Performance Characteristics of OpenMP Constructs, and Application Benchmarks on a Large Symmetric Multiprocessor

Nathan R. Fredrickson
fredrick@ee.queensu.ca

Ahmad Afsahi*
ahmad@ee.queensu.ca

Ying Qian
qiany@ee.queensu.ca

Department of Electrical and Computer Engineering
Queen's University
Kingston, ON K7L 3N6
(613) 533-3068

ABSTRACT

With the increasing popularity of small to large-scale symmetric multiprocessor (SMP) systems, there has been a dire need to have sophisticated, and flexible development and runtime environments for efficient and rapid development of parallel applications. To this end, OpenMP has emerged as the standard for parallel programming on shared-memory systems. It is very important to evaluate the performance of OpenMP constructs, kernels, and application benchmarks on large-scale SMP systems. We present the performance of the basic OpenMP constructs, class B of NAS OpenMP 3.0 benchmarks, and the SPEC OMPL2001 application benchmarks (large data set) on a contemporary 72-node Sun Fire 15K SMP node. We report the basic timings, scalability, and runtime profiles of different parallel regions within each benchmark in the NAS OpenMP 3.0, and the SPEC OMPL-2001 suites. We elaborate on the performance differences between the medium and large classes of the SPEC OMP2001 suites on our system, as well as a comparison among a number of large-scale symmetric multiprocessors for the SPEC OMPL2001.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems

General Terms

Performance, Measurement

*Correspondence to Ahmad Afsahi

ICS'03, June 23–26, 2003, San Francisco, California, USA.

Keywords

Performance Evaluation, SPEC OMPL2001, NAS OpenMP, High-Performance Computing, SMP, OpenMP

1. INTRODUCTION

Cache-coherent, shared-memory multiprocessor (SMP) systems have gained prominence in the market place. Considerable work has gone into the design of SMP systems, and several vendors such as IBM, Sun, Compaq, SGI, and HP offer small to large-scale shared memory systems. Recent trends clearly show that large-scale SMPs continue to become commercially available. Sun Microsystems has recently introduced its Sun Fire 15K SMP, supporting 72 to 106 processors, backed up with its Sun Fireplane crossbar interconnect [10]. The Sun Fireplane uses one to four levels of interconnects to provide better shared-memory performance.

With the increasing popularity of SMP systems, there has been a dire need to have sophisticated, and flexible development and runtime environments for efficient and rapid development of parallel applications. OpenMP [3] has emerged as the standard for parallel programming on shared-memory systems. The OpenMP application programming interface (API) consists of a set of compiler directives, library routines, and environment variables that are used to define parallel regions and share work between threads. The directive-based interface and the ability to incrementally parallelize a program make OpenMP one of the easiest ways to parallelize both existing and new applications. An OpenMP-aware compiler will use the directives to generate multi-threaded code suitable for a shared-memory machine.

OpenMP parallel applications are platform independent. However, this ease of use and portability is due to the hiding of many details from the user. The directives provide an abstraction that hides the details of creating a multithreaded parallel application. As a result, the compiler and runtime environment may have dramatic effects on the performance of an OpenMP application. Understanding the performance and scalability of OpenMP constructs on specific systems is critical to the development of efficient parallel programs. Previous research [9, 8, 16, 7, 14] has presented OpenMP constructs' overhead and scalability on a wide variety of

systems. Meanwhile, it is highly desirable to evaluate the performance of parallel applications that use such OpenMP constructs.

As SMPs become more commonplace, it is very important to assess their performance using microbenchmarks, kernels, and application benchmarks. The Edinburgh Parallel Computing Centre (EPCC) has released a set of OpenMP Microbenchmarks to evaluate the performance of different OpenMP constructs [9]. The NAS Parallel Benchmarks (NPB) [6] were designed to compare the performance of parallel computers. Recently, the NASA Ames Research Center has made publicly available an OpenMP implementation of its NAS Parallel Benchmarks in the NAS 3.0 suite [13]. The Standard Performance Evaluation Corporation (SPEC) has released a suite of OpenMP application benchmarks, called SPEC OMP2001, to evaluate the performance of contemporary SMP systems [1]. The SPEC OMPM2001 suite (referred to as the medium suite) is targeted toward mid-range parallel computers. Some performance results on a number of SMP systems have been reported in [4, 5, 19]. Recently, SPEC has released a larger data set for the suite, SPEC OMPL2001, for focusing on 16-way and larger systems. Some preliminary results have been presented in [17, 19].

There are two main contributions of this paper. First is to evaluate the performance of OpenMP constructs and application benchmarks on a 72-way Sun Fire 15K multiprocessor system. We present the performance of basic OpenMP constructs using the EPCC Microbenchmarks, class B of the NAS OpenMP 3.0 Parallel Benchmarks, and the large class of the SPEC OMP2001 benchmarks. Secondly, this paper presents the runtime profile characteristics of these application benchmarks including the parallel regions and directives invoked. To the best of our knowledge, this is the first attempt at profiling these applications and studying the scalability and performance characteristics of them on such an SMP system.

The rest of this paper is organized as follows. Section 2 gives an overview of the EPCC microbenchmark suite, the NAS OpenMP 3.0, and the SPEC OMPL2001 application benchmark suites. In Section 3, we describe our platform, and runtime environment. Section 4 presents the experimental results for different synchronization directives, scheduling policies, and other constructs in OpenMP. We report the basic timings, scalability, and runtime profiles of different parallel regions within each benchmark in the NAS OpenMP 3.0, and the SPEC OMPL2001 suites. We elaborate on the performance differences between the medium and large classes of the SPEC OMPL2001 benchmarks, as well as a comparison among a number of large-scale symmetric multiprocessors. In Section 5, we mention the related work. Finally, Section 6 concludes the paper.

2. BENCHMARKS

We use three benchmarks to evaluate the OpenMP performance on our large-scale share-memory system: the EPCC microbenchmarks, the NAS OpenMP benchmarks, and the SPEC OMPL2001 application benchmarks. Together, these benchmarks provide a comprehensive performance evaluation of the Sun Fire 15K system.

