Evaluating OpenMP on Chip MultiThreading Platforms

Chunhua Liao, Zhenying Liu, Lei Huang, and Barbara Chapman
Computer Science Department, University of Houston, Texas
{liaoch, zliu, leihuang, chapman}@cs.uh.edu

Abstract. Recent computer architectures provide new kinds of on-chip parallelism, including support for multithreading. This trend toward hardware support for multithreading is expected to continue for PC, workstation and high-end architectures. Given the need to find sequences of independent instructions, and the difficulty of achieving this via compiler technology alone, OpenMP could become an excellent means for application developers to describe the parallelism inherent in applications for such architectures. In this paper, we report on several experiments designed to increase our understanding of the behavior of current OpenMP on such architectures. We have tested two different systems: a Sun Fire V490 with Chip MultiProcessing and a Dell Precision 450 workstation with Simultaneous MultiThreading technology. OpenMP performance is studied using the EPCC Microbenchmark suite, subsets of the benchmarks in SPEC OMPM2001 and the NAS parallel benchmark 3.0 suites.

1 Introduction

OpenMP has been successfully deployed on small-to-medium shared memory systems and large-scale DSMs, and is evolving over time. The OpenMP specification version 2.5 public draft [19] was released by the Architecture Review Board (ARB) in November 2004. It merged C/C++ and FORTRAN and clarified some concepts, especially with regard to the memory model. OpenMP 3.0 is expected to follow, and to consider a variety of new features. Among the many open issues are some tough challenges including determining how best to extend OpenMP to SMP clusters, how best to support new architectures, making hierarchical parallelism more powerful, and parallel I/O. In this paper, we explore some aspects of current OpenMP performance on several recent platforms: a Sun Fire V490 [25] with Chip MultiProcessing and a Dell Precision 450 workstation with Simultaneous Multithreading technology.

As computer components decrease in size, architects have begun to consider different strategies for exploiting the space on a chip. A recent trend is to implement Chip MultiThreading (CMT) in the hardware. This term refers to the simultaneous execution of two or more threads within one processor. It may be implemented through several physical processor cores in one chip (Chip MultiProcessing, CMP)

[18], a single core processor with replication of features to maintain the state of multiple threads simultaneously (Simultaneous multithreading, SMT) [27] or the combination of CMP and SMT [10] [11]. OpenMP support for these new micro architectures needs to be evaluated and possibly enhanced.

In this paper, we report on the behavior of some OpenMP benchmarks on each of the two systems mentioned above: the Sun Fire V490 exploits CMP and the Dell Precision 450 workstation with SMT technology. The remainder of the paper is organized as follows. We first discuss the architectures that are used in this experiment, and comment on their implications for OpenMP. We then describe the methodology of how to run the benchmarks, followed by our results and a discussion of them. The paper then presents related work and reaches some conclusions.

2 Chip MultiThreading and Its Implications for OpenMP

CMT is emerging as the dominant trend in general-purpose processor design [24]. In a CMT processor, some chip-level resources are shared and further software approaches need to be explored to maximize overall performance. CMT may be implemented through Chip Multi-Processing (CMP) [15], Simultaneous Multithreading (SMT) [27] or the combination of CMP and SMT [10] [11]. In this section, we give a brief overview of CMP and SMT first, and discuss their implications for OpenMP.

Chip MultiProcessing enables multiple threads to be executed on several physical cores within one chip. Each processor core has its own resources as well as shared ones. The extent of sharing varies from one implementation to another. For example, the UltraSPARC IV [28]'s two cores are almost completely independent except for the shared off-chip data paths while the Power4 [15] processor has two cores with shared L2 cache to facilitate fast inter-chip communication between threads.

Simultaneous MultiThreading combines hardware multithreading with superscalar processor technology to allow several independent threads to issue instructions to a superscalar's multiple function units each cycle [27]. SMT permits all thread contexts to simultaneously compete for and share processor resources; it uses multiple threads to compensate for low single-thread instruction-level parallelism (ILP).

CMP and SMT are two closely related technologies. They can be simply seen as two different extents of sharing of on-chip resources among threads. However, they are also significantly different because the various types of resource sharing have different implications for application performance, especially when the shared pipelines on SMT are compared with the private pipelines on CMP. Moreover, new multithreaded chips such as Power5 [10] and the Sun Niagara 32-way CMT SPARC processor [11], tend to integrate both CMP and SMT into one processor. This kind of integration brings even deeper memory hierarchy and more complex relationship between threads.

