Media Applications on Hyper-Threading Technology

Yen-Kuang Chen, Microprocessor Research, Intel Labs
Matthew Holliman, Microprocessor Research, Intel Labs
Eric Debes, Microprocessor Research, Intel Labs
Sergey Zheltov, Microprocessor Research, Intel Labs
Alexander Knyazev, Microprocessor Research, Intel Labs
Stanislav Bratanov, Microprocessor Research, Intel Labs
Roman Belenov, Microprocessor Research, Intel Labs
Ishmael Santos, Software Solutions Group, Intel Corporation

Index words: Hyper-Threading Technology, multithreading, multimedia, MPEG, performance analysis

ABSTRACT

This paper characterizes selected workloads of multimedia applications on current superscalar architectures, and then it characterizes the same workloads on Intel® Hyper-Threading Technology. The workloads, including video encoding, decoding, and watermark detection, are optimized for the Intel® Pentium® 4 processor. One of the workloads is even commercially available and it performs best on the Pentium 4 processor. Nonetheless, due to the inherently sequential constitution of the algorithms, most of the modules in these well-optimized workloads cannot fully utilize all the execution units available in the microprocessor. Some of the modules are memory-bounded, while some are computation-bounded. Therefore, Hyper-Threading Technology is a promising architecture feature that allows more CPU resources to be used at a given moment.

Our goal, in this paper, is to better explain the performance improvements that are possible in multimedia applications using Hyper-Threading Technology. Our initial studies show that there are many unexplored issues in algorithms and applications for Hyper-Threading Technology. In particular, there are many techniques to develop better software for multithreading systems. We demonstrate different task partition/scheduling schemes and discuss their trade-offs so that a reader can understand how to develop efficient applications on processors with Hyper-Threading Technology.

INTRODUCTION

To date, computational power has typically increased over time because of the evolution from simple pipelined designs to the complex speculation and out-of-order execution of many of today's deeply-pipelined superscalar designs. While processors are now much faster than they used to be, the rapidly growing complexity of such designs also makes achieving significant additional gains more difficult. Consequently, processors/systems that can run multiple software threads have received increasing attention as a means of boosting overall performance. In this paper, we first characterize the workloads of video decoding, encoding, and watermarking on current superscalar architectures, and then we characterize the same workloads using the recently-announced Hyper-Threading Technology. Our goal is to provide a better understanding of performance improvements in multimedia applications on processors with Hyper-Threading Technology.

Figure 1 shows a high-level view of Hyper-Threading Technology and compares it to a dual-processor system. In the first implementation of Hyper-Threading Technology, one physical processor exposes two logical processors. Similar to a dual-core or dual-processor system, a processor with Hyper-Threading Technology appears to an application as two processors. Two applications or threads can be executed in parallel. The major difference between systems that use Hyper-Threading Technology and dual-processor systems is the

different amounts of duplicated resources. In today's Hyper-Threading Technology, only a small set of the microarchitecture state is duplicated[1], while the front-end logic, execution units, out-of-order retirement engine, and memory hierarchy are shared. Thus, compared to processors without Hyper-Threading Technology, the die-size is increased by less than 5% [7]. While sharing some resources may increase the latency of some single-threaded applications, the overall throughput is higher for multi-threaded or multi-process applications.

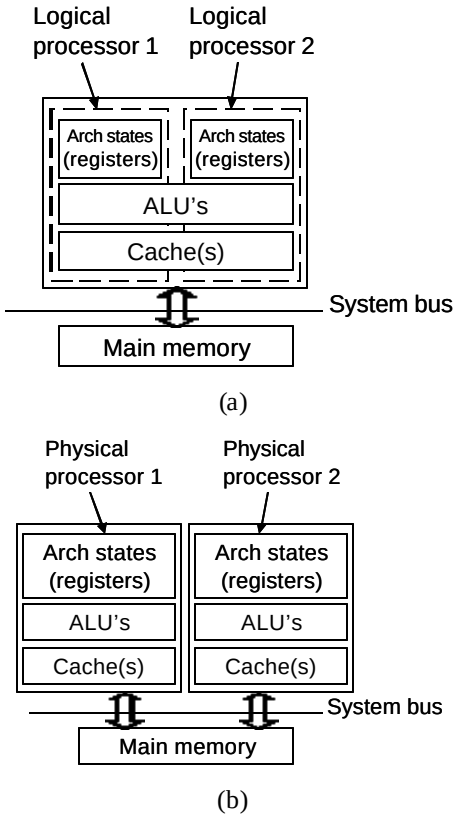


Figure 1: High-level diagram of (a) a processor with Hyper-Threading Technology and (b) a dual-processor system

This paper is organized as follows. First, we provide a brief review of the basic principles behind most current video codecs, describing the overall application behavior of video decoding/encoding/watermarking and the implications of the key kernels for current and emerging architectures. Then, we show the multi-threaded software architectures of our applications, including data-domain and functional decomposition. Additionally, we describe some potential pitfalls when developing software on processors with Hyper-Threading Technology and our techniques to avoid them. Finally, we provide some performance numbers and our observations.

