Thread level parallelism of desktop applications

University of Michigan
1301 Beal Ave.
Ann Arbor, MI
48109-2122
+1-734-764-0203

Intel Microprocessor Research Lab
5350 NE Elam Young Parkway
Hillsboro, OR
97123
+1-503-696-3154

Krisztián Flautner   manowar@engin.umich.edu
Rich Uhlig   richard.a.uhlig@intel.com
Steve Reinhardt   stever@eecs.umich.edu
Trevor Mudge   tnm@eecs.umich.edu

Abstract

Multiprocessing is already prevalent in servers, where multiple clients present an obvious source of thread level parallelism. The case for multiprocessing is less clear for desktop applications, however processor architects are already designing processors that count on the availability of concurrently runnable threads. In this paper we examine a wide variety of existing desktop workloads (over 50) across three different operating systems (Windows NT, BeOS and Linux) and quantify the amount and nature of thread level parallelism (TLP) in the system. Our results show that OS and application structure have a significant effect on TLP. While most of the workloads exhibit only moderate amounts of parallelism (under 1.5), there is evidence that show that many of these workloads are not inherently single threaded.

1. Introduction

Processor architects are increasingly looking at support for multithreaded workloads. A new generation of processors such as the IBM Power4 [3], Compaq EV8 [4] and Sun MAJC [5] are using simultaneous multi-threading [1] or single chip multiprocessing [2] to achieve high performance. While multiprocessing is already prevalent in servers, where multiple clients present an obvious source of parallelism, the case for multiprocessing is not as clear for desktop applications. Moreover, desktop applications are difficult to simulate on architectural simulators due to user interactions and the graphical user interface. For these reasons, researchers have often resorted to concurrently running multiple independent workloads to simulate these workloads. While this methodology is sufficient for certain studies, it has little correlation to real usage models.

Our goal in this paper is to gain insight into the amount and nature of thread level parallelism in a wide variety of existing desktop applications and to speculate on what can be expected in the near future. We are not only interested in the number of threads that are spawned during the run of a benchmark, but also whether these threads actually run concurrently. To this end, we have developed a methodology (described in Section 2.1) for measuring thread level parallelism (TLP) that is portable across operating systems and have measured a large number of desktop applications (over 50) under Windows NT, Linux and BeOS running on a multiprocessor machine. Our data collection methodology hooks into the kernel at a very low level, which allows us to see the activities of all threads in the system and takes all the synchronization and scheduling overhead into account. The collected data can also be used to estimate the TLP of machine configurations with fewer processors than the one on which data has been collected (Section 2.2). Some of the questions that we seek to answer are the following:

- To what degree are existing applications threaded?
- How much does the OS contribute to TLP?
- How do modern run-time environments affect TLP?

The chosen operating systems allow us to look at a large number and variety of applications. Windows NT (Section 4.1) gave us a platform that runs many of the existing Windows applications on multiprocessing hardware (Windows 98, currently a more prevalent desktop OS does not support multiprocessing) as well as a platform for looking at Java applications. We are interested in how the language-level concurrency support of Java applications and their modern run-time environments influence TLP. BeOS (Section 4.2) is a heavily threaded modern operating system, intended for desktop users. Since the operating system has supported SMP hardware from its early days and subsequently its applications have been written with multiprocessing in mind, we looked at it as an indicator of what can be reasonably expected in desktop applications in the near future. Unlike in the previous operating systems, the use of threads is not as pervasive in Linux (Section 4.3). While the operating system supports the use of kernel threads (which are just regular processes) through the pthreads [6] interface, most applications are not multithreaded.

Thread level parallelism of desktop applications   December 23, 1999   1 of 9

2. Methodology

2.1 Trace collection

Our goals for the measurement process were to come up with a portable technique that is minimally intrusive, works with any unmodified workload, and whose results can be processed off-line to generate interesting metrics and simulate different machine configurations. Instead of using a small number of automated workloads (where the run of the benchmark has been made to be reproducible), we wanted to examine a large cross-section of applications under realistic usage conditions.

The core of our methodology relies on invoking a measurement hook every time a thread starts executing on any of the processors on a multiprocessor system. Our benchmark machine is a four-way multiprocessor, with 500Mhz Pentium III processors and 512MB of main memory. When the hook is invoked, our function records a timestamp, the CPU number on which the switch occurred and the thread ID of the thread that was swapped in. This information is added to a trace and is saved to disk at the end of the observation period. The trace can then be processed off-line to derive TLP information accurately, since the timestamp counters on all processors are synchronized when the machine is booted. To make our traces more complete, we have added information about thread and process creation and deletion, process names and thread-process relationships (i.e., which threads belong to which processes).

The Windows NT implementation uses callbacks to a device driver to record all aforementioned information, while a user-level process controls the trace acquisition. Our benchmarks were run under Windows NT 4 Server (SP5) operating system, since Windows NT Workstation only supports a machine with up to 2 processors. One of the differences between the two operating systems is the time quantum that is given to foreground and background applications. We collected data with different time quantum settings, however it does not have a significant bearing on the TLP numbers.

