Scalability Analysis of the SPEC OpenMP Benchmarks on Large-Scale Shared Memory Multiprocessors

Karl Fürlinger[1,2], Michael Gerndt[1], and Jack Dongarra[2]

[1] Lehrstuhl für Rechnertechnik und Rechnerorganisation,
Institut für Informatik,
Technische Universität München
{fuerling, gerndt}@in.tum.de
[2] Innovative Computing Laboratory,
Department of Computer Science,
University of Tennessee
{karl, dongarra}@cs.utk.edu

Abstract. We present a detailed investigation of the scalability characteristics of the SPEC OpenMP benchmarks on large-scale shared memory multiprocessor machines. Our study is based on a tool that quantifies four well-defined overhead classes that can limit scalability – for each parallel region separately and for the application as a whole.

Keywords: SPEC, Shared Memory Multiprocessors.

1 Introduction

OpenMP has emerged as the predominant programming paradigm for scientific applications on shared memory multiprocessor machines. The OpenMP SPEC benchmarks were published in 2001 to allow for a representative way to compare the performance of various platforms. Since OpenMP is based on compiler directives, the compiler and the accompanying OpenMP runtime system can have a significant influence on the achieved performance.

In this paper we present a detailed investigation of the scalability characteristics of the SPEC benchmarks on large-scale shared memory multiprocessor machines. Instead of just measuring each application's runtime for increasing processor counts, our study is more detailed by measuring four well-defined sources of overhead that can limit the scalability and by performing the analysis not only for the overall program but also for each individual parallel region separately.

The rest of this paper is organized as follows. In Sect. 2 we provide a brief overview of the SPEC OpenMP benchmarks and their main characteristics. In Sect. 3 we describe the methodology by which we performed the scalability analysis and the tool which we used for it. Sect. 4 presents the results of our study, while we discuss related work in Sect. 5 and conclude in Sect. 6.

2 The SPEC OpenMP Benchmarks

The SPEC OpenMP benchmarks come in two variants. The *medium* variant (SPEC-OMPM) is designed for up to 32 processors and the 11 applications contained in this suite were created by parallelizing the corresponding SPEC CPU applications. The *large* variant (SPEC-OMPL) is based on the medium variant (with code-modifications to increase scalability) but two applications (galgel and ammp) have been omitted and a larger data set is used.

Due to space limitations we omit a textual description of the background, purpose, and implementation of each application, please refer to [7] for such a description. Instead, Table 1 lists the main characteristics of each application with respect to the OpenMP constructs used for parallelization (suffix _m denotes the medium variant, while suffix _l denotes the large variant of each application).

Table 1. The OpenMP constructs used in each of the applications of the SPEC-OpenMP benchmark suite

	BARRIER	LOOP	CRITICAL	LOCK	PARALLEL	PARALLEL_LOOP	PARALLEL_SECTIONS
wupwise_m/wupwise_l		11	1		7	3	
swim_m						8	
swim_l		2			2	10	
mgrid_m/mgrid_l		12			12		
applu_m	2	17			9	13	
applu_l	2	18			10	12	
galgel_m		7			2	26	1
equake_m		2			2	9	
equake_l		4			4	8	
apsi_m		23			23	1	
apsi_l		18			18	10	
gafort_m/gafort_l		6		400000	6	1	
fma3d_m		1	1		1	29	
fma3d_l		8			4	47	
art_m/art_l		2	1		1	3	
ammp_m		2			5	5	

3 Scalability Analysis Methodology

We performed the scalability study with our own OpenMP profiling tool, `ompP` [4,5]. `ompP` delivers a text-based profiling report at program termination that is meant to be easily comprehensible by the user. As opposed to standard subroutine-based profiling tools like gprof [6], `ompP` is able to report timing data and execution counts directly for various OpenMP constructs.

In addition to giving flat region profiles (number of invocations, total execution time), ompP performs overhead analysis, where four well-defined overhead classes (synchronization, load imbalance, thread management, and limited parallelism) are quantitatively evaluated. The overhead analysis is based on the categorization of the execution times reported by ompP into one of the four overhead classes. For example, time in an explicit (user-added) OpenMP barrier is considered to be synchronization overhead.

