Evolution of Thread-Level Parallelism in Desktop Applications

Geoffrey Blake, Ronald G. Dreslinski, Trevor Mudge
University of Michigan, Ann Arbor
[blakeg,rdreslin,tnm]@umich.edu

Krisztián Flautner
ARM
krisztian.flautner@arm.com

ABSTRACT

As the effective limits of frequency and instruction level parallelism have been reached, the strategy of microprocessor vendors has changed to increase the number of processing cores on a single chip each generation. The implicit expectation is that software developers will write their applications with concurrency in mind to take advantage of this sudden change in direction. In this study we analyze whether software developers for laptop/desktop machines have followed the recent hardware trends by creating software for chip multi-processing. We conduct a study of a wide range of applications on Microsoft Windows 7 and Apple's OS X Snow Leopard, measuring *Thread Level Parallelism* on a high performance workstation and a low power desktop. In addition, we explore graphics processing units (GPUs) and their impact on chip multi-processing. We compare our findings to a study done 10 years ago which concluded that a second core was sufficient to improve system responsiveness. Our results on today's machines show that, 10 years later, surprisingly 2-3 cores are more than adequate for most applications and that the GPU often remains under-utilized. However, in some application specific domains an 8 core SMT system with a 240 core GPU can be effectively utilized. Overall these studies suggest that many-core architectures are not a natural fit for current desktop/laptop applications.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Measurement

Keywords

Benchmarking, Multi-core, Thread Level Parallelism, Desktop applications

ISCA'10, June 19–23, 2010, Saint-Malo, France.

1. INTRODUCTION

Ten years ago the idea of *chip multi-processing* was still being explored primarily in research labs with projects such as the Stanford Hydra [38] and the Compaq Piranha [29]. The first commercial chip multi-processor, the POWER4 [6] was still a few years away. Meanwhile, Intel and AMD were engaged in the megahertz war as chips from these two leading vendors were approaching 1GHz and looked to continue well beyond. Consumer PCs were almost exclusively built as uni-processor systems. Multi-processor systems were a niche product relegated to high performance workstations and, of course servers. At that time Flautner et al. [33] presented a study of *"Thread Level Parallelism"* (TLP) on interactive desktop applications for modest Symmetric Multi-Processor (SMP) workstations of 2-4 CPUs. These were still not chip multi-processors as each CPU was on its own die. The original work was performed to gauge whether multi-processors provided any advantages to desktop applications. They found that two processors improved responsiveness of general interactive programs and larger systems were beneficial only for domain specific applications like video encoding.

Approximately five years ago chip manufactures began to reach the limits of frequency scaling as a result of power constraints. At that point, Intel and AMD turned to making multi-core chips for the desktop market. These were dual-core chip multi-processors [7, 8]. Since then, chip multi-processors have become the flagship products for the two leading desktop chip makers [23, 25, 17], currently using upwards of eight cores per die. In addition low power chip-multiprocessors for small sub-notebooks have been created. Examples include Intel's Atom [12] with 2-cores, Intel's Consumer Ultra Low Voltage (CULV) [28] chips with 2-cores, and Qualcomm's Snapdragon processor with 2 ARM v7 ISA based cores [27]. Chip multi-processor systems are now poised to enter the mobile phone market to run applications on smart phones, e.g., TI's OMAP4 [21]. Historically mobile phone chips had one application core and many accelerators.

Current chip multi-processing systems are more complex than the original SMP systems studied 10 years ago. They may include simultaneous multi-threading (SMT) support, cores sharing caches on chip, multiple chips in a system, and advanced GPUs that function as offload engines for highly parallel general purpose code, such as those from AMD [9] and NVIDIA [11]. Future desktop/laptop chips in advanced development now include heterogenous chip-multiprocessors, for example AMD's Llano [15] and Intel's Sandy Bridge [24] architectures that integrate programmable GPU cores onto the die with several regular general purpose

CPU cores. Manufacturers are designing these systems with increasing numbers of cores expecting application developers to write code that makes profitable use of them. This belief that wide-spread parallel software will materialize is reminiscent of the movie "Field of Dreams", where the protagonist is compelled to build a baseball diamond in the middle of a corn-field by a mysterious voice that says: "If you build it, they will come".

Now that multi-processor systems are the norm rather than the exception, we present this study to determine to what extent the growth in number of cores has been accompanied by an increase in software parallelism. We repeat the experiments of Flautner et al. [34, 33] investigating a broad range of desktop applications under two desktop/laptop operating systems: Microsoft Windows 7 and Apple's OS X Snow Leopard. In addition to the original study, we look at the impact of SMT, GPUs and the effects of moving to a low power architecture by performing the same tests on an embedded processor based desktop. One of the outcomes of our study is to provide a ten year perspective on desktop multi-processing. Our goal is to answer the following questions:

- To what degree does the overall system leverage concurrency, and how has that changed from 10 years ago?
- What impact does SMT have on parallel performance?
- How are GPU's being used to improve system performance, and do opportunities exist to further exploit them?
- How does architectural sophistication and clock frequency impact TLP?

In the original study, Flautner et al. showed that desktop applications leveraged TLP very sparingly, and the majority of gains came from improving user perceived responsiveness with one additional core. We have also chosen TLP as a metric because it indicates how efficiently we are using parallel resources when at least one core needs to do work, and indicates how many cores would be needed to fully support the parallel portions of an application. In our study we find that the number of cores that can be profitably used now approaches 3. In addition we find that SMT can be beneficial, the GPU is often under-utilized (although it has potential for providing additional parallelism), and TLP is insensitive to processor speed.

The rest of the paper is organized as follows: In Section 2 we present background from the original work and discuss other related work. Section 3 describes our measurement environment and metrics. In Section 4 we present the benchmarks used. Section 5 presents and analyzes the results for the mainstream desktop. Section 6 presents our results for our experiments on the low power desktop. In Section 7 we discuss the implications of the results and present our concluding remarks in Section 8.

2. BACKGROUND AND RELATED WORK

We base our work on the original studies done by Flautner et al. [34, 33] in early 2000. The authors investigated whether a modest SMP system of 2 to 4 desktop processors offered advantages for interactive applications over a single processor. Their work was performed on the commodity operating systems of the time: Windows NT 4.0, Linux, and BeOS. They found that developers had programmed their applications with threads, but the behavior was still primarily single threaded. Only in a few domain specific applications, such as Adobe Photoshop for image manipulation, used more than two processors. The final conclusion was that the majority of benefit came from the increased responsiveness provided by two processors. This responsiveness came from running the graphical user interface (GUI) portion of the program concurrently with the processing back end.

Other studies have been done in a similar vein. Works from Zhou et al. [48], Lee et al. [43] Endo et al. [32] and Chen et al. [31] from the mid to late 1990's all looked at profiling applications in the desktop environment. They note that the workload characteristics of this environment is very different from server or scientific workloads, where accesses to memory and disk are more random and latency is a more important metric. More recent sources for desktop application evaluation has come from non-academic sources such as enthusiast web sites like XBit Labs [30], or TomsHardware.com. These sites provide insight into how desktop applications (primarily games) perform on modern hardware, and their results reflect the results of Flautner et al. in that a two processor system is sufficient for most applications.

Work by Hauser et al. [39] looks at the programming paradigms for threads used in interactive applications. They found that threads are used primarily to structure programs into a comprehensible organization rather than for performance. This was confirmed by Flautner et al. when they observed that the GUI thread was separate from the back end of the application. Work by Giacaman et al. [37] proposes the restructuring of desktop applications using threads to gain performance from modern chip multi-processors. Work in parallelizing web browsers for desktops and mobile devices was investigated by Jones et al. [42].

Other works by Hung et al. [40], Frachtenberg et al. [36, 35], Nguyen et al. [45] look at multiprocessor applications for the desktop from an Operating System (OS) perspective. They all seek to characterize these applications in order to improve OS schedulers to provide better quality of service. For example if the OS detected a multimedia playback application is in focus, it could give it priority over a background task in order to reduce frame drops seen by the user.

