A

Seminar report

On

Hyper-Threading

Submitted in partial fulfillment of the requirement for the award of degree
Of Mechanical

SUBMITTED TO:

www.studymafia.org

SUBMITTED BY:

www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

Preface

I have made this report file on the topic **Hyper-Threading;** I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

CONTENTS

INTRODUCTION

Hyper-Threading (HT) Technology is ground breaking technology from Intel that allows processors to work more efficiently. This new technology enables the processor to execute two series, or threads, of instructions at the same time, thereby improving performance and system responsiveness while delivering performance headroom for the future.

Intel Hyper-Threading Technology improves the utilization of onboard resources so that a second thread can be processed in the same processor. Hyper-Threading Technology provides two *logical* processors in a single processor package.

Hyper-Threading Technology offers:

- improved overall system performance
- increased number of users a platform can support
- improved reaction and response time because tasks can be run on separate threads
- increased number of transaction that can be executed
- compatibility with existing IA-32 software

Code written for dual-processor (DP) and multi-processor (MP) systems is compatible with Intel Hyper-Threading Technology-enabled platforms. A Hyper-Threading Technology-enabled system will automatically process multiple threads of multi-threaded code.

UTILIZATION OF PROCESSOR RESOURCES

Intel Hyper-Threading Technology improves performance of multi-threaded applications by increasing the utilization of the on-chip resources available in the Intel® NetBurst™ microarchitecture. The Intel NetBurst microarchitecture provides optimal performance when executing a single instruction stream. A typical thread of code with a typical mix of Intel® IA-32-based instructions, however, utilizes only about 35 percent of the Intel NetBurst microarchitecture execution resources.

By adding the necessary logic and resources to the processor die in order to schedule and control two threads of code, Intel Hyper-Threading Technology makes these underutilized resources available to a second thread of code, offering increased throughput and overall system performance.

Hyper-Threading Technology provides a second *logical processor* in a single package for higher system performance. Systems containing multiple Hyper-Threading Technology-enabled processors further improve system performance, processing two code threads for each processor.

HYPER-THREADING TECHNOLOGY IMPROVES PERFORMANCE

Intel Hyper-Threading Technology offers better performance improvement as additional processors are added. Multi-processor systems with Hyper-Threading Technology can outperform multi-processor systems without Hyper-Threading Technology.

MULTI-THREADED APPLICATIONS

Virtually all contemporary operating systems (including Microsoft Windows* and Linux*) divide their workload up into processes and threads that can be independently scheduled and dispatched. The same division of workload can be found in many high-performance applications such as database engines, scientific computation programs, engineering-workstation tools, and multi-media programs.

To gain access to increased processing power, programmers design these programs to execute in dual-processor (DP) or multiprocessor (MP) environments. Through the use of symmetric multiprocessing (SMP), processes and threads can be dispatched to run on a pool of several physical processors. With multi-threaded, MP-aware applications, instructions from several threads are simultaneously dispatched for execution by the processors' core. In processors with Hyper-Threading Technology, a single processor core executes these two threads concurrently, using out-of-order instruction scheduling to keep as many of its execution units as possible busy during each clock cycle.

Intel® NetBurst™ Microarchitecture Pipeline

Without Hyper-Threading Technology enabled, the Intel NetBurst microarchitecture processes a single thread through the pipeline. Recall that the typical mix of typical instructions only utilizes about 35% of the resources in the Intel NetBurst microarchitecture.

MULTIPROCESSOR PERFORMANCE ON A SINGLE PROCESSOR

The Intel Xeon processor introduces a new technology called Hyper-Threading (HT) that, to the operating system, makes a single processor behave like two logical processors. When enabled, the technology allows the processor to execute multiple threads simultaneously, in parallel within each processor, which can yield significant performance improvement. We set out to quantify just how much improvement you can expect to see. The current Linux symmetric multiprocessing (SMP) kernel at both the 2.4 and 2.5 versions was made aware of Hyper-Threading, and performance speed-up had been observed in multithreaded benchmarks

This article gives the results of our investigation into the effects of Hyper-Threading (HT) on the Linux SMP kernel. It compares the performance of a Linux SMP kernel that was aware of Hyper-Threading to one that was not. The system under test was a multithreading-enabled, single-CPU Xeon. The benchmarks used in the study covered areas within the kernel that could be affected by Hyper-Threading, such as the scheduler, low-level kernel primitives, the file server, the network, and threaded support.