MULTIMEDIA WORKLOADS

This section describes the workload characterization of selected multimedia applications on current superscalar architectures. Although the workloads are well optimized for Pentium® 4 processors, due to the inherent constitution of the algorithms, most of the modules in these workloads cannot fully utilize all the execution resources available in the microprocessor. The particular workloads we target are video decoding, encoding, and watermark detection[2], which are key components in both current and many future applications and are representative of many media workloads.

MPEG Decoder and Encoder

The Moving Pictures Expert Group (MPEG) is a standards group founded in 1988. Since its inception, the group has defined a number of popular audio and video compression standards, including MPEG-1, MPEG-2, and MPEG-4 [3]. The standards incorporate three major compression techniques: (1) predictive coding; (2) transform-based coding; and (3) entropy coding. To implement these, the MPEG encoding pipeline consists of motion estimation, Discrete Cosine Transform (DCT), quantization, and variable-length coding. The MPEG decoding pipeline consists of the counterpart operations of Variable-Length Decoding (VLD), Inverse Quantization (IQ), Inverse Discrete Cosine Transform (IDCT), and Motion Compensation (MC), as shown in Figure 2.

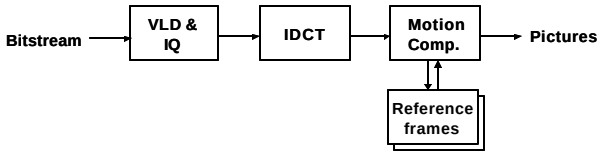


Figure 2: Block diagram of an MPEG decoder

[1] Nearly all the architectural state is duplicated, however.

[2] A digital video watermark, which is invisible and hard to alter by others, is information embedded in the video content. A watermark can be made by slightly changing the video content according to a secret pattern. For example, when just a few out of the millions of pixels in a picture are adjusted, the change is imperceptible to the human eye. A decoder can detect and retrieve the watermark by using the key that was used to create the watermark.

Table 1: MPEG decoding kernel characterization on 2 GHz Pentium® 4 processors (9 Mb/s MPEG-2, 720x480)

Kernel	IPC	UPC	MMX/SSE/SSE-2 per instructions	Cond. Branch/ instr.	Mispred. Cond./ Instr.	Mispred. Cond./ Clock	L1 misses/ Instr.	FSB activity
VLD	0.76	0.99	0.074	1/9	1/120	**1/158**	1/92	11.1%
IDCT	0.59	0.89	**0.90**	1/141	1/2585	1/4381	1/193	2.4%
MC	0.24	0.40	0.42	1/17	1/142	1/592	1/11	**30. 3%**

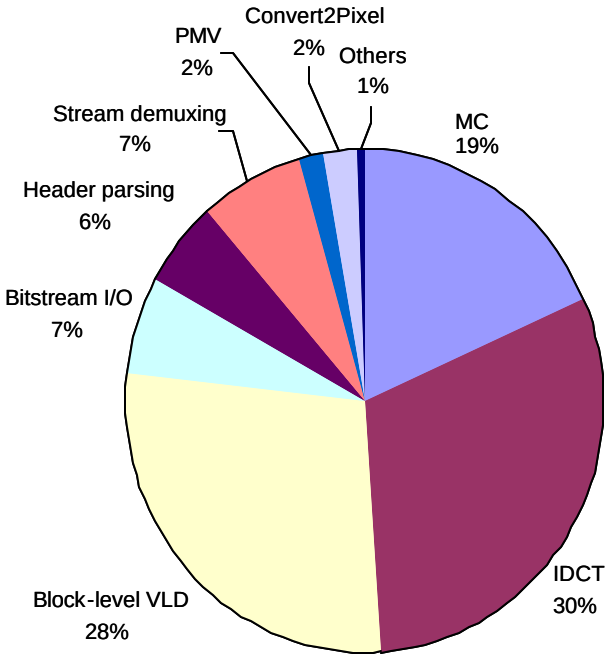


Figure 3: MPEG-2, 720x480 decoding breakdown by time on 2GHz Pentium® 4 processors

The behavior of the MPEG decoder can be highly dependent on the characteristics of the video stream being decoded. Figure 3 shows an example of the CPU time breakdown of our MPEG decoder for a typical DVD resolution video sequence. VLD, IDCT, and MC are the main components in the process. The decoder used in the study is part of the Intel Media Processing Library (MPL)[3], which was developed by Intel Labs. The software was analyzed using the Intel VTune™ Performance Analyzer on an Intel® Pentium® 4 processor with a 400 MHz system bus, an 8 KB first-level data cache, a 256 KB second-level shared instruction/data cache, and 512 MB of main memory. We run our applications on Windows* XP.

