Performance Implications of a CMT Processor for OpenMP Applications

Myungho Lee[*†] Yeonseung Ryu[‡] Sugwon Hong[§] Chungki Lee[**]

Department of Computer Software, MyongJi University,
38-2 San Namdong, Cheo-In Gu, Yong-In, Gyung Gi Do 449-728, Korea.

Abstract. Shared Memory Multiprocessor (SMP) systems incorporating processors with Chip-level MultiThreading (CMT) technology are becoming mainstream servers in High Performance Computing (HPC) and commercial business applications as well. With the thread-level parallelism on chip, CMT servers can deliver higher aggregate performance than servers without the CMT technology. However, resource sharing among the threads on the same processor chip can cause conflicts and hurt the overall performance. Thus, in order to realize full performance potentials of CMT servers, it is crucial to understand the performance impact that CMT processors have on the target applications. In this paper, we evaluate the performance, scalability, and the impact of resource conflicts for standard OpenMP benchmarks on an example high-end CMT server.

Keywords: Chip-MultiThreading (CMT), SMP, High Performance Computing, OpenMP, Scalability

1 Introduction

OpenMP is an Application Program Interface (API) to express threaded, shared-memory parallelism in C, C++, and Fortran [7]. With its ease of use and good performance, it has become de-facto industry standard for programming shard-memory multiprocessors. OpenMP consists of (1) compiler directives to express parallelism, synchronization, data scoping information; (2) runtime routines to check thread ID's, number of threads used, passing information related to the nested parallelism; (3) environment variables to set the number of threads to be used, scheduling types, thread/processor binding, among many others. Current OpenMP implementations yield quite impressive performance and scalability on mid-range (up to 32 CPU's) servers. Further improvements to achieve high performance and scalability in large-scale servers are still anticipated.

With the ever increasing transistor density for the last few decades, microprocessor designers have been considering many design choices to utilize the effectively enlarged silicon area. Among many design choices, the most outstanding one in recent years is to support thread-level parallelism on a chip. This includes Chip Multi-Processors (CMP's) and/or Multi-Threaded (MT) processors. Both CMP's and MT processors allow multiple software threads executing on a chip in parallel. Thus they support Chip-level Multithreading (CMT). CMT processors provide a larger capacity of computations performed per chip for a given time interval (or throughput). Examples are Dual-Core Intel Xeon [3], AMD Opteron [1], UltraSPARC IV, IV+ microprocessors from Sun Microsystems [11] (CMP's), and Intel Pentium IV with Hyperthreading technology [4] (MT processors). The thrust for a higher degree of thread-level parallelism on chip will lead to a development of general-purpose CMT microprocessors with 16~32 threads in a few years. Shared-Memory Multiprocessor (SMP) systems will be built around such CMT processors in the near future also. Thus CMT servers executing a few hundred threads system-wide in parallel will be in common practices.

A SMP server incorporating CMT processors has higher degrees of hardware parallelism, thus can potentially lead to higher scalability for OpenMP applications. However, current OpenMP implementations do not generally yield scalable performance on large-scale servers. Furthermore, resources such as cache, cache/memory bus, functional units, etc., are shared among the threads on the same CMT processor chip,

[*] This work was supported by 2005 research fund of MyongJi University.
[†] myunghol@mju.ac.kr
[‡] ysryu@mju.ac.kr
[§] swhong@mju.ac.kr
[**] cklee@mju.ac.kr

which can cause thrashing and hurt performance. This problem will be magnified in the future as the degree of thread-level parallelism will increase dramatically in the years to come. Thus, in order to realize high performance and scalability of OpenMP applications on the CMT server, it is important to first understand the performance impact that the CMT processors have on the target applications.

In this paper, we evaluate the overall performance, scalability, and the impact of resource conflicts of OpenMP applications, SPEC OMPL [9], on a high-end CMT server, Sun Fire E25K. We use the Sun Studio 9 compiler suite [12] to generate fairly high optimized executables for SPEC OMPL programs. We use features of Solaris 10 Operating System [8] friendly to OpenMP applications and run the optimized executables on E25K. When running the SPEC OMPL, we measure the overall performance and scalability by using 72 and 143 threads. Furthermore, in order to evaluate the performance impact of resource conflicts on the CMT processor, we run the programs using 72 threads with and without the CMT feature. The experimental results show a decent overall scalability of 1.66 from 72 threads to 143 threads. Also, the results show the performance impact of the resource conflicts for each test program.