3. METHODOLOGY

For this study we required a system profiling method that would allow us to observe the entire system and measure how effectively threads were being used on a modern chip multi-processor. Statistics currently provided by the OS are insufficient in characterizing the effectiveness of chip-multiprocessing. For example, machine utilization is a poor metric for interactive applications due to the large periods of idle time. Statistics about the number of threads created or the number of threads active in a system, like those presented in Table 1, illustrate that programmers do program with threads, but these statistics offer no insight into whether the observed threads execute concurrently. In the following sub-sections we present our metrics, tracing utilities and system setup.

3.1 Metrics

The principle metrics we are interested in for this study are the GPU Utilization and Thread Level Parallelism (TLP).

Benchmark	Created	Avg Live
Handbrake 0.9	22511	24
Call of Duty 4	77	44
Photoshop CS4	82	75
Adobe Reader 9	239	24
Quicktime-HD	53	52
Firefox 3.5	522	38

Table 1: Summary of number of threads created and average that were being used by a benchmark from each category of applications tested.

Hardware	Software
2009 Mac Pro	OS X 10.6.2
2x Intel Xeon E5520	Snow Leopard
6GB RAM	Dual boot
NVIDIA GTX285 -or-	Windows 7
NVIDIA GT120	Enterprise
ASUS ASRock	Windows 7
1x Intel Atom 330	Enterprise
4GB RAM	
NVIDIA ION	
Intel X-25M SSD	

Table 2: Benchmark environment system setup

GPU utilization is merely the average of GPU use over time. Thread Level Parallelism is a variation of machine utilization that factors out idle time and accurately describes usage of the parallel resources for the entire system.

Because there is a large amount of idle time in interactive applications, as illustrated in Figure 1, we use the metric Flautner et al. term as *TLP*. TLP is calculated by summing up c_i's that are the fraction of time that exactly $i = 0, ..., n$ (where n is the number of thread contexts in the machine) threads are executed concurrently. That number is then divided by fraction of non-idle time, $1 - c_0$, to get the TLP. TLP characterizes the average amount of concurrency exhibited by the program during its execution when at least one core is active. The formula for TLP is given in Equation 1.

$$TLP = \frac{\sum_{i=1}^{n} c_i i}{1 - c_0} \quad (1)$$

We have chosen TLP as our metric because it is an indication of efficiency in using the resources in a chip multiprocessor. A low TLP does not necessarily indicate that performance or responsiveness of the application is poor, it provides information about what portion of the machine is idle. Alternatively, it indicates the minimum number of processors needed to support an applications parallel workload. We seek not to measure the idle time, but how much of the system is utilized when at least one core is performing work. As more cores are added to a chip multi-processor, it is important to gauge how we are using them and TLP remains the best metric for determining this.

3.2 Trace Collection

We collect system wide traces of thread context switch activity and GPU utilization which are then processed to get the final TLP and GPU utilization statistics. We found that our collection techniques had minimal impact on our test system and therefore easily factored out of the final results. The details are presented in the following sub-sections.

3.2.1 Processor Context Switches

To profile whether threads are being used concurrently we use existing functionality in the two-targeted operating systems to track when a context switch happens on any processor. By tracing context switches we get very detailed information on how the parallel resources are used.

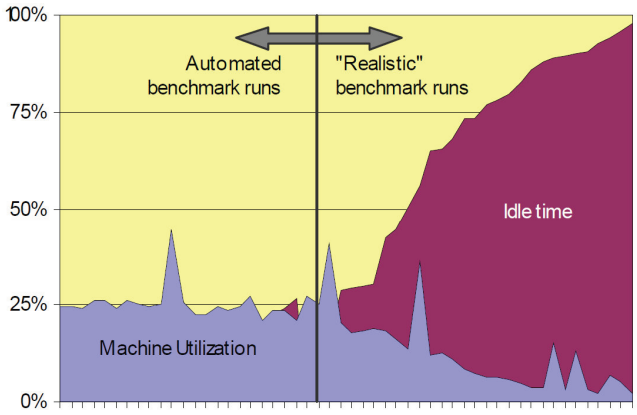

Figure 1: Figure qualitatively describing the utilization differences between automated desktop workloads and "realistic" runs performed by a user from [33]

On OS X, we use DTrace [10], which was first implemented by Sun Microsystems in the Solaris OS. DTrace works by dynamically modifying the OS binary (and even user space binaries) with hook instructions at user specified points anywhere in the kernel. This gives us immense flexibility, but presents the problem of determining where to trace. We found two functions in the OS X kernel that provided us with sufficient information to trace all context switches in the system: `timer_switch` and `thread_dispatch`. The DTrace data we gather contains a *timestamp*, *CPUID*, *ProcessID*, *ThreadID* and *executing image* name.

On Windows we use Event Tracing for Windows(ETW) [41, 46]. It is similar to DTrace, but limited in comparison. Instead of tracing function calls by modifying the kernel binary like DTrace, it records specific events exposed and reported by the OS. This is less flexible than DTrace, but more intuitive as the events provided are well documented. We traced the `CSwitch`, `Thread` and `Process` events provided by the Windows kernel using ETW. These events gave us the same information as DTrace.

3.2.2 GPU Utilization

We also trace GPU usage, as they are now very capable general purpose chip multi-processors in their own right. To accomplish this we employ two similar methods.

On OS X we use the functionality in the Apple I/O Kit [19] interface to probe performance statistics present in the GPU driver. Because we were limited to a small set of statistics, and by the interface to the GPU driver, we could only measure GPU utilization by polling. For OS X we measured GPU utilization every 10 milliseconds to minimize the overhead from calling into the driver through the I/O Kit. We then used DTrace to instrument our executable that polled the GPU driver statistics to combine it with the CPU trace so we could correlate CPU and GPU activity.

In Windows we use the NVPerfKit [13] to collect similar statistics that were available in OS X. In the NVPerfkit, we gain access to the onboard GPU counters. We used the GPU idle cycle and clock cycle counters to derive GPU utilization. As in OS X, we could only poll the GPU (the

polling frequency was again 10ms). Similarly to OS X, the binary that polled the GPU also registered with the Windows ETW subsystem. This provided an additional ETW event that we collect to enable correlation of GPU and CPU activity.

3.3 System Setup

The system setup we use is substantially different from the original work by Flautner et al. In the ten years that have passed, major advances in manufacturing technology and years of incremental improvements to the x86 architecture and GPUs have given us a system that is an order of magnitude faster than the original tested system. In 2000, a standard consumer system ran at around 1GHz and had a 3-way out-of-order processor (Intel Pentium III [4] or AMD Athlon [3]) with 256kB-512kB of last level cache. Our system runs at 2.26GHz, has four cores per chip, is 4-way out-of-order, has 8MB of last level cache and uses two chips. The amount of RAM commonly used then was about 256MB, whereas this system has 6GB. In terms of GPU, the increase is more astounding, NVIDIA's first GPU, the GeForce 256 [5] was a fixed graphics pipeline ASIC with 32MB of memory. NVIDIA's GTX285 is a fully programmable 240-core chip multi-processor with 1GB of onboard memory. We also test NVIDIA's GT120 [20] GPU which is a 32-core chip multi-processor. In ten years we have progressed from small 2-4 processor SMP machines running around 1GHz to very powerful many-core (100+ cores) heterogeneous machines.

In contrast to the very powerful desktop we tested, we also tested an Intel Atom based desktop system. We did so to gain insight into the effects on leveraged parallelism when the processor architecture looks more like the future chip-multiprocessors slated to be used in smart phones. The Atom is a 2-way superscalar in-order design that is reminiscent of the original Intel Pentium [2] processor. The Atom architecture also trends toward what academic researchers believe will represent future many-core chip-multiprocessors: a large number of simple processor tiles [18]. The Atom tested runs at 1.6Ghz, has 2 cores with 2-way SMT. The Intel Atom desktop also uses an NVIDIA ION low power GPU which has 16 fully programmable cores. In this study we were unable to report GPU numbers for the ION as it is currently unsupported by the NVPerfKit.