On all platforms the driver has very low overhead, since it uses no locks and does very little computation. Trace events belonging to each CPU are saved in separate buffers so that the driver can be invoked from each CPU concurrently without using locks. The Linux (kernel 2.2.3) implementation differs from the one for Windows NT in that instead of writing a device driver, we inserted the required calls into the kernel directly and added two system calls to control the measurement process.

While we use the same post-processor on the BeOS as on the other operating systems, trace collection is done using an existing facility for recording scheduler events. While this approach has certain limitations, we could not find any way of directly hooking the desired thread events. The downside of using the scheduler event log is threefold: not every piece of information is preserved in the trace, it only works with at most a dual processor machine and the size of the collected trace is limited. To figure out process names, and thread-process relationships, we implemented a user-level program that periodically wakes up and records this information. To avoid disturbing the results, we do not collect thread creation and destruction times and only look at the state of threads in processes that are significant for the workload. This monitor process inflates our TLP numbers by at most 0.02. Using two processors instead of four is not a great limitation, since most desktop applications exhibit a TLP of less than 2. The size of the collected trace is limited to 240K, due to a built-in buffer size.

2.2 Metrics

The data collected for each workload is summarized by specifying the total execution time, machine utilization (MU), thread level parallelism (TLP) numbers and information about the threads and processes that were active during the run of the benchmark.

The TLP data is summarized in four fields. The first field shows what percentage of total execution time (c_i) that exactly $i = 0, ..., n$ (the number of CPUs in the machine) threads are executed concurrently. Correspondingly, c_0 specifies the amount of idle time experienced by the benchmark. The second field shows the measured amount of thread-level parallelism of the benchmark using four processors and the expected amount of TLP on a dual-processor machine. The third and fourth fields describe what percentage of the concurrency comes from within a single process or multiple processes. When possible, processes are classified into two categories: application or system processes, depending on whether they are spawned by the running application or the operating system.

Whenever possible, we show data about the number of active threads and processes in the machine as well as their lifetimes. A thread or process is active if it was scheduled to run at least once during the run of the benchmark. The thread and process lifetime is given as a percentage of the runtime of the benchmark. Thread affinity is the measure of how often a thread is scheduled to run on the same processor as the last time.

TABLE 1. **Concurrency on a dual processor vs. a quad-processor machine**

Benchmark	Expected 2 processors	Actual	
		2 processors	4 processors
Elastic Reality - Sys98	1.07	1.07	1.07
Photoshop - Sys98	1.15	1.12	1.23
PowerPoint - Sys98	1.07	1.07	1.07
SoundForge - Wins99	1.12	1.12	1.14
Visual C++ - Wins99/MP	1.75	1.74	1.98

The data in the first parallelism field is used to derive machine utilization and TLP. Machine utilization is a measure of how well the processing resources are taken advantage of during the execution of the benchmark. Equation 1 shows how machine utilization is derived from the collected data. Machine utilization is 1 if all processors in the machine were fully utilized.

$$MU = \frac{\sum_{i=1}^{n} c_i i}{n} \qquad (EQ\ 1)$$

The problem with this metric is that if the benchmark incurs a significant amount of idle time (due to I/O activity, the user interface or periodic activity), its value can be misleading. For this reason, we have defined *TLP* (Equation 2) as the number of concurrent threads that are executing on the machine if at least one non-idle thread is executing.

$$TLP_n = \frac{\sum_{i=1}^{n} c_i i}{1 - c_0} \qquad (EQ\ 2)$$

*TLP* is between 1 and *n* (the number of processors in the machine). It is important to recognize that the par-allelism metric does not necessarily mean that given *n* processors, the workload will get done *TLP* times faster, it only states that the non-idle (processor bound) portions of the program will benefit from the specified speedup. We would have liked to separate idle time due to I/O, however we have been unable to find a portable and reliable way of doing this.

Knowing how the concurrency of an application would scale up to more processors requires information about thread synchronization, however our intuition was that scaling down is a simpler problem. If four threads could execute concurrently on a quad processor machine, then the same threads would also fully utilize a machine with fewer number of processors, but would execute for a longer time. We make the conservative assumption that there is a join point between parallel episodes of different cardinality. This means that when scaling down the workload, parallel episodes with higher TLPs cannot be overlaid with less parallel ones; they must complete before moving on to the next episode. Figure 1 illustrates a hypothetical example where data from a quad-processor is scaled to a dual-processor machine. Note that episodes 4 and 5 have an increased run-time on the simulated machine due to the above stated assumptions.