The CMP and SMT technology introduces new opportunities and challenges for OpenMP. The current flat view of OpenMP threads is not able to reflect these new features and thus need to be revisited to ensure continuing applicability in the future. Previous research on SMT [16, 22, 27, 29, 30] has developed some strategies for efficiently sharing of key resources, especially caches. In OpenMP, we need to spec-

ify sibling[1] processors to perform the work cooperatively with proper scheduling and load balancing mechanisms, explore optimizations to avoid inter-thread competition for shared resources, and select the best number of processors from a group of multiple cores. On CMP, first of all, for systems with only one multicore processor, they are indeed a slim implementation of SMP on a chip. While chip level integration has the benefits of fast synchronization and lower latency communication among threads, the shared resources may lead to conflicts between threads and unsatisfactory performance. Secondly, for SMPs composed of several multicore processor systems, the relationship between processing units is no longer strictly symmetric. For example, cores within one processor chip may have faster data exchange speed than cores crossing processor boundaries. Multithreading based on those cores has to take this asymmetric feature into account in order to achieve optimal performance. Altogether, the integration, resource sharing and hierarchical layout in SMP systems with CMT bring additional complexity to the tasks of data set partition, thread scheduling, work load distribution, and cache/memory locality maintenance.

3. Methodology

We have chosen the Sun Fire V490 with UltraSPARC IV [28] processors and a Dell Precision 450 workstation with Xeon [12] processors as test beds to explore the impact of CMT technology. In this section, we describe these two machines, benchmarks that we ran and how to execute the benchmarks. We attempt to understand the scalability of OpenMP applications on the new platforms, and the performance difference between SMPs with CMT and traditional SMPs via the designed experiments.

3.1 Sun Fire V490 with UltraSPARC IV and Dell Precision 450 with Xeon

Sun UltraSPARC IV was derived from earlier uniprocessor designs (UltraSPARC III) and the two cores do not share any resources, except for the off-chip data paths [24]. The UltraSPARC IV processor is able to execute dual threads based on two 14-stage, 4-way superscalar UltraSPARC III [8] pipelines of two individual cores. Each core has its own private L1 cache and exclusive access to half of an off-chip 16MB L2 cache. The L1 cache has 64KB for data and 32K for instructions. L2 cache tags and a memory controller are integrated into the chip for faster access.

The Sun Fire V490 server for our experiments has four 1.05 GHz UltraSPARC IV processors and 32 GB main memory. The basic building block is a dual CPU/Memory module with two UltraSPARC IV cores, an external L2 cache and a 16 GB interleaved main memory. The Sun Fireplane™ Interconnect is used to connect processors to Memory and I/O devices. It is based on a 4-port crossbar switch with a 288-bit (256-bit data, 32-bit Error-Correcting Code) bus and clock rate of 150 MHz. The maximum transfer rate is thus 4.8 GB/sec. The Sun Fire V490 is loaded with Solaris 10 operating system [23] and Sun Studio 10 [26] integrated development

[1] We use the term “sibling” to refer to the two cores in the same processor for CMP, and the two logical processors in the same physical processor for SMT.

environment which support OpenMP 2.0 APIs for C/C++ and FORTRAN 95 programs. Solaris 10 allows superusers to enable or disable individual processor cores by using processor administration tool. An environment variable SUNW_OMP_PROBIND is available for users to bind OpenMP threads to processors. More CMP-specific features are described in [16].

Xeon processors have Simultaneous MultiThreading (SMT) technology, which is called HyperThreading [12]. HyperThreading makes a single physical processor appear as two logical processors; most physical execution resources including all 3 levels of caches, execution units, branch predictors, control logic and buses are shared between the two logical processors, whereas state resources such as general-purpose registers are duplicated to permit concurrent execution of two threads of control. Since the vast majority of micro-architecture resources are shared, the additional hardware consumes less than 5% of the die area.

The Dell Precision 450 workstation, on which we carry out the experiments, has dual Xeon 2.4 GHZ CPUs with 512K L2 cache, 1.0GB memory and HyperThreading technology. The system runs Linux with kernel 2.6.3 SMP. The Omni compiler [20] is installed to support OpenMP applications and GCC 3.3.4 acts as a backend compiler. Linux Kernel 2.6.3 SMP [31] in our 2-way Dell Precision workstation has a scheduler which is aware of HyperThreading. This scheduler can recognize that two logical processors belong to the same physical processor, thus maintaining the load balance per physical CPU, not per logical CPU. We confirmed this via simple experiments that showed that two threads were always allocated to two different physical processors unless there was a third thread or process involved.

