Analyzing the Interaction of OpenMP Programs Within Multiprogramming Environments on a Sun Fire E25K System with PARbench

Rick Janda, Wolfgang E. Nagel, and Bernd Trenkler

Center of Information Services and
High Performance Computing
Dresden University of Technology
01162 Dresden, Germany
`rick.janda@zih.tu-dresden.de`, `bernd.trenkler@tu-dresden.de`,
`wolfgang.nagel@tu-dresden.de`

Abstract. Nowadays, most high performance computing systems run in multiprogramming mode with several user programs simultaneously utilizing the available CPUs. Even though most current SMP systems are implemented as ccNUMA to reduce the bottleneck of main memory access, the user programs still interact as they share other system resources and influence the scheduler decisions with their generated load. PARbench was designed to generate complete load scenarios based on synthetic jobs and to measure the job behavior during the execution of these scenarios. The E25K is a ccNUMA system with up to 72 dual core CPUs and a crossbar-based connection network. This paper describes the results of the examination of such a Sun Fire E25K system with PARbench. First, PARbench was used to investigate the performance impact caused by the interactions of jobs on fully loaded and overloaded machines. Second, the impact of operating system tasks to the performance of OpenMP parallelized programs in scenarios of full load as created by the cluster batch engine is quantized, especially when these system tasks are not considered in the initial load calculation. Additionally, the generated scenarios were used for a statistical analysis of the scheduling of OpenMP programs, focusing on data locality and migration frequency.

1 Introduction

Current installations of high performance computing systems often contain systems with several hundred processors. Not all user programs need this huge amount of CPUs. Thus, the systems run several user jobs simultaneously in multiprogramming mode. While these jobs can often use a subset of the available CPUs almost exclusively, they nevertheless share common resources like data connections, caches, or the I/O subsystem. The scheduler may also migrate jobs to other CPUs in order to assign resources equitably. Such migrations and saturated memory connections are major causes of performance degradations. The performance impact becomes more and more substantial as the processor/memory speed gap widens.

OpenMP is a widely used approach to parallelize calculations on SMP systems. In the past a lot of work has been presented that assesses the performance of OpenMP programs on dedicated SMP systems or measures the runtime of different OpenMP directives. However, these benchmarks assess the machine's performance under quite favorable circumstances and say nothing about the interaction of several user jobs in production environments. They also do not consider the load generated by the operating system itself, that may be much higher in real production environments than in benchmark situations.

PARbench was designed to address exactly this issue. It permits the user to generate synthetic jobs with various characteristics based on sequences of simple benchmark kernels. After that, several of these jobs can be executed simultaneously. This permits the composition of almost arbitrary workload scenarios. During the execution of the whole scenario the runtime and the CPU time of each job is measured. Thereby, PARbench can not only generate sequential jobs but even tightly coupled parallel programs based on OpenMP. The first version of PARbench had been designed from 1988-1990 to measure the interaction of several programs during execution in multiprogramming mode [1], [2]. In 2001 PARbench was ported to the SGI Origin 3800 by Sebastian Boesler and the use of OpenMP as a standard for parallelization was introduced [3]. Meanwhile, the vector systems NEC SX-4 and SX-5 and the IBM p690 series were also analyzed [4], [5].

Object of this investigation were the Sun Fire E25Ks of the RWTH Aachen with 72 dual core UltraSPARC IV CPUs each, running on Solaris 9. The PARbench code was compiled with the Sun Studio 9 Compiler Collection. A comprehensive view of the architecture of the Sun Fire E25K can be found in [6]. More details about the UltraSAPRC IV CPU can be found in [7].

The work is parted in the following sections: The first part of the investigation will be a general assessement of the scalability of the crossbar-based ccNUMA architecture with dual core CPUs. The second part will examine the operation mode driven by the cluster batch engine (Sun N1 Grid Engine 5.3). The performance impact to OpenMP programs due to operating system tasks, which are not considered in the load calculation of the batch engine, will be elaborated. The last part is dedicated to a statistical analysis of the scheduling and will reveal some weaknesses in the treatment of OpenMP programs.