Equation 3 is an elaboration of Equation 2 and can be used to estimate the expected TLP of the measured benchmark on computers with fewer processors than on the measurement machine. In the equation *k* specifies the reduced number of processors and *n* is the number of processors for which the data has been collected ($n > 1$ and $1 <= k <= n$).

$$TLP_k = \frac{\sum_{i=1}^{n} c_i i}{\sum_{i=1}^{k} c_i + \frac{1}{k} \sum_{i=k+1}^{n} c_i i} \qquad (EQ\ 3)$$

FIGURE 1. **Scaling TLP results to a dual processor machine**
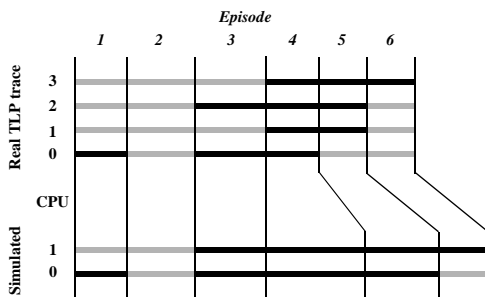
To check the validity of our assumptions, we reran the subset of our benchmarks that were completely automated on the same machine with two processors turned off. Table 1 on page 3 shows the measured values along with the expected values predicted by Equation 3. As can be seen, this simple formula can be successfully used to predict the TLP of the applications on a dual processor machine. The values for PhotoShop show the greatest difference between predicted and actual values. In this benchmark there is a significant reduction of dead-time (c_0), which causes the overall proportion of c_1 to increase and c_2 to decrease.

3. Benchmarks

We chose Windows NT Server 4.0 (SP5) as our primary platform for these investigations, since it runs a large variety of applications, has robust support for threading and can run on multiprocessor hardware. The Sysmark 98 [10] and Winstone 99 [11] benchmark sets provided a large number of applications for our initial observations. The advantage of these benchmark suites is that they consist of commercial applications that are driven by a GUI automation tool to achieve life-like behaviour. The reproducibility of the runs of these benchmarks has allowed us to fine-tune our methodology and to make comparisons between runs on different computers. However, running only fully-automated benchmarks would have obscured the fact that many of the applications incur a lot of idle-time when running in interactive mode.

Java applications were run on Windows NT using the Java 1.3beta SDK with the HotSpot Client VM. The BeOS applications were run on BeOS 4.5.2 and we used the RedHat 6.0 distribution with our modified 2.2.3 kernel for Linux benchmarks.

4. Results

4.1 Windows NT

Most Windows NT desktop applications exhibit only moderate amounts of TLP (see Table 2 on page 5). The highest numbers come from multimedia applications (both authoring and playback), program development tools as well as from scientific and engineering applications. The only applications that achieve significantly more TLP than 1.3 are applications that are either hand parallelized (Parallel MPEG player) or that have "obvious" areas of parallelism such as Visual C++, where projects are compiled concurrently. However, even this application misses out on some opportunity for concurrency, since files are not compiled in parallel,

only projects. When compiling a single project the TLP is 1.18, compiling two projects it is 1.98. The gnu make and gcc based compilation environment on the other hand achieves a concurrency of 3.27 when compiling a single project.

In their study of the characteristics of desktop applications [12], Lee et al. observed that most of the instructions are executed from a single dominant thread. The TLP numbers in our study confirm this observation, however it is worth noting that results can vary widely based on the workload. For example when running Adobe Photoshop, we observed that our TLP numbers varied from 1.23 to 2.36 implying that a single thread did not always dominate during execution. However, Lee et al. reported that during their run 97.16% of instructions came from the primary thread. While the two metrics are not directly comparable, it is unlikely that one could observe an average of two concurrent threads during execution, while executing nearly all instructions from only one of them.

The amount of concurrency can fluctuate greatly when running the same application with different workloads. An example of this is Photoshop, which has a concurrency of 1.23 when running as part of Sysmark 98, and 2.36 under the Winstone 99/MP workload.

Java applications do not seem to exhibit significantly more TLP than their traditional counterparts. It is true that they use more threads than other applications on average, however these threads tend not to run concurrently. The pervasive use of threads in Java does translate into a large percentage of the concurrency coming from within the application process (50% to 80% in our benchmark). This number includes both threads from the Java application and the Java VM.

The database workloads in our measurements do not show a significant amount of concurrency as long as there is only a single client accessing the data. However, using two clients on the Cloudscape database, the TLP jumps to 1.59. This usage scenario could become realistic on the desktop as loosely coupled object based environments (such as Jini and JavaSpaces) become more prevalent.

