used between nodes.

As more transistors are integrated onto bigger die, an on-chip multiprocessor will become a promising alternative to the complex superscalar microprocessor that is commonly used today. While a trend seems to be toward CPUs with wider instruction issues and support for larger amount of speculative execution, an on-chip multiprocessor composed of simpler processors may exploit thread-level parallelism efficiently. For such on-chip multiprocessor, parallel (or multi-threaded) programming is required. OpenMP will offer an every ease-to-use simple parallel programming environment to make use of an on-chip multiprocessor.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISSS'02, October 2–4, 2002, Kyoto, Japan.
Copyright 2002 ACM 1-58113-562-9/02/0010…$5.00.

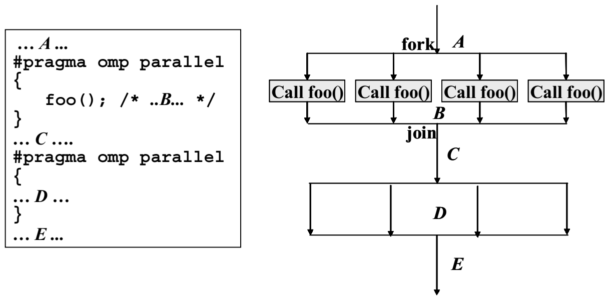
The Omni OpenMP compiler [2] is a production-level research prototype compiler for OpenMP, which supports C and Fortran 77. One of our project objectives of the Omni OpenMP compiler project is providing a portable implementation of OpenMP for SMPs and several native and experimental thread libraries.

In this paper, we describe a brief introduction of OpenMP API and its parallel programming in the next section. In section 3, we present our Omni OpenMP complier and performance of some applications on a shared memory multiprocessor. In section 4, a role of OpenMP for modern on-chip multiprocessors is discussed.

2. PROGRAMMING IN OPENMP

OpenMP provides three kinds of directives: parallelism/work sharing, data environment, and synchronization. We only describe a brief introduction of OpenMP here. Refer to the OpenMP standard for the full specification [1].

OpenMP uses the fork-join model of parallel execution. An OpenMP program begins execution as a single process, called the master thread of execution. The fundamental directive for expressing parallelism is the parallel directive. It defines a parallel region of the program, which is executed by multiple threads. When the master threads enters a parallel region, it forks a team of t threads (one of them being the master thread), and work is continued in parallel among these threads. Upon exiting the parallel construct, the threads in the team synchronize (join the master), and only the master continues execution. The statements in the parallel region, including functions called from within the enclosed statements, are executed in parallel by each thread in the team. The statements enclosed lexically within a construct define the static extent of the construct. The dynamic extent further includes the functions called from within the construct.



```
... A ...
#pragma omp parallel
{
    foo(); /* ..B... */
}
... C ....
#pragma omp parallel
{
... D ...
}
... E ...
```

Figure 1. OpenMP for-Execution Model

All threads replicate the execution of the same code, unless a work sharing directives is specified within the parallel region. Work sharing directives, such as for divide the computation among threads. For example, “for” directive (DO directive in FORTRAN) specifies that the iterations of the associated loop should be divided among threads so that each iteration is performed by a single thread. The data environment directives control the sharing of program variables defined outside of a parallel region. Variables default to shared, which means shared among all threads in a parallel region. A private variable has a copy per thread. The reduction directive identifies reduction variables. In the example shown in Figure 2, the most outer loop for sparse matrix-vector multiplication is parallelized by “for” directives. The temporary variables such as “j”, “t”, “start”, “end” are specified as private variables in the parallel region.

```
Matvec(double a[],int row_start,int col_idx[],
 double x[],double y[],int n)
{
   int i,j,start,end; double t;
#pragma omp parallel for private(j,t,start,end)
   for(i=0; i<n;i++){
      start=row_start[i];
      end=row_start[i+1];
      t = 0.0;
      for(j=start;j<end;j++)
         t += a[j]*x[col_idx[j]];
      y[i]=t;
  }
}
```

Figure 2. Example of OpenMP programming:

Sparse matrix-vector routine parallelized by OpenMP

3. OMNI OPENMP COMPILER AND PERFORMANCE ON SMP

The Omni OpenMP compiler is a translator which takes OpenMP programs as input to generate the multithreaded C program with runtime library calls.