The rest of the paper is organized as follows: Section 2 describes the architecture of a generic CMT processor and an example CMT server, Sun Fire E25K. Section 3 describes the OpenMP execution model and our test benchmark suite, SPEC OMPL. Section 4 shows our methodologies to generate optimized executables for SPEC OMPL and to exploit other compiler, OS features for obtaining high performance for SPEC OMPL. Section 5 shows the experimental results on E25K. Section 6 wraps up the paper with conclusions.

2 CMT Server

We first describe the architecture of the state-of-the-art CMT processor. We then describe an example high-end CMT server, Sun Fire E25K, which we used for our performance experiments in this paper.

The current main design for CMT processors is based on CMP's such as Dual-Core Intel Xeon [3], AMD Opteron [1], Sun Microsystems' UltraSPARC IV, IV+, among others. Some CMT processors go one step further to incorporate Simultaneous MultiThreading (SMT) [14] or similar technologies on a processor core. Examples are IBM Power5 [5] and UltraSPARC T1 microprocessor [13] from Sun. Figure 1 shows the architecture of a generic CMT processor [2]. On each processor chip, there are *N*-processor cores, with each core having its own cache on chip. The *N*-cores share a larger cache on or off the processor chip. Each core also has *M* hardware threads performing SMT or similar features. Thus it supports two levels of parallelism. Also, it has a cache hierarchy of private (to each core) and shared (among threads). For example, the UltraSPARC T1 from Sun includes 8 cores on a chip, with each core supporting 4 hardware threads. In total, 32 (= 8 x 4) threads can execute on a chip at the same time. Each core has 8KB private data cache. The level-2 unified cache is 3MB in size. T1 is targeted to the web applications and has a simple core. In a next few years, similar designs as illustrated in Figure 1 will be widely adopted in general-purpose microprocessors with up to 32 threads available on chip.

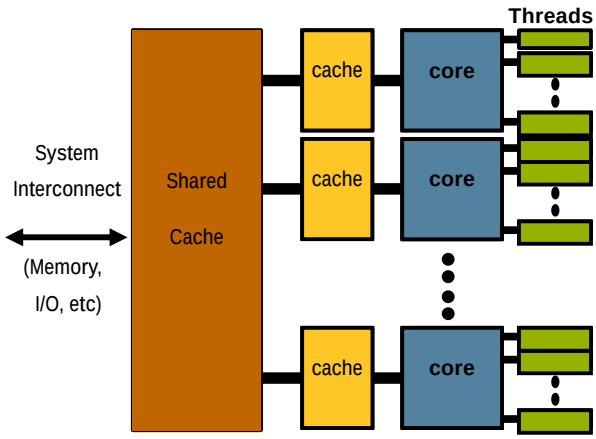

Fig 1. Architecture of a generic CMT processor

The Sun Fire E25K server is the first generation throughput computing server from Sun Microsystems which aims to dramatically increase the application throughput via CMT technology. The server is based on the UltraSPARC IV processor with two UltraSPARC III Cu (predecessor to UltraSPARC IV processor) cores. It can scale up to 72 UltraSPARC IV processors executing 144 threads (two threads per processor) simultaneously. The system offers up to twice the compute power of the UltraSPARC III Cu based high-end systems. Besides the two cores, the UltraSPARC IV contains a memory controller and the necessary cache tag for 8 MB of external L2 cache per core. The off-chip L2 cache is 16 MB in size (8 MB per core). The two cores share the Fireplane System Interconnect, as well as the L2 cache bus.

The basic computational component of the Sun Fire E25K server is the UniBoard [11]. Each UniBoard consists of up to four UltraSPARC IV processors, their L2 caches, and associated main memory. Sun Fire E25K can contain up to 18 UniBoards, thus at maximum 72 UltraSPARC IV processors. In order to maintain cache coherency system wide, the snoopy cache coherency protocol is used within the UniBoard

and directory-based cache coherency protocol is used among different UniBoards. The memory latency, measured using lat_mem_rd ( ) routine of lmbench, to the memory within the same UniBoard is 240nsec and 455nsec to the memory in different Uniboard (or remote memory).

3 SPEC OMPL Benchmarks

The underlying thread execution model for OpenMP is fork-join [7]. A *master* thread executes sequentially until a parallel region of code is encountered. At that point, the *master* thread forks a team of *worker* threads. All threads participate in executing the parallel region concurrently. At the end of the parallel region (the *join* point), the team of worker threads and the master synchronize. After then the *master* thread alone continues sequential execution. OpenMP parallelization incurs an overhead cost that does not exist in sequential programs: cost of creating threads, synchronizing threads, accessing shared data, allocating copies of private data, bookkeeping of information related to threads, and so on.