2.1 EPCC Microbenchmarks

The performance of OpenMP constructs vary across different architectures, and even different compilers on the same machine. The performance of an OpenMP application is at least partially dependent on the implementation of the OpenMP runtime library and the thread runtime environment provided by the operating system. The directive-based nature of OpenMP makes measurement of overhead more difficult than other parallel programming paradigms, such as message passing interface (MPI) [2]. Since it is not possible to directly measure the overhead of many directives, their overhead must be measured indirectly. First a reference time is measured for the execution of a parallel section or loop without OpenMP directives. This reference is then compared to the execution time of the same code with OpenMP directives to find the overhead. The EPCC Microbenchmarks [9] measure the overheads of synchronization and loop scheduling in the OpenMP runtime library.

The synchronization benchmark measures the overhead incurred by work-sharing and mutual exclusion directives. The work-sharing directives include PARALLEL, DO/FOR, PARALLEL DO/FOR, and BARRIER. The mutual exclusion directives include SINGLE, CRITICAL, LOCK/UNLOCK, ORDERED, and ATOMIC. Overhead is defined in terms of the sequential time, T_s, and the parallel time, T_p, on p processors. The overhead is given by $O_p = T_p - T_s/p$.

The loop scheduling benchmark compares the scheduling policies available with OpenMP. Specifically, it compares the overhead of the DO directive when used with three scheduling policies: STATIC, DYNAMIC, and GUIDED. The STATIC policy determines scheduling at compile time and is well suited for programs with static workloads that can be easily divided among threads. The DYNAMIC and GUIDED scheduling policies are intended for programs with dynamic workloads that must be balanced between threads at runtime. All three policies have an additional parameter, chunk size, which specifies the size of a single work unit in terms of loop iterations. A smaller chunk size allows for finer-grained scheduling at the cost of more scheduling overhead. The GUIDED policy attempts to balance this tradeoff by dynamically decreasing the chunk size. The overhead of the scheduling benchmark is defined and measured the same as the synchronization benchmark.

2.2 NAS OpenMP Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) [6] has been widely used to characterize high-performance computers. Recently, the NAS 3.0 OpenMP suite [13] has been released. The suite consists of five kernels, (CG, MG, FT, IS, EP), and three simulated CFD applications (BT, SP, LU). The five kernels mimic the computational core of five numerical methods used by CFD applications. The three simulated CFD applications reproduce much of data movement and computation found in full CFD codes.

2.3 SPEC OMPL2001 Benchmarks

The SPEC OMP2001 suite of benchmarks was developed by the SPEC High-Performance Group [1]. The suite consists of a set of OpenMP-based scientific applications. These programs were originally part of the SPEC CPU2000 suite and were parallelized by inserting OpenMP directives.

In June 2002, SPEC OMPL2001, was released with larger data sets and modified code to achieve better scaling. It

Table 1: Overview of SPEC OMPL2001.

Benchmark	Field	Language	Lines
wupwise	Quantum Chromodynamics	Fortran	2200
swim	Weather Prediction	Fortran	400
mgrid	Fluid Dynamics	Fortran	500
applu	Fluid Dynamics	Fortran	4000
equake	Earthquake Simulation	C	1500
apsi	Pollution Modeling	Fortran	7500
gafort	Genetic Algorithm	Fortran	1500
fma3d	Crash Simulation	Fortran	60000
art	Image Recognition	C	1300

is intended to measure the performance of large shared-memory systems with at least sixteen processors. The suite typically requires 6.4GB of memory for execution, and currently consists of nine programs that are listed in Table 1.

We give a brief overview of each application. The interested reader is referred to [1]. WUPWISE is a physics program in the field of Quantum Chromodynamics. It solves a problem in the area of lattice gauge theory. SWIM is a weather prediction program. It uses the finite difference method to solve the shallow water equations. MGRID is a program in the field of computational fluid dynamics. It is a very simple multigrid solver for computing a three dimensional potential field. APPLU solves five coupled nonlinear PDEs on a 3-dimensional logically structured grid, using the symmetric successive Over-Relaxation implicit time-marching scheme. EQUAKE is a simulator of seismic wave propagation in large basins. The goal is to recover the time history of the ground motion everywhere within the valley due to a specific seismic event. APSI is an environmental modeling program that simulates a lake environment. GAFORT computes the global maximum fitness using a genetic algorithm. FMA3D is a finite element method computer program designed to simulate the inelastic, transient dynamic response of three-dimensional solids and structures subjected to impulsively or suddenly applied loads. ART is an image recognition program to recognize objects in a thermal image. The objects are a helicopter and an airplane.

3. EXPERIMENTAL METHODOLOGY

We tested the performance of the OpenMP constructs, and applications on a 72-node Sun Fire 15K at the High Performance Computing Virtual Laboratory (HPCVL) at Queen's University. The 900MHz UltraSPARC III processors in Sun Fire 15K are arranged four per system board. Each processor has 8 MB of ECC-protected external cache. The system has 144 GB of RAM provided by 150-MHz DIMM memory modules connected by a 128-bit-wide data path, and 11.7 TB of Sun StorEdge T3 disk storage. The software environment included Sun Solaris 8, and the Sun Forte Developer 6, update 2.

We had exclusive access to the Sun Fire 15K system during our experimentation. We experimented with the EPCC microbenchmark suite [9], and the class B of the NAS OpenMP suite, version 3.0 [13]. Each program in SPEC OMPL2001 [1] was run with 12, 24, 48, 64, and 70 threads/processors. Ideally speedups should be calculated relative to the execution time on a single thread. However due to the size of the SPEC OMPL2001 applications, running with a single thread was not feasible. We observed that running the applications with 72 threads (equal to the number of processors) caused extremely poor performance and thus was avoided in experiments. Also note that we did not modify the codes. Due to limited exclusive access to the system and the amount of time needed to run these large benchmarks, we only present results for seven of the nine SPEC OMPL2001 programs at this time. We will report the results for the APPLU and GAFORT applications, in future.

