Desktop Workload Characterization for CMP/SMT and Implications for Operating System Design

Sven Bachthaler, Fernando Belli and Alexandra Fedorova
School of Computing Science, Simon Fraser University
Burnaby, B.C. Canada

Abstract—Much recent research has focused on operating system scheduling algorithms for managing shared resource contention on chip multiprocessors (CMPs) and simultaneous multithreaded (SMT) systems. While the relevance of those algorithms is apparent for server workloads, it is less obvious for desktop workloads. As CMP/SMT processors are becoming increasingly deployed in desktops, it is important to understand whether such sophisticated algorithms are required in a desktop OS. Those algorithms would be required if desktop workloads exhibited sufficient parallelism, as this would imply contention for CMP/SMT shared resources which should be managed by the OS. To determine whether such parallelism is present in desktop workloads, we conducted a study of workloads by collecting live performance data from home, university staff and lab computers. We concluded that most workloads have low parallelism, which justifies scheduling algorithms that perform load balancing and power consumption management on lightly loaded CMP/SMT systems. Furthermore, these workloads do not have sufficiently high parallelism to justify performance-optimizing contention management algorithms that rely on having many runnable threads. An important implication of our findings is that more work is needed on runtime parallelization and speculative multithreading so that desktop workloads can benefit from CMP/SMT systems with larger degrees of parallelism.

Index Terms—workload characterization, desktop workloads, chip multiprocessors, simultaneous multithreading.

1 Introduction

Multi-core technology is quickly becoming mainstream, and systems based on chip multiprocessing (CMP) and simultaneous multithreading (SMT) are made commercially available by leading hardware manufacturers [2] [6] [7] [8] [12]. Following that trend, much recent research in operating systems has focused on the design of scheduling algorithms that manage resource contention on CMP/SMT systems [4] [10] [13] [14] [15] [16] [17]. Unmanaged contention, as previous work showed, leads to inefficient resource use, performance degradation and poor quality of service for interactive applications [3] [4] [14] [15]. Scheduling algorithms for CMP/SMT systems understandably target environments where many threads or processes are runnable *at the same time*. While abundant thread-level parallelism is characteristic for server workloads, there has not been a comprehensive study showing either the presence or absence of parallelism in desktop workloads. Although it is commonly known that desktop computers are mostly idle with only short periods of CPU activity, the extent of parallelism *during* those periods of activity has not been evaluated. Presence of parallelism during those periods would indicate the need to use contention management algorithms in desktop operating systems on CMP/SMT hardware for improved user experience and power management. Given that CMP and SMT processors are increasingly used in desktop processors [11] studying the parallelism properties of desktop workloads is timely and important.

This paper presents the first (to the best of our knowledge) such study. We collected data on live systems for three user groups: home users, university lab users, and university staff users. We addressed these different groups because they could exhibit different characteristics with respect to parallelism. All our users ran the Microsoft Windows XP operating system. We collected various runtime statistics using built-in performance counters on Windows as well as other tools available from Microsoft. The key data that was collected includes the number of threads in the run queue, processor utilization, and available memory. The analysis of this data allows us to determine the amount of parallelism in the considered workloads. In particular, we use the size of the run queue as the key statistic to estimating the degree of parallelism. Most of the computers in our study were not CMP/SMT, so the number of threads in the run queue indicates how many threads *could be* running in parallel had the computer been CMP/SMT.

To address the objective of our study, it is important to look at both *low* parallelism (i.e., two-three threads ready

to run at once) and *high* parallelism (i.e., more than three threads ready to run at once). The presence of low parallelism would justify the use of scheduling algorithms that manage load balancing and power consumption on lightly loaded systems [10] [14]. The presence of high parallelism would justify the use of the performance-optimizing algorithms that determine which threads to co-schedule based on their resource use; those algorithms rely on having a large number of threads in the run queue in order to be able to select the co-schedule with the desired properties [13] [16] [17].

We found that the workloads generated by university lab and staff users show a presence of low parallelism (35-52% of the time), but almost no presence of high parallelism (<13% of the time). On the other hand, the workloads generated by home users, show a higher degree of parallelism, with low parallelism present 62-69% of the time and high parallelism present 26-35% of the time. More analysis is needed, however, to pinpoint the exact causes for a higher parallelism for this group of users.