Table 1 shows a high-level analysis of the MPEG-2 decoder. The first stage of the decoding pipeline, VLD/IQ, is characterized by substantial data dependency, limiting opportunities for instruction, data, and thread-level parallelism. The kernel is entirely computation-bound, and it shows excellent performance scaling over increasing frequencies on the Pentium 4 processor. The next stage, IDCT, is also completely computation-bound. The kernel is dominated by MMX/SSE/SSE2 (Streaming SIMD Extension) operations, with interspersed register-to-register moves and stores; e.g., a sequence of `movaps`, `addps`, and `subps` is a typical recurring theme, corresponding to the well-known butterfly operation, surrounded by associated prescaling/multiply operations. Because 90% of the instructions are executed in the MMX/SSE/SSE2 unit, the integer execution unit is idle most of the time in the IDCT module[4]. The final stage of the decoding pipeline, MC, is memory intensive compared to the other modules in the pipeline. The front-side bus is busy 30% of the time in this module. Although the out-of-order execution core in the Pentium 4 processor can tolerate some memory latencies, the module shows an equal distribution of time between computation and memory latency because there are too many memory operations. All these modules are well-optimized, but still cannot utilize 100% of the execution units available in the microprocessors. While the Pentium 4 processor can execute multiple uops in one cycle, the uops retired per cycle (UPC) is only 0.74 in the MPL decoder.

MPEG encoders, similar to the decoder, consist of some MMX/SSE/SSE2 intensive modules (e.g., motion estimation, DCT) and some data-dependent modules (e.g., variable-length coding). All these modules are well optimized, but a UPC of 1.05 again indicates that the

[3] More information about the MPL can be found at http://www.intel.com/research/mrl/research/mpl/. Additionally, the MPL MPEG-2 decoder is commercially available as part of the Ligos* GoMotion* SDK.

™ VTune is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

® Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

*Other brands and names may be claimed as the property of others.

[4] See Figure 4 in [4], integer operations and floating-point/MMX/SSE/SSE2 operations are executed in different units.

encoder cannot fully utilize all the execution units available in the microprocessor.

Video Watermarking

Another application that we studied is video watermark detection [1]. Our watermark detector has two basic stages: video decoding and image-domain watermark detection. The application is optimized with MPL (as the video decoder) and the Intel IPL (for the image manipulations used during watermark detection) [5]. A UPC of 1.01 also indicates that there is room for improvement.

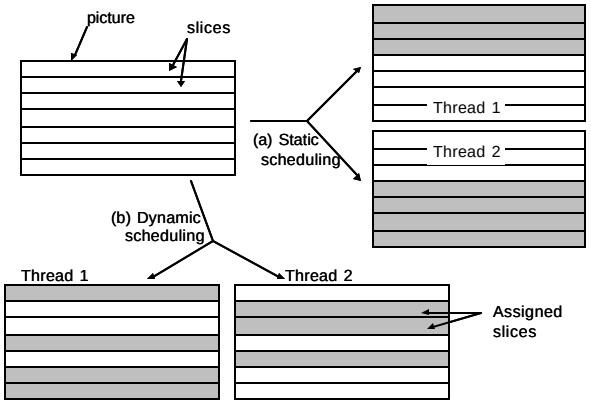


Figure 4: Two slice-based task partitioning schemes between two threads: (a) half-and-half dispatching (static scheduling); and (b) slice-by-slice scheduling (dynamic scheduling)

TASK PARTITIONING AND SCHEDULING

In general, multimedia applications, such as video encoding and decoding, exhibit not only data- and instruction-level parallelism, but also the possibility for substantial thread-level parallelism. Such workloads are good candidates for speed-up on a number of different multithreading architectures. This section discusses the trade-offs of different software multithreading methods.

Data-Domain Decomposition–Slice-Based Dispatching

As shown in Figure 4, a picture in a video bit stream can be divided into slices of macroblocks. Each slice, consisting of blocks of pixels, is a unit that can be decoded independently. Here we compare two methods to decode the pictures in parallel:

1. Half-and-half (aka static partitioning): In this method, one thread is statically assigned the first half of the picture, while another thread is assigned the other half of the picture (as shown in Figure 4 (a)). Assuming that the complexity of the first half and second half is similar, these two threads will finish the task at roughly the same time. However, some areas of the picture may be easier to decode than others. This may lead to one thread being idle while the other thread is still busy.

2. Slice-by-slice (aka dynamic partitioning): In this method, slices are dispatched dynamically. A new slice is assigned to a thread when the thread has finished its previously assigned slice. In this case, we don't know which slices will be assigned to which thread. Instead, the assignment depends on the complexity of the slices assigned. As a result, one thread may decode a larger portion of the picture than the other if its assignments are easier than those of the other thread. The execution time difference between two threads, in the worst case, is the decoding time of the last slice.

In both cases, each thread performs Variable-Length Decoding (VLD), Inverse Discrete Cosine Transform (IDCT), and Motion Compensation (MC) in its share of the pictures, macroblock by macroblock. While one thread is working on MC (memory intensive), the other thread may work on VLD or IDCT (less memory intensive). Although the partitioning does not explicitly interleave computations and memory references, on average, it better balances the use of resources.