The SPEC OMPL benchmark suite consists of nine application programs written in C and Fortran, and parallelized using the OpenMP API [9]. These benchmarks are representative of HPC applications from the areas of chemistry, mechanical engineering, climate modeling, and physics. Each benchmark requires a memory size up to 6.4 GB when running on a single processor. Thus the benchmarks target large-scale systems with 64-bit address space. Following table lists the benchmarks and their application areas.

Table 1. SPEC OMPL Benchmarks

Benchmark Programs	Application Areas	Programming Languages
311.wupwise_l	Quantum chromodynamics	Fortran
313.swim_l	Shallow water modeling	Fortran
315.mgrid_l	Multi-grid solver	Fortran
317.applu_l	Partial differential equations	Fortran
321.equake_l	Earthquake modeling	C
325.apsi_l	Air pollutants	Fortran
327.gafort_l	Genetic algorithm	Fortran
329.fma3d_l	Crash simulation	Fortran
331.art_l	Neural network simulation	C

4 Optimizing Executables and System Settings

Using Sun Studio 9 compiler suite [12], we've generated executables for the benchmarks in SPEC OMPL suite. By using a common set of compiler options provided by the Sun Studio 9, fairly high level of compiler optimizations are enabled and applied to the benchmarks. The common set of compiler flags are –fast –openmp –xipo=2 –autopar –xprofile –xarch=v9a. These options provide many common and advanced optimizations such as scalar optimizations, loop transformations, data prefetching, memory hierarchy optimizations, interprocedural optimizations, profile feedback optimizations, among others. The –openmp option processes openmp directives and generate parallel code for execution on multiprocessors. The –autopar option provides automatic parallelization by the compiler beyond user-specified parallelization. This can further improve the performance.

The Solaris 10 Operating System provides features which help improve performance of OpenMP applications. They are Memory Placement Optimization (MPO) and Multiple Page Size Support (MPSS). For programs with intensive data accesses to localized regions of memory, efficiently utilizing MPO feature in Solaris 10 can significantly improve the performance. With the default MPO policy called first-touch, memory accesses can be kept on the local board most of the time, whereas, without MPO, those accesses would be distributed all over the boards (both local and remote) which can become very expensive. For programs which use a large amount of memory, using large size pages (supported by MPSS) can significantly reduce the number of TLB entries needed for the program and the number of TLB misses, thus significantly improve the performance [8]. We are enabling both MPO and MPSS for our runs of SPEC OMPL executables.

OpenMP threads can be bound to processors using the environment variable SUNW_MP_PROCBIND which is supported by thread library in Solaris 10. Processor binding, when used along with the static scheduling, benefits applications that exhibit a certain data reuse pattern where data accessed by a thread in a parallel region will either be in the local cache from a previous invocation of a parallel region, or in local memory due to the first-touch memory allocation policy of Solaris 10.

5 Performance Results

As described in Section 4, we've generated fairly high optimized executables for SPEC OMPL. We also enabled MPO and MPSS features in Solaris 10. We then run the SPEC OMPL programs on Sun Fire E25K while using the processor binding. In the following subsections, we show performance results for mainly scalability and the impact of resource conflicts.

5. 1 Overall Performance and Scalability

Using 72 threads and 143 threads, we've run SPEC OMPL suite on Sun Fire E25K. In both runs, both cores of the UltraSPARC IV processors (clock rate = 1050 Mhz) were used. Thus 72 cores in 36 UltraSPARC IV processors were used in the 72 threads run and 143 cores in 72 processors were used for the 143 threads run. In 143 thread run, one core is left idle to take care of system processes, daemons, or interrupts.

The performance of SPEC OMPL is computed as follows [9]:

- For each benchmark, the run time (wall clock time) is measured. Then we divide the run time with the reference run time given for each benchmark. The resulting number is multiplied with 16,000. This is the score for each benchmark.

- After computing scores for the all 9 benchmarks, we then compute their geometric mean. This geometric mean is the overall performance score of the whole benchmark suite.

Table 2. Performance Comparisons for SPEC OMPL—143 threads vs. 72 threads

Benchmark Programs	72 threads	143 threads	Scalability: 143/72
311.wupwise_l	282508	504121	1.78
313.swim_l	151927	287973	1.9
315.mgrid_l	178496	302214	1.69
317.applu_l	156009	278397	1.78
321.equake_l	96142	120950	1.26
325.apsi_l	106463	172362	1.62
327.gafort_l	147356	222660	1.51
329.fma3d_l	155068	288431	1.86
331.art_l	1001238	1653466	1.65
Geometric Mean	186995	310772	1.66