2 Methodology

2.1 Process

In order to evaluate typical workloads of desktop computers, we collected data from three different types of user groups: home users, university lab users, and university staff users. Collection of data from the university computers was performed with collaboration and under the supervision of the Department of Computing Science at our university. The department was in charge of setting up the data collection and ensuring that all the data collected was made anonymous before it was handed to us. Data collection from home computers was performed from volunteers who agreed to participate in our study. Those volunteers were given a package to install on their computers which contained the necessary programs to setup, collect, and return the data. To reassure the volunteers that the collected data was indeed limited to the purpose of our study, it was converted to human-readable text format at the end of the collection. That way the users could review it before it was sent back to us. The data was also made anonymous as soon as we received it.

The collection of data was performed over a period of two weeks. Twenty computers were selected at random from a group of 72 from one of the Computing Science labs and monitored 24 hours a day for a period of one week. Due to privacy policies, the selection of the ten staff machines to monitor was done by the Department of Computing Science. These computers were also monitored 24 hours a day for a period of one week. For the home users, twelve people volunteered to collect data from their own home computers. They were asked to install the monitoring program and send the information back to us after one week of data collection. As we will see in the result section, not all the data collected from all computers was suitable for analysis and it was also not possible to collect data from all computers due to technical problems.

The data collection was performed locally and remotely. Local collection was used for the home computers, while remote collection was used for the university computers. Local collection means that the data is collected by the monitored computer itself, and this data is stored on the hard drive of that computer. Remote collection is performed over a network from a different computer than the one that is monitored. The monitored computer only provides the counter values. The computer doing the collection is in charge of performing the pulling and storing the collected data at predefined intervals. Remote collection has the major advantage of creating less overhead since the storage and processing of data is done on the remote computer that collects the data. Its main disadvantage is that it can generate substantial amount of network traffic; especially if the pulling of data is done frequently and/or the collected data has considerable size. Also, if the pulling interval is short, network latency can affect the quality and accuracy of the data. This was not the case in our collection since the university network is very fast. However, as indicated in the analysis section, the collected data from the home computers includes some overhead which is caused by the use of local collection mechanisms. The overhead manifested itself in an increase of the number of threads in the system whenever one of our data collection scripts ran.

2.2 Tools

In this section we describe the various tools that were used to perform data collection. All these programs are provided by Microsoft as part of the Microsoft Windows XP OS or provided as separate support tools for this family of operating systems.

The most important information is collected by Microsoft's Performance Monitor. This application is shipped with the operating system and allows collecting information about internal states of the operating system and the computer hardware. Performance Monitor was first introduced when Microsoft released its first 32-bit operating system (Windows NT 3.1) and since then it has been part of all newer versions of Windows. The application was intended to be part of the operating system from the very beginning; therefore it is designed to capture data with the smallest amount of overhead possible. Some of its main features include: local and remote collection of data, ability to schedule different polling intervals (the minimum interval is one second), and different ways to collect and store data. Performance Monitor is extensively used by IT departments to monitor performance and health of Windows servers and to

troubleshoot problems with the hardware and/or the operating system in any computer running Windows.

PsList and PsInfo are two additional programs that were called by the scripts for local data collection (and therefore only for the group of home users). Both programs are part of the Sysinternals Suite and are available for download from the Microsoft website [1]. PsList is a command-line tool that gathers additional performance information from Windows' built-in Performance Monitor, such as the information on running processes, memory, and threads. PsInfo is a command-line tool that gathers key information about the Windows operating system and the host computer. This information includes the operating system name, type and version, the number of processors, their type and speed, and the total amount of physical memory available.

2.3 Data Collection

Besides finding information about the parallelism in current desktop workloads, we were also interested in the nature of these workloads and the demand that it generates for hardware resources (in particular for the processor). Therefore, we collected more data than was explicitly needed for this project. We intend to analyze this additional data in future projects as well.

We repeatedly queried information from Performance Monitor to collect the information regarding the processor's run queue length. To get more detailed information while keeping the collection overhead low, we decided to use four different sampling intervals for the data collection. Every two seconds we queried only the length of the processor's run queue. For every 15 seconds we queried a more detailed snapshot of the current operating system state. Every 30 seconds we collected detailed information about the number of threads running in the system including their status. Finally, every five minutes a summary of the number of processes and sub-processes was gathered.