The video playback applications in our benchmark suite all exhibit different kinds of concurrency characteristics. The Parallel MPEG player has been hand parallelized to play back an HDTV video stream. The existing hardware is barely enough to accomplish this task (TLP is 3.76). The other playback applications have not been parallelized to nearly the same degree. Quick-Time uses multiple threads to decode and play back the multiple streams of multimedia information in a movie:

TABLE 2.   Windows NT benchmark results

Category	Application	Workload	MU	Thread Level Parallelism (TLP)											
				Threads					TLP_4	TLP_2	Intra process		Inter process		
				c_0	c_1	c_2	c_3	c_4			App	Sys	App-App	App-Sys	Sys-Sys
Database	Cloudscape - Java[a]	embedded	18.23%	29.7%	68.2%	1.7%	0.3%	0.1%	1.04	1.03	22.14%	0.19%	46.59%	30.37%	0.71%
		rmijdbc 1c	24.39%	11.8%	81.0%	5.4%	1.4%	0.4%	1.11	1.09	12.41%	0.34%	47.32%	38.20%	1.72%
		rmijdbc 2c	36.96%	7.1%	45.5%	41.4%	4.2%	1.7%	1.59	1.53	15.72%	0.13%	48.65%	33.80%	1.70%
		rmijdbc 4c	51.53%	8.6%	39.0%	13.2%	16.1%	23.1%	2.26	1.68	19.19%	0.12%	55.65%	24.48%	0.56%
	Paradox 8.0	Sys98	21.19%	19.2%	77.3%	3.1%	0.3%	0.1%	1.05	1.05	16.60%	0.39%	27.98%	53.58%	1.45%
Design	Bryce 2	Sys98	24.78%	1.9%	97.2%	0.9%	0%	0%	1.01	1.01	0%	0%	63.34%	36.66%	0%
	CorelDRAW 8.0	Sys98	24.65%	4.4%	92.7%	2.7%	0.1%	0%	1.03	1.03	0.01%	0%	14.22%	85.75%	0.02%
	Elastic Reality 3.1	Sys98	24.88%	7.3%	86.3%	6.0%	0.3%	0.1%	1.07	1.07	0%	0.23%	32.35%	67.22%	0.19%
	Extreme 3D	Sys98	24.32%	4.6%	93.7%	1.6%	0.1%	0%	1.02	1.02	0%	0.01%	32.05%	67.79%	0.16%
	Image/J - Java		18.13%	42.2%	44.4%	12.4%	0.8%	0.2%	1.25	1.24	63.67%	0.02%	6.66%	29.48%	0.17%
	Photoshop 4.0.1	Sys98	23.3%	24.2%	68.4%	2.0%	0.9%	4.5%	1.23	1.15	1.14%	0.01%	2.17%	96.67%	0.02%
		Win99[b]	21.87%	52.5%	31.2%	3.6%	1.6%	11.0%	1.84	1.47	28.16%	0.29%	35.64%	31.16%	4.76%
		Win99/MP	34.59%	41.5%	28.5%	4.0%	2.1%	23.9%	2.36	1.66	49.67%	0.33%	28.12%	21.81%	0.07%
Dev.	Visual C++ 5.0	Win99	24.69%	16.0%	72.4%	9.3%	1.4%	0.9%	1.18	1.15	0.18%	0.73%	31.40%	67.51%	0.17%
		Win99/MP	44.28%	10.4%	24.9%	46.8%	13.3%	4.7%	1.98	1.75	0.16%	1.42%	49.18%	49.00%	0.24%
	GNU make / g++[c]	gzip	36.10%	55.8%	8.8%	2.1%	1.5%	31.7%	3.27	1.88	0.07%	0.22%	62.66%	36.85%	0.20%
G	Quake 2		24.22%	5.3%	92.8%	1.8%	0.2%	0.0%	1.02	1.02	1.92%	0.02%	15.26%	82.40%	0.40%
Internet	Frontpage 98	Win99	9.32%	70.0%	25.0%	3.4%	0.9%	0.7%	1.24	1.20	0.59%	0.85%	55.87%	42.63%	0.05%
	HotJava - Java	Browsing	6.44%	78.0%	19.1%	2.3%	0.5%	0.2%	1.17	1.15	75.63%	0.14%	5.41%	18.23%	0.59%