3.2 Experiments

Since it was not clear whether the Solaris 10 on our Sun Fire V490 with 4 dual-core processors is aware of the asymmetry among underlying logical processors, we used the Sun Performance *Analyzer* [26] to profile a simple OpenMP Jacobi code running with 2, 3, and 4 threads. We set the result timelines to display data per CPU instead of per thread in *Analyzer*, and found that Solaris 10 is indeed aware of the differences in processors of a multicore platform and tries to avoid scheduling threads to sibling cores. Therefore we can roughly assume the machine acts like a traditional SMP for OpenMP applications with only 4 or less active threads. For applications using 5 or more threads, there must be sibling cores working at the same time. As a result, any irregular performance change from 4 to 5 threads might be related to the deployment of sibling cores.

The EPCC Microbenchmark [4] Suite, SPEC OMPM2001 [2], and NAS parallel benchmark (NPB) 3.0 [9] were chosen to discover the impact of CMT technology on OpenMP applications. The EPCC microbenchmark is a popular program to test the overhead of OpenMP directives on a specific machine while the SPEC OMPM2001 and NAS OpenMP benchmarks are used as representative codes to help us understand the likely performance of real OpenMP applications on SMP systems with CMT.

For experiments running on the Sun Fire V490 machine, we compiled all the selected benchmarks using the Sun Studio 10 compiler suite with the generic compilation option “-fast -xopenmp” and ran them from 1 to 8 threads in multi-user mode.

This round of experiments gave us a first sense of the OpenMP behavior on the CMP platform and exposed problematic benchmarks. After that, we ran the problematic benchmarks using 2 threads when only 2 cores were enabled: the two cores were either from the same CMP processor or from different processors (an approximate traditional SMP). For both cases, Sun performance tools were used to collect basic performance metrics as well as related hardware counter information in order to find the reasons for the performance difference between a CMP SMP and a traditional SMP. The experiments on the Dell Precision workstation were designed in the same fashion whenever possible. For example, we measured the performance of the EPCC microbenchmarks using 2 logical processors of the same physical processor and on 2 physical processors with HyperThreading disabled. This way, we can understand the influence of HyperThreading better.

4. Results and Analysis

This section illustrates the results for the experiments on the two machines using the selected benchmarks, and our analysis. Some major performance problems are explained with the help of performance tools. We are especially interested in the effects of the sibling cores and sibling processors. In other words, particular attention is paid to performance gaps when the number of threads changes from 4 to 5 on the Sun Fire V490 and from 2 to 3 on the Dell Precision 450.

4.1 The EPCC Microbenchmark Suite

4.1.1 Sun Fire V490

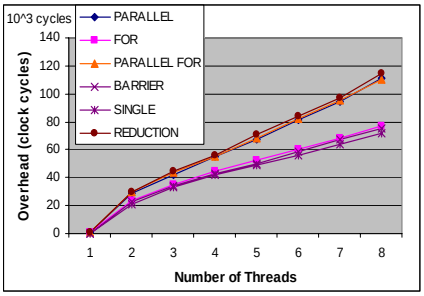

Fig. 1. Synchronization overhead on Sun Fire V490

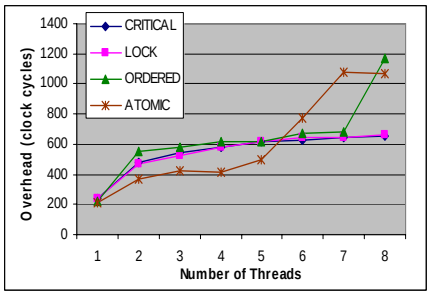

Fig. 2. Mutual exclusion + ORDERED overhead on Fire 490

Fig. 1 and 2[2] shows the results of the EPCC microbenchmark on the Sun Fire V490. Most OpenMP directives have higher overhead than on the Sun HPC 3500 [4], a traditional SMP machine, for some unknown reason. PARALLEL, PARALLEL FOR

[2] We display the overhead of ORDERED and mutual exclusive directives in Fig. 2 as they have the same order of magnitude overhead.

and REDUCTION have similar overhead on the Sun Fire V490. For mutual exclusion synchronization results shown in Fig. 2, LOCK and CRITICAL show similar overhead and scale well. The ORDERED directive has a high cost when the full eight threads are used, although it scales as well as LOCK and CRITICAL when less than eight threads are involved. The exception of ORDERED may come from hardware, OpenMP compiler implementation, or scheduler in Solaris 10. The ATOMIC directive is noticeably cheaper than LOCK and CRITICAL when the number of threads is up to five, however, it is more expensive than the other two if we use more than five threads. We checked the corresponding assembly code of an ATOMIC construct (see Table 1), and found that the Sun Studio 10 compiler uses runtime library calls **__mt_b_atomic_** and **__mt_e_atomic_** to start and finish an ATOMIC operation, rather than a single hardware primitive. The *Analyzer* indicated that the execution time of those two calls is sensitive to the physical layout of processor cores: ATOMIC does not have good performance if both sibling cores are involved.