The data which is collected every 15 seconds includes the following statistics:

- Size of the run queue
- Number of running processes
- Number of running threads
- Available main memory
- Percentage of time when the processor is busy

This detailed list helps us to identify the times when the processor is heavily utilized and the workload shows a high degree of parallelism. At the same time we can sort out the cases when this high parallelism is just due to insufficient main memory (which causes paging and requires lots of additional computational power). The collection which is done every two seconds is used to validate the data collected every 15 seconds. The scripts that run every 30 seconds and every five minutes used PsList v1.28 to gather the required data. PsInfo was only run once in order to collect general information about the monitored computer.

2.4 Data Preprocessing

Whenever it was possible, we tried to capture data only when users were working with their computers. Since students and staff at the university are required to logon into their computers before they can start working with them, we could make use of that information to identify the times when users were actually working on a computer. This kind of information was not available for home users. Many home users configure their computer for automatic login, and usually they do not turn off their computer when they finish using it. This implies that part of the information collected from the home users includes time intervals where the computer was not used by a person.

2.5 Types of Hardware

Most workloads examined in our study ran on computers with conventional, non-CMP/SMT processors (to which we later refer as superscalar processors). However, there were some computers enabled with SMT (i.e., Intel Hyper-Threading (HT) Pentium). We report our data separately for superscalar and hyper-threaded computers. As expected, hyper-threaded computers usually show a lower degree of parallelism—we use the size of the run queue as the metric for parallelism. As shown in Figure 1, for the same workload the run queue is likely to be smaller on hyper-threaded computers than on superscalar ones, because there are more hardware contexts available for runnable threads.

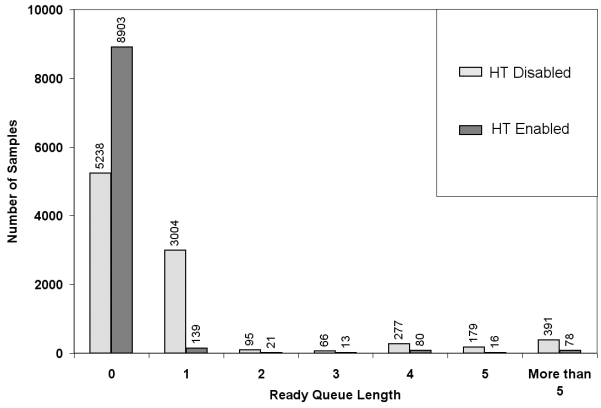

Figure 1. Two lab computers with HT enabled and disabled.

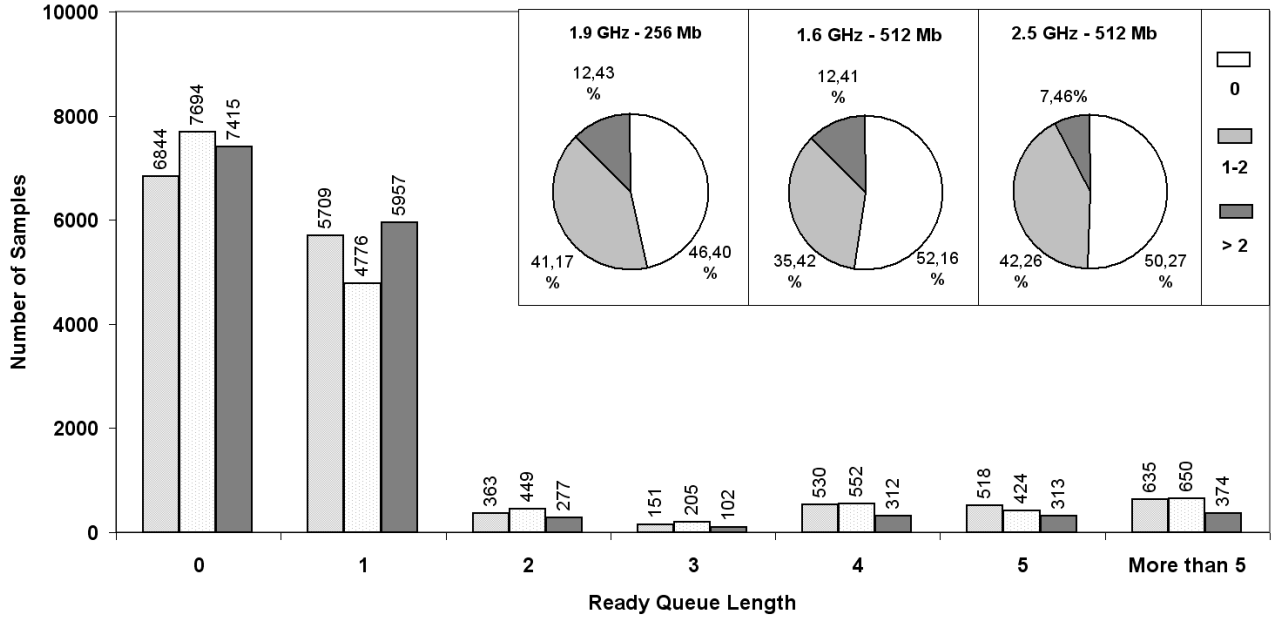

Figure 2. Lab computers - three computers per group (14750 samples per group).