For software, we test Windows 7 Enterprise and OS X Snow Leopard on the Mac Pro. For the Atom machine, only Windows 7 Enterprise was available to test. We decided to use these desktop/laptop operating systems to be able to test a wide range of free and commercial programs that consumers would be likely to use. Our exact system specifications are shown in Table 2.

The three video cards we test are from NVIDIA and support CUDA [11] enabled applications. CUDA allows general purpose programs to offload kernels of execution to the GPU. The kernels can be any type of general purpose code that could benefit from massive parallelism offered by our tested GPUs that have 16, 32, and 240 cores respectively(for the ION, GT120 and GTX285). In our tests we study two CUDA enabled applications and evaluate real-world results obtained from using low-end GPUs and a high-end GPU.

Both systems we test also have 2-way SMT for each core that exposes up to 16 hardware thread contexts to the OS on the Mac Pro and 4 hardware contexts for the Atom. We also had the ability in both OS's to turn individual cores on and off to experiment with different core counts on the same setup. In Windows we could turn cores on and off at boot time, but only had SMT available when all eight cores were activated. In OS X we were able to try all combinations of cores and SMT support.

4. BENCHMARKS

We performed a variety of experiments for interactive programs on both the Windows and OS X operating systems. We tried to pick our benchmarks to have as much overlap as possible between Windows and OS X to provide insights to the effective use of TLP on modern day chip multi-processor machines. All the tests were carried out by hand where the user worked on the program with a strict set of timing, input, and usage constraints to allow for repeatable tests. No user interface (UI) automation tool we found was able to handle the nature of the interactive sessions because timings can experience subtle changes between runs—such as differing latencies for loading web pages or different play outcomes in game scenarios. Neither could any UI automation tool handle events such as frequently changing web content. Accordingly we elected to do the tests with a real user. Because of the possible impacts of human interaction and the variability in some of the benchmarks we ran each test a minimum of 5 times to obtain a measure of variance. The test results proved to be repeatable as the measured standard deviation was low. For the sake of brevity we briefly describe each benchmark, but each test is documented in detail at [26]. Unless otherwise noted, all benchmarks were tested on both operating systems.

4.1 3D Games

3D games have long been a driver for faster hardware. Game developers continually strive for more realistic graphics, artificial intelligence, physics, and immersive game play. Games were primarily single threaded until around 2005 when chip multi-processors were released on both PC and the current generation gaming consoles.

Activision Call of Duty 4: Modern Warfare- Is a tactical first person shooter (FPS) game that showcases high definition graphics with lots of enemies and background activity on screen. This game was originally developed for the Xbox360 and Playstation 3 game consoles which both use chip multi-processing.

2K Games Bioshock[1]- A slower paced action-adventure FPS game. This game was originally developed for the Xbox360 and Playstation 3.

EA Crysis[1]- Is a standard FPS from EA. It showcases advanced graphics, AI, and physics. It was originally developed for the PC.

4.2 Image Authoring

Image authoring has been known to use multiple processors effectively and a large amount of TLP was present in the original study. We test a 2D image authoring program and a 3D modeling program for these tests.

Adobe Photoshop CS4- Photoshop is an advanced image authoring and composition tool. We tested Photoshop by applying 5 hand picked filters in succession to a 20 megapixel photograph.

[1]Only available on Windows platform

Autodesk Maya 3D 2010- Maya 3D is a three-dimensional modeling tool. The test consisted of opening up a complex model, rotating, zooming and panning the workspace camera. Then the user smoothed the model and rendered the scene using the rasterizing renderer and the raytracing renderer.

4.3 Office

For the Office productivity tests we selected applications that are commonly used by users for basic office tasks.
Adobe Reader 9- We tested Adobe Reader 9 by opening up an eight megabyte PDF file, browsing and searching for phrases in it.
Microsoft Excel 2007/2008- Microsoft Excel was tested with a large 10,000 row spreadsheet to make commands like calculating the standard deviation take an appreciable amount of time. The test consisted of performing common operations on the data, such as calculations and plotting.
Microsoft PowerPoint 2007/2008- Microsoft PowerPoint was tested on a large presentation with a large amount of illustrations and animation. We performed common operations such as adding/deleting slides, previewing animations, inserting pictures and changing the slide formatting.
Microsoft Word 2007/2008- The Microsoft Word test involved adding/deleting text, changing format, and inserting/deleting/moving embedded images.
Microsoft Streets and Trips 2010[1]**-** Microsoft Streets and Trips is a full featured mapping and route planning application. The user was tasked with navigating the user interface briefly and planning a cross country trip to fifteen randomly selected cities in the United States.

4.4 Multimedia Playback

For multimedia playback we picked two media players that are popular and common across both platforms: iTunes and QuickTime player.
iTunes 9- The iTunes test consisted of searching and playing an MP3 files from a local database and also searching and playing videos from the same database.
QuickTime Player- We tested QuickTime by playing a 480p video and a 1080p high definition video encoded in the H.264/AVC format.

4.5 Video Authoring

Video authoring is becoming an increasingly popular application on the desktop as consumer grade camcorders can now shoot high definition video and transferring video to a user's machine for editing is as simple as putting the flash card in a reader. Video authoring is also important because users now want to transcode their personal videos to multiple different formats to put on YouTube.com or share on a DVD.
CyberLink PowerDirector v8[1]**-** PowerDirector v7 from CyberLink is a full featured video editor and production program. It also leverages CUDA for the video transcoding. For testing PowerDirector we imported some video clips and composed them into a short video complete with transitions and titles and rendered it with and without CUDA support.
Handbrake 0.9- Handbrake is an open source licensed video transcoder that can take many input formats and convert it to numerous other formats. It does not have any editing capability. We tested Handbrake by encoding a portion of a source DVD to a high profile H.264 format.
Elemental Badaboom[1]**-** Badaboom is another video transcoder. It uses CUDA to do the video transcoding. To test Badaboom we repeated the Handbrake test as it is a transcoder with no editing ability.

4.6 Web Browsing

In the past ten years web browsing has become a very common workload for desktop users so we do a variety of tests on multiple web browsers. Web browsers are now very complicated pieces of software, having to support many standards such as Flash, Java etc to display rich content. Because of this we perform four distinct types of tests to profile the selected browsers. All the tests take approximately 5 minutes to perform.

The first test (Tabs) has the user first watch a video on YouTube.com, then browse to ESPN.com, go to CNN.com, browse the BestBuy.com site, and finally play a flash game. Each new page is visited by creating a new tab in the browser. An additional test (Sequential) is to perform the Tabs test, but without using tabs and simply leaving the current site for the next in succession. The final two browser tests are to see the difference between sites with lots of active content like Java and Flash and one without much active content. One test was to browse ESPN.com, and the other to browse a Facebook.com profile. We test two browsers: Mozilla's Firefox 3.5, and Apple's Safari 4.0.

5. RESULTS: INTEL XEON SYSTEM

In this section we present our analysis of the data we collected from our experiments. We will cover overall results, present breakdowns for certain applications, and present an analysis of GPU usage characteristics.

5.1 Overall Results

A summary of all the 8 Core simulations with SMT support and the GTX285 GPU are presented in Table 3. The TLP data is summarized in three fields. The first field shows what percentage of total execution time (c_i) that exactly i=0, ...,n threads are executed concurrently, where n is the number of thread contexts in the machine. 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 entire system averaged over at least 5 runs of the test. The third field reports the standard deviation. The small values indicate the tests are reproducible and insensitive to the variations induced by a real user. The fourth field presents GPU utilization percentages of the system. At the end of the table an average TLP and average GPU utilization is calculated for each of the benchmark categories.