The results on Linux kernel 2.4.19 show Hyper-Threading technology could improve multithreaded applications by 30%. Current work on Linux kernel 2.5.32 may provide performance speed-up as much as 51%.

HYPER-THREADING SPEEDS LINUX

Intel's Hyper-Threading Technology enables two logical processors on a single physical processor by replicating, partitioning, and sharing the resources within the Intel NetBurst microarchitecture pipeline.

Hyper-Threading support in the Xeon processor

The Xeon processor is the first to implement Simultaneous Multi-Threading (SMT) in a general-purpose processor. To achieve the goal of executing two threads on a single physical processor, the processor simultaneously maintains the context of multiple threads that allow the scheduler to dispatch two potentially independent threads concurrently.

The operating system (OS) schedules and dispatches threads of code to each logical processor as it would in an SMP system. When a thread is not dispatched, the associated logical processor is kept idle.

When a thread is scheduled and dispatched to a logical processor, LP0, the Hyper-Threading technology utilizes the necessary processor resources to execute the thread.

When a second thread is scheduled and dispatched on the second logical processor, LP1, resources are replicated, divided, or shared as necessary in order to execute the second thread. Each processor makes selections at points in the pipeline to control and process the threads. As each thread finishes, the operating system idles the unused processor, freeing resources for the running processor.

The OS schedules and dispatches threads to each logical processor, just as it would in a dual-processor or multi-processor system. As the system schedules and introduces threads into the pipeline, resources are utilized as necessary to process two threads.

Hyper-Threading support in Linux kernel 2.4

Under the Linux kernel, a Hyper-Threaded processor with two virtual processors is treated as a pair of real physical processors. As a result, the scheduler that handles SMP should be able to handle Hyper-Threading as well. The support for Hyper-Threading in Linux kernel 2.4.x began with 2.4.17 and includes the following enhancements:

Kernel performance measurement

To assess the effects of Hyper-Threading on the Linux kernel, we measured the performance of kernel benchmarks on a system containing the Intel Xeon processor with HT. The hardware was a single-CPU, 1.6 GHz Xeon MP processor with SMT, 2.5 GB of RAM, and two 9.2 GB SCSI disk drives. The kernel under measurement was stock version 2.4.19 configured and built with SMP enabled. The existence of Hyper-Threading support can be seen by using the command cat /proc/cpuinfo to show the presence of two processors, processor 0 and processor 1. Note the ht flag in Listing 1 for CPUs 0 and 1. In the case of no Hyper-Threading support, the data will be displayed for processor 0 only.

Linux kernel benchmarks

To measure Linux kernel performance, five benchmarks were used: LMbench, AIM Benchmark Suite IX (AIM9), chat, dbench, and tbench. The LMbench benchmark times various Linux application programming interfaces (APIs), such as basic system calls, context switching latency, and memory bandwidth. The AIM9 benchmark provides measurements of user application workload. The chat benchmark is a client-server workload

modeled after a chat room. The dbench benchmark is a file server workload, and tbench is a TCP workload. Chat, dbench, and tbench are multithreaded benchmarks, while the others are single-threaded benchmarks.

Effects of Hyper-Threading on Linux APIs

The effects of Hyper-Threading on Linux APIs were measured by LMbench, which is a microbenchmark containing a suite of bandwidth and latency measurements. Among these are cached file read, memory copy (bcopy), memory read/write (and latency), pipe, context switching, networking, filesystem creates and deletes, process creation, signal handling, and processor clock latency. LMbench stresses the following kernel components: scheduler, process management, communication, networking, memory map, and filesystem. The low level kernel primitives provide a good indicator of the underlying hardware capabilities and performance.