3 Results

We analyzed the collected data for each of the groups separately. The hardware configuration for each of the groups or individual computers is shown in each figure. All our results are reported using the data that was collected every 15 seconds.

We found that five out of the 20 lab computers did not collect valid data due to technical problems. Similar problems arose for three home computers and two staff computers. For the remaining eight staff computers we received valid data, but one computer was found to be running an indexing service for a website. We do not consider that kind of application to be part of a normal workload of a desktop computer. Therefore we treated this as a special case and analyzed its workload separately.

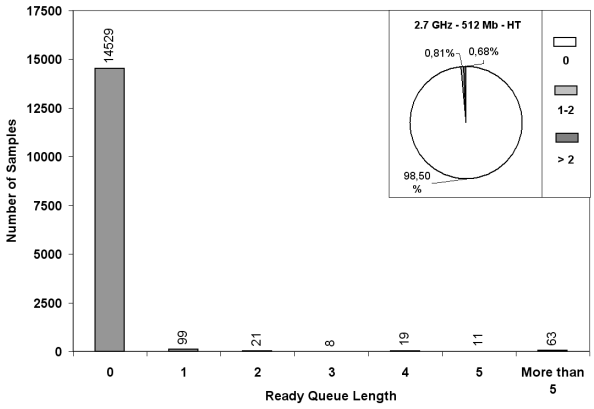

Figure 3. Three HT lab computers (14750 samples in total)

Every graph of this section shows histograms and pie charts for different data sets. Each histogram shows the number of data samples for different lengths of the run queue. Each bar of the histogram shows the sum over the samples of the computers in that group. Each pie chart shows the percentage of data set samples that belong to one of the following groups: no parallelism (zero threads in the run queue), low parallelism (one or two threads in the run queue) and high parallelism (three or more threads in the run queue). The white slice shows the percentage of samples with no parallelism, the light-gray slice shows the percentage of low parallelism and the dark-gray slice shows the percentage of high parallelism. For graphs with more than one pie chart, the order for the pie charts is the same as for the histograms.

3.1 University Lab Computers

Figure 2 shows samples collected for university computers. We group the data according to the hardware configurations of the computers, resulting in three groups of three computers. The graph for superscalar computers shows that this workload has low or high parallelism approximately half of the time, and no parallelism the rest of the time. As we will see later, university lab workloads have a compar-

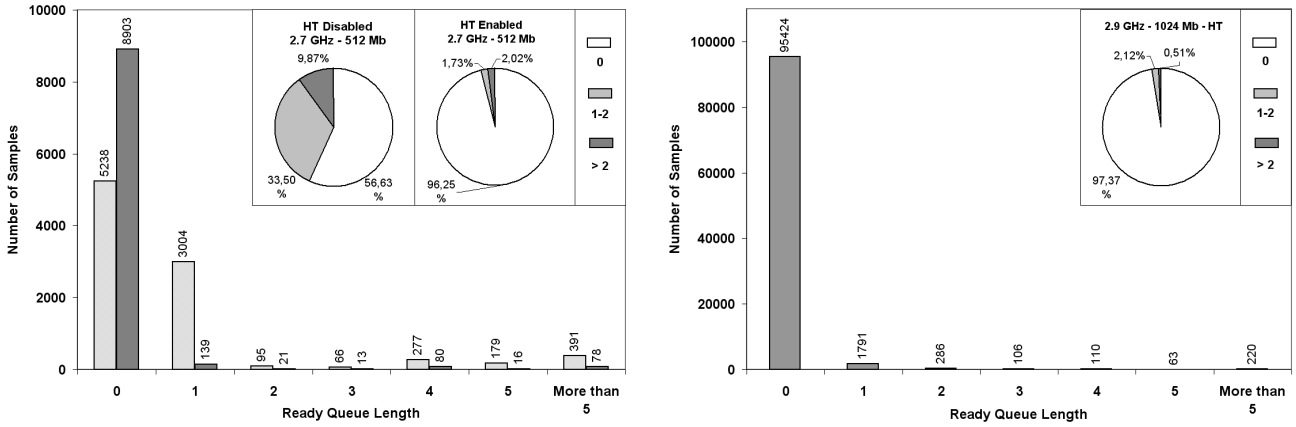

Figure 4. Left graph: Two lab computers with HT enabled and disabled (9250 samples per computer). Right graph: Six HT staff computers (98000 samples in total).