On 143 threads run, we've obtained an overall performance of 310,772. Then we've conducted 72 threads run which results in the overall performance of 186,995. The scalability is computed as 1.66 (310772/186995) when the number of threads is almost doubled from 72 to 143. Table 2 summarizes the performance results and scalability. Four of the benchmarks show scalabilities greater than 1.78 (311.wupwise, 313.swim_l, 317.applu_l, 329.fma3d_l). In the case of 313.swim_l which consumes a lot of memory bandwidth, the high peak aggregate memory bandwidth available on E25K helps obtain the high scalability. The L2 cache miss rate is high for 329.fma3d_l in 72 threads run. When the number of threads gets doubled from 72 to 143, the amount of data loaded onto L2 cache gets diminished significantly. This results in a significant reduction in L2 cache misses, due to effectively reduced working set sizes.

The scalabilities for 321.equake_l and 327.gafort_l are noticeably low: 1.26, 1.51. In the case of 321.equake_l, the synchronization overhead increased significantly as the number of threads increased from 72 to 143. 327.gafort_l has two hottest loops with critical sections inside. The critical section loops take a long time to execute while performing intensive load and store operations from/to the memory. As they are sequential portions, they act as the "serial bottleneck" in Amdahl's law and leads to low scalability. Figure 2 shows the scalability bar graph for each benchmark for better illustrations of the results.

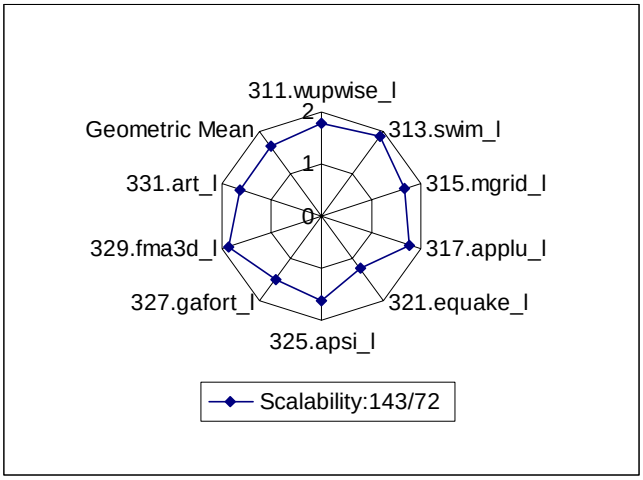

Fig 2. Scalability bar graph per each benchmark program

5.2 Performance Impact of Resource Conflicts on CMT

In order to measure the performance impact of resource conflicts on CMT (level-2 cache bus, memory bus) on SPEC OMPL, we've measured the performance of 72 threads runs in two ways:

1. Using 36 UltraSPARC IV processors (experiments conducted in section 5.1), thus using both cores of the processor

2. Using 72 UltraSPARC IV processors, thus using only one core per processor. There are no resource conflicts between the two cores on the same processor chip.

Table 3 (and also Figure 3) shows the performance for both 1 and 2, and also shows the speedup of 2 over 1. Overall, 2 performs 1.16x better than 1. Benchmarks with greater performance gains from 2 consume high memory bandwidths and/or use large amounts of memory:

Table 3. 72 threads case—36x2 vs. 72x1

Benchmark Programs	36 chip x 2 cores	72 chip x 1 core	1 core vs. 2 cores
311.wupwise_l	282508	299246	1.06
313.swim_l	151927	211008	1.39
315.mgrid_l	178496	213833	1.20
317.applu_l	156009	160670	1.03
321.equake_l	96142	98853	1.03
325.apsi_l	106463	130035	1.22
327.gafort_l	147356	187597	1.27
329.fma3d_l	155068	155060	1.00
331.art_l	1001238	1293218	1.29
Geometric Mean	186995	216592	1.16

- 313.swim_l is a memory bandwidth-intensive benchmark as mentioned in section 5.1. When only one core is used per processor, it can fully utilize the memory bandwidth available on the processor chip, whereas when two cores are used the bandwidth is effectively halved between the two cores. Thus 2 performs 1.39x better than 1.

- 315.mgrid_l, like in 313.swim_l, requires high memory bandwidth. Thus using only one core can have much higher memory bandwidth which led to 1.20x gain. However, as seen in section 5.1, the low scalability from the 72 threads run to the 143 threads run stems from the low iteration counts of the two hottest loops. The maximum iteration counts for the two hottest loops are 512 only. The iteration count 512 is relatively small compared with the number of threads used (143, 72). Thus the portion for parallelization overheads such as synchronization also increases as the number of threads is increased. This prevented the benchmark from achieving high scalability in Section 5.1.