4. EXPERIMENTAL RESULTS

In this section, we present results for the EPCC microbenchmarks, the NAS OpenMP benchmarks, and the SPEC OMPL-2001 application benchmarks.

4.1 EPCC

4.1.1 Synchronization Directives

The synchronization benchmark measures the overhead of the most common OpenMP directives. The PARALLEL directive defines a parallel region at which threads are created upon entry and rejoined upon completion. The DO and FOR directives are used to mark a parallel loop, in Fortran and C, respectively. A DO/FOR directive can be combined with the PARALLEL directive into a single PARALLEL DO/FOR directive. The BARRIER and mutual exclusion directives must appear within a parallel region and represent a logical barrier among all threads and a mutually exclusive section, respectively.

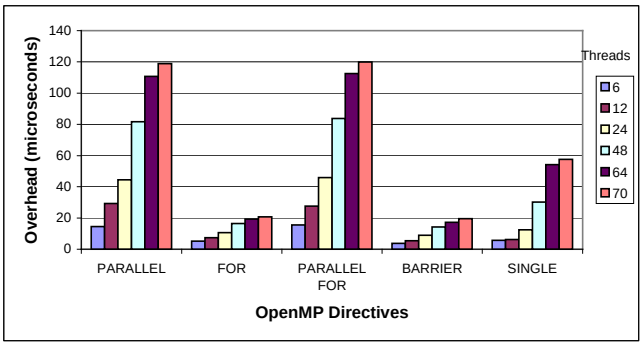

Figure 1: Overhead of OpenMP synchronization and work-sharing directives (C version).

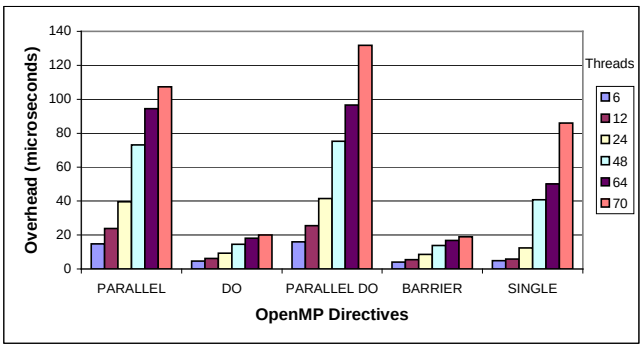

Figure 2: Overheads of OpenMP synchronization and work-sharing directives (Fortran version).

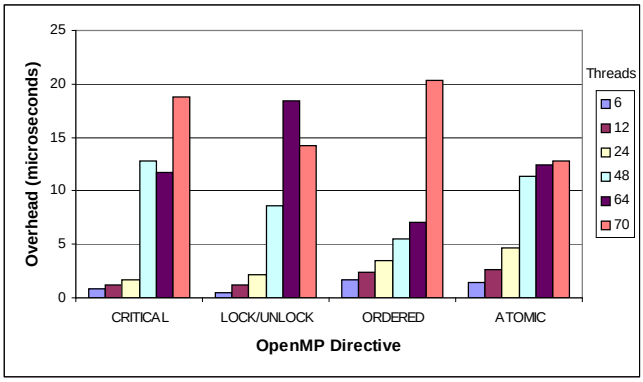

Figure 3: Overheads of OpenMP mutual exclusion directives (C version).

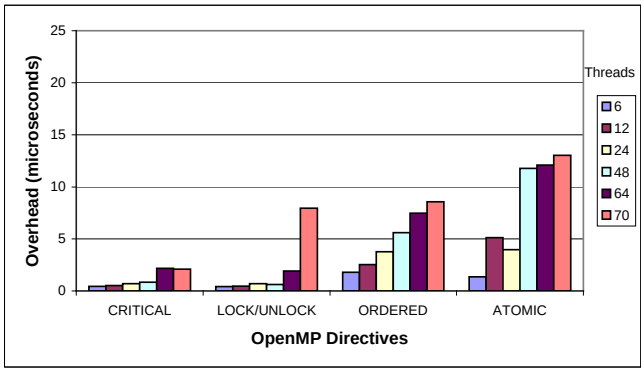

Figure 4: Overheads of OpenMP mutual exclusion directives (Fortran version).

Figures 1 and 2 present the overheads of the OpenMP synchronization and work-sharing directives for C, and Fortran, respectively. The overheads for the work-sharing directives in C are relatively similar to the overheads in Fortran. It seems there is no clear benefit to either language. The basic and combined PARALLEL directives have the largest overhead. This is expected since this is when the threads are initially spawned and eventually rejoin. The scalability of the statically scheduled DO/FOR directive is similar to a BARRIER. The DO/FOR overhead is only slightly higher than BARRIER for all numbers of threads. This implies that nearly all the cost of the DO/FOR is the implicit BARRIER at the end of the loop.

A common question among OpenMP developers is whether to parallelize a sequence of loops with multiple PARALLEL DO directives or with a single PARALLEL directive containing multiple DO directives. The answer is entirely system dependent. For our system, it is clearly better to use a single PARALLEL directive due to the high overhead of the PARALLEL directive relative to the DO/FOR directive.

Figures 3 and 4 present the overheads of the OpenMP mutual exclusion directives for C, and Fortran, respectively. These directives all have relatively low overheads ranging from a few microseconds up to at most 20 microseconds. For the most part, overhead increases with the number of threads as expected. The Fortran versions of these directives clearly scale better than the C version. The overhead of the

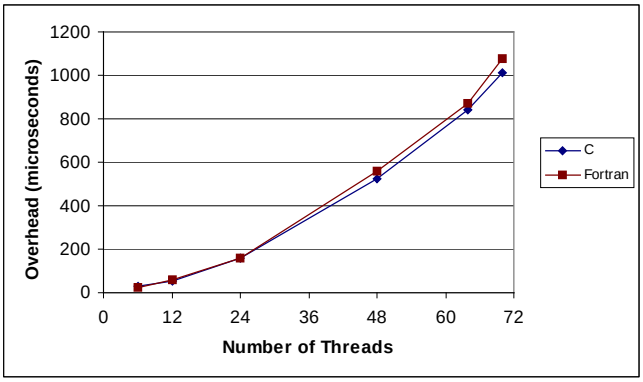

Figure 5: Overhead of REDUCTION directive.