Table 2. The timing categories reported by ompP for the different OpenMP constructs and their categorization as overheads by ompP's overhead analysis. (S) corresponds to synchronization overhead, (I) represents overhead due to imbalance, (L) denotes limited parallelism overhead, and (M) signals thread management overhead.

	seqT	execT	bodyT	exitBarT	enterT	exitT
MASTER	•					
ATOMIC		• (S)				
BARRIER		• (S)				
USER_REGION		•				
LOOP		•		• (I)		
CRITICAL		•	•		• (S)	• (M)
LOCK		•	•		• (S)	• (M)
SECTIONS		•	•	• (I/L)		
SINGLE		•	•	• (L)		
PARALLEL	•	•		• (I)	• (M)	• (M)
PARALLEL_LOOP	•	•		• (I)	• (M)	• (M)
PARALLEL_SECTIONS	•	•	•	• (I/L)	• (M)	• (M)

Table 2 shows the details of the overhead classification performed by ompP. This table lists the timing categories reported by ompP (execT, enterT, etc.) for various OpenMP constructs (BARRIER, LOOP, etc.) A timing category is reported by ompP if a '•' is present and S, I, L, and M indicate to which overhead class a time is attributed. A detailed description of the motivation for this classification can be found in [5].

A single profiling run with a certain thread count gives the overheads according to the presented model for each parallel region separately and for the program as a whole. By performing the overhead analysis for increasing thread numbers, scalability graphs as shown in Fig. 1 are generated by a set of perl scripts that come with ompP. These graphs show the accumulated runtimes over all threads, the "Work" category is computed by subtracting all overheads form the total accumulated execution time. Note that a perfectly scaling code would give a constant total accumulated execution time (i.e., a horizontal line) in this kind of graph if a fixed dataset is used (as is the case for our analysis of the SPEC OpenMP benchmarks.

4 Results

We have analyzed the scalability of the SPEC benchmarks on two cc-NUMA machines. We ran the medium size benchmarks from 2 to 32 processors on a 32 processor SGI Alitx 3700 Bx2 machine (1.6 GHz, 6 MByte L3-Cache) while the tests with SPEC-OMPL (from 32 to 128 processors, with increments of 16) have been performed on a node of a larger Altix 4700 machine with the same type of processor. The main differences to the older Altix 3700 Bx2 are an upgraded interconnect network (NumaLink4) and a faster connection to the memory subsystem.

Every effort has been made to ensure that the applications we have analyzed are optimized like "production" code. To this end, we used the same compiler flags and runtime environment settings that have been used by SGI in the SPEC submission runs (this information is listed in the SPEC submission reports) and we were able to achieve performance numbers that were within the range of variations to be expected from the slightly different hardware and software environment.

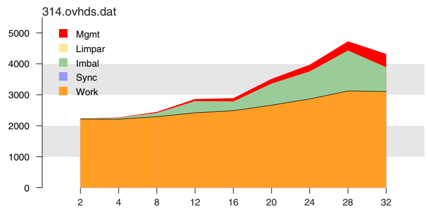
The following text discusses the scalability properties we were able to identify in our study. Due to space limitations we can not present a scalability graph for each application or even for each parallel region of each application. Fig. 1 shows the most interesting scalability graphs of the SPEC OpenMP benchmarks we have discovered. We also have to limit the discussion to the most interesting phenomena visible and can not discuss each application.