Functional Decomposition of Video Watermark Detection

Besides data-domain decomposition, an application can also be partitioned functionally into multiple threads. For example, our video watermark detector consists of two basic stages: video decoding and watermark detection. Hence, we assign different threads to decode the video and to detect the watermark, as shown in Figure 5.

One method is to use two threads: one for video decoding and another for watermark detection, as shown in Figure 5 (b). However, this method does not have very good load balance. This is because in our video watermark detector, video decoding takes roughly one-third of the CPU time, while watermark detection takes two-thirds of the CPU time.

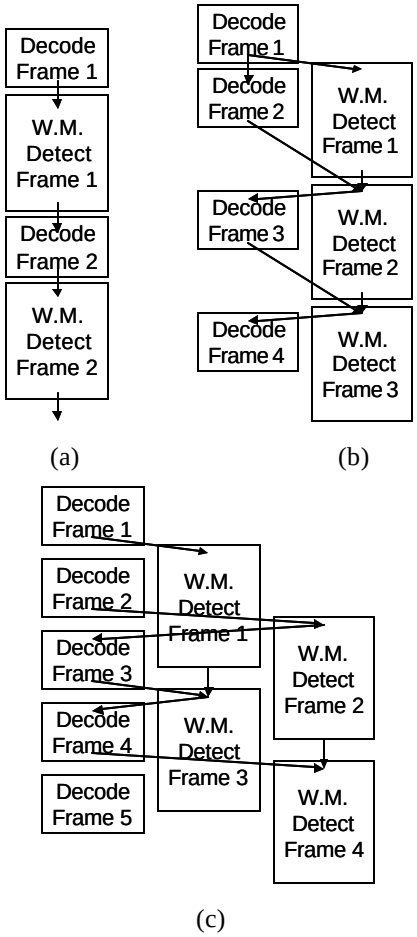


Figure 5: Three threading methods for video watermark detection: (a) single-threaded mode; (b) two-threaded mode; and (c) three-threaded mode

Because watermark detection takes twice as much computation time as video decoding, we use two threads for watermark detection for better load balancing, as shown in Figure 5 (c). While one thread decodes the video sequence, two threads work on watermark detection. The lines in the figure indicate the dependency between functional blocks. We can see that at any moment, there are at least two threads running in the three-threaded mode. In contrast to the data-domain video decoding decomposition described above, threads in this implementation are assigned to different functions.

IMPLICATIONS OF SOFTWARE DESIGN FOR HYPER-THREADING TECHNOLOGY

During the implementation of our applications on processors with Hyper-Threading Technology, we had a number of observations. In this section, we discuss some general software techniques to help readers design their applications better on systems with Hyper-Threading Technology.

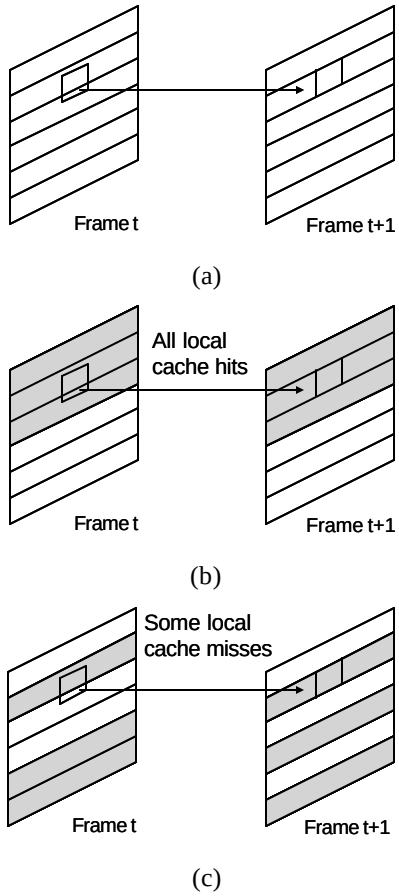


Figure 6: Cache localities, during; (a) motion compensation, in; (b) static partitioning, and in; (c) dynamic partitioning

Using Hyper-Threading Technology, performance can be lost when the loads are not balanced. Because two logical processors share resources on one physical processor with Hyper-Threading Technology, each logical processor does not get all the resources a single processor would get. When only a single thread of the application is actively working and the other thread is waiting (especially, spin-waiting), this portion of the application could have less than 100% of the resources when compared to a single processor, and it might run slower on a processor with simultaneous multithreading capability than on processors without simultaneous multithreading capability. Thus, it is important to reduce the portion in which only one thread is actively working. For better performance, effective load balancing is crucial.

The foremost advantage of the dynamic scheduling scheme (see Figure 4) is its good load balance between the two threads. Because some areas of the picture may be easier to decode than others, one thread under the static

partitioning scheme may be idle while another thread still has a lot of work to do. In the dynamic partitioning scheme, we have very good load balance. As we assign a new slice to a thread only when it has finished its previous slice, the execution time difference between the two threads, in the worst case, is the decoding time of a slice.

Because two logical processors share one physical processor, the effective sizes of the caches for each logical processor are roughly one half of the original size. Thus, it is important for multithreaded applications to target one half of the caches for each application thread. For example, when considering code size optimization, excessive loop unrolling should be avoided.