atively low number of simultaneous processors (on average there are only 30 processes and 411 threads running). The reason for that is due to the fact that the users do not have the administrative permissions to install new programs. They are restricted to the installed educational programs. Nevertheless, a significant presence of low parallelism (35-42% of the time) suggests that there will be a presence of simultaneous threads if this workload were run on a CMP/SMT system. But since the presence of high parallelism is not significant (under 13%) we believe that this workload will justify only the need for CMP/SMT scheduling algorithms targeted for lightly loaded systems.

Figure 3 and Figure 4 (left graph) show the results for university lab workloads running on the hyper-threaded computers. Figure 3 shows the computers with identical hardware configurations, Figure 4 (left graph) shows different hyper-threaded configurations, including the computer where hyper-threading was disabled. The latter is equivalent to a superscalar computer. These computers are running a similar workload as the ones that are shown in Figure 2. For computers with hyper-threading enabled, the run queue is empty for 98% of the total time. We conclude that the hyper-threaded processors are able to handle that workload appropriately. At the same time, this offers a great opportunity to benefit from scheduling algorithms that optimize power consumption of the cores. Since the average processor utilization is approximately 2.6% for an empty run queue, it would be advantageous to reduce the power consumption as suggested in [14]. As mentioned in that paper as well, such a scheduling algorithm could also improve the utilization of the cores (on CMP systems) by balancing their workload.

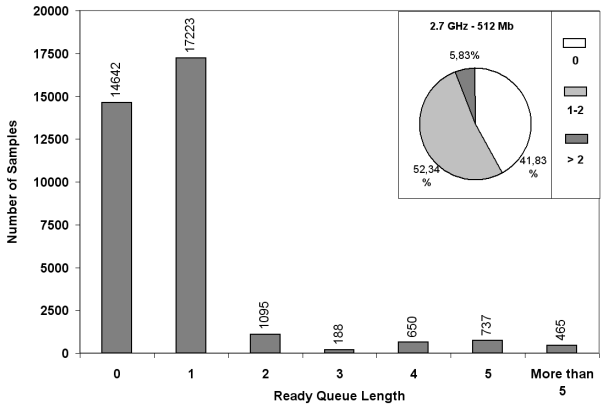

Figure 5. University staff computer (35000 samples).

3.2 University Staff Computers

The result for university staff computers is shown in Figure 4 (right graph). The data was collected for a group of 6 hyper-threaded computers that are used by the administrative personnel. The workload on these computers is heavier than for the lab users; on average 44 processes and 536 threads are running. Nevertheless, we see a similar trend as for the hyper-threaded lab computers: The run queue is empty for 97.37% of the time. It is worth mentioning that these computers have faster processors and twice the memory than the lab computers. This could further contribute to efficient emptying of the run queue.

There was only one computer among the staff computers that did not have hyper-threading. The results for this computer are shown in Figure 5. This workload shows sim-