To study the effects of Hyper-Threading, we focused on latency measurements that measure time of message control, (in other words, how fast a system can perform some operation). The latency numbers are reported in microseconds per operation.

Effects of Hyper-Threading on Linux single-user application workload

The AIM9 benchmark is a single user workload designed to measure the performance of hardware and operating systems . Most of the tests in the benchmark performed identically in Hyper-Threading and non-Hyper-Threading, except for the sync file operations and Integer Sieves. The three operations, Sync Random Disk Writes, Sync Sequential Disk Writes, and Sync Disk Copies, are approximately 35% slower in Hyper-Threading. On the other hand, Hyper-Threading provided a 60% improvement over non-Hyper-Threading in the case of Integer Sieves.

Effects of Hyper-Threading on Linux multithreaded application workload

To measure the effects of Hyper-Threading on Linux multithreaded applications, we use the chat benchmark, which is modeled after a chat room. The benchmark includes both a client and a server. The client side of the benchmark will report the number of messages sent per second; the number of chat rooms and messages will control the workload. The workload creates a lot of threads and TCP/IP connections, and sends and receives a lot of messages. It uses the following default parameters:

Number of chat rooms = 10
Number of messages = 100
Message size = 100 bytes
Number of users = 20

By default, each chat room has 20 users. A total of 10 chat rooms will have 20x10 = 200 users. For each user in the chat room, the client will make a connection to the server. So since we have 200 users, we will have 200 connections to the server. Now, for each user (or connection) in the chat room, a "send" thread and a "receive" thread are created. Thus, a 10-chat-room scenario will create 10x20x2 = 400 client threads and 400 server threads, for a total of 800 threads. But there's more.

Each client "send" thread will send the specified number of messages to the server. For 10 chat rooms and 100 messages, the client will send 10x20x100 = 20,000 messages. The server "receive" thread will receive the corresponding number of messages. The chat room server will echo each of the messages back to the other users in the chat room. Thus, for 10 chat rooms and 100 messages, the server "send" thread will send 10x20x100x19 or 380,000 messages. The client "receive" thread will receive the corresponding number of messages.

Effects of Hyper-Threading on Linux multithreaded file server workload

The effect of Hyper-Threading on the file server was measured with dbench and its companion test, tbench. dbench is similar to the well known NetBench benchmark from the Ziff-Davis Media benchmark program, which lets you measure the performance of file servers as they handle network file requests from clients. However, while NetBench requires an elaborate setup of actual physical clients, dbench simulates the 90,000 operations typically run by a NetBench client by sniffing a 4 MB file called client.txt to produce the same workload. The contents of this file are file operation directives such as SMBopenx, SMBclose, SMBwritebraw, SMBgetatr, etc. Those I/O calls correspond to the Server Message Protocol Block (SMB) that the SMBD server in SAMBA would produce in a netbench run. The SMB protocol is used by Microsoft Windows 3.11, NT and 95/98 to share disks and printers.

In our tests, a total of 18 different types of I/O calls were used including open file, read, write, lock, unlock, get file attribute, set file attribute, close, get disk free space,

get file time, set file time, find open, find next, find close, rename file, delete file, create new file, and flush file buffer.

dbench can simulate any number of clients without going through the expense of a physical setup. dbench produces only the filesystem load, and it does no networking calls. During a run, each client records the number of bytes of data moved and divides this number by the amount of time required to move the data. All client throughput scores are then added up to determine the overall throughput for the server. The overall I/O throughput score represents the number of megabytes per second transferred during the test. This is a measurement of how well the server can handle file requests from clients.

dbench is a good test for Hyper-Threading because it creates a high load and activity on the CPU and I/O schedulers. The ability of Hyper-Threading to support multithreaded file serving is severely tested by dbench because many files are created and accessed simultaneously by the clients. Each client has to create about 21 megabytes worth of test data files. For a test run with 20 clients, about 420 megabytes of data are expected. dbench is considered a good test to measure the performance of the elevator algorithm used in the Linux filesystem. dbench is used to test the working correctness of the algorithm, and whether the elevator is aggressive enough. It is also an interesting test for page replacement.

tbench

tbench is another file server workload similar to dbench. However, tbench produces only the TCP and process load. tbench does the same socket calls that SMBD would do under a netbench load, but tbench does no filesystem calls. The idea behind tbench is to eliminate SMBD from the netbench test, as though the SMBD code could be made fast. The throughput results of tbench tell us how fast a netbench run could go if we eliminated all filesystem I/O and SMB packet processing. tbench is built as part of the dbench package.