While sharing caches may be a drawback for some applications running on processors with Hyper-Threading Technology, it can provide better cache locality between the two logical processors for other applications. For example, Wang *et al*. use one logical processor to prefetch data into the shared caches to reduce a substantial amount of the memory latency of the application in the other logical processors [8]. We now illustrate the advantage of sharing caches in our application.

On dual-processor systems, each processor has a private cache. Thus, there may be a drawback to dynamic partitioning in terms of cache locality. Figure 6 illustrates the cache locality in multiple frames of video. During motion compensation, the decoder uses part of the previous picture, the referenced part of which is roughly co-located in the previous reference frame, to reconstruct the current frame. It is faster to decode the picture when the co-located part of the picture is still in the cache. In the case of a dual-processor system, each thread is running on its own processor, each with its own cache. If the co-located part of the picture in the previous frame is decoded by the same thread, it is more likely that the local cache will have the pictures that have just been decoded. Since we dynamically assign slices to different threads, it is more likely that the co-located portion of the previous picture may not be in the local cache when each thread is running on its own physical processor and cache, as shown in Figure 6 (c). Thus, dynamic partitioning may incur more bus transactions[5]. In contrast, the cache is shared between logical processors on a processor with Hyper-Threading Technology, and thus, cache localities are preserved. We obtain the best of both worlds with dynamic scheduling: there is load balancing between the threads, and there is the same effective cache locality as for static scheduling on a dual-processor system.

RESULTS

This shows some performance numbers and analysis of our applications on multithreading architectures. In general, our results show that Hyper-Threading Technology offers a cost-effective performance improvement (7%-18%) for multithreading without doubling hardware cost (see Figure 7) as in dual-processor systems.

Our Hyper-Threading Technology system has an experimental 1.7GHz Intel Pentium® 4 processor with Hyper-Threading Technology capability, which is a pre-production prototype, running Windows* XP. The processor has a 512KB second-level cache, but no third-level cache. To contrast the performance with single-thread performance on the system experimentally in lab setting, we disable the support of Hyper-Threading Technology from the CPU, motherboard, BIOS, and the operating system. Our dual-processor system has two 1.7GHz Intel Xeon™ processors, each of which has a 256KB second-level cache and a 1MB third-level cache, running Windows XP. To measure single-thread performance on the dual-processor system, we disable one physical processor and run a single-thread version of the application. The relative speed between Hyper-Threading Technology systems and dual-processor systems is not measured in our experiment.

To measure the performance of the encoder, we use five 720x480 YVU 4:2:0 benchmark sequences. To measure the performance of the decoder, we use one 640x480, three 704x480, three 720x480, one 1280x720, and two 1920x1080 MPEG-2 sequences. Moreover, three 704x480 MPEG-2 sequences are used to measure the performance of the video watermark detectors. The speed-ups are sequence dependent, but within a small variation. We report only the average numbers in Figure 7.

[5] On dual-processor systems, an alternative method of keeping cache locality in dynamic scheduling is to dispatch slices to one thread from top-down and slices to the other thread from bottom-up. However, it is hard to generalize the method for four-way or eight-way multi-processor systems. In this paper, we did not show the results of this method.

® Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

* Other brands and names may be claimed as the property of others

™ Xeon is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Data-Domain Decomposition

This section describes the performance of the data-domain decomposition of the video decoding described earlier.

First, Figure 7 (b) shows that it is better to use the dynamic scheduling method than the static scheduling method on a processor with Hyper-Threading Technology, as it is very important to have a balanced load. Because resources are shared between the logical processors, the relative performance of each logical processor can be less than 1.0 compared to an equivalent processor without simultaneous multithreading capability. When only one thread is busy, the overall throughput is less than that of a single processor. To have the best performance, it is important to have a balanced workload between threads. Hence, the dynamic scheme is better than static scheduling.

On the other hand, Figure 7 (b) shows that the static scheduling method is better than the dynamic scheduling method on a dual-processor system. It is faster to decode the picture when the co-located parts of the pictures are still in the cache. As mentioned earlier, although dynamic scheduling has better load balance, co-located parts of the pictures may not be decoded by the same processor when using dynamic scheduling. This scheduling scheme incurs more bus transactions, as shown in Table 2, with the result that the overall speed using dynamic scheduling is slower.