Table 1. An OpenMP source and the corresponding assembly code segment for an ATOMIC construct

An OpenMP code segment	The corresponding assembly code for the ATOMIC construct

…
float aaaa=0;
#pragma omp parallel private(j)
{
for (j=0; j<innerreps/nthreads;
j++){
#pragma omp atomic
aaaa += 1;
}
} ….
	call **__mt_b_atomic_** ! params = ! Result =
	nop
	ld [%sp+92],%f2
	add %l4,1,%l4
	ld [%l5],%f0
	fadds %f2,%f0,%f0
	call **__mt_e_atomic_** ! params = ! Result =
	…

We show OpenMP loop scheduling results on the Sun Fire V490 in Fig. 3. Block cyclic scheduling (STATIC, n) and block scheduling (STATIC) have similar performance when the chunk size is not too small. The cost of GUIDED decreases noticeably when the chunk size is incremented up to 1024.

Altogether, the synchronization overhead for OpenMP directives does not show sensitivity to the change from 4 to 5 threads. We further designed some experiments to closely examine the effect of different combination of cores. We compare the benchmark performance on two sibling cores and non-sibling cores (in fact, the former reflects the effects of CMP and the latter the effects of the traditional SMP), and on four cores from two processors and from four different processors, respectively. In order to ensure the desired core/processor layout, we take advantage of the *prsadm* utility from Sun Solaris 10 to turn off the cores that we will not use. Fig 4 shows the overhead ratio for OpenMP directives on cores belonging to the same processor(s) and different processor(s). OpenMP directives on CMP take slightly less time than on a traditional SMP except for three mutual exclusion directives: CRITICAL, LOCK

and ATOMIC. The overall overhead difference tends to be smaller (close to ratio of 1) when more threads are used, except for the ATOMIC directive.

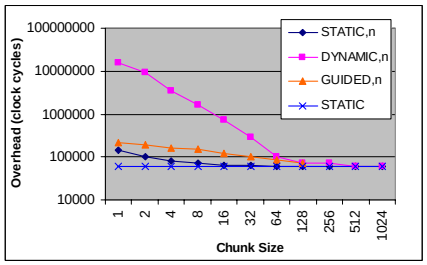

Fig. 3. Scheduling overhead on Sun Fire V490

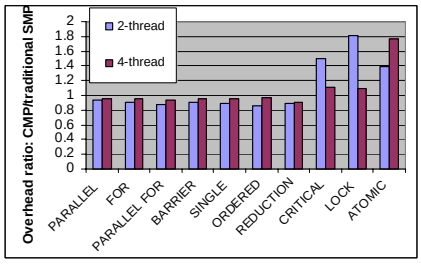

Fig. 4. Synchronization overhead ratio: CMP/traditional SMP

Therefore, we may conclude that sibling cores do not bring significantly faster synchronization or fewer overhead for OpenMP constructs on the Sun Fire V490 machine. This is because the sibling cores in the UltraSPARC IV do not share L1 and/or L2 cache to facilitate faster communications.