To translate a sequential program annotated with OpenMP parallel directives into a fork-join parallel program, the compiler encapsulates each parallel region into a separate function. The master thread calls the runtime library to invoke the slave threads which are executed this function in parallel. Pointers to shared variables with auto storage class are passed to slaves at the fork. Private variables are re-declared in the functions generated by compiler. The work sharing and synchronization constructs are

Table 1. Performance of SPEC Climate on COMPAQ ES40 (Alpha EV67 666MHz, L2 8M, 1GB memory, Linux2.2.14-6.0smp)

#threads	1	2		4	
Small	405	213	(1.90)	130	(3.11)
Medium	4972	2568	(1.94)	1380	(3.60)
Large	40984	21216	(1.93)	11598	(3.53)

Table 2. Performance of SPECClimate on COMPAQ ProLiant6500 (Pentium II Xeon 450MHz, 4CPU, 1GB memory, Linux2.2.16)

#threads	1	2		4	
Small	1203	657	(1.83)	375	(3.20)
Medium	14691	7816	(1.88)	4239	(3.47)
Large	121462	63058	(1.93)	34708	(3.50)

translated into codes that contain the corresponding runtime library calls.

This transformation pass is written in Java using the Exc Java toolkit. The generated program is compiled by the native back-end C compiler linked with the Omni OpenMP runtime library.

For SMP platforms, the runtime library includes microtasking and synchronization primitives on top of the different thread libraries including POSIX threads and Solaris thread, “sproc” of SGI IRIX, StackThreads/MP. StackThreads/MP [5] is a thread library, developed by the University of Tokyo, which supports fine-grain multithreading in GCC/G++. It tolerates a large number of threads far beyond the number of processors, and imposes a very small overhead for creating and terminating a thread. The runtime library using the StackThreads supports the nested and irregular parallelism efficiently.

Table 1 and 2 show the performance of the SPECClimate benchmark from SPEC HPC suites on COMPAQ ES40, ProLiant6500 respectively, using our Omni OpenMP compiler. This program is known as the climate simulation program MM5, which is parallelized by using OpenMP. The detail of the performance of Omni OpenMP compiler is reported in [3].

4. A ROLE OF OPENMP FOR ON-CHIP MULTIPROCESSORS APPLICATION

An on-chip multi-processor with simple processors is a promising approach to exploit thread-level coarse-grain parallelism. In [3], Olukotun et. al. discussed the design trade-offs between wide-issue processors and on-chip multiprocessors in the same dai-area. They found that for applications with large amount of parallelism on-chip multiprocessor architecture outperforms the superscalar architecture. On-chip multiprocessors offer localized implementation of high-clock rate and low latency communication. To exploit thread-level parallelism, OpenMP can serve as programming environment to write multithreaded applications.

Another opportunity to use OpenMP is modern multi-threaded architecture. For instance, HyperThreading, Intel's simultaneous multithreading technique yields thread-level parallelism on a single processor. The availability of the HyperThreading feature within the Intel Xeon MP (Prestonia) server/workstation chip should aid in threaded desktop application software development. Intel was also promoting the OpenMP API, which supports writing multithreaded applications.

The current most OpenMP compilers and runtime system are designed for shared memory multiprocessors. For on-chip multiprocessors and multithreaded microprocessors, an OpenMP implementation could be re-designed to make use of fast synchronization and low latency communications between thread in run-time system and compiler. Flexible thread runtime scheduling is also an interesting design issue of OpenMP to exploit nested dynamic parallelism using fast synchronization of on-chip multiprocessor.

OpenMP can be used as a backend of some automatic parallelizing compiler. The automatic paralleling compiler translates sequential code into multi-threaded code with OpenMP directives. It helps users to port an existing application on on-chip multiprocessors.

5. CONCLUDING REMARKS

The OpenMP Application Programming Interface is an emerging standard for parallel programming on shared-memory multiprocessors. On-chip multiprocessor architecture will be easier to implement with a high clock rate and low latency communication. For such on-chip multiprocessors, OpenMP offers an easy-to-use parallel programming environment to develop multi-threaded applications on on-chip multiprocessors. We are developing a portable OpenMP compiler, called Omni OpenMP compiler system. While the current OpenMP compiler and runtime system are designed for shared memory multiprocessors, OpenMP should be re-designed for on-chip multiprocessors to make use of fast synchronization and low latency communication and to support flexible thread scheduling for nested dynamic parallelism.

6. REFERENCES

[1] OpenMP ARB, "OpenMP API", http://www.openmp.org/

[2] Omni OpenMP Compiler Project, http://www.hpcc.jp/Omni/

[3] Kunle Oluktun, Basem A. Nayfeh, Lance Hammond, Ken Wilson and Kunyung Chang, "The Case for a Single-Chip Multiprocessor", ASLOPS VII, 1996.

[4] Kazuhiro Kusano, Shigehisa Satoh and Mitsuhisa Sato, "Performance Evaluation of the Omni OpenMP Compiler", WOMPEI (part of ISHPC2K), LNCS 1940, pp. 403-414, Tokyo, Oct, 2000.

[5] Yoshizumi Tanaka, Kenjiro Taura, Mitsuhisa Sato, Akinori Yonezawa, "Performance Evaluation of OpenMP Applications with Nested Parallelism". LCR 2000, pp. 100-112.