C version significantly increases between 24 and 48 threads (except for the ORDERED).

Figure 5 presents the overhead of the REDUCTION directive. This directive has a high overhead relative to the other synchronization directives. However it does perform more work by combining multiple instances of variables as threads rejoin upon completion of a parallel section. The REDUCTION directive scales smoothly with no sudden jumps and the C and Fortran results are very close.

4.1.2 *Scheduling*

OpenMP provides three options for scheduling loop iterations among threads: STATIC, DYNAMIC, and GUIDED. Figures 6 and 7 show the scheduling overheads with 24, and 6 threads, respectively, for the C version of the scheduling benchmark. The most fine-grained scheduling policy is DYNAMIC with a chunk size of one, or DYNAMIC(1). As expected, this policy has the highest overhead. The overhead of DYNAMIC(n) decreases as the chunk size increases.

The overhead of GUIDED(n) is better than DYNAMIC(n) for small chunk sizes. This is expected since n under the GUIDED policy represents the minimum chunk size. The initial chunks scheduled will be larger than n resulting in less scheduling overhead. Since n is a minimum chunk size the overhead of GUIDED(n) with large n would be the same as DYNAMIC(n). EPCC only reports the overhead of GUIDED(n) for small values of n.

The STATIC scheduling is multiple orders of magnitude faster than DYNAMIC scheduling, but of course does not have the benefit of dynamic load balancing. The overhead of STATIC(n) scheduling is nearly constant across all chunk sizes n. The overhead of STATIC(n) matches the overhead of STATIC.

As shown in the Figures 6 and 7, the overhead of both the DYNAMIC and GUIDED policies increase by an order of magnitude between 6 and 24 threads. The overhead of the STATIC policy increases only slightly. It is clear that if the workload is balanced between threads, then STATIC scheduling with its low overhead is clearly the best choice. However if the workload is dynamic and unbalanced, then either GUIDED or DYNAMIC scheduling could perform better if the benefit realized outweighs the extra overhead incurred by those policies.

It is worth mentioning that only one SPEC OMPL2001 program, ART, contains an OpenMP directive that specifies a scheduling policy. It simply specifies DYNAMIC without

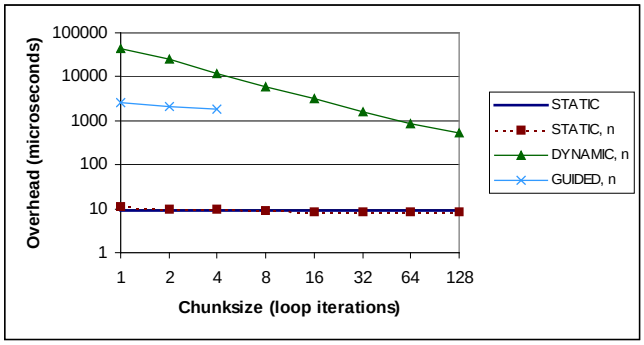

Figure 6: Overhead of thread scheduling policies in OpenMP with 24 threads (C version).

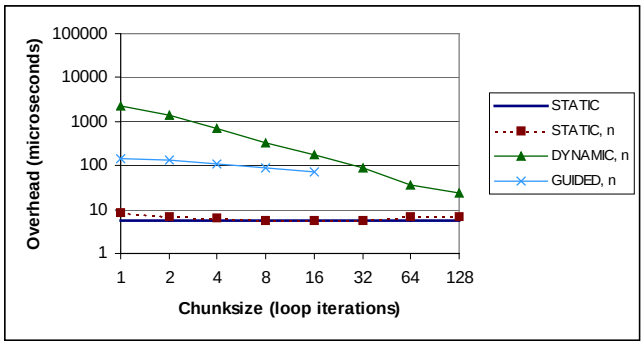

Figure 7: Overhead of thread scheduling policies in OpenMP with 6 threads (C version).

specifying a chunk size. This will result in the same block scheduling as STATIC, except that the scheduling decisions are made at runtime. All of the other SPEC OMPL2001 application benchmarks do not specify a scheduling and will default to the STATIC policy.

4.1.3 *Semantically Equivalent Directives*

The two most commonly used OpenMP directives in the SPEC OMPL2001 programs are PARALLEL and DO (FOR directive in C). These directives can either be used separately or combined into a single PARALLEL DO directive. Table 2 compares the overheads of these three directives for both C and Fortran with 64 threads. In both cases, the PARALLEL directive is significantly more costly than the DO directive. As discussed in Section 4.1.1, if a section of code contains multiple parallel loops it is clearly better to use multiple DO directives within a single PARALLEL region.

Table 2: Overhead of most commonly used directives with 64 threads.

	Overhead (microseconds)	
Directive	Fortran	C
`PARALLEL`	94.5	110.7
`DO (FOR)`	18.2	19.3
`PARALLEL DO (FOR)`	96.6	112.5
`PARALLEL + DO (FOR)`	112.6	130.0

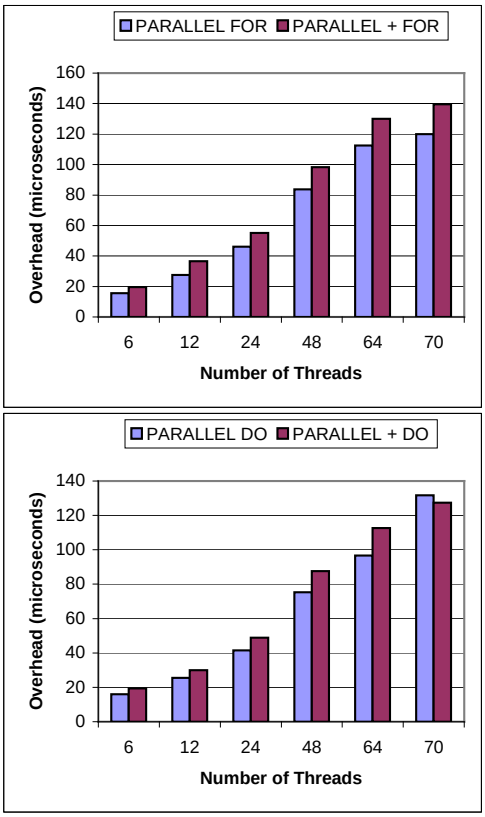

Figure 8: In general, there is approximately 20 percent more overhead using separate PARALLEL and DO directives than combining into a single PARALLEL DO directive (PARALLEL FOR in C).