- 325.apsi_l and 331.art_l allocate large amount of memory per thread at run-time. Thus, instead of allowing 8 threads to allocate large amounts of memory on the same UniBoard's memory, allowing only 4 threads can have significant performance benefit.

- 327.gafort_l suffers from intensive memory loads and stores issued in the critical section loops. Allocating 8 threads on two different UniBoards reduces the pressure on the memory bandwidth significantly compared with allocating 8 threads on the same UniBoard.

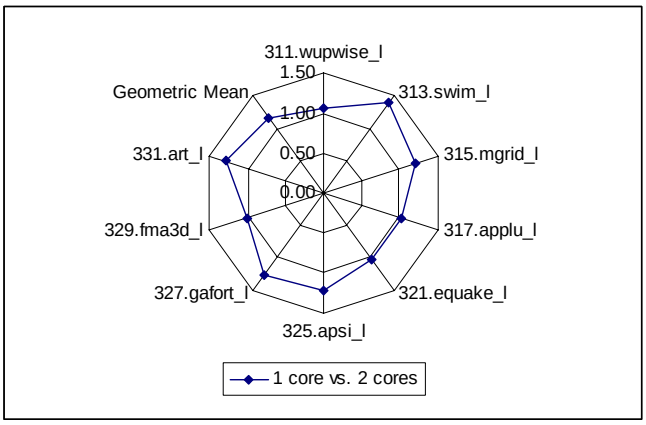

Fig 3. Performance comparisons of 1 core vs. 2-cores

Benchmarks other than the above (311.wupwise_l, 317.applu_l, 321.equake_l, 329.fma3d_l) relatively give less pressure on the memory bandwidth and/or consume smaller amount of memory. Thus both 1 and 2 result in the same level of performance. These benchmarks are not heavily affected by the resource conflicts and more suitable for execution on CMT servers.

6 Conclusions

In this paper, we first introduced the architecture of an advanced CMT processor and an example CMT server, Sun Fire E25K, in detail. Then we introduced the OpenMP execution model along with the SPEC OMPL benchmark suite used to evaluate the CMT server Sun Fire E25K. We showed our methodology to generate highly optimized executables for SPEC OMPL benchmarks using the Sun Studio 9 compiler. We also described the features in Solaris 10 OS (MPO, MPSS) which help improve OpenMP application performance. Using the executables for SPEC OMPL generated by the Sun Studio 9 compilers, and also the MPO and MPSS features in Solaris 10, we've obtained high scalability on E25K. Benchmarks which incur a large synchronization overheads (321.equake_l) or large overheads due to critical section loops (327.gafort_l) show low scalabilities. We've also measured the performance impact of the resource conflicts on CMT processor for SPEC OMPL using either one core or both cores of a CMT processor. It turned out that the benchmarks which require large memory bandwidths and/or consume large amounts of memory suffer when both cores are used due to the saturations on the memory bus and the main memory.

References

[1] AMD Multi-Core: Introducing x86 Multi-Core Technology & Dual-Core Processors, *http://multicore.amd.com/2005*

[2] Shailender Chaudhry, Paul Caprioli, Sherman Yip, and Marc Tremblay, High-Performance Throughput Computing, *IEEE Micro, May-June, 2005*.

[3] Intel Dual-Core Server Processor, *http://www.intel.com/business/bss/products/server/dual-core.htm*

[4] Intel Hyperthreading Technology, *http://www.intel.com/technology/hyperthread/index.htm*

[5] R. Kalla, B. Sinharoy, and J. Tendler, IBM POWER5 chip: a dual core multithreaded processor, *IEEE Micro, March-April 2004.*

[6] K. Olukotun et. al., The Case for a single Chip-Multiprocessor, *International Conference on Architectural Support for Programming Languages and Operating Systems, 1996.*

[7] OpenMP Architecture Review Board, *http://www.openmp.org*

[8] Solaris 10 Operating System, *http://www.sun.com/software/solaris*

[9] The SPEC OMP benchmark suite, *http://www.spec.org/omp*

[10] L. Spracklen and S. Abraham, Chip MultiThreading: Opportunities and Challenges, *11th International Symposium on High-Performance Computer Architecture (HPCA-11), pp 248-252, 2005.*

[11] Sun Fire E25K server, *http://www.sun.com/servers/highend/sunfire_e25k/index.xml*

[12] Sun Studio 9 Software, *http://www.sun.com/software/products/studio/index.html*

[13] Sun UltraSPARC T1 microprocessor, *http://www.sun.com/processors/UltraSPARC-T1*

[14] D. Tullsen, S. Eggers, and H. Levy, Simultaneous MultiThreading: Maximizing On-Chip Parallelism, *International Symposium on Computer Architecture, 1995.*