4.1.2 Dell Precision 450

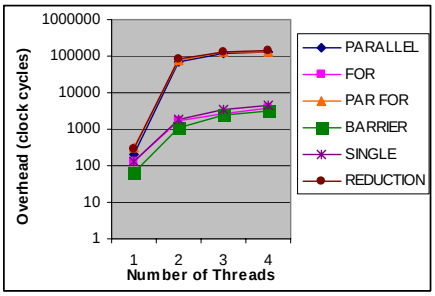

Fig. 5. Synchronization overhead on Dell Precision 450

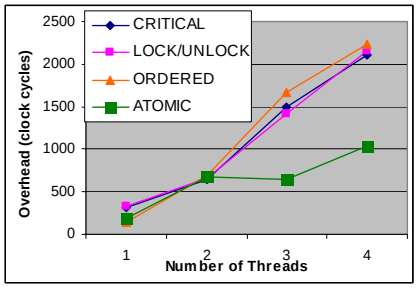

Fig. 6. Mutual exclusion + ORDERED overhead on Dell Precision 450

We depict the results obtained by using the EPCC microbenchmarks to measure OpenMP synchronization overhead on the Dell Precision 450 in Fig. 5 and 6, and the results for OpenMP scheduling overhead in Fig. 7. Overall, the results are similar to those on a traditional SMP system. In Fig. 8, we display the overhead ratio between the two sibling processors and two physical processors. Only the overhead of ATOMIC is smaller in the case of sibling processors. The OpenMP synchronization directives are mostly implemented using *spin-wait*, which leads to more competition for shared resources on Xeon systems.

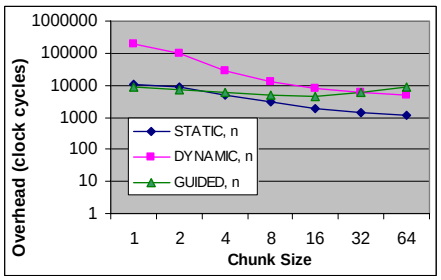

Fig. 7. Scheduling overhead on Dell Precision 450

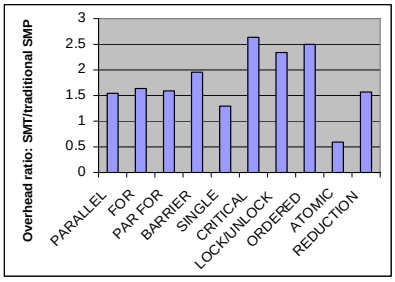

Fig. 8. Synchronization overhead ratio: SMT vs. traditional SMP

4.2 SPEC OMPM2001 and NAS NPB 3.0

4.2.1 Sun Fire V490

We show the speedup of a subset from SPEC OMPM2001 and NPB 3.0 with class B dataset on the Fire V490 machine in Fig. 9 and Fig. 10 separately. In Fig. 9, WUPWISE, EQUAKE, and APSI show good scalability on Sun Fire V490, with only slight changes after 4 threads. AMMP scales as poorly as described on traditional SMPs [21]. The APPLU demonstrates super-linear speedup for no less than 6 threads, which start to provide enough L2 caches to accommodate its critical dataset as mentioned in [21]. As can be seen in Fig. 10, most NAS OpenMP benchmarks show a more consistent behavior during the change from 4 to 5 threads than those from SPEC. Only EP achieves linear speedup because its dataset is much smaller than the size of L2 cache.

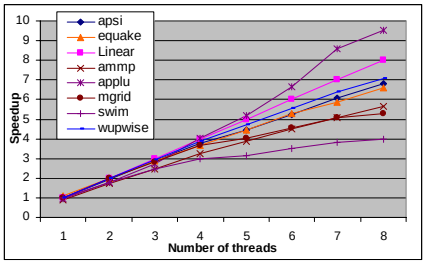

Fig. 9. Speedup of SPEC OMPM2001 on Sun Fire V490

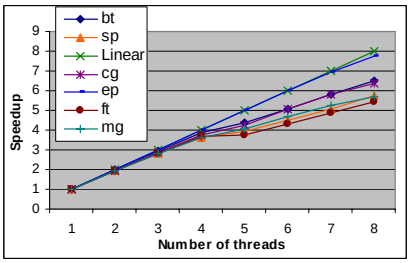

Fig. 10. Speed up of NAS NPB 3.0 on Sun Fire V490

Several benchmarks do not scale very well on this platform, particularly SWIM from SPEC and FT from NPB 3.0. We profiled SWIM and FT using 2 threads on sibling and non-sibling cores, and the profiling results confirmed the negative impact of multicores, as sibling cores did cause longer L2 stalls in the major loops. Both SWIM and FT are memory-intensive so that they cannot benefit from the multicore architecture. For the medium dataset, the major loops of SWIM perform computations on fourteen arrays, each of which contains 3802 * 3802 double precision float-

ing point numbers; hence the total memory requirement is 38022*8*14=1.51G bytes. Similarly, the FT benchmark with class B dataset also requires over 1 GB memory during its execution. Moreover, non-contiguous memory loads and stores of FT's major function cause high cache miss rates and competition for the shared data path of the sibling cores. Therefore, performance degradation occurs when the number of involved threads is increased from 4 to 5.