	Internet Explorer 5	Misc	3.27%	89.5%	8.5%	1.5%	0.3%	0.2%	1.25	1.21	23.02%	1.03%	31.30%	44.16%	0.49%
	Netscape 4.05	Sys98	24.71%	17.8%	67.6%	12.8%	1.7%	0.1%	1.20	1.19	17.25%	0.18%	19.15%	62.78%	0.65%
Authoring	Premiere 4.2	Sys98	24.93%	5.8%	89.2%	4.7%	0.3%	0.1%	1.06	1.06	65.39%	0.01%	12.89%	21.71%	0.00%
		Win99	25.69%	13.0%	72.0%	14.2%	0.7%	0.1%	1.18	1.18	0.06%	0.03%	19.38%	80.53%	0%
	SoundForge 4.0	Win99	20.80%	26.9%	64.4%	7.5%	1.0%	0.1%	1.14	1.13	0%	6.56%	14.73%	78.19%	0.52%
	MPEG Encoder	Sys98	22.27%	15.9%	79.7%	4.0%	0.3%	0.1%	1.06	1.06	13.09%	0.02%	45.19%	41.67%	0.04%
Multimedia playback	Parallel MPEG	HDTV	94.02%	0.0%	4.4%	2.4%	5.8%	87.4%	3.76	1.98	97.94%	0.03%	0.93%	1.07%	0.03%
	QuickTime 4.0.3	MP3	3.78%	85.9%	13.1%	0.9%	0.1%	0.0%	1.07	1.07	17.41%	0.32%	24.87%	57.10%	0.30%
		QT	16.39%	44.4%	46.2%	8.9%	0.5%	0.0%	1.18	1.17	77.91%	0.02%	9.56%	12.46%	0.05%
		QT - 2[d]	25.03%	18.5%	64.9%	14.8%	1.7%	0.2%	1.23	1.21	43.26%	0.01%	29.27%	27.47%	0%
	RealJukebox	MP3	3.64%	87.9%	10.2%	1.6%	0.3%	0.1%	1.20	1.18	5.14%	0.29%	32.48%	61.97%	0.12%
	Windows Media	AVI-local	6.43%	76.9%	21.0%	1.8%	0.3%	0.1%	1.11	1.10	53.76%	0.08%	26.69%	19.44%	0.03%
	Player	AVI-net	5.93%	79.6%	17.7%	2.0%	0.5%	0.1%	1.17	1.15	20.28%	0.64%	22.46%	56.36%	0.25%
		AVI-crypt	7.56%	73.5%	23.2%	2.9%	0.4%	0.0%	1.14	1.13	36.70%	1.96%	27.76%	33.42%	0.16%
		MP3	1.97%	92.8%	6.6%	0.5%	0.1%	0.0%	1.09	1.09	18.15%	0.37%	48.85%	32.14%	0.49%
Lang	NatSpeaking 2.02	Sys98	24.34%	5.0%	93.0%	1.7%	0.3%	0.0%	1.02	1.02	33.49%	0.01%	29.91%	36.57%	0.02%
	PowerTranslator		17.87%	29.4%	69.9%	0.5%	0.1%	0.0%	1.01	1.01	0%	0%	70.54%	29.46%	0%
Productivity	Excel 97	Sys98	25.97%	5.5%	85.6%	8.5%	0.4%	0.0%	1.10	1.10	2.57%	0.03%	4.43%	92.70%	0.27%
	Lotus SmartSuite	Win99	14.24%	48.6%	46.8%	3.8%	0.7%	0.1%	1.11	1.10	2.28%	0.68%	8.37%	88.29%	0.39%
	Microsoft Office[e]	Win99	23.49%	22.4%	63.0%	13.1%	1.3%	0.2%	1.21	1.20	2.14%	0.28%	74.33%	22.86%	0.39%
	PowerPoint 97	Sys98	24.85%	7.2%	86.6%	5.6%	0.4%	0.0%	1.07	1.07	2.83%	0.34%	10.45%	82.89%	3.49%
	Word 97	Sys98	26.42%	4.8%	85.2%	9.7%	0.3%	0.0%	1.11	1.11	0.16%	0.48%	21.40%	77.20%	0.76%
	OmniPage Pro 8.0	Sys98	22.29%	14.9%	81.6%	3.0%	0.4%	0.1%	1.05	1.04	0.01%	0%	67.22%	32.75%	0.02%
Sci/Eng	AVS Express 3.4	Win99	26.03%	4.7%	86.7%	8.4%	0.2%	0.0%	1.09	1.09	0.02%	0.01%	27.54%	72.4%	0.03%
	JSV - Java[f]		25.37%	7.8%	84.9%	5.9%	1.0%	0.4%	1.10	1.09	56.24%	0.04%	7.54%	36.16%	0.03%
	MicroStation SE	Win99	23.57%	17.7%	70.9%	10.9%	0.4%	0.0%	1.15	1.14	9.52%	0.13%	26.41%	63.86%	0.08%
		Win99/MP	27.17%	18.4%	55.0%	26.2%	0.4%	0.1%	1.33	1.33	71.85%	0.09%	2.05%	25.97%	0.03%