The first thing to note is that every benchmark shows some portion of time with concurrent execution but in many cases this concurrent execution is a small percentage. Meaning programmers are writing applications with threads that can use a multi-processor machine. However, in many cases the applications do not have a high percentage of concurrent execution. For instance, in *Office* and *Web Browsing* the average TLP is between 1 and 2 for our test system. Some web tests show a surprising amount of concurrency with numbers above 2. Even for more compute intensive applications like *Games*, *Playback* and *Image Authoring* the average TLP,

[1]Only available on Windows platform

			Idle	8 Core SMT - System Wide TLP																		GPU	Average GPU	
	Application	OS	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	TLP	σ	Utilization	Average TLP	Utilization
Game	Bioshock	Windows	1%	57%	31%	7%	1%	2%	1%											1.6	0.05	75%	1.6	69%
	Call of Duty 4	Windows	0%	12%	35%	20%	14%	8%	7%	2%	1%									2.1	0.21	86%		
	Call of Duty 4	OS X	6%	77%	16%	1%														1.2	0.02	29%		
	Crysis	Windows	1%	72%	23%	4%	1%													1.4	0.07	84%		
Image Authoring	Maya3D 2010	Windows	55%	34%	6%	1%					1%								2%	2.4	0.53	18%	2.1	12%
	Maya3D 2010	Mac	57%	32%	5%	1%													1%	2.2	0.14	12%		
	Photoshop CS4	Windows	43%	43%	7%	1%					3%								1%	2.0	0.56	17%		
	Photoshop CS4	Mac	50%	38%	7%	1%	1%											1%	1%	1.9	0.19	1%		
Office	Adobe Reader 9	Windows	65%	25%	8%	1%														1.3	0.05	23%	1.2	11%
	Adobe Reader 9	OS X	70%	24%	6%															1.2	0.02	4%		
	Excel 2007	Windows	72%	23%	4%															1.2	0.02	10%		
	Excel 2008	OS X	57%	38%	5%															1.1	0.01	2%		
	PowerPoint 2007	Windows	69%	25%	5%	1%														1.2	0.03	16%		
	PowerPoint 2008	OS X	66%	30%	4%															1.1	0.01	7%		
	Streets & Trips 2010	Windows	68%	23%	7%	1%														1.4	0.01	14%		
	Word 2007	Windows	74%	22%	4%															1.2	0.04	16%		
	Word 2008	OS X	70%	27%	3%															1.1	0.01	3%		
Playback	iTunes 9	Windows	71%	23%	5%	1%														1.3	0.16	22%	1.5	22%
	iTunes 9	OS X	79%	18%	3%															1.2	0.02	5%		
	Quicktime 7.6	Windows	50%	38%	10%	2%														1.3	0.01	43%		
	Quicktime X	OS X	79%	14%	6%	1%														1.4	0.01	0%		
	Quicktime 7.6 - HD	Windows	66%	22%	5%	1%	1%				3%									2.1	0.06	40%		
	Quicktime X - HD	OS X	67%	13%	17%	3%														1.7	0.02	19%		
CUDA	Badaboom	Windows	54%	35%	9%	2%														1.3	0.03	95%	2.2	62%
	PowerDirector v8	Windows	42%	20%	12%	6%	5%	4%	3%	2%	3%	1%								3.2	0.52	28%		
Video Authoring	Handbrake 0.9	Windows	1%				1%	3%	9%	17%	22%	20%	14%	8%	4%	1%				8.4	0.02	8%	7.4	9%
	Handbrake 0.9	OS X	1%				1%	3%	8%	16%	21%	20%	15%	9%	4%	1%				9.0	0.44	0%		
	PowerDirector v8	Windows	27%	20%	11%	6%	4%	3%	2%	6%	8%	5%	5%	3%						4.8	0.15	18%		
Web Browsing	Firefox 3.5*	Windows	66%	24%	6%	1%					1%									1.5	0.05	24%	2.0	16%
	Firefox 3.5*	OS X	49%	33%	10%	3%	1%						1%	1%	1%	1%				2.2	0.12	10%		
	Safari 4.0*	Windows	50%	34%	11%	3%	1%				1%									1.6	0.06	24%		
	Safari 4.0*	OS X	50%	27%	10%	3%	1%	1%			1%	1%	1%	1%	1%	1%	1%			2.5	0.19	6%		

Table 3: Table of System TLP and GPU Utilization results. Concurrency values less than 1% are not displayed.

though higher, is still approximately 2. Only in the domain specific application of *Video Authoring* do we observe a high amount of TLP, in this case an average of approximately 7. Because of the average TLP being primarily just below or above 2, this leads to the conclusion that a small multi-processing machine of 2-3 cores is adequate for all but a few domain specific applications like *Video Authoring*. The following subsections will provide more detailed studies of the applications run on our test system.

In terms of GPU usage only a handful of applications, *Games* and *CUDA*, fully exploit the GPU with average utilizations near 70%. All the other benchmarks leave it primarily under-utilized, offering a wealth of resources for future parallel programmers. A more detailed discussion of the GPU can be found in Section 5.6.

5.2 Ten Year Perspective

A comparison of similar applications from the 2000 study and our study is presented in Figure 2. The biggest gains made in the past ten years has been in *Video Authoring*. This type of application has high computation requirements, and is inherently parallel, making it a prime target for parallelization. In fact, the ParallelMPEG decoder from the original study was only a research project at the time, but this type of code has now found itself in mainstream video players. On the other hand, some applications have shown only modest improvements, such as the *Web Browsing* and *Office* areas. For *Office* applications the major improvement has come from the virus scan interface of the system, successfully leveraging the additional cores, but in all cases it can be concluded that current single-threaded performance is adequate for *Office* applications. For *Web Browsing* it appears that there are benefits to be gained by moving to parallel code, but little progress has been made with TLP approaching 2 on average for these applications. There has been some improvement in 3D image rendering but it is offset in the *Image Authoring* applications by the decrease in 2D rendering (Photoshop). The decrease in both the Photoshop and Quicktime applications can be attributed to the gains of single core performance in terms of both frequency and architectural features.

5.3 OS Threading Support

OS X and Windows are two substantially different operating systems. OS X is derived from a combination of the BSD Unix [44] and Mach [47] operating systems. Because of its Unix heritage it natively supports the POSIX [1] threads (pthreads) interface for concurrency. Windows has a proprietary kernel and uses its own custom threading interface. These subtle differences mean that some applications, when written for one OS, may perform sub-optimally when ported to the other OS.

For all applications that run both in Windows and OS X a TLP comparison is made in Figure 3. The interesting trend to note is that OS X is better at *Video Authoring*, *Playback* and *Web Browsing*. The Quicktime and Safari applications are developed by Apple and tuned for OS X, which is a main contributor to why their performance is better in OS X and also use more features of the system. For instance, Quicktime X leverages the onboard video decode ASICs found on

			Idle	8 Core SMT - System Wide TLP																		GPU	Average GPU
	Test	OS	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	TLP	σ	Utilization	Average TLP	Utilization
Firefox 3.5	Tabs	Windows	55%	29%	11%	3%	1%													1.7	0.06	29%	1.5	24%
	ESPN	Windows	67%	26%	5%	1%														1.4	0.04	23%		
	Facebook	Windows	77%	19%	3%															1.3	0.05	15%		
	Sequential	Windows	66%	24%	6%	1%					1%									1.7	0.05	28%		
	Tabs	OS X	29%	41%	15%	5%	1%					1%	1%	1%	1%	1%	1%	1%		2.7	0.13	13%	2.2	10%
	ESPN	OS X	60%	33%	6%	1%														1.6	0.13	7%		
	Facebook	OS X	71%	25%	4%															1.3	0.06	7%		
	Sequential	OS X	36%	33%	14%	5%	1%				1%	1%	2%	2%	3%	2%	1%			3.3	0.14	13%		
Safari 4.0	Tabs	Windows	37%	38%	16%	5%	1%	1%			1%									1.7	0.05	31%	1.6	24%
	ESPN	Windows	53%	34%	9%	2%	1%													1.5	0.07	20%		
	Facebook	Windows	64%	29%	6%	1%														1.3	0.09	16%		
	Sequential	Windows	45%	37%	12%	3%	1%				2%									1.7	0.02	30%		
	Tabs	OS X	26%	32%	18%	7%	3%	1%	1%	1%	1%	1%	2%	2%	2%	2%	1%			3.2	0.33	9%	2.5	6%
	ESPN	OS X	64%	26%	6%	1%														1.9	0.18	3%		
	Facebook	OS X	76%	20%	3%															1.4	0.10	2%		
	Sequential	OS X	34%	30%	15%	5%	2%	1%	1%	1%	1%	1%	1%	2%	2%	2%	2%			3.5	0.14	10%		

Table 4: Table of System TLP, and Graphics Utilization results for Web Browser workloads. Concurrency values less than 1% are not displayed.