Hyper-Threading support in Linux kernel 2.5.x

Linux kernel 2.4.x was made aware of HT since the release of 2.4.17. The kernel 2.4.17 knows about the logical processor, and it treats a Hyper-Threaded processor as two physical processors. However, the scheduler used in the stock kernel 2.4.x is still considered naive for not being able to distinguish the resource contention problem between two logical processors versus two separate physical processors.

Consider a system with two physical CPUs, each of which provides two virtual processors. If there are two tasks running, the current scheduler would let them both run on a single physical processor, even though far better performance would result from migrating one process to the other physical CPU. The scheduler also doesn't understand that migrating a process from one virtual processor to its sibling (a logical CPU on the same physical CPU) is cheaper (due to cache loading) than migrating it across physical processors.

HT-aware passive load-balancing:

The IRQ-driven balancing has to be per-physical-CPU, not per-logical-CPU. Otherwise, it might happen that one physical CPU runs two tasks while another physical CPU runs no task; the stock scheduler does not recognize this condition as "imbalance." To the scheduler, it appears as if the first two CPUs have 1-1 task running while the second two CPUs have 0-0 tasks running. The stock scheduler does not realize that the two logical CPUs belong to the same physical CPU.

"Active" load-balancing:

This is when a logical CPU goes idle and causes a physical CPU imbalance. This is a mechanism that simply does not exist in the stock 1:1 scheduler. The imbalance caused by an idle CPU can be solved via the normal load-balancer. In the case of HT, the situation is special because the source physical CPU might have just two tasks running, both runnable. This is a situation that the stock load-balancer is unable to handle, because

running tasks are hard to migrate away. This migration is essential -- otherwise a physical CPU can get stuck running two tasks while another physical CPU stays idle.

HT-aware task pickup:

When the scheduler picks a new task, it should prefer all tasks that share the same physical CPU before trying to pull in tasks from other CPUs. The stock scheduler only picks tasks that were scheduled to that particular logical CPU.

HT-aware affinity:

Tasks should attempt to "stick" to physical CPUs, not logical CPUs.

HT-aware wakeup:

The stock scheduler only knows about the "current" CPU, it does not know about any sibling. On HT, if a thread is woken up on a logical CPU that is already executing a task, and if a sibling CPU is idle, then the sibling CPU has to be woken up and has to execute the newly woken-up task immediately.

EACH PROGRAM HAS A MIND OF ITS OWN

The OS and system hardware not only cooperate to fool the user about the true mechanics of multi-tasking, but they cooperate to fool each running program as well. While the user thinks that all of the currently running programs are being executed simultaneously, each of those programs thinks that it has a monopoly on the CPU and memory. As far as a running program is concerned, it's the only program loaded in RAM and the only program executing on the CPU. The program believes that it has complete use of the machine's entire memory address space and that the CPU is executing it continuously and without interruption. Of course, none of this is true. The program actually shares RAM with all of the other currently running programs, and it has to wait its turn for a slice of CPU time in order to execute, just like all of the other programs on the system.

A few terms: process, context, and thread

Before continuing our discussion of multiprocessing, let's take a moment to unpack the term "program" a bit more. In most modern operating systems, what users normally call a program would be more technically termed a **process**. Associated with each process is a **context**, "context" being just a catch-all term that encompasses all the information that completely describes the process's current state of execution (e.g. the contents of the CPU registers, the program counter, the flags, etc.).

Processes are made up of **threads**, and each process consists of at least one thread: the main thread of execution. Processes can be made up of multiple threads, and each of these threads can have its own local context in addition to the process's context, which is shared by all the threads in a process. In reality, a thread is just a specific type of stripped-down process, a "lightweight process," and because of this throughout the rest of this article I'll use the terms "process" and "thread" pretty much interchangeably.