4.2.2 Dell Precision 450

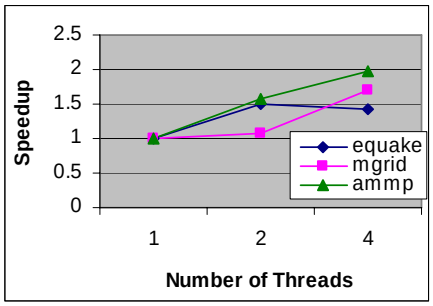

Fig. 11. Speedup of SPEC OMPM2001 on Dell Precision 450

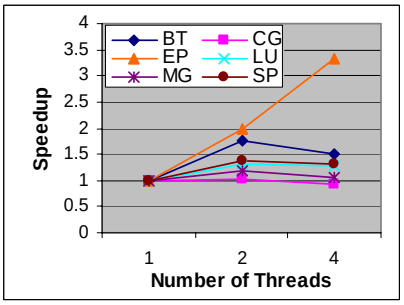

Fig. 12. Speed up of NAS NPB 3.0 on Dell Precision 450

We tested a subset of the SPEC OMPM2001 and NAS NPB 3.0 benchmarks on our dual-Xeon Linux workstation. Results are given in Fig. 11 and Fig. 12 respectively. EQUAKE from SPEC is a memory-intensive application in which more SMT threads lead to performance degradation due to the memory competition. The Xeon's halved memory bandwidth does not fulfill the demands of more threads [6]. We also observed that MGRID did not have a speedup with 2 threads. The reason is that the OS schedules the two threads onto a single physical processor and resource conflicts occur. From our experiments, it seems the HyperThreading-aware scheduler in Linux kernel 2.6.3 is not well implemented. We did not get good speedup for the NAS benchmarks except EP, which requires much less memory than others.

5. Discussion and Future Work

The Sun Fire V490 is a successful platform for OpenMP as one of the first generation CMP (multicore) machines. We find its overall scalability to be comparable to a traditional SMP machine since each core has the similar capability as a regular uniprocessor. Most OpenMP applications from SPEC OMPM2001 and NPB 3.0 scale very well. For memory-intensive applications using 5 and more threads, scalability may be compromised to an acceptable degree due to the competition for the shared data path between sibling cores. There should be a threshold for the applications' memory demand to be intensive enough to cause performance degradation on a specific machine. Unfortunately, the EPCC microbenchmark's results did not show profitable

faster synchronization among sibling cores within one processor in this machine. We believe it is mostly due to the fact that the L2 cache is not really shared between sibling cores. Meanwhile, the Solaris operating system schedules OpenMP threads very well taking the asymmetry between cores into consideration. The Sun Studio Performance Analyzer is a very useful and handy tool to help us understand the underlying reasons for performance problems. However, the caller-callee tab leaves some room to be improved and we used our own tool [7] for better understanding the benchmarks.

Compiler optimizations are also key factors in OpenMP performance on the Sun platforms. For example, we observed a performance degradation from 4 to 5 threads for Jacobi code compiled with "–fast –xopenmp" even for a small data set (3 500*500 arrays and 1000 iterations). The *Analyzer* showed that the Sun Studio 10 compiler performed loop unrolling on the major loops and inserted PREFETCH instructions into them. Data cache stall information collected from hardware counters hints that some PREFETCH operations initiated from two sibling cores stressed the shared data path connected to L2 cache and memory, thus incurred longer stall cycles than usual. But it is not always the case for all PREFETCH operations. Further work is needed to understand the exact conditions and effects of these traditional optimizations with regard to the OpenMP performance on machines with CMP capabilities.

We observed that a straightforward OpenMP implementation for traditional SMP architecture may not achieve good scalability on the Xeon system. The main reasons are memory bandwidth bottleneck [6] and competition for the shared computing resources, which degrade the overall SMT performance. More OpenMP scheduling policies [30] and precomputation and a prefetch approach via a helper thread [29] which utilize new SMT features are able to lead to a better performance. Our experiments also showed that a HyperThreading-aware OS is important for maintaining load balance and efficiently utilizing the resources of an SMT system. The EPCC microbenchmarks results also indicated that the overhead of OpenMP synchronization implementation in a SMT system is higher than that in an SMP system. The OpenMP implementation needs to take the SMT features into account.

Our experiments for CMP and SMT were carried out separately to understand the implications of CMT. We plan to analyze those OpenMP benchmarks and consider the interaction between CMP and SMT when we have access to a machine with both technologies. The ultimate objective is to obtain a quantitative model, which is capable of predicting and explaining the OpenMP performance on CMT machines, considering machine parameters, application features and compiler optimizations.