a.   An object database system. Modes: embedded (server and client in single program) and client-server (rmijdbc) with 1, 2 and 4 clients.

b.   This benchmark did not complete, due to a bug in the Winstone 99 benchmark driver.

c.   make -j 8 run on the gzip sources under cygwin (sourceware.cygnus.com/cygwin).

d.   Playing back two QuickTime movie streams concurrently (Sorensen compression, 640x288, millions of color).

e.   The data presented here is for the middle portion of the run, since the entire trace was too big for the post-processor.

f.   Scientific Visualization. Displaying and rotating various molecules.

TABLE 3.   System processes contributing to inter application TLP[a]

Benchmark	TLP$_4$	App-Sys TLP	Thread from operating system process		
			1.	2.	3.
Internet Explorer 5	1.25	44.16%	System (85.55%)	UI (6.78%)	Win32 (5.39%)
JSV - Java	1.10	36.16%	System (83.74%)	Win32 (13.92%)	UI (1.06%)
Microstation SE	1.15	63.86%	System (71.86%)	Win32 (26.43%)	Services (1.33%)
Netscape 4.05	1.20	62.78%	System (84.30%)	Win32 (10.74%)	UI (3.99%)
PhotoShop 4.0.1	1.23	96.67%	System (98.5%)	Win32 (1.37%)	UI (0.10%)
PowerPoint 97	1.07	82.89%	System (75.64%)	Spooler (11.47%)	Win32 (6.37%)
RealJukebox - MP3	1.20	61.97%	System (77.19%)	Win32 (11.97%)	Services (8.67%)
Visual C++ 5.0	1.18	67.51%	System (84.48%)	Win32 (14.90%)	Services (0.24%)

a. The System process contains kernel and device driver threads, UI stands for Explorer.exe, which is responsible for the Windows NT user interface, Win32 is responsible for handling the Win32 API calls, Services is the service controller (starts, stops and controls service daemons) and Spooler (spoolss.exe) is the spooler service daemon.

it spawns a thread for audio and one for video (depending on the movie, there could be more or less threads). However, neither one of these threads is very resource intensive and the audio decoder has about 20% of the processing requirements of the video thread (based on its 1.18 TLP, 78% of which is intra application concurrency). The machine idles 44% of the time during playback, since once it reaches the desired service quality (the highest in our case), it has to ensure that frames are presented at a uniform rate. The Windows Media Player has similar characteristics; however, it was only able to play our AVI movie at a low framerate. Displaying two video streams concurrently estimates the video workload of a high end video conferencing software. The measured TLP of two high-quality QuickTime movie streams increases to 1.23 and includes a significant reduction of idle time (18.5% of total execution time).

When running interactive workloads under realistic conditions, the system is idle a majority of the time. This fact is obscured by the automated benchmarks; a case in point is the comparison of Netscape from the Sysmark 98 benchmark suite to Internet Explorer and HotJava, both of which were run by hand. While the Netscape benchmark spends only 17.8% of its execution time idling, the other two web browsers wait for something to do about 80% of the time.

VMware allows one to run operating systems inside a virtual machine. To accomplish this task, it has to emulate the behaviour of the entire machine, including all the necessary devices. In our measurements, the VM was a significant source of TLP when device emulation was used heavily, however under most conditions its contribution to TLP was not significant. Moreover since it only emulates a uniprocessor, applications running within the VM are also limited to uniprocessor performance, even if they are multithreaded.

A negligible fraction of concurrency comes from running operating system threads (from one or more processes) in parallel. However, application and system threads running at the same time make up a large portion of overall TLP. The greatest single contribution comes from running an application thread in parallel with a thread from the System process, which contains the device driver and kernel threads. Table 3 shows the operating system processes that contribute the most to interprocess TLP between application and system threads. As can be seen from the table, the operating system threads of Windows NT contribute significantly to the concurrency of the platform.

4.2   BeOS

Given the BeOS' reputation as an operating system designed with multiprocessing and multithreading in mind, we expected it to yield higher TLP numbers than Windows NT. Overall this expectation turned out to be justified but the range and sophistication of the applications we measured under the BeOS is not nearly as great as under Windows NT. The results are shown in Table 4 on page 7.

Two of the measured applications have exact counterparts under Windows: Abuse 2.0 and Quake2. Quake was a straight port to the BeOS, and any increase in concurrency is due to the OS: the TLP improves from 1.02 to 1.08, not a significant change.

The difference is greater in Abuse. Abuse under Windows runs as a DOS application (unfortunately it crashes under Windows NT), which is effectively sin-

TABLE 4.   BeOS benchmark results

Application	Workload	Total time (s)	Utilization MU	Threads			TLP$_2$	Process		Active	Created	Affinity
				c_0	c_1	c_2		Intra	Inter			
Media-Player	AVI	11.94	26.99%	50.5%	45.0%	4.5%	1.09	19.6%	80.4%	44	0	81.22%
	Quick-Time	11.26	9.70%	82.7%	15.1%	2.1%	1.12	13.4%	86.6%	55	0	87.82%
3dmov	Pulse -QT	15.06	40.84%	28.7%	60.9%	10.4%	1.15	5.1%	94.9%	44	0	78.27%
Abuse 2.0	Normal	3.71	16.54%	73.2%	20.6%	6.2%	1.23	25.7%	74.3%	53	0	84.76%
	Fast	4.74	73.38%	1.4%	50.4%	48.2%	1.49	35.4%	64.6%	57	0	71.36%
Axia 1.01	Normal	2.89	37.78%	30.5%	63.5%	6.0%	1.09	47.1%	52.9%	32	0	87.41%
	Fast	3.2	62.63%	1.8%	71.1%	27.1%	1.28	50.9%	49%	41	0	90.38%
Quake2	Demo	29.79	53.99%	0.2%	91.5%	8.2%	1.08	3.7%	96.3%	52	0	96.34%
Life	2 threads	14.48	81.51%	15.9%	5.2%	78.9%	1.94	55.2%	44.8%	44	6	56.4%
	8 threads	8.8	70.56%	25.8%	7.4%	66.9%	1.90	70.9%	29.1%	49	12	47.48%
Flight		10.6	55.04%	0.0%	89.9%	10.1%	1.10	8.7%	91.3%	42	0	70.04%
Charts - space	LoFi	12.04	6.61%	90.4%	5.9%	3.6%	1.38	36.1%	63.9%	42	0	93.4%
	HiFi	15.08	96.89%	0.0%	6.2%	93.8%	1.94	46.2%	53.8%	43	0	78.87%
NetPositive	Apple	3.05	24.92%	65.2%	19.7%	15.1%	1.43	27.6%	72.4%	100	51	76.13%
	Be	3.82	21.85%	67.9%	20.5%	11.6%	1.36	24.1%	75.9%	140	86	72.02%
	Wired	2.21	23.57%	65.1%	22.7%	12.2%	1.35	21.6%	78.4%	73	31	84.13%