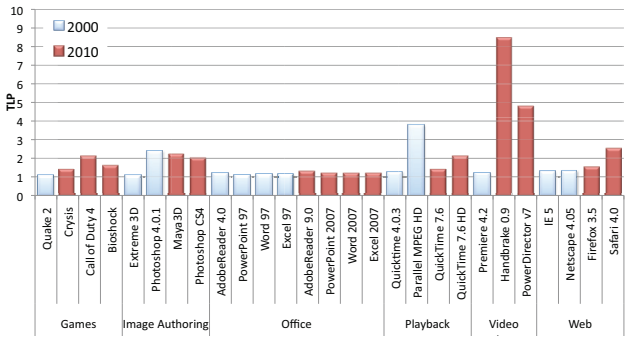

Figure 2: System TLP comparison for 2000 vs. 2010.

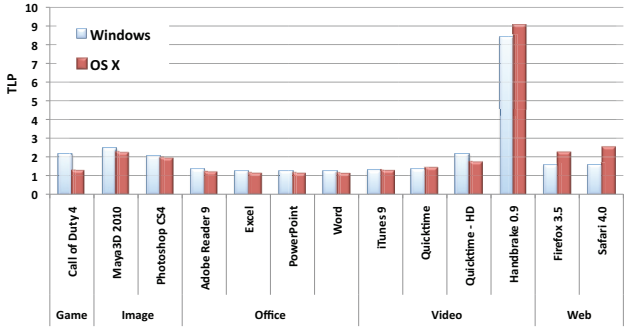

Figure 3: System TLP of 8 Cores w/ SMT on applications in both Windows and OS X.

current generation NVIDIA GPUs, as shown by 0% GPU utilization for the 480P resolution test. The Handbrake benchmark is an open source benchmark developed mostly on Linux and is tuned well to work using pthreads, which OS X supports natively and Windows emulates. On the other hand, *Games* and *Office* applications are initially developed for Windows, yielding better performance than on OS X. The port of these applications to OS X has not leveraged the parallelism realized in its Windows counterpart. For example, Call of Duty 4 has a TLP of 2 on Windows whereas the port to OS X exhibits significant single-thread behavior. In addition, the Windows *Office* applications gain in overall TLP by invoking the virus scan application in a separate thread. Overall, the trend is that applications show slight preference to one or the other OS, typically the one they were initially developed for.

With the introduction of Windows 7 and Mac OS X Snow Leopard, there has been considerable effort to push high level parallel programming initiatives and include them as integral parts of the OS. These initiatives include: DirectCompute [14] for Windows to leverage GPUs; OS X Snow Leopard supports OpenCL [22] for GPUs and introduces Grand Central Dispatch (GCD) [16] for the main multicore CPU. OpenCL and DirectCompute style parallel programming are targeted towards data-parallel applications like video transcoding, which as shown in this study is already taking advantage of such parallelism with CUDA and threads. GCD targets general applications such as web browsing, which while heavily threaded still exhibits primarily single threaded behavior. It remains to be seen whether these new programming initiatives will be used heavily in the future.

5.4 Core Scaling

The TLP performance as the system scales from 2 to 8 cores is another important trend to consider. Figures 4 and 5 show how TLP and GPU utilization changes over time for a subset of the benchmark run. First, the Handbrake application in Figure 4 illustrates the scalability of a highly parallel program which heavily utilizes the system. From the graph it can be seen that the TLP is at a maximum for most of the run on 2,4, and 8 core systems. When SMT is enabled on an 8 core system the resultant TLP fluctuates around 8. The theoretical maximum would be a TLP of 16, but in the case of Handbrake the algorithm used for the H.264 encoding has reached the peak of its scaling—it appears the code as written has bottlenecks that prevent scaling past 8-9 concurrent threads on average. This was confirmed by reading the developer notes present for Handbrake's H.264 encoder. It is also important to note that the transcoding rate of this application in frames-per-second (fps) scales up linearly with the number of cores until more than 8 thread contexts become available.

The second trend is illustrated by the Photoshop benchmark in Figure 5. In this case the benchmark is often filled with idle time as the user navigates menus, and configures

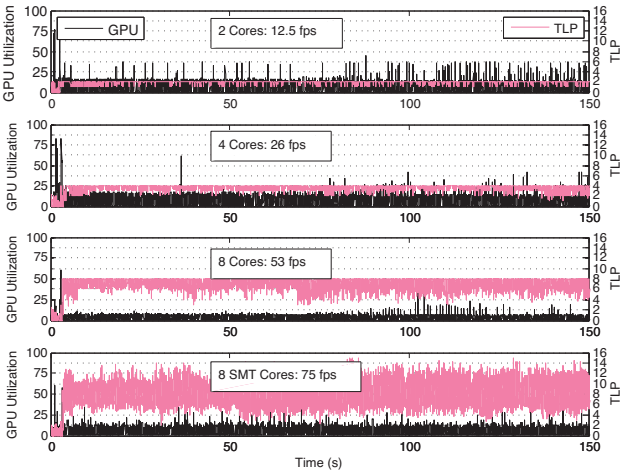

Figure 4: System TLP and GPU Utilization over time for the Handbrake benchmark in Windows for differing numbers of cores. Note as cores increase the benchmark takes almost full advantage showing good scalability for video transocde workloads.

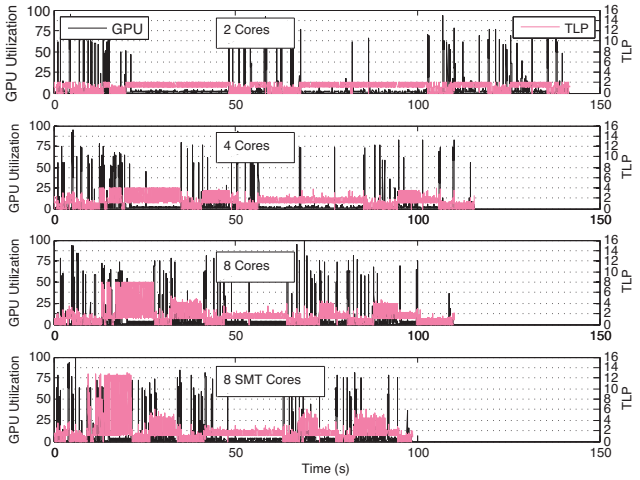

Figure 5: System TLP and GPU Utilization over time for the Photoshop benchmark in Windows for differing numbers of cores. Note that as cores increase, the peaks of the filtering operations become taller and narrower, yielding a shorter runtime. As SMT is enabled a large improvement is seen.

filters. The plot shows the user performing several filter operations, these correspond to the peaks seen in the 8 core graph. In the 2 core system the runtime is long and the TLP is flat between 1 and 2. As the number of cores is increased the peaks where the filters are performed grow both higher, due to increased resources for the parallel part, and narrower, due to the decrease in runtime from the parallelism. Overall, this reduces the runtime to complete these filters and the shortened total execution time can be observed in the figure.

In summary, there are two trends that increased resources provide: 1) for non-interactive applications where the system can be fully utilized, application performance increases, in this case fps was increased; and 2) for interactive applications, the time a user waits for small tasks to complete—responsiveness of the application—can be improved and total execution time reduced.

5.5 Simultaneous Multi-Threading

The impact of SMT on parallelism is another important feature to evaluate when examining concurrent architectures for desktop applications. Figure 7 shows the resulting fps of the transcoding of the Handbrake benchmark in Windows, OS X, and OS X with SMT support. It also shows the performance of a CUDA offload application, Badaboom, that will be discussed in Section 5.6. The Handbrake benchmark transcoding rate increases with the addition of cores. The graph is not ideally linear due to the way the cores are interconnected. In particular, some cores share cache on-die while others need to communicate between chips in different sockets. In our SMT 4 core system all 8 thread contexts are on the same die and can communicate through the shared on-die cache. While our non-SMT 8 core system needs to communicate between the two sockets, increasing latency to complete the operations. This is illustrated in Figure 7 where the SMT 4 core system outperforms the non-SMT 8 core system. This shows that SMT is beneficial for parallel applications when cache can be shared and is fast to access.