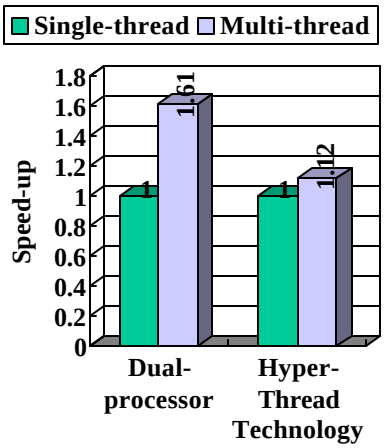
Compared to dual-processor systems, processors with Hyper-Threading Technology have the advantage of sharing the second-level cache between two logical processors. Even when the same logical processor does not decode the co-located part of the reference picture, that part of the picture can still be read from the shared second-level cache. Table 3 shows that the numbers of bus activities are similar between static scheduling and dynamic scheduling. In this case, the overall speed of dynamic scheduling is faster because the workload is

Table 2: The numbers of front-side bus (FSB) data activities per second between static scheduling and dynamic scheduling on a dual-processor system

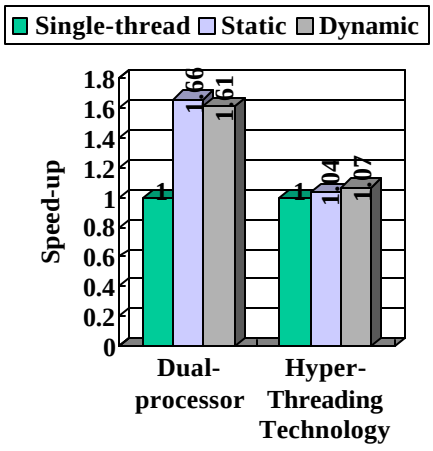
Event	Static scheduling	Dynamic scheduling
FSB_data_activity	8,604,511	12,486,051

Table 3: The numbers of FSB data activities per second between static scheduling and dynamic scheduling on a processor with Hyper-Threading Technology

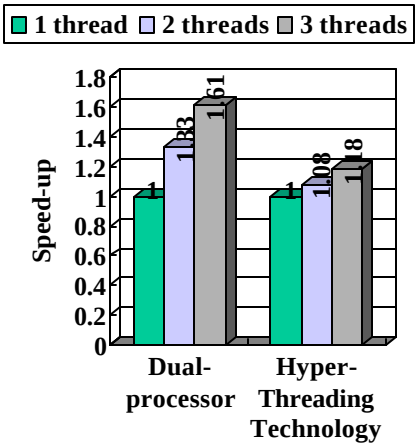
Event	Static scheduling	Dynamic scheduling
FSB_data_activity	8,474,022	8,536,838



(a)



(b)



(c)

Figure 7: Performance of; (a) our video encoder; (b) our video decoder; and (c) our watermarking detection with software configurations

Table 4: The workload characteristics of our applications on single-threaded processors and processors with Hyper-Threading Technology

Event	MPEG encoding		MPEG decoding		Video watermarking	
	Single-thread	Hyper-threading	Single-thread	Hyper-threading	Single-thread	Hyper-threading
Clockticks (Millions)	13,977	11,688	7,467	6,687	23,942	20,162
Instructions retired (Millions)	11,253	11,674	3,777	3,921	17,728	17,821
Uops retired (Millions)	14,735	15,539	5,489	5,667	24,120	24,333
MMX/SIMD uops retired (Millions)	6,226	6,220	1,119	1,120	5,334	5,341
IPC (instructions per clock)	0.80	1.00	0.51	0.59	0.74	0.88
UPC (uops per clock)	1.05	1.33	0.74	0.85	1.01	1.21
Trace cache misses (Millions)	20.8	29.0	13.3	24.1	7.6	13.3
First-level cache misses (Millions)	132	145	132	166	510	638
Bus utilization	8.5%	8.5%	14.7%	16.4%	14.2%	22.3%

better balanced.

Functional Decomposition

Here, we describe the performance of the video watermark detection functional decomposition described earlier in Figure 5. Figure 7 (c) shows the performance comparisons. 2-thread denotes one video-decoding thread and one watermark detection thread, and 3-thread denotes one video-decoding thread and two watermarking threads (see Figure 5 (c)). Similar to the results of the video decoder, better performance is obtained with better balanced workloads.

Overall Performance Characteristics

As mentioned earlier, different modules have been interleaved in the application to utilize more execution resources in the machine at a given time. Hence, it is hard to break down the workload characteristics in individual modules. Rather, it is better to consider the application as a whole.

As shown in Table 4, although the numbers of instructions retired and cache misses (e.g., trace and first-level) increase in both applications after threading, because of threading overhead and capacity misses in each thread, the overall application performance still increases. To verify that resource utilization is better balanced on a processor with Hyper-Threading Technology, we compare UPC for single-threaded and multi-threaded applications. UPC increases from 1.05 to 1.33 in video encoding, from 0.78 to 0.85 in video decoding, and from 1.01 to 1.21 in watermark detection, confirming the more efficient resource utilization possible with Hyper-Threading Technology. (These numbers include the overhead of thread synchronization; however, this overhead is relatively small, being on the order of 0.5% for watermark detection, approximately 3-4% for video decoding, and 4-5% for video encoding.)

POWER CONSUMPTION ISSUES

In this paper, we have mainly discussed methods to improve the application throughput on processors with Hyper-Threading Technology. In addition to throughput, power consumption is also an important performance factor for the next generation of processors. This is especially true for battery-run mobile systems, in which the average power consumption for a given fixed application is a crucial parameter to consider for the evaluation of the overall performance of the system.