Results for the medium variant (SPEC-OMPM):

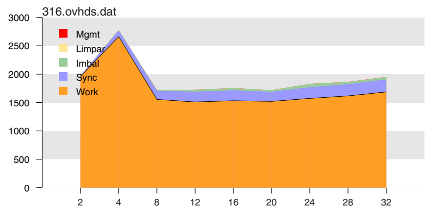
wupwise_m: This application scales well from 2 to 32 threads, the most significant overhead visible is load imbalance increasing almost linearly with the number of threads used (it is less than 1% for 2 threads and rises to almost 12% of aggregated execution time for 32 threads). Most of this overhead is incurred in two time-consuming parallel loops (`muldoe.f 63-145` and `muldeo.f 63-145`).

swim_m: This code scales very well from 2 to 32 threads. The only discernible overhead is a slight load imbalance in two parallel loops (`swim.f 284-294` and `swim.f 340-352`), each contributing about 1.2% overhead with respect to the aggregated execution time for 32 threads.

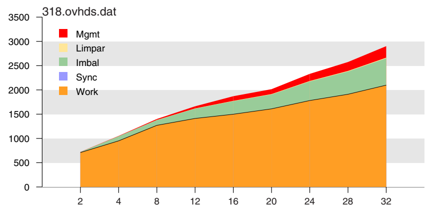
mgrid_m: This code scales relatively poorly (cf. Fig. 1a). Almost all of the application's 12 parallel loops contribute to the bad scaling behavior with increasingly severe load imbalance. As shown in Fig. 1a, there appears to be markedly reduced load imbalance for 32 and 16 threads. Investigating this issue further we discovered that this behavior is only present in three of the application's parallel loops (`mgrid.f 265-301`, `mgrid.f 317-344`, and `mgrid.f 360-384`). A source-code analysis of these loops reveals that in all three instances, the loops are always executed with an iteration count that is a power of two (which ranges from 2 to 256 for the `ref` dataset). Hence, thread counts that are not powers of two generally exhibit more imbalance than powers of two.



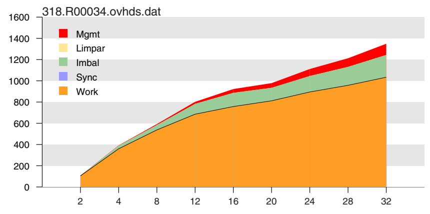
(a) mgrid_m.



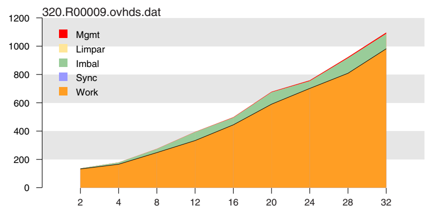
(b) applu_m.



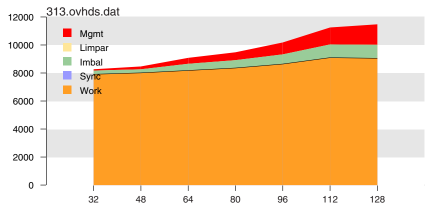
(c) galgel_m.



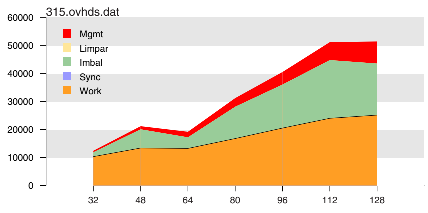
(d) galgel_m (lapack.f90 5081-5092).



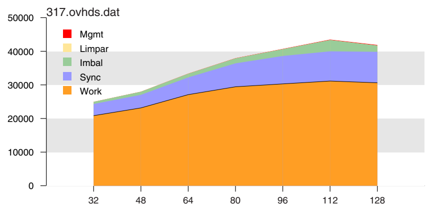
(e) equake_m (quake.c 1310-1319).



(f) swim_l.



(g) mgrid_l.



(h) applu_l.