2 Performance of Sequential Jobs Under Full Load and Overload

The main reason for applying the ccNUMA for large symmetric multiprocessor systems is to effectively widen the memory access bottleneck in comparison to UMA systems. Therefore, ccNUMA systems should scale significantly better for a larger number of CPUs. The first question, that was to be answered by this investigation, is how well the E25Ks do scale. The amount of interaction between the user jobs, due to sharing system resources and maintaining cache coherency in the system, was examined.

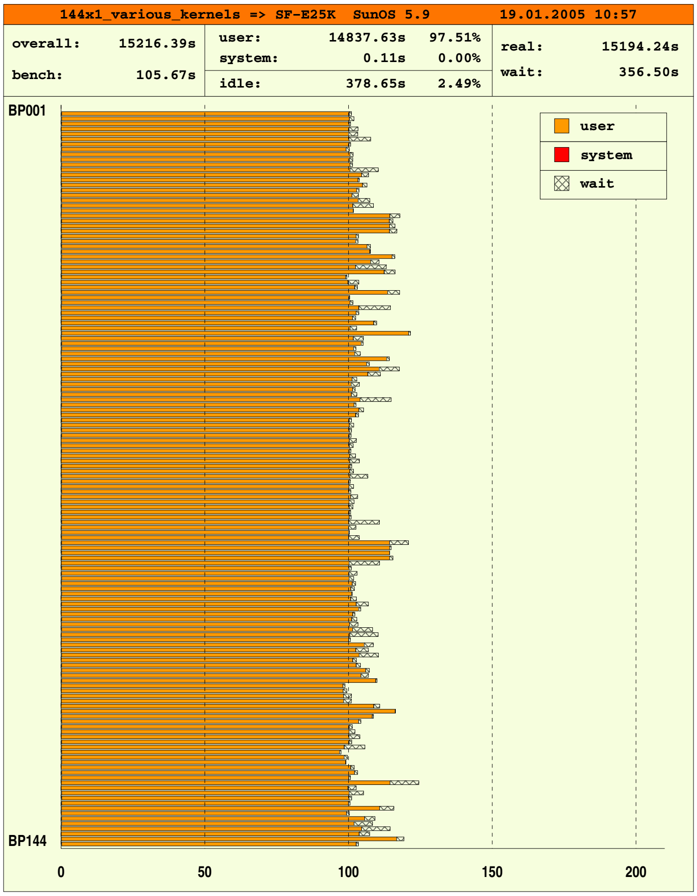

Fig. 1. 144 different jobs on the Fire E25K

2.1 Full Load

To obtain a first overview about the scalability of the architecture, 36 jobs with different kernel numbers and 100 seconds runtime each were generated. These kernels covers all of the 17 internally used math cores, do not perform any I/O operations and span as much as possible of the available data matrices to run out of the caches and stress the memory subsystem. After the generation all generated jobs were executed simultaneously with four copies each to achieve 144 jobs. The result is shown in fig. 1. Some jobs clearly distinguish from other by their increased CPU usage but none of the jobs suffers from a crucial performance impact. Even the most affected job took only 120 seconds which correspond to a relative increase of just 20%. From this point of view the hardware scales very well.

2.2 Specifying the OS Load

In the previous test the jobs showed some waiting time. This was caused by operating system tasks that need a CPU from time to time to do their work. The short interruptions of the user jobs may lead to increased rates of cache misses due to the system jobs replacing the cache lines with their own data or the user jobs being migrated to other CPUs by the scheduler to assure fair resource assignment. Therefore, 144 user jobs at once as well as the system tasks results in a slightly overloaded system. The impact on sequential jobs by these circumstances will be investigated in a next step. Additionally it may be of interest, to what extend the influence increases within clearly overloaded systems.

For this reason the load created by the operating system itself is quantified in order to build tests, that consider this load and avoid interruption of the user jobs. PARbench measures CPU usage time and real time for every job and calculate the overall waiting time for a scenario. A simple scenario with 144 sequential jobs shows, that the jobs remained without CPU in about 2% of their runtime. The experiment was repeated with gradually reduced number of user jobs until the overall waiting time became almost zero at 140 user jobs.