In this section, we show that Hyper-Threading Technology can not only improve system throughput but can also save energy for applications with fixed duties. As an introduction to this new research topic, we give some hints on how to design "power-aware" applications on processors with Hyper-Threading Technology and we show the first results of this ongoing work.

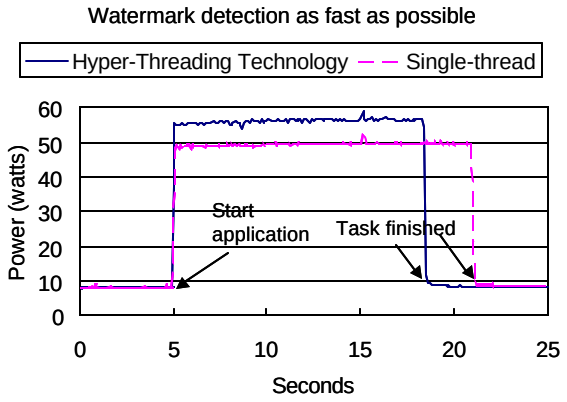


Figure 8: Measured power consumption of our watermark detector on a processor with Hyper-Threading Technology and a normal system at the same frequency and voltage

In various situations, Hyper-Threading Technology consumes additional power while improving the performance, as shown in Figure 8. When idle, the execution units in Intel Pentium® 4 processors consume less power because of clockgating [2]. Hyper-Threading Technology makes the execution units busier, and thus, they consume slightly more power. The graphs also show that the task finishes earlier on a system with Hyper-Threading Technology. Because the task finishes in fewer cycles, the overall energy consumption is slightly less on a system with Hyper-Threading Technology even with the same voltage and frequency. This is because powering up additional execution units for two simultaneous threads is more economical than powering the whole pipeline with fewer execution units to run serial threads.

In the case of real-time applications[6], where we need only a fixed amount of throughput, we can reduce the frequency and the voltage. As Hyper-Threading Technology increases the throughput, and we have more spare cycles, we can further reduce the frequency and the voltage. Because the active power consumption is proportional to frequency*(voltage)^2, we can have a cubic effect on energy saving.

Nonetheless, a common thread scheduling pitfall in multithreading real-time applications can reduce the overall energy gain on the system with Hyper-Threading Technology. Figure 9 (a) shows a common, but less than optimal, multithreading method of the watermark detection application–the watermark detector is active immediately after the video frame is decoded. Due to a large cycle period, there may be no overlapping between two threads (see Figure 5 (b)). While Figure 9 (b) has the same cycle period as Figure 9 (a), by delaying the starting time of the second thread, we increase the overlapping period of two threads. That is, we queue the tasks and dispatch together to maximize the overlap. In this case, the halted period in CPU is increased. Because powering up additional execution units for two simultaneous threads is more economical and the physical processor consumes less power when it is halted (or when both logical processors are halted), Figure 9 (b) consumes less energy. (In our real-time watermark detector, the measured CPU power is 22.8 watts vs. 23.6 watts[7].) The key is to overlap the busy cycles of one logical processor with those of the other.

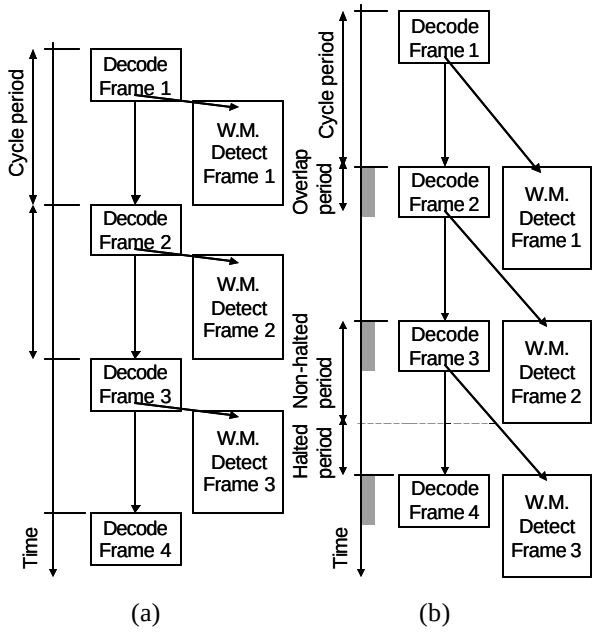


Figure 9: Two different methods of multithreading real-time applications. (a) uses more energy than (b)

CONCLUSION

In this paper we explained how typical media applications can benefit from Hyper-Threading Technology. From the increases in UPCs, we have observed that Hyper-Threading Technology can increase the utilization of processor resources by 15 to 27%, even for well-optimized multimedia applications. The results given in this paper also show that it is possible to benefit from Hyper-Threading Technology to save power when executing a fixed task.

Moreover, it has been shown that it is crucial to reach an optimal load balancing for an efficient implementation on Hyper-Threading Technology. This can usually be done for media applications exploiting both data and functional decompositions. Such partitioning, especially with a dynamic scheduling scheme, benefits in most cases from the fact that, unlike in symmetric multiprocessor systems,

® Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

[6] Real-time in this work means that applications need to perform some tasks periodically, while throughput-oriented applications just finish all the tasks as fast as possible.