gle-threaded. However, under BeOS in "normal" mode, the TLP is 1.23. This increase in concurrency is due to the improvements made during the porting process and shows that this workload is not inherently single threaded. Abuse also provides a "fast" mode in which the game attempts to use as much of the available resources as possible without limiting the speed of the action to playable levels. The fast mode measurement shows even greater concurrency of 1.49. These TLP numbers might be achievable on a similar game by making the graphics and game logic more sophisticated.

The MediaPlayer benchmarks look very similar to their Windows NT counterparts: the concurrency is low and there is a lot of time when nothing is executing on the machine. This is due to the fact that playback of these streams is tied to a given service quality (frame rate and sound quality) and if they are met, there is simply nothing more to be done. Somewhat of a surprise was the 3dmov benchmark, which includes a 3D rendered moving pulse, onto which a QuickTime movie is projected. While there was a significant decrease in dead time compared to movie playback by itself, the application does not show much increase in concurrency.

The web browsing benchmarks on the other hand exhibit a significant amount of TLP, even when display-ing simple web pages. The displayed web pages do not contain Java applets but simply graphics and text. Compared to similar benchmarks under Windows NT, the BeOS comes out about 0.1 TLP ahead even in the worst case. This is a significant increase and is due to a different rendering approach on the BeOS; every object on the web page is rendered in parallel, as data from the network becomes available. This contrasts with the approach with the other web browsers under Windows NT that do not spawn new threads for objects within a page.

It is not as easy to classify the concurrency of BeOS applications by application and system threads as on Windows NT. The reason is that almost all applications register threads in other processes for handling events and the user interface. As a result, concurrency that would show up as intra process concurrency in NT shows up as inter process concurrency on the BeOS.

4.3 Linux

Linux is not a threaded operating system in the same sense as the Windows NT and BeOS. In Linux, the basic unit of execution is a process, however lightweight processes can be created using the clone system call with small overhead, and those can be used as threads.

TABLE 5.   Linux benchmark results

Application	Workload	Total time (s)	MU	Thread Level Parallelism (TLP)						
				Threads					TLP_4	TLP_2
				c_0	c_1	c_2	c_3	c_4		
Gimp	Cubism	24.06	25.27%	0%	99%	1%	0%	0%	1.01	1.01
	Predator	4.54	21.64%	23.5%	68.2%	6.7%	1.4%	0.2%	1.13	1.12
	Weave	15.7	27.04%	2.9%	87.9%	7.6%	1.4%	0.3%	1.11	1.10
Battalion		44.83	25.26%	0%	99%	1%	0%	0%	1.01	1.01
Xanim	AVI	22.6	0.33%	98.7%	1.3%	0%	0%	0%	1.00	1.00
Mozilla M10		29.09	13.19%	49.9%	47.5%	2.6%	0%	0%	1.05	1.05
Make / gcc	kernel src	69.11	82.18%	4.5%	13.2%	3.3%	7.1%	71.9%	3.44	1.92
Netscape 4.5	Apple	20.01	2.4%	91.5%	7.5%	1.1%	0%	0%	1.13	1.13
	Intel	20.01	1.9%	92.9%	6.5%	0.5%	0%	0%	1.08	1.08
	Wired	20.01	4.19%	84.5%	14.4%	1.2%	0%	0%	1.08	1.08
Quake2	Demo	20.01	25.06%	0.1%	99.7%	0.3%	0%	0%	1.00	1.00

However, the use of kernel threads is not yet pervasive in the operating system and its utilities. Even applications that are threaded do not always use Linux' kernel threads; usually because of unresolved issues regarding the thread safety of standard libraries. For example Netscape Communicator relies on its own user level thread library, which prevents it from taking advantage of multiple processors.

The only workload in our suite that exhibits a significant amount of TLP is the compiler benchmark, which uses parallel make to compile files concurrently. The benchmarks execute three or four threads less frequently than the other two OSs, because there is not as much opportunity to overlay execution of system and application threads.

Unlike Adobe Photoshop on Windows NT, the Gimp image manipulation utility does not exploit the thread level parallelism inherent in its algorithms but derives some concurrency from loading and running plug-ins and the user interface.