2.3 Overload

For a next test, a core sequence merely based on only one kernel version was generated to take 100 seconds. Then several copies of this jobs were run at once, first with 140 copies to consider the system tasks, second with 144 copies which comply to the load factor achieved by the cluster batch engine and do not consider system task and third with 164 jobs to overload the system. This test was repeated for various kernel versions. Table 1 compares the results. As one can see, most of the kernels do not show any relevant influence from slight or conspicuous overload in comparison to their CPU usage time in a full loaded system with 140 user jobs plus system jobs. Only version 241, 281 and 301 consume recognizable more CPU time on the clearly overloaded system. For a analysis of this behavior, some additional data from the jobs, like cache usage and percentage of write-accesses, is needed. This can be achieved with the performance counters of the UltraSPARC IV CPU but is not yet included in PARbench. The code of the math cores, however, reveals that these cores contains some really odd access patterns to confuse the cache usage and stress the memory subsystem. Hence, their generated load in the memory subsystem is not typical for scientific computations. In the outcome sequential jobs are practical not affected by the system tasks.

3 Influence of Operating System Processes on OpenMP Programs

The cluster batch engine of the RWTH Aachen uses a simple scheme to avoid overloading the systems. The users have to specify the number of processes or threads

Table 1. Relative increase in CPU usage time with raising load factor for different kernel versions

Kernel version	MREFS	FLOPS	Relative average CPU time			
			single	140x	144x	164x
126	765.7	770.1	100%	100.1%	100.1%	100.1%
111	406.9	638.6	100%	100.2%	100.4%	100.2%
101	451.3	702.3	100%	100.4%	100.3%	100.4%
151	919.8	893.2	100%	100.4%	100.3%	100.4%
81	205.2	410.6	100%	102.5%	102.6%	102.4%
226	669.0	79.0	100%	104.5%	104.5%	103.0%
61	71.5	286.0	100%	106.7%	106.5%	106.5%
246	311.2	1.4	100%	107.4%	107.4%	107.5%
221	485.1	54.1	100%	111.5%	113.0%	108.4%
121	309.5	309.6	100%	109.1%	108.9%	108.8%
141	130.7	126.7	100%	118.6%	118.6%	118.4%
281	8.5	0.0	100%	**117.6%**	**120.0%**	**122.0%**
241	325.2	0.1	100%	**109.1%**	**109.1%**	**124.7%**
286	75.1	0.6	100%	124.6%	124.7%	124.8%
201	593.9	119.3	100%	125.3%	125.3%	124.8%
301	63.3	31.7	100%	**124.4%**	**127.0%**	**135.2%**

that they want to use for their MPI or OpenMP jobs. The batch engine then executes only as many jobs simultaneously on the according system that each MPI instance and each thread of the OpenMP programs can theoretically use a CPU exclusively. For the E25K systems the batch engine adjust the number of user programs and threads to 144, which is the number of CPU cores. However, operating system tasks are not considered at all, thus, the system runs slightly overloaded. As it could be seen in the previous section, sequential programs do not suffer much from the short interruptions caused by system tasks. On the other hand, parallelized programs, especially fine-grained parallelized programs like OpenMP programs, may experience much more impact from these short interruptions. To ensure data validity, OpenMP programs contain a lot of synchronization barriers, that have to be reached by all worker threads before the calculation can proceed. In order to obtain efficiently parallelized programs, the work will be spread to every worker thread equally. If every thread can use a CPU exclusively, all worker threads will reach the barriers at the same time and the next part of the work is spread to the worker threads immediately. But if some worker threads are interrupted by system tasks, all other threads of the OpenMP program will have to wait on the next barrier for the interrupted thread. Part of the investigation was to quantify these impact on the performance of OpenMP programs.