[7] Here, we use average power as the indicator for energy saving. In real-time applications, power saving and energy saving can be used interchangeably.

threads share the cache in a processor with Hyper-Threading Technology.

Finally, the results show that for complex media applications running on Hyper-Threading Technology, in which multiple threads typically interact together and access memory concurrently, the thread synchronization issues and the overall data and functional partitioning are more important than the individual function characteristics.

ACKNOWLEDGMENTS

We acknowledge the exceptional efforts of the people at the Intel Nizhny Novgorod Lab in developing the encoder/decoder used in this study, especially Valery Kuriakin. Additionally, we thank Doug Carmean, Mike Upton, Per Hammarlund, Russell Arnold, Shihjong Kuo, George K. Chen, and Stephen Gunther for their help in setting up our Hyper-Threading Technology hardware and software environments and for valuable discussions during this work.

REFERENCES

[1] E. Debes, M. Holliman, W. Macy, Y.-K. Chen, and M. Yeung, "Computational Analysis and System Implications of Video Watermarking Applications," in *Proceedings of SPIE Conference on Security and Watermarking of Multimedia Contents IV*, Jan. 2002.

[2] S. Gunther, F. Binns, D. Carmean, and J. Hall, "Managing the Impact of Increasing Microprocessor Power Consumption," *Intel Technology Journal, Q1 2001*.

[3] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*, MA: Kluwer, 1997.

[4] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The Microarchitecture of the Pentium® 4 Processor," *Intel Technology Journal, Q1 2001*.

[5] Intel Corp., *Intel® Performance Library Suite*, (available on-line: http://developer.intel.com/software/products/perflib/index.htm)

[6] Intel Corp., *Intel® Pentium® 4 Processor Optimization Reference Manual*, Order Number: 248966 (also available on-line: http://developer.intel.com/design/pentium4/manuals/24896604.pdf)

[7] D. Marr, et al., "Hyper-Threading Technology Microarchitecture and Performance," *Intel Technology Journal, Q1 2002*.

[8] H. Wang, P. Wang, R. D. Weldon, S. Ettinger, H. Saito, M. Girkar, S. Liao, and J. Shen, "Speculative Precomputation: Exploring the Use of Multithreading Technology for Latency," *Intel Technology Journal, Q1 2002*.

AUTHORS' BIOGRAPHIES

Yen-Kuang Chen is a researcher in the Media Systems Lab, Microprocessor Research, Intel Labs. His research interests include video compression and processing, architecture and algorithm design in multimedia computing, video and graphics hardware design, and performance evaluation. He received a Ph.D. in electrical engineering from Princeton University. His e-mail is yen-kuang.chen@intel.com.

Matthew J. Holliman is a researcher in the Media Systems Lab, Microprocessor Research, Intel Labs. His research interests include media and Internet technology, focusing on content delivery and protection. His e-mail is matthew.holliman@intel.com.

Eric Debes is a researcher in the Media Systems Lab, Microprocessor Research, Intel Labs. His research interests include media coding, processing, communications and content protection as well as microarchitecture design and parallelism in computer architecture. He received an M.S. degree in electrical and computer engineering from Supélec, France, an M.S. degree in electrical engineering from the Technical University Darmstadt, Germany and a Ph.D. degree from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland. His e-mail is Eric.Debes@intel.com.

Sergey Zheltov is a project manager in Microprocessor Research, Intel Labs. His research interests include media compression and processing, software and platforms architecture, signal processing, high-order spectra. He received a Diploma in radio-physical engineering and MS degree in theoretical and mathematical physics from Nizhny Novgorod State University. His e-mail is Sergey.Zheltov@intel.com.

Alexander Knyazev is a software engineer in Microprocessor Research, Intel Labs. His research interests include video compression and processing, multimedia software architecture, platforms architecture, test, rough set and fuzzy logic theories. He received a Master's Degree of Applied Mathematics and Computer Science from Nizhny Novgorod State University. His e-mail is Alexander.Knyazev@intel.com.

Stanislav Bratanov is a software engineer in Microprocessor Research, Intel Labs. His research interests include multi-processor software platforms, operating system environments, and platform-dependent media data coding. He graduated from Nizhny Novgorod

State University, Russia. His e-mail is Stanislav.Bratanov@intel.com.

Roman Belenov is a software engineer in Microprocessor Research, Intel Labs. His research interests include video compression and processing, multimedia software architecture and wireless networking. He received a Diploma in physics from Nizhny Novgorod State University. His e-mail is Roman.Belenov@intel.com

Ishmael Santos is a Hardware Engineer for Power and Trace Technologies in Software Solutions Group, Intel Corporation. His interests include computer architecture with an emphasis on microprocessor power consumption and performance. Ishmael received his B.S. in Electrical Engineering and Computer Science from the University of California, Los Angeles. His e-mail is Ishmael.F.Santos@intel.com.