6. Related Work

CMP technology exists in IBM Power4, Sun UltraSPARC IV, AMD Opteron [1], and some embedded systems [14]. The research on Power4 [5] showed that the improved data locality of the L2 cache made the SP program from NAS NPB scale from 16 to 32; otherwise, the program runs on 32 processors are slower. [13] gives an overview of the Sun Fire E25K with UltraSPARC IV and its OpenMP support; in particular, the base and peak performance of SPEC OMPL benchmarks using almost maximum

threads was demonstrated and compared with a traditional SMP. A system-on-chip (SOC) design from Cradle Technologies, Inc. integrates processors into one chip and OpenMP was selected in [14] to deal with the heterogeneity of CMP: OpenMP is extended to support Cradle's Digital Signal Engine (DSE) and optimized via data prefetching and privatization. [3] explores the performance impact of asymmetric multicore architectures using a wide range of applications on a hardware prototype.

Simultaneous multithreading techniques can be dated back to 10 years ago as a means to improve the utilization of superscalar processors [27]. Work related to compiler support for SMT has focused on the problems of synchronization, memory allocation and program optimizations for shared caches. Researchers explored the performance of symbiosis [22], a group of two sequential programs, or a parallel program other than OpenMP running on SMT. In particular, hand-written program transformations, namely dynamic tiling, copying and blocking, were presented in [17] to partition the shared caches on SMT processors and the performance gain of a parallel program other than OpenMP or multiple programs is 16-29%. For OpenMP programs, speculative precomputation and thread-level parallelism are used together [29] to achieve more efficient execution on real SMT processors using a runtime approach, however, it is hard to control the parameters (runahead distance and the span of the precomputation chunk) of speculative precomputation, and to transform OpenMP programs to include speculative precomputation. [30] presented an adaptive OpenMP loop scheduler on top of the Omni OpenMP compiler. Its runtime system adds affinity and trapezoidal scheduling, and enables the dynamic selection of the number of threads on each processor for each parallel region.

7. Conclusions

SMP technology is increasingly widely used. SMT and/or CMP technology are alternative strategies for increasing performance on a chip and exploiting space on the die to get better throughput. Applications need to be able to profit from the additional power by using multiple threads of execution and by adapting to best utilize the memory hierarchy. However, it will be hard for a compiler to automatically derive suitably large regions of code to obtain good performance. OpenMP is a relatively straightforward way to specify multithreading in a code and appears to be ideal for this purpose. But OpenMP was not designed with this kind of system and memory hierarchy specifically in mind, and there are to date few reports on the performance of OpenMP codes on such platforms. More experience, an analysis of the architectures and reconsideration of compilation strategies, are needed to determine the amount of speedup that can be expected, to derive suitable approaches to parallelization for these architectures, and to decide whether there are any modifications to OpenMP itself that would facilitate the execution of OpenMP applications on this kind of hardware.

Acknowledgements

We thank Sun Microsystems Inc. for loaning the Sun Fire V490 machine to the Computer Science Department at University of Houston (UH). Chandler Wilkerson from UH coordinated the loan and provided timely installation and technical support with the help of Tony Curtis in the High Performance Computing Center of UH.