However, if a section of code contains only a single parallel loop, it is not clear what style is better to use. On our system, there is approximately 20% less overhead when the directives are combined into a single PARALLEL DO directive, even though the code is functionally equivalent as having two directives. Depending on the overall OpenMP overhead in a given program, this benefit may or may not be significant. The extra overhead is increasingly significant on larger systems where more threads are run, as shown in Figure 8.

4.2 NAS OpenMP 3.0

Table 3 presents the execution time (in seconds) for the class B of the six different benchmarks in the NAS OpenMP 3.0 suite. The scalability is shown in Figure 9. The CG benchmark shows a superlinear scalability. The LU benchmark achieves a perfect scalability (except for 70 threads). The BT, SP, and MG benchmarks show relatively good performance. However, the performance of FT is very poor.

4.3 SPEC OMPL2001

The execution time was measured during runs of seven SPEC OMPL2001 benchmark applications with 12, 24, 48, 64, and 70 threads. Table 4 presents the execution time results, and Figure 10 presents the results in terms of application speedup. The speedup is normalized to twelve threads, which was the smallest test run.

Table 3: Execution time for the NAS OpenMP 3.0 parallel benchmarks (in seconds).

	Benchmark	
Threads	BT	SP
1	2586	2929
4	663	740
9	303	312
16	174	180
25	105	107
36	78	81
49	73	76
64	52	61
70	70	59

	Benchmark			
Threads	LU	CG	MG	FT
1	4295	2872	154	763
2	1921	1664	81	418
4	773	754	35	201
8	387	369	19	102
16	198	147	10	53
32	108	52	5	36
48	82	41	4	45
64	61	33	3	57
70	62	30	3	62

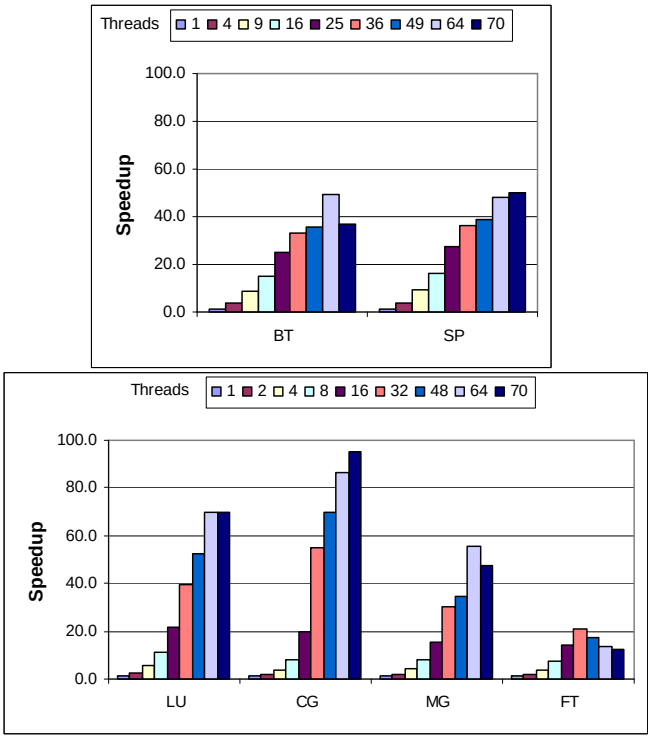

Figure 9: Scalability of NAS OpenMP 3.0 parallel benchmarks.

Table 4: Execution times for the SPEC OMPL2001 benchmarks (in seconds).

	Threads				
Benchmark	12	24	48	64	70
wupwise	6584	3531	1801	1418	1353
swim	6164	4197	2977	2762	2749
mgrid	12577	5316	3103	2553	2707
equake	23310	11146	6275	5157	5031
apsi	6306	4605	4144	4920	5434
fma3d	23484	10680	5657	4417	4047
art	37634	17643	9107	6812	6281

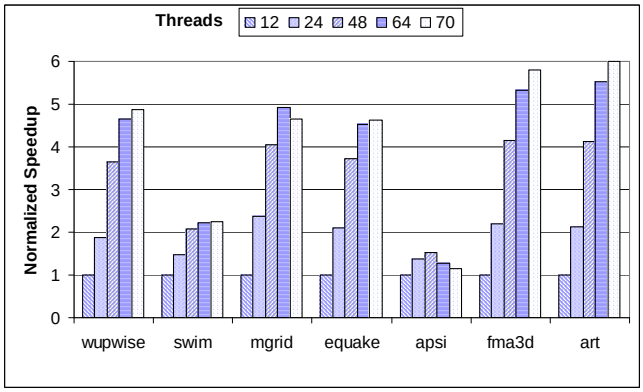

Figure 10: Scalability of the SPEC OMPL2001 benchmarks; speedup is relative to smallest test run, 12 threads.

Five of the seven programs achieved good scalability. The two exceptions were SWIM and APSI which both scaled poorly. APSI scaled the worst with performance decreases for number of threads greater than 48. ART had the best scalability of the applications, achieving better than ideal linear speedup for all numbers of threads.

The FT application of NAS 3.0 OpenMP, and the SWIM and APSI applications of SPEC OMPL2001 scaled poorly on the Sun Fire 15K. There are many factors, including OpenMP overhead, instruction and data cache misses, and synchronization delays, that might affect the performance. In section 4.4 we show that the OpenMP overhead is minimal for most of the applications. We are currently investigating other factors that might have affected the performance of these applications and we will report results in the future.