For this purpose a job with 200 seconds sequential runtime and no I/O was created. This job was generated to average 500 MREFS and 500 MFLOPS and contains about 5000 kernels, which means 5000 implicit barriers for the OpenMP parallelization. The scenario contains nine copies of this job which were parallelized to eight threads each. First these jobs were executed with 72 sequential

Table 2. Average CPU usage time (user+system) and runtime of the parallelized jobs (9x8) with spinning threads (busy waiting) and different background load in the system

Total number of user threads	Test setup	CPU usage time [s] of the par. jobs	Runtime [s] (average)
8	1x8	232.4	29.1
72	9x8	234.7	29.3
138	66x1 9x8	238.7	30.0
140	68x1 9x8	239.5	30.1
142	70x1 9x8	248.2	31.4
144	72x1 9x8	**272.7**	**34.9**

Table 3. Runtime differences of the parallelized jobs between considering operating system tasks and not

Threads/job	Average runtime of the par. jobs [s] for total number of user threads		Rel. extension [%]
	140	144	
8	30.1	34.9	15.9%
16	19.3	26.3	36.3%
24	14.1	22.3	58.2%
32	10.7	20.6	92.5%

jobs simultaneously and then with only 68 sequential jobs in parallel to ensure four free CPUs for the system tasks. Fig. 2 and fig. 3 show the results. Some of the OpenMP jobs ran notably longer without free CPUs for system jobs. To ensure that the impact does not originate from other influences, the measurement was also repeated without the simultaneous sequential jobs to compare CPU usage and runtime with these values. Table 2 contains the average runtime and the average CPU usage for all of the parallelized jobs and the different scenarios. As one can see, the runtime does not differ much for lower system loads but does notably increase with more than 140 user threads in the system[1]. But then again, an increase to 35 seconds compared to 30 seconds is practically neglectable.

After that the examination was expanded to a higher degree of parallelization with up to 32 threads. Again the OpenMP jobs were executed with some sequential jobs simultaneously to fill the system and then with four sequential jobs lesser to avoid interruption by system tasks. Table 3 compares the average runtime of the parallel jobs in the two scenarios for the an increasing number of parallel threads. One can see, that the impact increases rapidly with higher parallelization. The slight overload caused by the operating system has a dramatic impact on the performance of the OpenMP programs with higher parallelization. OpenMP jobs with 32 threads will already run almost twice as long, if the operating system tasks are not considered in the load calculation and slightly overload the system.

[1] Sequential programs are counted as one thread.

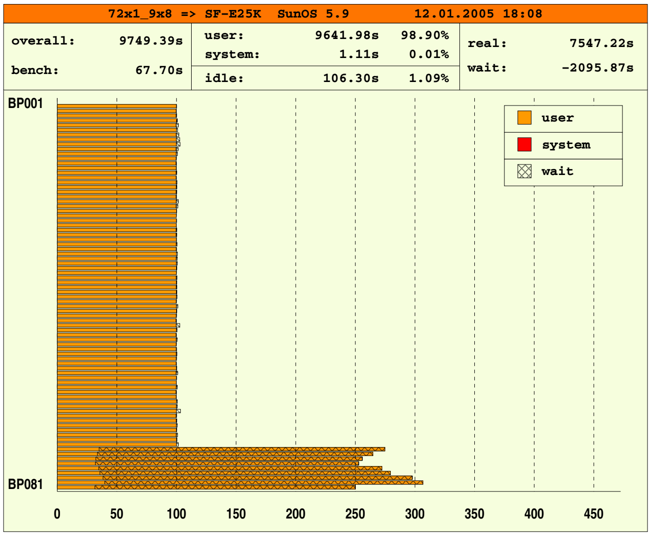

Fig. 2. 144 user threads within the SF-E25K (144 CPUs)