Fig. 1. Scalability graphs for some of the applications of the SPEC OpenMP benchmark suite. Suffix _m refers to the medium size benchmark, while _l refers to the large scale benchmark. The x-axis denotes processor (thread) count and the y-axis is the accumulated time (over all threads) in seconds.

applu_m: The interesting scalability graph of this application (Fig. 1b) shows super-linear speedup. This behavior can be attributed exclusively to one parallel region (`ssor.f 138-209`) in which most of the execution time is spent (this region contributes more than 80% of total execution time), the other parallel regions do not show a super-linear speedup. To investigate the reason for the super-linear speedup we used `ompP`'s ability to measure hardware performance counters. By common wisdom, the most likely cause of super-linear speedup is the increase in overall cache size that allows the application's working set to fit into the cache for a certain number of processors. To test this hypothesis we measured the number of L3 cache misses incurred in the `ssor.f 138-209` region and the results indicate that, in fact, this is the case. The total number of L3 cache misses (summed over all threads) is at 15 billion for 2 threads, and at 14.8 billion at 4 threads. At 8 threads the cache misses reduce to 3.7 billion, at 12 threads they are at 2.0 billion from where on the number stays approximately constant up to 32 threads.

galgel_m: This application scales very poorly (cf. Fig. 1c). The most significant sources of overhead that are accounted for by `ompP` are load imbalance and thread management overhead. There is also, however, a large fraction of overhead that is not accounted for by `ompP`. A more detailed analysis of the contributing factors reveals that in particular one small parallel loop contributes to the bad scaling behavior: `lapack.f90 5081-5092`. The scaling graph of this region is shown in Fig. 1d. The accumulated runtime for 2 to 32 threads increases from 107.9 to 1349.1 seconds (i.e., the 32 thread version is only about 13% faster (wall-clock time) than the 2 processor execution).

equake_m: Scales relatively poorly. A major contributor to the bad scalability is the small parallel loop at `quake.c 1310-1319`. The contribution to the wall-clock runtime of this region increases from 10.4% (2 threads) to 23.2% (32 threads). Its bad scaling behavior (Fig. 1e) is a major limiting factor for the application's overall scaling ability.

apsi_m: This code scales poorly from 2 to 4 processors but from there on the scaling is good. The largest identifiable overheads are imbalances in the application's parallel loops.

Results for the large variant (SPEC-OMPL):

wupwise_l: This application continues to scale well up to 128 processors. However, the imbalance overhead already visible in the medium variant increases in severity.

swim_l: The dominating source of inefficiency in this application is thread management overhead that dramatically increases in severity from 32 to 128 threads (cf. 1f). The main source is the reduction of three scalar variables in the small parallel loop `swim.f 116-126`. At 128 threads more than 6 percent of total accumulated runtime are spent in this reduction operation. The time for the reduction is actually larger than the time spent in the body of the parallel loop.

mgrid_l: This application (cf. 1g) shows a similar behavior as the medium variant. Again lower numbers are encountered for thread counts that are powers of two. The overheads (mostly imbalance and thread management) however, dramatically increase in severity at 128 threads.

applu_l: Synchronization overhead is the most severe overhead of this application (cf. 1h). Two explicit barriers cause most of this overhead with severities of more than 10% of total accumulated runtime each.

equake_l: This code shows improved scaling behavior in comparison to the medium variant which results from code changes that have been performed.

5 Related Work

Saito et al. [7] analyze the published results of the SPEC-OMPM suite on large machines (32 processors and above) and describe planned changes for the – then upcoming – large variant of the benchmark suite.

A paper of Sueyasu [8] analyzes the scalability of selected components of SPEC-OMPL in comparison with the medium variant. The experiments were performed on a Fujitsu Primepower HPC2500 system with 128 processors. A classification of the applications into good, poor, and super-linear is given and is more ore less in line with our results. No analysis on the level of individual parallel regions is performed and no attempt for a overhead classification is made in this publication.

The work of Aslot et al. [1] describes static and dynamic characteristics of the SPEC-OMPM benchmark suite on a relatively small (4-way) UltraSPARC II system. Similar to our study, timing details are gathered on the basis of individual regions and a overhead analysis is performed that tries to account for the difference in observed and theoretical (Amdahl) speedup. While the authors of this study had to instrument their code and analyze the resulting data manually, our `ompP` tool performs this task automatically.