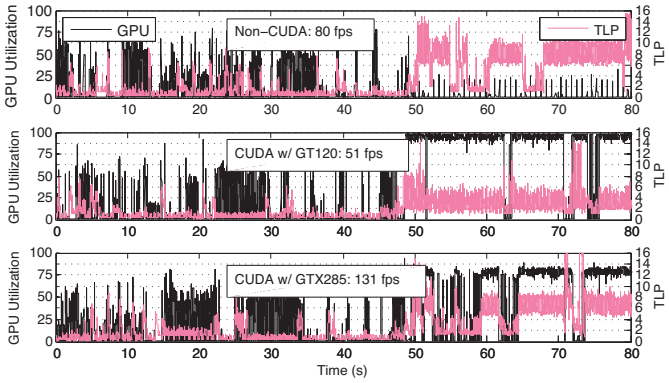

Figure 6: A time plot of a subsection of the PowerDirector benchmark, transcoding begins around 50s. The three plots show the non-CUDA enhanced version, and CUDA enhanced for a low-end GPU and a high-end GPU. Note the improvement in transcoding rate with CUDA enhancements on the high end graphics card.

5.6 Graphics

One of the main questions this work wanted to answer was: "Are GPU's being used to improve system performance, and what opportunities are there to further use them?" The following subsections will break down the measurements of our CUDA applications and typical GPU utilization. These will show there remains potential to leverage the additional computational resources of the GPU.

5.6.1 Graphics Offloading

CUDA support is a promising new programming technique where the goal is to offload highly parallel work to the GPU. For this study the PowerDirector and Badaboom benchmarks offer two video transcoding applications with CUDA support that can be compared to non-CUDA transcoders. First the transcoding rate, fps, of the Badaboom

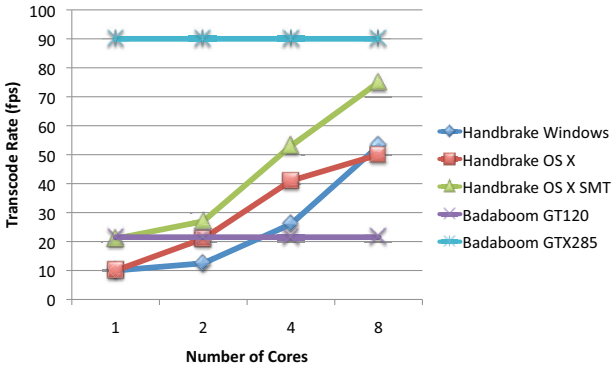

Figure 7: Transcoding Rate of Handbrake and CUDA Application Badaboom for 1 to 8 Cores. Illustrating the performance afforded by GPU offloading, and the impact of SMT on performance.

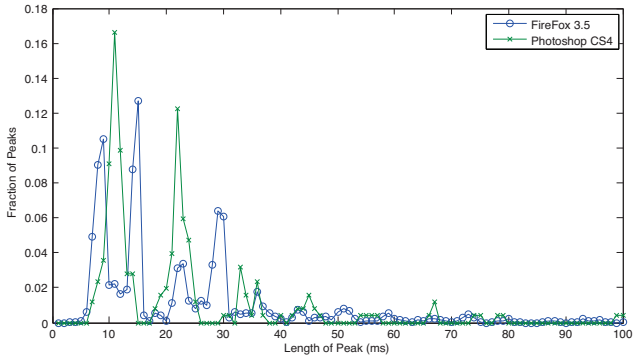

Figure 8: Frequency of GPU Utilization Peaks above 10% of various durations for the Firefox and Photoshop Benchmarks.

benchmark is shown in Figure 7. It is plotted both for the low-end graphics card, GT120, and the high-end graphics card, GTX285. From the graph it can be seen that the system (running the non-CUDA Handbrake application) easily outperforms a low-end graphics card for core counts greater than 2. However, a high end card is capable of transcoding at 125% the speed of even an 8 core system with SMT.

Figure 6 shows a time breakdown of the TLP and graphics utilization of the PowerDirector benchmark with and without CUDA support. The first thing to note is the transcoding rate achieved by each application, and the trends are similar to the Badaboom application. In terms of graphics utilization it can be seen that the non-CUDA version uses almost no graphics support during the transcoding phase of the application. When the GT120 is used with CUDA the graphics utilization reaches 100% and the transcoding becomes GPU limited. Once the graphics card is improved to the GTX285 the utilization of the GPU falls off and the TLP of the system increases to 8 leaving the transcoding once again core limited because the graphics card has enough resources to match the transcoding rate of the cores.

In conclusion CUDA is promising when high-end GPUs with many cores are available. For low-end GPUs, CUDA may not offer much improvement, and may even perform worse than a typical 2 or 4 core chip multi-processor.

5.6.2 *Graphics Utilization*

In Table 3 the GPU column presents the average utilization of the GPU for each benchmark. It is noteworthy that for most applications the GPU utilization is low. The majority of the time there are short spikes when the screen needs to be updated, e.g. moving a window. The typical spike in utilization for the Firefox benchmark in Windows is approximately 20ms in length and 40% utilization. Figure 8 shows a breakdown for the length of GPU utilization spikes higher than 10% for the Photoshop and Firefox applications. It shows that for these applications there is plenty of parallel resources being under-utilized in the GPU. If GPU offloading enhancements could be made for the benchmarks, there is significant opportunity to further utilize the GPU. There are, however, a couple of application classes, *Playback* and *Game*, where the GPU is already highly utilized and there are less opportunities for off-loading. The other area with high utilization is the *CUDA* applications, this is, of course, due to the fact that they are already leveraging off-load capabilities.

5.7 Web Browser Workload

A detailed breakdown of the individual web browser tests is presented in Table 4. The interesting observation from the results is that the sequential page access test actually observes a higher TLP than the tab based page access under OS X. The tab based approach is keeping alive pages, including some that are streaming content, and does exhibit higher TLP for that portion of the run. However, in the sequential version of the tests a garbage collection thread runs each time a site is navigated away from. This garbage collection thread is used to reduce the memory footprint of each web browser by cleaning up allocated memory and writing back cacheable items to the file cache, e.g., pictures. The information still resides in the system so it is faster to access compared to retrieving the site again over the network. The tab based version keeps the memory alive, and the tabs can be thought of as a user controlled cache of the sites likely to be revisited soon. Overall, web-browsing is a more CPU intensive application than it previously was 10 years ago. Then, it was primarily network dominated, but now network bandwidth is plentiful and the amount of content present takes appreciable cpu time to render. Behavior is still primarily single threaded, as indicated by the low TLP. Jones et al. [42] see the same behavior where the browser is CPU dominated instead of waiting idle for the network and are working towards developing methods to parallelize web browsing.