5. Conclusions

Our survey shows that most desktop applications only exhibit moderate amounts of thread level parallelism. The amount of parallelism can vary greatly between similar kinds of applications and is dependent on the operating system. Applications on the BeOS tend to exhibit higher amounts of TLP than similar applications on Windows NT. Java applications on the other hand, do not exhibit significantly higher degrees of concurrency than their Windows NT counterparts. Most desktop applications incur a large amount of idle time, which can be as much as 90% of the total execution time.

Moore's law predicts a doubling of uniprocessor performance every 18 months. In practice this implies that if there is a workload that runs with TLP of 2 on a dual processor machine today, then in a year and a half there will be a uniprocessor capable of the same feat (to a first order approximation). However, even the more parallel of the existing desktop applications exhibit a concurrency of only around 1.5, which brings the lead time of a dual processor over an equivalent performing uniprocessor to about three quarters of a year.

The amount of concurrency in most of the measured applications is less than two. This suggests that without substantial software changes to expose TLP to hardware, desktop multiprocessors with more than two processors are unwarranted in most situations. The only applications that exhibit a higher amount of concurrency have either an obvious source of concurrency (such as compiling using make -j) or ones that rely on a parallel algorithm at their core (e.g. Parallel MPEG player). While most of the results were collected on a quad processor machine, the expected TLP_2 number shows that in most cases a dual processor machine would exhibit similar amounts of concurrency.

The availability of idle time presents an opportunity to provide a richer software layer for applications. Dynamic profile-feedback based optimizations could be performed during the available time to improve the efficiency of the software.

It is clear from the survey of existing applications that most of the programs use threads as a convenient abstraction for the programmer, not as way to maximize performance. This is partly due to the current prevalence of uniprocessor systems — optimizing for the uncommon multiprocessor machine is not beneficial — and also because programmer time is expensive. Unless

writing an application that takes advantage of the parallel hardware is on the same order of difficulty as writing the program in the straight-forward way, it is unlikely that we will see a great increase in parallelism. If increased concurrency does not come from the applications, it could come from the platform and the user interface.

The use of soft devices (mostly software implementation of hardware devices, such as modems) could be significant source of concurrency on multiprocessors. The nature of these applications is to put a constant load on the machine to provide the required signal processing. These types of devices currently include soft modems, soft DSL and DVD players. Unfortunately the current implementations that we know of do not run under Windows NT (most run under Windows 98, which is not a multiprocessor capable OS), so we could not directly measure their contribution. Also, the current implementations usually rely on DPCs (delayed procedure calls) to perform work, which does not show up as thread level concurrency. However, based on data from Rockwell [9], a soft modem requires about 50MIPS of processing, while a soft DSL would require about 500MIPS. This means that on the measurement hardware, the presence of a soft modem would add approximately 0.1 and the soft DSL about 1 to the concurrency numbers. A theoretical client for video-on-demand services (soft DSL with video playback) could operate with an average concurrency of about 2.

The voice recognition and understanding benchmark are both single-threaded CPU intensive workloads. Even if the applications themselves are not parallelized, they would significantly contribute to the concurrency of the platform if used as part of the user interface. It would mean at least one additional CPU bound thread in addition to anything the user is doing on the machine.

In this paper we have focused on the overall thread level parallelism of our workloads. However, our early results indicate that multiprocessing can have a significant impact on the user perceived response time of interactive applications, which is the focus of our ongoing research.

6. Acknowledgements

References

[1] D. M. Tullsen, S. J. Eggers, H. M. Levy. Simultaneous Multithreading: Maximizing On-chip Parallelism. *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 206-218, June 1995.

[2] L. Hammond, K. Olukotun. *Considerations in the Design of Hydra: a Multiprocessor-on-a-Chip Microarchitecture.* Stanford University Technical Report No. CSL-TR-98-749.

[3] 1999 Microprocessor Forum

[4] 1999 Microprocessor Forum

[5] *Microprocessor Architecture for Java Computing.* http://www.sun.com/microelectronics/MAJC

[6] B. Lewis, D. J. Berg, *Multithreaded programming with pthreads*. Sun Microsystems, 1998.

[7] Y. Endo, Z. Wang, J. B. Chen, M. I. Seltzer, "Using Latency to Evaluate Interactive System Performance" *2nd Symposium on Operating Systems Design and Implementation*, pp. 185-199, October 1996.

[8] D. A. Solomon, *Inside Windows NT Second Edition.* Microsoft Press, 1998.

[9] E. Cota-Robles, J. P. Held, "A Comparison of Windows Driver Model Latency Performance on Windows NT and Windows 98" *3nd Symposium on Operating Systems Design and Implementation,* February, 1999.

[10] http://www.bapco.com/sys98k.htm

[11] http://www.zdnet.com/zdbop/

[12] D. C. Lee, P. J. Crowley, J. Baer, T. E. Anderson, and B. N. Bershad. Characteristics of Desktop Applications on Windows NT. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.