Performance characteristics of application programs on large-scale systems are often different from those on smaller systems. We compare our results for the large dataset SPEC OMPL2001 to the existing results for the medium dataset SPEC OMPM2001 for the Sun Fire 6800. These results are reported by Sun Microsystems and obtained from the SPEC [1], as of February 17, 2003. It is worth mentioning that the Sun Fire 6800 is a mid-size SMP with twenty four 900 MHz UltraSPARC III processors, each with 8 MB E-cache, and a total of 24GB RAM. Its runtime environment is exactly the same as our runtime system for the Sun Fire 15K.

Figure 11 shows the speedup results for the SPEC OMPM-2001 applications on a Sun Fire 6800. By comparing Figure 10 to Figure 11, we notice a couple of differences. APSI

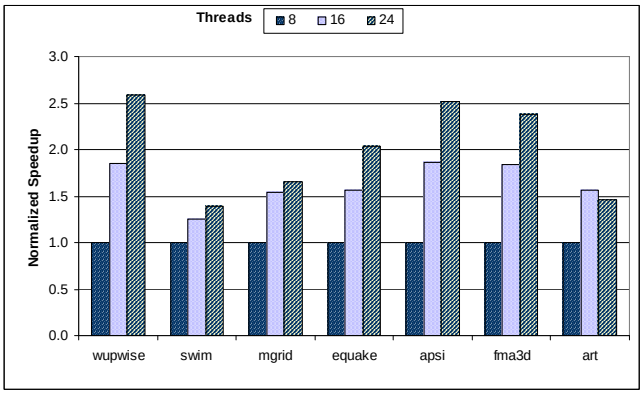

Figure 11: Scalability of the SPEC OMPM2001 benchmarks on a Sun Fire 6800.

scales much better to 24 threads than it did on our large-scale system. However ART, which performed the best on our system, did not scale as well. The other programs exhibited similar scalability including SWIM which was poor in both cases.

4.3.1 Comparison to Other Large-Scale Systems

We are interested in comparing the SPEC OMPL2001 performance on the Sun Fire 15K with the reported results on other large-scale systems. The results for the HP Superdome and the SGI Origin 3800 were obtained from the SPEC [1], as of February 17, 2003. Figure 12 shows the scalability of each application benchmark on the systems. The results for each system have been normalized with the smallest run; 12 threads for the Sun Fire 15K (900 MHz), 32 threads for the Superdome (750 MHz, PA-8700), 16 threads for Superdome (875 MHz, PA-8700+), and 32 threads for SGI 3800 (400 MHz, R12K).

WUPWISE shows good scalability on all platforms. SWIM, and APSI perform very poor after 24 threads on Sun Fire 15K, but they show good scalability on other systems. APSI has a linear scalability on Superdome (PA-8700). MGRID performs very good on all platforms, expect with a slightly worse performance on SGI 3800 with 128 threads. EQUAKE and FMA3D perform the best on Sun Fire 15K, but EQUAKE has a poor performance on SGI with 128 threads. ART shows slightly superlinear performance on Sun Fire 15K, and has almost linear scalability on other platforms.

4.4 OpenMP Overhead in the Application Benchmarks

It is interesting to discover the overhead of OpenMP in real applications. This can be done by counting the number of directives executed and multiplying by the overhead time for each directive. For this we profiled the applications to find the parallel regions, the directives each region contained, and the number of times each region is invoked. These profiles are shown in tables 5 through 8. We then used the profiles in conjunction with the EPCC microbenchmark results for 64 threads to estimate the OpenMP overhead in the NAS OpenMP, and SPEC OMPL2001 applications.

4.4.1 OpenMP Overhead in the NAS OpenMP

Table 5 summarizes the runtime characteristics for each of the NAS programs. The numbers of parallel regions vary among programs: FT contains 111 parallel regions while SP contains 3617. Using the overhead measurements from EPCC synchronization benchmarks, we estimate the total OpenMP overhead of the NAS programs. The OpenMP overhead is low for most programs, ranging from less than one percent to five percent of the total execution time. The only exception is CG with an estimated overhead of 12%. Table 6 shows which OpenMP directives are used by each program and how often. The NAS OpenMP programs use only a small subset of availble OpenMP directives. The complete runtime characteristics including the directives used in each parallel region can be found in [11].

Table 5: Runtime characteristics and estimation of OpenMP overhead in NAS OpenMP 3.0 application benchmarks with 64 threads.

Benchmark	Number of Parallel Regions	Estimated Overhead (microseconds)	Fraction of Total Execution Time
LU	514	2852012	4.65%
CG	2210	4057921	12.24%
MG	1276	152243	5.50%
FT	111	10724	0.02%
BT	1013	175894	0.34%
SP	3617	503313	0.82%

Table 6: Number of OpenMP directives invoked by the NAS OpenMP 3.0 application benchmarks with 64 threads.

Directive	LU	MG	FT	CG	BT	SP
parallel	510	487		2053	209	409
do	151526	974		7831	1237	2437
parallel do	1	785	111	81	804	3208
parallel do reduction	3	4		76		
critical	4				2	2
barrier	1000					
reduction				3952		
Total	153044	2250	111	13993	2252	6056

4.4.2 OpenMP Overhead in the SPEC OMPL2001

Table 7 summarizes the runtime characteristics for each of the SPEC programs. The number of parallel regions vary dramatically among programs. ART enters only 4 paralle regions while MGRID enters 299100 parallel regions, each consisting of a pair of PARALLEL and DO directives. Using the overhead measurements from EPCC synchronization benchmarks, we estimate the total OpenMP overhead of the SPEC programs. The OpenMP overhead is very low for all programs; less than one percent of the total execution time. The program with the largest overhead (0.95%) is MGRID; ART has the smallest overhead, less than 0.01%. Table 8 shows which OpenMP directives are used by each program and how often. The SPEC OMPL2001 programs use only a small subset of availble OpenMP directives. The complete runtime characteristics including the directives used in each parallel region can be found in [11].