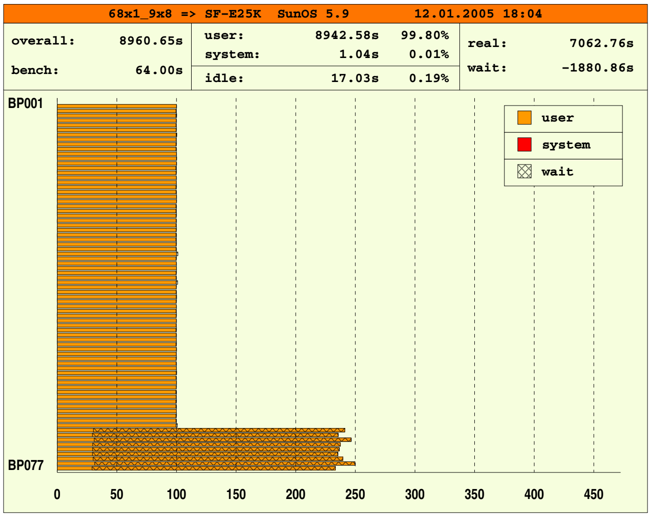

Fig. 3. 140 user threads within the SF-E25K (144 CPUs)

4 Scheduling Analysis

PARbench allows assessing the performance of jobs in different multiprogramming scenarios. The measured interaction, however, may caused by hardware limitation as well as by unfavorable scheduling. To obtain a direct access to the decisions of the scheduler the PARbench startup script was extended in order to run the Solaris `prstat` tool during the workload execution. The tested workloads were reduced by one sequential job to achieve the same load factor as without `prstat`. The `prstat` tool is a program similar to the widely known `top` utility but additionally offers precise information about every thread of the running programs. In this case `prstat` was run in batch mode to log the process and thread assignment to the CPUs every second. After the experiments, the gathered data was analyzed regarding migrations and data locality. The sequence of used CPUs was determined for every user job and for each of their threads.

A migration always occurred, if the CPU number changed between two snapshots. Since most of the performance of the UltraSPARC IV CPU is related to its huge level two cache, programs will only obtain this performance, if they utilize this cache effectively. If a process or thread is migrated to another CPU, the cache usage will be disturbed. To compare the migrations, a *migration rate*, which set the counted migrations in ratio to the total number of snapshots made for each thread or process, was calculated.

The *home board rate* is the second ratio calculated from the gathered data. The home board of a process is the system board, where a process was first executed and where it allocated its data structures in main memory (first touch policy). Within a system board, the latency to main memory is almost the same but data access across the central crossbar switch takes much longer. Thus, the Solaris scheduler tries to bind processes to its home board and avoid migrations to other system boards. The board number is related to the CPU numbers, so it is quite simple to determine, if a process was being executed on its home board during a snapshot. The threads of an OpenMP program should also gain fast access to the data, consequently they should also be executed on the home board of the according process. As a system board contains only 4 UltraSPARC IV CPUs with two cores each, it makes only sense to examine OpenMP programs with up to 8 threads. Hence, the home board rate for a thread or process was calculated by counting the number of snapshots the thread or process was executed on its home board and setting this value in ratio to the total number of snapshots for the according thread or process.

Table 4 lists the results for the full load scenarios with and without consideration of the operating system tasks, divided into the sequential and the parallel jobs. For the sequential jobs the average was taken over the values of each sequential job. For parallel jobs the according rate was calculated for every thread separately and then averaged. As one can see, the home board binding works very well for sequential jobs. The migration ratio is very low, too, and

Table 4. Average home board rate and thread migrations rate separated for sequential an parallelized jobs

Total number of user threads	Average home board rate [%]		Average migration rate [%]	
	sequential (x1)	parallel (x8)	sequential (x1)	parallel (x8)
140	98.8	**38.4**	2.6	8.2
142	96.6	**36.3**	3.1	11.3
144	**95.7**	**38.2**	5.7	**16.5**

Table 5. Average executions ratio on the home board separated for the thread numbers

Total number of user theads	Test setup	Home board rate [%] for thread							
		1 (master)	2	3	4	5	6	7	8
140	67x1, 9x8, prstat	**98.7**	83.2	44.4	33.3	**0.0**	11.1	19.1	11.5
142	69x1, 9x8, prstat	**96.7**	62.2	56.3	12.2	**11.5**	22.2	0.0	22.2
144	71x1, 9x8, prstat	**95.7**	77.3	67.6	29.6	**10.1**	21.6	9.3	3.4

does only slightly increase with the small overload caused by the system tasks in the scenario with 144 user threads. The tide turns for parallelized jobs. All threads of the OpenMP programs were only executed one third of the time on the home board and suffer from significantly more migrations than the sequential jobs. OpenMP programs compiled with the Sun Studio 9 compilers allocate all threads at the program's start. In order to save the time for stopping and starting the threads over and over again, temporarily idle threads do busy waiting per default in order not to loose their CPU. This approach is reasonable, if the system does not become overloaded and no other jobs are available to use the freed CPUs. Accordingly, starting and stopping of threads is not the reason for the higher migration rate.