Fredrickson et al. [3] have evaluated, among other benchmark codes, the performance characteristics of seven applications from the OpenMP benchmarks on a 72 processor Sun Fire 15K. In their findings, all applications scale well with the exception of swim and apsi (which is not in line with our results, as well as, e.g. [7]). This study also evaluates "OpenMP overhead" by counting the number of parallel regions and multiplying this number with an empirically determined overhead for creating a parallel region derived from an execution of the EPCC micro-benchmarks [2]. Compared to our approach, this methodology of estimating the OpenMP overhead is less flexible and accurate, as for example it does not account for load-imbalance situations and requires an empirical study to determine the "cost of a parallel region". Note that in our study all OpenMP-related overheads are accounted for, i.e., the work category does not contain any OpenMP related overhead.

6 Conclusion and Future Work

We have presented a scalability analysis of the medium and large variants of the SPEC OpenMP benchmarks. The applications show a widely different scaling behavior and we have demonstrated that our tool `ompP` can give interesting, detailed insight into this behavior and can provide valuable hints towards an explanation for the underlying reason. Notably, our scalability methodology encompasses four well-defined overhead categories and offers insights into how the overheads change with increasing numbers of threads. Also, the analysis can be performed for individual parallel regions and as shown by the examples, the scaling behavior can be widely different. One badly scaling parallel region can have increasingly detrimental influence on an application's overall scalability characteristics.

Future work is planned along two directions. Firstly, we plan to exploit `ompP`'s ability to measure hardware performance counters to perform a more detailed analysis of memory access overheads. All modern processors allow the measurement of cache-related events (misses, references) that can be used for this purpose. Secondly, we plan to exploit the knowledge gathered in the analysis of the SPEC benchmarks for an optimization case study. Possible optimizations suggested by our study include the privatization of array variables, changes to the scheduling policy of loops and avoiding the usage of poorly implemented reduction operations.

References

1. Vishal Aslot and Rudolf Eigenmann. Performance characteristics of the SPEC OMP2001 benchmarks. *SIGARCH Comput. Archit. News*, 29(5):31–40, 2001.
2. J. Mark Bull and Darragh O'Neill. A microbenchmark suite for OpenMP 2.0. In *Proceedings of the Third Workshop on OpenMP (EWOMP'01)*, Barcelona, Spain, September 2001.
3. Nathan R. Fredrickson, Ahmad Afsahi, and Ying Qian. Performance characteristics of OpenMP constructs, and application benchmarks on a large symmetric multiprocessor. In *Proceedings of the 17th ACM International Conference on Supercomputing (ICS 2003)*, pages 140–149, San Francisco, CA, USA, 2003. ACM Press.
4. Karl Fürlinger and Michael Gerndt. ompP: A profiling tool for OpenMP. In *Proceedings of the First International Workshop on OpenMP (IWOMP 2005)*, Eugene, Oregon, USA, May 2005. Accepted for publication.
5. Karl Fürlinger and Michael Gerndt. Analyzing overheads and scalability characteristics of OpenMP applications. In *Proceedings of the Seventh International Meeting on High Performance Computing for Computational Science (VECPAR'06)*, Rio de Janeiro, Brasil, 2006. To appear.
6. Susan L. Graham, Peter B. Kessler, and Marshall K. McKusick. gprof: A call graph execution profiler. *SIGPLAN Not.*, 17(6):120–126, 1982.
7. Hideki Saito, Greg Gaertner, Wesley B. Jones, Rudolf Eigenmann, Hidetoshi Iwashita, Ron Lieberman, G. Matthijs van Waveren, and Brian Whitney. Large system performance of SPEC OMP2001 benchmarks. In *Proceedings of the 2002 International Symposium on High Performance Computing (ISHPC 2002)*, pages 370–379, London, UK, 2002. Springer-Verlag.
8. Naoki Sueyasu, Hidetoshi Iwashita, Kohichiro Hotta, Matthijs van Waveren, and Kenichi Miura. Scalability of SPEC OMP on Fujitsu PRIMEPOWER. In *Proceedings of the Fourth Workshop on OpenMP (EWOMP'02)*, 2002.