6. RESULTS: INTEL ATOM SYSTEM

We tested the Atom to see if decreasing the overall performance of the cores made an appreciable difference in the observed TLP. The hypothesis was: The Intel Xeon system has very high single thread performance therefore individual tasks may be completing quickly enough that they were not running concurrently. The Intel Atom's single thread performance compared to the Intel Xeon is very low and could lead to increases in TLP. This performance differential is seen in the Handbrake video transcoder, a single core of the Xeon processor can maintain a 10fps average encode rate

		Idle	System TLP						
	Application	C0	C1	C2	C3	C4	TLP	σ	Average TLP
Game	Bioshock	2%	25%	35%	27%	11%	2.2	0.04	2.0
	Call of Duty 4	1%	37%	27%	26%	10%	2.1	0.05	
	Crysis	0%	39%	44%	14%	2%	1.8	0.02	
Image Authoring	Maya3D 2010	20%	41%	13%	5%	21%	2.1	0.05	1.9
	Photoshop CS4	8%	59%	17%	6%	10%	1.6	0.11	
Office	Adobe Reader 9	40%	36%	19%	5%	1%	1.5	0.03	1.4
	Excel 2007	45%	40%	12%	2%		1.3	0.01	
	PowerPoint 2007	38%	42%	16%	4%	1%	1.4	0.01	
	Streets & Trips 2010	39%	34%	20%	5%	2%	1.6	0.02	
	Word 2007	35%	49%	13%	3%		1.3	0.01	
Playback	iTunes 9	24%	45%	24%	6%	1%	1.5	0.09	2.1
	Quicktime 7.6	4%	28%	39%	23%	6%	2.1	0.10	
	Quicktime 7.6 - HD	11%	19%	22%	22%	26%	2.6	0.01	
CUDA	Badaboom	68%	23%	9%	1%		1.3	0.04	2.0
	PowerDirector v8	8%	17%	20%	23%	32%	2.8	0.07	
Video Authoring	Handbrake 0.9	0%		2%	10%	88%	3.8	0.04	3.6
	PowerDirector v8	3%	8%	11%	19%	58%	3.3	0.04	
Web Browsing	Firefox 3.5*	25%	42%	19%	9%	5%	1.6	0.04	1.7
	Safari 4.0*	23%	35%	21%	12%	8%	1.8	0.08	

Table 5: Table of System TLP for the Intel Atom machine on Windows 7 only. Concurrency values less than 1% are not displayed.

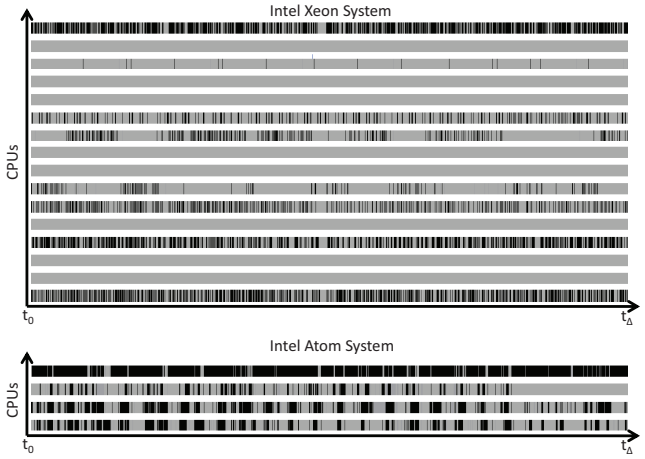

Figure 9: Trace of 1 second of execution for the *Call of Duty 4* benchmark for the Xeon and Atom systems on Windows 7. Black is active processing, and Gray is idle time. Each bar corresponds to a logical CPU visible to the OS.

while the Atom processor averages 3fps. Therefore a single Atom core is roughly 30% of the performance of a Xeon core.

The overall experiment results are presented in Table 5. The tests were carried out in the exact same manner as the tests reported in Table 3, but only numbers for Windows 7 are reported, OS X is not supported on the atom platform. We are also not able to report GPU statistics for the NVIDIA ION chip as the version of NVPerfKit we used did not support it.

The first major difference between the Atom results and the Xeon results is the marked decrease in *idle* time between the two platforms, except for *Games*. This further reinforces that the Atom's single thread performance is inferior to the Xeon. Further confirmation of this lack of performance was reported by the human tester as almost all the tests produced poor user experiences with long wait times.

The *Playback* test sees an increase in TLP to around 2 on average, but the *idle* time is very small on the Atom. The *Games* tests have TLP results that are almost identical between the two platforms. The similar TLP numbers indicate that the software is the main factor in the low amount of parallelism exposed. This can be seen in Figure 9 which shows an execution trace for both the Xeon and Atom on the Call of Duty 4 test. For both architectures, the amount of idle time is large with one main thread appearing to spawn multiple small tasks, which explains why the TLP is relatively low.

The *Office* tests show the same TLP as the Xeon tests with averages between 1 and 1.5. The idle times are surprising as they are much lower than the Xeon tests. This emphasizes that single-thread performance is still important. The tester noted that *Office* applications were particularly sluggish. Interestingly, the results for performing *Video Authoring* or *Image Authoring* on the Atom are the same as on the Xeon; these applications are able to successfully leverage all the contexts available.

The *Web Browsing* tests are again interesting. The idle time is less than one quarter of the execution time on the Atom whereas on the Xeon it was more than one half. This implies web browsing is more CPU limited than commonly perceived, as the network link used was low latency and high bandwidth. This is especially apparent the for *Tabs* and *Sequential* tests where the user is going to many content and feature rich sites. The idle time for these tests was close to zero, but the TLP only approached 2. This also indicates the software is the main cause of low TLP.

The Atom results show that low single-thread performance can have a small impact on the TLP observed. The Xeon did hide some TLP as it completed tasks very quickly compared to the Atom. But overall, this effect was small. The main observation was that the software is the main limiting factor, even on a "small" chip-multiprocessor with 2 cores and 4 hardware thread contexts.

7. DISCUSSION

Most modern designs have adopted multi-cores in an attempt to further increase performance. However, there has been surprisingly little increase in actual concurrency for desktop/laptop applications since the original study by Flautner et al. ten years ago. This is true in spite of the fact that programmers create a large number of threads in their programs. It appears that these threads exist primarily to structure code to be comprehensible and that desktop/laptop applications are mainly single threaded in behavior. This lack of parallelism in desktop applications such as games and web browsing may be inherent, but if it is not, then programmers must find significantly more parallelism to take advantage of future many-core chip multiprocessors. If the past is a predictor of the future, this seems unlikely.

In the scientific and server application domains, leveraging multi-processing is well understood and is backed by a large body of literature. As a product of this research and development, current architectures are well suited for these two areas. Multi-core SMP systems on a chip work very well for servers, and current GPUs work well for scientific applications that need abundant floating point performance. Unfortunately, as shown by our results, applications from the desktop/laptop space fit very poorly with these architectures because of their low TLP and GPU utilization. The only ex-

ception is applications like video transcoding which are very similar to scientific applications.

The apparent lack of progress in creating usable concurrency in the desktop/laptop application domain points to a lack of deeper understanding of the applications. To yield a more applicable design, architects should consider more relevant workloads, e.g. web browsing as benchmark choices. Future designs may require different characteristics from than those of server or scientific architectures to continue allowing increases in performance and energy efficiency. The tests with the Atom platform further indicate that software is the main factor in the lack of parallelism. The continued progression to larger chip-multiprocessors with less capable cores may be applicable for scientific computers and some servers, but seems less of a viable path to follow for desktop/laptop computers that value latency over throughput.

8. CONCLUSION

In the time since the original study, much has changed in the computing landscape. Architectures have become radically different and multi-processing is now the norm on the desktop from low power processors like the Intel Atom to the high performance Intel Xeon, offering up to hundreds of threads. On the other hand, as shown in the results, little has changed from the software perspective. In domain specific applications like video transcoding, many-core processors are being fully utilized, whether it be the general purpose cores with Handbrake, or the GPU with its hundreds of cores for Badaboom. However, in general purpose applications the average TLP 10 years later has shown only a modest improvement to $\sim$2, and continuing to add cores to either the GPU or CPU may not be the answer for this application space. For the desktop/laptop environment multicore processor hardware is out-stripping software developers ability to keep up even for low performance systems like the Intel Atom. For 3D games, they seem to be able to use just over 2 processors. For web browsing, it can utilize multiple cores, though only up to about four as rendering a web page requires lots of threads to gather the content to render. In other areas like office productivity, multiple cores are hardly used as the level of parallelism has changed little in the last 10 years.