The assumption was that the scheduler does not distinguish between a sequential program and the master thread of an OpenMP program but has no favorable strategy for further threads. Thus, the same data was analyzed again but the values were grouped by the thread number. The values of the sequential jobs and the values of the master thread of the parallel jobs become one group for averaging, the second group contains the second thread of each parallel program and so on. Table 5 lists these average home board rates.

The values for the first thread resemble the values of the sequential jobs in the previous table, which supports the assumption. The binding to the home board rapidly decrease for larger thread numbers. Starting with thread five the threads are practically no more subject to any home board binding and will suffer from slow remote access to their data in the main memory of another CPU board.

5 Conclusions

In this paper, the performance of the Sun Fire E25K in multiprogramming mode was evaluated. The tests with workloads containing different sequential jobs indicate that the hardware scales very well and the ccNUMA architecture with crossbar-based connection network provide sufficient throughput to effectively eleminate the memory access bottleneck.

In contrast to sequential programs, that are not very influenced by the interruptions caused by system jobs, tightly coupled parallel programs suffer much more from those CPU losses. The performance impact increases rapidly with larger number of threads and it was shown how the consideration of the system tasks in the load calculation already reduces the runtime of OpenMP jobs with 32 threads in half. Thus, system tasks could not be ignored in the batch engine's load calculation. On the E25Ks the batch engine should limit the number of user processes and threads to 140 instead of 144, which is the current configuration.

The statistical analysis of the thread scheduling reveals some weaknesses. The first thread, which is the only thread for sequential jobs, is tightly coupled to the board, where the program data was allocated. This ensures fast access to the data and reduce the usage of the central crossbar switch. Further threads are not subject to this tight home board binding and suffer from higher memory access latency for that reason. They also show higher migration rates, which reduce cache utilization. Better thread scheduling could offer some performance increases here. Sun promises much improved thread handling with Solaris 10, which will be subject for further research.

Acknowledgments

The research described in this work was developed using the resources of the computing center of the RWTH Aachen. We would like to thank Dieter an Mey, Andrea Lorenz and Hans-Jürgen Schnitzer as members of the HPC team for providing useful advices and technical assistance.

References

1. Linn, M.A.: Eine Programmierumgebung zur Messung der wechselseitigen Einflüsse von Hintergrundlast und parallelem Programm. Techn. Report Jül-2416, Forschungszentrum Jülich (1990)
2. Nagel, W.E.: Performance evaluation of multitasking in a multiprogramming environment. Techn. Report KF-ZAM-IB-9004, Forschungszentrum Jülich (1990)
3. Boesler, S.: Performance-Analyse von Hochleistungsrechnern im Multiprogramming-Betrieb: Untersuchungen auf der SGI Origin. Diplomarbeit, Center for High Performance Computing, Dresden University of Technology (2001)
4. Kowarz, A.: Performance-Untersuchungen mit dem PARbench-System auf unterschiedlichen Parallelrechnern. Diplomarbeit, Zentrum für Hochleistungsrechnen, Technische Universität Dresden (2003)
5. Dietze, H.: Das PARbench-System: Untersuchungen zum Scheduling von parallelen Programmen auf der IBM p690. Diplomarbeit, Zentrum für Hochleistungsrechnen, Technische Universität Dresden (2004)
6. Sun Microsystems, Inc.: Sun Fire E25K/E20K Systems Overview. (2004)
7. Sun Microsystems, Inc.: UltraSPARC IV Whitepaper. (2004)