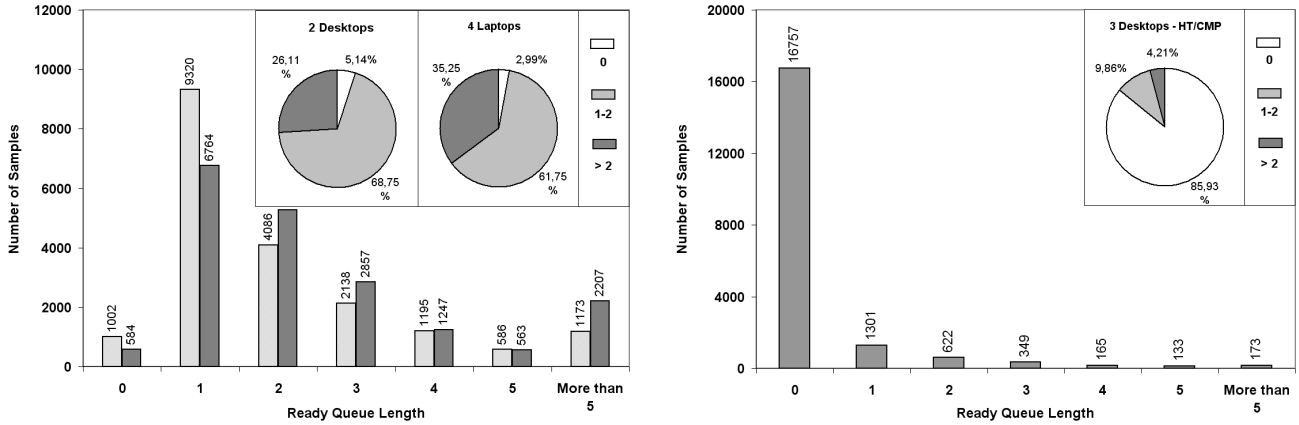

Figure 6. Left graph: Two home laptops and four home desktops (19500 samples per group). Right graph: Three HT/CMP home computers. (19500 samples in total)

ilar properties as the university lab workloads: the runtime is dominated by low parallelism (52% of the time), there is no parallelism for 42% of the time and high parallelism for roughly 6% of the time. These similarities persist despite the heavier load on the staff computers (43 processes and 449 threads on average).

It is worth mentioning that we found in the collected data the special case of a hyper-threaded computer that was used by administrative personnel and performed the indexing of web pages (in addition to running other user programs). That indexing service runs in the background and utilized the processor for 52% of the total time. Figure 7 shows the run queue for this particular computer. Even though this computer has two hardware threads, it cannot serve all ready-to-execute threads in time. It is clear that this computer shows high parallelism and will benefit from running on CMP/SMT processor with a larger degree of parallelism

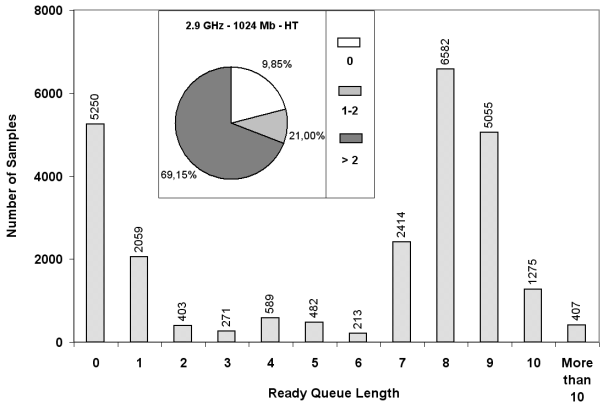

Figure 7. Hyper-threaded staff computer indexing a website. (35000 samples)

and from OS contention management algorithms. However, this kind of workload is very rare for desktop computers.

Our conclusions for this group of users echo those for lab users: there are opportunities to apply load balancing and power consumption using the scheduling algorithms for lightly loaded systems, but the use of contention-managing algorithms for highly loaded systems is not necessary.

3.3 Home Computers

After analyzing the hardware configuration of the home user computers, we decided to group these systems into three categories. The first group consists of two superscalar desktop computers; the second group consists of three computers that have two hardware threads (either via hyper-threading or CMP). The last group includes four laptops with mobile superscalar processors.

The results for the laptops and superscalar desktop computers are shown in Figure 6 (left graph). We can observe a different trend here: the workload shows significant presence of low and high parallelism (62-69% and 26-35% respectively). Furthermore, the situations with no parallelism are almost absent. We hypothesize that the larger parallelism in this workload could be due to following reasons: (1) home user workloads are more prone to viruses and spyware that run in the background; (2) home users are more likely than university users to run a large variety of programs (i.e. multimedia, IP telephony, games); (3) some laptop computers had limited free memory—this could result in an increased CPU utilization due to paging; (4) higher parallelism could be in part due to the overhead generated by our data collection script that periodically ran on the background (this is the artifact of the local data collection method used for home workloads). More analysis is needed to understand the exact cause of higher parallelism in this

workload. Finally, Figure 6 (right graph) shows the results for three computers with either hyper-threaded or dual-core processors. Every system is equipped with two gigabytes of main memory and high-end graphics cards—these computers are mainly used for graphic applications and 3D games. Therefore the workload of these computers is above the average that we have seen so far: on average there are 61 processes and 722 threads running. Similarly to what we have seen for university workloads, these computers largely show no presence of parallelism (85% of the time). This demonstrates again that CMP/SMT processors are able to handle this workload well and there is no need for a sophisticated scheduler that employs on a performance-optimizing contention management.