Our final conclusion is that software developers are still playing catch up, perhaps as a result of the relatively abrupt transition from high performance single-thread machines to chip multi-processors. In practice software developers use large numbers of threads in most applications in the desktop/laptop application space. However, they have not made appreciable steps in exploiting chip multi-processors, because the threads rarely run in parallel or the work distributed among threads is very unbalanced. This imbalance in work leads to Amdahl's effects resulting in the low TLP seen. Similar problems plague GPU vendors, because they are developing massive chip multi-processors with the majority of the silicon being left under-utilized. Developers have started to test the waters by writing multi-threaded applications that leverage both CPUs and GPUs, but have yet to take the plunge into the many-core chip multi-processing era. If many-core is to be the solution for future desktop/laptop applications, then programmers and algorithm designers must discover much greater levels of balanced parallelism. Our ten years perspective suggests this may be a significant challenge.

9. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their suggestions and comments. This work was supported by ARM.

10. REFERENCES

[1] IEEE. Standard for Threads Interface to POSIX. P1003.1c, 1996.

[2] Intel Pentium Processor. http://datasheets.chipdb.org/Intel/x86/Pentium/24199710.PDF, 1997.

[3] AMD Athlon Processor Product Brief. http://www.amd.com/us-en/Processors/ProductInformation/0,„30_118_1260_759%5E1151,00.html, 1999.

[4] Intel Pentium III Processor. http://www.intel.com/design/intarch/pentiumiii/pentiumiii.htm, 1999.

[5] NVIDIA GeForce 256. http://www.nvidia.com/page/geforce256.html, 1999.

[6] Power4 system microarchitecture. http://www-03.ibm.com/systems/p/-hardware/whitepapers/power4.html, 2001.

[7] AMD Announces World's First 64-Bit, x86 Multi-Core Processors For Servers And Workstations At Second-Anniversary Celebration Of AMD Opteron Processor. *AMD News Room*, 2005.

[8] Intel Has Double Vision: First Multi-Core Silicon Production Begins. *Intel Press Room*, 2005.

[9] AMD "Close to Metal" Technology Unleashes the Power of Stream Computing . *AMD News Room*, 2006.

[10] DTrace User Guide. *Sun Microsystems Inc.*, 2006.

[11] NVIDIA Unveils CUDA - the GPU Computing Revolution Begins. *NVIDIA News Releases*, 2006.

[12] Intel Atom Processor. http://www.intel.com/products/processor/atom/specifications.htm, 2008.

[13] NVIDIA PerfKit. *Nvidia Developer Zone*, 2008.

[14] The Direct3D11 Compute Shader. Microsoft WINHEC Session GRA-T517, 2008.

[15] AMD Displays Llano Die: 4 x86 Cores, 480 Stream Processors. http://www.xbitlabs.com/news/cpu/display/20091111143547_AMD_Displays_Llano_Die_4_x86_Cores_480_Stream_Processors.html, 2009.

[16] Grand Central Dispatch:A better way to do multicore. Apple Inc. Technical Breif, 2009.

[17] Intel Previews Intel Xeon 'Nehalem-EX' Processor. *Intel Press Room*, 2009.

[18] International Technology Roadmap For Semiconductors - System Drivers. *Iternational Technology Roadmap for Semiconductors*, 2009.

[19] Leopard Reference Library. *Apple Inc. Developer Connection*, 2009.

[20] NVIDIA GeForce GT 120 (OEM Product). http://www.nvidia.com/object/product_geforce_gt_120_us.html, 2009.

[21] OMAP 4: Mobile applications platform. *Texas Instruments Product Bullentin*, 2009.

[22] OpenCL:Parallel Computing for Hetergeneous Devices. http://www.khronos.org/developers/library/overview/opencl_overview.pdf, 2009.

[23] AMD Sets the New Standard for Price, Performance, and Power for the Datacenter. *AMD Newsroom*, 2010.

[24] Intel Sandy Bridge. http://en.wikipedia.org/wiki/Intel_Sandy_Bridge_%28microarchitecture%29, 2010.

[25] Intel Spotlights New Extreme Edition Processor, Software Developer Resources at Game Conference. *Intel Press Room*, 2010.

[26] Interactive TLP Bench. http://itlpbench.eecs.umich.edu, 2010.

[27] The Snapdragon Platform. http://www.qctconnect.com/products/snapdragon.html, 2010.

[28] Ultra-Thin Notebooks: Powered by ultra-low-voltage Intel Core processors. http://www.intel.com/in/irdonline/ultra_low.htm, 2010.

[29] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 282–293, New York, NY, USA, 2000. ACM.

[30] A. Berillo. Multi-Core Processors in 3D Games. http://ixbtlabs.com/articles3/video/quadcore-p6.html, 2008.

[31] B. Chen, Y. Endo, K. Chan, D. M. A. Dias, A. Dias, M. Seltzer, and M. D. Smith. The measured performance of personal computer operating systems. *ACM Transactions on Computer Systems*, 14:3–40, 1995.

[32] Y. Endo, Z. Wang, J. B. Chen, and M. Seltzer. Using latency to evaluate interactive system performance. In *OSDI '96: Proceedings of the second USENIX symposium on Operating systems design and implementation*, pages 185–199, New York, NY, USA, 1996. ACM.

[33] K. Flautner, R. Uhlig, S. Reinhardt, and T. Mudge. Thread-level parallelism and interactive performance of desktop applications. *SIGARCH Comput. Archit. News*, 28(5):129–138, 2000.

[34] K. Flautner, R. Uhlig, S. Reinhardt, and T. Mudge. Thread-level parallelism of desktop applications. *Workshop on Multi-threaded Execution, Architecture and Compilation*, 2000.

[35] E. Frachtenberg. Process scheduling for the parallel desktop. In *ISPAN '05: Proceedings of the 8th International Symposium on Parallel Architectures,Algorithms and Networks*, pages 132–139, Washington, DC, USA, 2005. IEEE Computer Society.

[36] E. Frachtenberg and Y. Etsion. Hardware Parallelism: Are Operating Systems Ready? (Case Studies in Mis-Scheduling). *Workshop on the Interaction between Operating System and Computer Architecture*, 2006.

[37] N. Giacaman, O. Sinnen, N. Giacaman, and O. Sinnen. Inhibitors for desktop parallelisation, 2006.

[38] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, and K. Olukotun. The stanford hydra cmp. *IEEE Micro*, 20(2):71–84, 2000.

[39] C. Hauser, C. Jacobi, M. Theimer, B. Welch, and M. Weiser. Using threads in interactive systems: a case study. *SIGOPS Oper. Syst. Rev.*, 27(5):94–105, 1993.

[40] L. D. Hung and S. Sakai. Dynamic estimation of task level parallelism with operating system support. In *ISPAN '05: Proceedings of the 8th International Symposium on Parallel Architectures,Algorithms and Networks*, pages 358–363, Washington, DC, USA, 2005. IEEE Computer Society.

[41] R. Isaacs, P. Barham, J. Bulpin, R. Mortier, and D. Narayanan. Request extraction in magpie: events, schemas and temporal joins. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 17, New York, NY, USA, 2004. ACM.

[42] C. G. Jones, R. Liu, L. Meyerovich, K. Asanovic, and R. Bodik. Parallelizing the Web Browser. *First USENIX Workshop on Hot Topics in Parallelism*, 2009.

[43] D. C. Lee, P. J. Crowley, J.-L. Baer, T. E. Anderson, and B. N. Bershad. Execution characteristics of desktop applications on windows nt. *SIGARCH Comput. Archit. News*, 26(3):27–38, 1998.

[44] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The design and implementation of the 4.4BSD operating system*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1996.

[45] T. D. Nguyen, R. Vaswani, and J. Zahorjan. Parallel application characterization for multiprocessor scheduling policy design. In *of Lectures Notes in Computer Science*, pages 105–118. Springer-Verlag, 1996.

[46] I. Park and R. Buch. Improve Debugging And Performance Tuning With ETW. *MSDN Magazine*, 2007.

[47] R. Rashid, R. Baron, R. Forin, D. Golub, and M. Jones. Mach: A system software kernel. In *In Proceedings of the 1989 IEEE International Conference, COMPCON*, pages 176–178. Press, 1989.

[48] M. Zhou and A. J. Smith. Analysis of personal computer workloads. In *MASCOTS '99: Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 208, Washington, DC, USA, 1999. IEEE Computer Society.