References

1. AMD Multi-Core: Introducing x86 Multi-Core Technology & Dual-Core Processors from AMD. http://multicore.amd.com/, 2005.
2. V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. B. Jones, and B. Parady. SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance. *Workshop on OpenMP Application and Tools, WOMPAT2001*, Lecture Notes in Computer Science, 2104, pp. 1-10, 2001.
3. S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The Impact of Performance Asymmetry in Emerging Multicore Architectures. *32nd Annual International Symposium on Computer Architecture (ISCA)*, June 2005.
4. J. M. Bull and D. O'Neill. A Microbenchmark Suite for OpenMP 2.0. *Proceedings of the Third European Workshop on OpenMP (EWOMP'01)*, Barcelona, Spain, September 2001.
5. M. Frumkin. Efficiency and Scalability of an Explicit Operator on an IBM POWER4 System. *Technical Report NAS-02-008*, NASA Ames Research Center, August 2002.
6. C. Guiang and A. Purkayastha and K. Milfeld and J. Boisseau. Memory performance of dual-processor nodes: comparison of Intel Xeon and AMD Opteron memory subsystem architectures. *Proceedings for ClusterWorld Conference & Expo 2003*, San Jose, CA June 2003.
7. O. Hernandez, C. Liao, and B. M. Chapman. Dragon: A Static and Dynamic Tool for OpenMP. *WOMPAT 2004*: 53-66, 2004.
8. T. Horel, and G. Lauterbach. UltraSPARC-III: designing third-generation 64-bit performance. *IEEE Micro*, 19(3): 73-85, 1999.
9. H. Jin, M. Frumkin and J. Yan. The OpenMP Implementation of NAS Parallel Benchmarks and its Performance. *Technical Report NAS-99-011*. NASA Ames Research Center, 1999.
10. R. Kalla, B. Sinharoy, and J. Tendler. IBM POWER5 chip: a dualcore multithreaded processor. *IEEE Micro*, 24(2): 40–47, 2004.
11. P. Kongetira. A 32-way Multithreaded SPARC Processor. *Hot Chips 16*, http://www.hotchips.org/archives/hc16/.
12. D. Koufaty and D. T. Marr. Hyperthreading Technology in the NetBurst Microarchitecture. *IEEEMicro*, 2003.
13. M. Lee, B. Whitney, and N. Copty. Performance and Scalability of OpenMP Programs on the Sun Fire™ E25K Throughput Computing Server. *WOMPAT 2004*, pp. 19-28, 2004.
14. F. Liu and V. Chaudhary. Extending OpenMP for heterogeneous chip multiprocessors Parallel Processing. *Proceedings of International Conference on Parallel Processing*, pp. 161-168, Oct. 2003.
15. C. Moore. POWER4 System Microarchitecture. *Microprocessor Forum*, 2000.

16. N. Nagarajayya. Improving Application Efficiency Through Chip Multi-Threading. http://developers.sun.com/solaris/articles/chip_multi_thread.html, March 10, 2005.
17. D. S. Nikolopoulos. Code and Data Transformation for Improving Shared Cache Performance on SMT Processors. *Proceedings of 5th International Symposium on High Performance Computing, (ISHPC 2003)*, Tokyo-Odaiba, Japan, October 20-22, 2003. Lecture Notes in Computer Science 2858, Springer, 2003.
18. K. Olukotun et al. The Case for a Single-Chip Multiprocessor. *Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 2-11, 1996.
19. OpenMP Application Program Interface, Version 2.5, public draft, November 2004.
20. M. Sato, S. Satoh, K. Kusano, and Y. Tanaka. Design of OpenMP compiler for an SMP cluster. *Proc. of the 1st European Workshop on OpenMP*, pp.32-39, September 1999.
21. H. Saito, G. Gaertner, W. Jones, R. Eigenmann, H. Iwashita, R. Lieberman, M. V. Waveren, and B. Whitney. Large System Performance of SPEC OMP2001 Benchmarks. *Proceedings of 4th International Symposium (ISHPC 2002)*, Kansai Science City, Japan, May 15-17, 2002.
22. A. Snavely, N. Mitchell, L. Carter, J. Ferrante, and D. Tullsen. Explorations in Symbiosis on Two Multithreaded Architectures. *Workshop on Multi-Threaded Execution, Architecture, and Compilers* (*M-TEAC99)*, January 1999.
23. Solaris 10. http://www.sun.com/software/solaris/
24. L. Spracklen, S.G. Abraham. Chip Multithreading: Opportunities and Challenges. *11th International Symposium on High-Performance Computer Architecture (HPCA-11)*, pp. 248-252, 2005.
25. Sun Fire™ V490 and V890 Servers Architecture. http://www.sun.com
26. Sun Studio 10. http://www.sun.com/software/products/studio/index.xml
27. D. Tullsen, S. Eggers, and H. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. *Intl. Symp. on Computer Architecture*, pp. 392–403, 1995.
28. UltraSPARC®IV Processor Architecture Overview. http://www.sun.com
29. T. Wang, F. Blagojevic and D. S. Nikolopoulos. Runtime Support for Integrating Precomputation and Thread-Level Parallelism on Simultaneous Multithreaded Processors. *The Seventh Workshop on Languages, Compilers, and Run-time Support for Scalable Systems (LCR 2004)*. Houston, TX, Oct. 2004.
30. Y. Zhang, M. Burcea, V. Cheng, R. Ho and M. Voss. An Adaptive OpenMP Loop Scheduler fro Hyperthreded SMPs. *Proc. of International Conference on Parallel and Distributed Systems (PDCS-2004)*, San Francisco, CA, September 2004.
31. J. Pranevich. The Wonderful World of Linux 2.6. http://www.kniggit.net/wwol26.html