4 Related Work

We are not aware of another study that evaluated the parallelism characteristics of desktop workloads. Research has been more focused on the area of database or web server workloads. For example Lo et al [5] examine database performance on SMT processors with regard to memory behavior. Menasce [9] performed a workload characterization for an e-commerce website to analyze the scalability of such web sites.

5 Conclusions and Future Work

We have demonstrated that a significant number of the analyzed workloads show only low parallelism for both superscalar and CMP/SMT systems. As a result of this, we conclude that there is not too much benefit in implementing advanced scheduling algorithms that further optimize performance. Instead, desktop workloads would benefit more from techniques such as runtime parallelization and speculative multithreading. There is also considerable leeway for implementing scheduling algorithms that optimize power consumption management and load-balancing of the core workloads. Even though our results do not have a statistical significance due to the small number of computers sampled, we have shown that there is a particular trend that is worth further research. This trend indicates that most current desktop workloads do not fully benefit from multiple hardware threads.

For the future, we consider to expand the data collection to a scale that provides statistical significance. Furthermore we plan to investigate better ways for the local data collection in order to reduce the data collection overhead.

6 Acknowledgements

We want to thank for the work and cooperation provided by the department of Computing Science at Simon Fraser University. Special thanks to Lee Greenough for allowing this project to go ahead and Ching Tai Wong for configuring and troubleshooting the data collection on the university computers. We also want to thank Michael Letourneau for providing guidelines for the data analysis. Finally, we want to thank to the people who volunteered for the collection of data from their home computers for time and collaboration.

References

[1] Microsoft Corporation. Windows Sysinternals. *http://www.microsoft.com/technet/sysinternals/default.mspx*, 2007.

[2] Advanced Micro Devices. AMD Demonstrates Dual Core Leadership. *http://www.amd.com*, 2004.

[3] F. Cazorla, P. Knijnenburg, R. Sakellariou, E. Fernandez, A. Ramirez and M. Valero. Predictable Performance in SMT Processors. *Proceedings of the 1st Conference on Computing Frontiers*, 2004.

[4] A. Fedorova. Operating System Scheduling for Chip Multithreaded Processors. *PhD Thesis*, 2006.

[5] J. Lo, L. Barroso, S. Eggers, K. Gharachorloo, H. Levy and S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. *Proceedings of the 25th Annual International Symposium on Computer Architecture*, 1998.

[6] P. Kongetira. A 32-way Multithreaded SPARC(R) Processor. *Proceedings of the 16th Symposium On High Performance Chips (HOTCHIPS)*, 2004.

[7] T. Maruyama. SPARC64 VI: Fujitsu's Next Generation Processor. *Microprocessor Forum 2003*, 2003.

[8] C. McNairy and R. Bhatia. Montecito - the Next Product in the Itanium Processor Family. *Hot Chips*, http://hotchips.org/archive(16), 2004.

[9] D. Menasce. Workload characterization. *IEEE Internet Computing*, (7):89–92, 2003.

[10] J. Nakajima and V. Pallipadi. Enhancements for Hyper-Threading Technology in the Operating System - Seeking the Optimal Scheduling. *Proceedings of the Second Workshop on Industrial Experiences with Systems and Software*, 2002.

[11] P. Otellini. Intel Keynote. *Intel Developer Forum*, 2004.

[12] R. Kalla, B. Sinharoy and J. Tendler. IBM POWER5 chip: a dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, 2004.

[13] S. Kim, D. Chandra and Y. Solihin. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2004.

[14] S. Siddha, V. Pallipadi and A. Mallick. Chip Multi Processing aware Linux Kernel Scheduler. *Proceedings of the Linux Symposium*, pages 337–347, 2005.

[15] A. Snavely and D. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreaded Processor. *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[16] T. Moseley, J. Kihm, D. Connors and D. Grunwald. Methods for Modeling Resource Contention on Simultaneous Multithreading Processors. *Proceedings of the International Conference on Computer Design*, 2005.

[17] R. Thekkath and S. Eggers. Impact of Sharing-Based Thread Placement on Multithreaded Architectures. *Proceedings of the 22nd Annual International Symposium On Computer Architecture (ISCA)*, 2004.