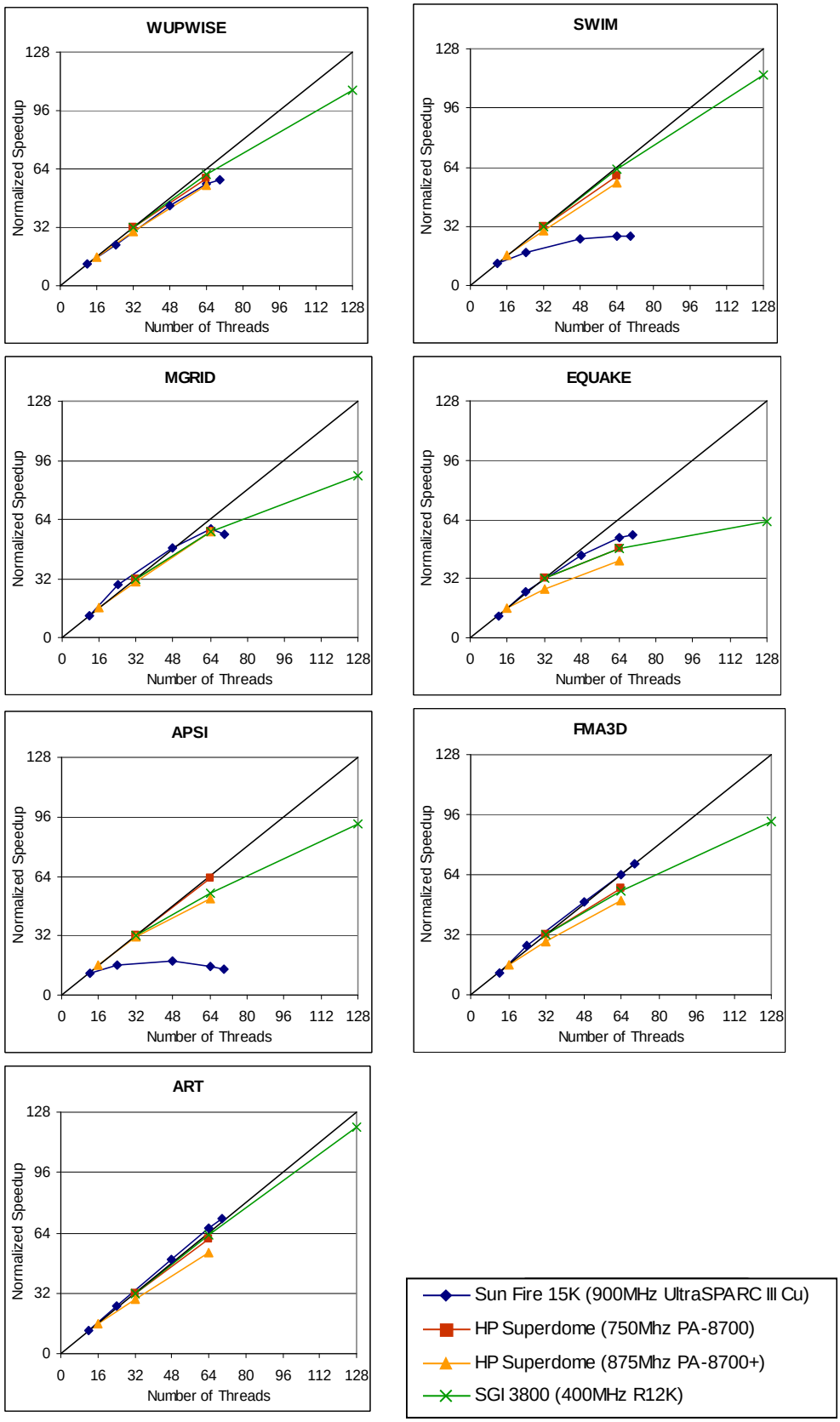

Figure 12: Scalability comparison of the applications in SPEC OMPL2001 on different SMP systems.

Table 8: Number of OpenMP directives invoked by the SPEC OMPL2001 benchmarks with 64 threads.

Directive	art	mgrid	equake	swim	apsi	fma3d	wupwise
parallel	1	229100	20004	4800	1903	5	5416
do	35	229100	650130	4800	1903	5	10530
parallel do	3		20039	9601	3678	9037	4
parallel do reduction				2400	600	3001	1201
critical							302
Total	39	458200	690173	21601	8084	12048	17453

Table 7: Runtime characteristics and estimation of OpenMP overhead in SPEC OMPL2001 application benchmarks with 64 threads.

Benchmark	Number of Parallel Regions	Estimated Overhead (microseconds)	Fraction of Total Execution Time
art	4	1124	0.00%
mgrid	229100	25805824	0.95%
equake	40043	4661514	0.09%
swim	16801	3552505	0.06%
apsi	6181	1090756	0.02%
fm3d	12043	3769489	0.05%
wupwise	6621	1882749	0.13%

5. RELATED WORK

Since the introduction of the OpenMP, there has been several research work on the performance evaluation of the OpenMP constructs. A performance monitoring interface for OpenMP is presented in [15]. In [9, 8], EPCC describes their OpenMP microbenchmarks and presents results for the Sun HPC 3500, Sun HPC 6500, the SGI Origin 3000. An enhanced set of OpenMP benchmarks that are derived from the EPCC benchmarks and use the SKaMPI framework are presented in [16]. This work reports results for the IBM SP3 and the Sun Fire 6800. Performance of OpenMP is reported in [7] for the Cray T90, SGI Origin 2000, and IBM R50; and in [14] for the Pentium 4 PC, Hitachi SR8K, and HP N-Class. OpenMP performance using the PARKBENCH benchmark is reported in [18].

The process of creating the SPEC OMP2001 suite from the SPEC CPU2000 suite is described in [4, 17]. Execution times are reported for medium-sized OMP2001 on a generic 8-processor system in [4]. In [5], the authors present performance characteristics of the medium-sized version of SPEC OMP2001 on a small SMP machine. Large system scalability of the SPEC OMP2001 benchmarks is reported in [17, 12]. Researchers in [19] evaluate the performance of the Hitachi SR8000 by SPEC OMP2001 and NAS OpenMP Parallel Benchmarks.

6. CONCLUSIONS AND FUTURE RESEARCH

Large shared-memory systems are common in high performance computing. Understanding the large system scalability of applications is necessary to use such systems efficiently. In this paper, we presented scalability results for seven of the new large dataset programs in the SPEC OMPL2001 benchmark suite. We find that five of the seven programs evaluated scale very well on Sun Fire 15K. Only SWIM and APSI show poor performance. ART has the best scalability among the applications, achieving superlinear speedup for all numbers of threads.

We also experimented with the class B of the NAS OpenMP 3.0. The CG benchmark shows a superlinear scalability. The LU benchmark achieves a perfect scalability (except for 70 threads). The BT, SP, and MG benchmarks show relatively good performance. However, the performance of FT is very poor.

As a complement to the SPEC OMPL2001 and NAS 3.0 OpenMP application benchmarks, we also presented results for the EPCC Microbenchmarks. These microbenchmarks provide insight into the scalability of individual directives. We used the results to estimate the OpenMP overhead in the SPEC OMPL2001, and the NAS OpenMP suites. We found that the scientific applications in the SPEC OMPL2001 suite have very low OpenMP overhead; less than one percent in all cases. The OpenMP overhead is also low for most programs in the NAS OpenMP suite, ranging from less than one percent to five percent of the total execution time. The only exception is CG with an estimated overhead of 12%.

We find that the SPEC OMP2001 suite is a better metric of total SMP system performance than specifically OpenMP performance. The ability to parallelize a program to sixty-four threads at almost no cost (less than one percent of total execution time) allows an HPC application to maximize its use of the available system resources.

As for the future research, we would like to complete our experimentation with the two remaining benchmarks, APPLU, and GAFORT, in the SPEC OMPL2001 suite. We also intend to run the SPEC OMPM2001 benchmarks on the 72-node Sun Fire 15K system to compare the results with the large set. We plan to evaluate the performance of NAS OpenMP suite with the larger class, C. Our next goal is to carefully discover the reasons behind why some of the applications do not scale well. For this, we will instrument the applications using the hardware counters of the UltraSPARC III CPU to find out the different metrics of the applications as well as the system that may need to be tuned or enhanced.

7. ACKNOWLEDGMENTS

This work was supported by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Queen's University. The authors would like to thank Dr. Ken Edgecombe, and Dr. Hartmut Schmider at High Performance Computing Virtual Laboratory (http://www.-hpcvl.org) at the Queen's University, and Mr. Gary Braida at Sun Microsystems for their kind help in accessing the Sun Fire 15K system.

8. REFERENCES

[1] SPEC OMP Benchmark Suite. (http://www.spec.org/omp/).

[2] MPI: A Message Passing Interface Standard, 1997. Version 1.2.

[3] OpenMP C/C++ Application Programming Interface, March 2002. Version 2.0.

[4] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. B. Jones, and B. Parady. SPEComp: A new benchmark suite for measuring parallel computer performance. *Lecture Notes in Computer Science*, 2104:1–10, 2001.

[5] V. Aslot and R. Eigenmann. Performance characteristics of the SPEC OMP2001 benchmarks. In *Proceedings of the European Workshop on OpenMP (EWOMP2001)*, 2001.

[6] D. H. Bailey, T. Harsis, W. Saphir, R. V. der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0: Report NAS-95-020. Technical report, Nasa Ames Research Center, December 1995.

[7] R. Berrendorf and G. Nieken. Performance characteristics for OpenMP constructs on different parallel computer architectures. *Concurrency: Practice and Experience*, 12(12):1261–1273, 2000.

[8] J. Bull. Measuring synchronisation and scheduling overheads in OpenMP. In *Proceedings of the First European Workshop on OpenMP*, 1999.

[9] J. M. Bull and D. O'Neill. A microbenchmark suite for OpenMP 2.0. In *Proceedings of the Third European Workshop on OpenMP (EWOMP'01)*, 2001.

[10] A. Charlesworth. The sun fireplane interconnect. *IEEE Micro*, 22(1):36–45, 2002.

[11] N. R. Fredrickson, A. Afsahi, and Y. Qian. Performance characteristics of openmp constructs, and application benchmarks on a large symmetric multiprocessor, ECE-0302. Technical report, Dept of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada, Feburary 2003.

[12] H. Iwashita, E. Yamanaka, N. Sueyasu, M. van Waveren, , and K. Miura. The SPEC OMP2001 benchmark on the Fujitsu PRIMEPOWER system. In *Proceedings of the European Workshop on OpenMP (EWOMP2001)*, 2001.

[13] H. Jin, M. Frumkin, and J. Yan. The OpenMP implementation of NAS parallel benchmarks and its performance, Report NAS-99-011. Technical report, Nasa Ames Research Center, October 1999.

[14] M. S. Mller. A shared memory benchmark in OpenMP. In *Proceedings of the International Workshop on OpenMP Experiences and Implementations (WOMPEI)*, 2002.

[15] B. Mohr, A. Mallony, H.-C. Hoppe, F. Schlimbach, G. Haab, and S. Shah. A performance monitoring interface for OpenMP. In *Proceedings of Fourth European Workshop on OpenMP (EWOMP'02)*, 2002.

[16] A. Prabhakar, V. Getov, and B. M. Chapman. Performance comparisons of basic OpenMP constructs. In *Proceedings of the fourth International Symposium on High Performance Computing (ISHPC)*, pages 413–424, 2002.

[17] H. Saito, G. Gaertner, W. Jones, R. Eigenmann, H. Iwashita, R. Lieberman, M. van Waveren, , and B. Whitney. Large system performance of SPEC OMP2001 benchmarks. In *Proceedings of the Workshop on OpenMP (WOMPEI2002): Experiences and Implementations*, 2002.

[18] M. Sato, K. Kusano, and S. Satoh. OpenMP benchmark using PARKBENCH. In *Proceedings of the European Workshop on OpenMP (EWOMP2000)*, 2000.

[19] D. Takahashi, M. Sato, and T. Boku. Performance evaluation of the Hitachi SR8000 using OpenMP benchmarks. In *Proceedings of International Workshop on OpenMP Experiences and Implementations (WOMPEI)*, 2002.