Even though threads are bundled together into processes, they still have a certain amount of independence. This independence, when combined with their lightweight nature, gives them both speed and flexibility. In an SMP system like the ones we'll discuss in a moment, not only can different processes run on different processors, but different threads from the same process can run on different processors.

IMPLEMENTING HYPER-THREADING

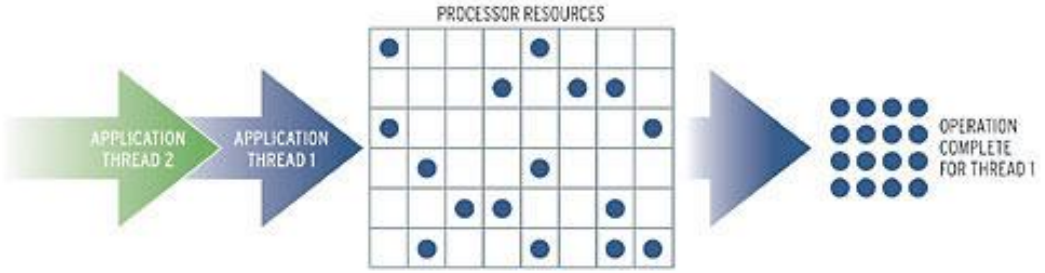
Although hyper-threading might seem like a pretty large departure from the kind of conventional, process-switching multithreading done on a single-threaded CPU, it actually doesn't add too much complexity to the hardware. Intel reports that adding hyper-threading to their Xeon processor added only %5 to its die area.

Intel's Xeon is capable of executing at most two threads in parallel on two logical processors. In order to present two logical processors to both the OS and the user, the Xeon must be able to maintain information for two distinct and independent thread contexts. This is done by dividing up the processor's microarchitectural resources into three types: replicated, partitioned, and shared.

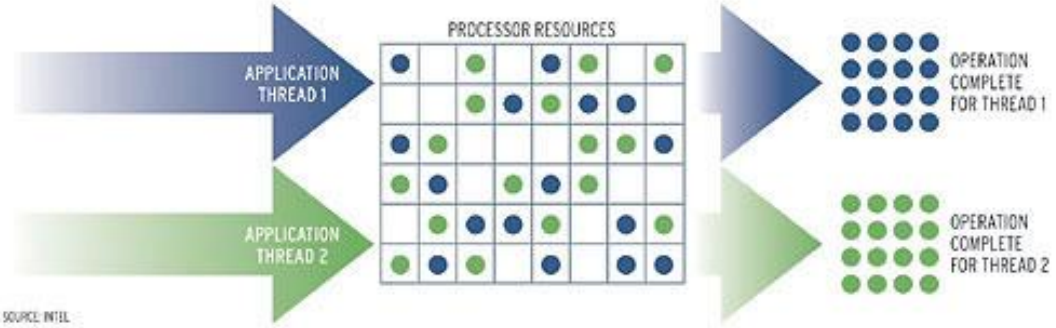
HOW HYPERTHREADING WORKS

THE AIM OF INTEL'S new hyperthreading technology is to use a CPU's resources more efficiently to get the job done. As Brian Fravel, Intel's desktop marketing manager, puts it, "Today's processors are one cook with one pan; the new chip is still one cook, but with two pans." A cook who works on two dishes at the same time should be able to complete work on both in less time overall.

WITHOUT HYPERTHREADING: When a multithreaded application or multiple applications send requests to the CPU, they line up and get taken care of sequentially, so one app stays idle until the other's task is complete, leaving some of the CPU's resources unused.



WITH HYPERTHREADING: The processor and operating system can see multiple requests from apps and feed them to the CPU so that it handles both simultaneously. This should yield a more-efficient use of resources and permit quicker completion of both applications' tasks.



SOURCE: INTEL

WHAT HYPERTHREADING CAN (AND CAN'T) DO FOR YOU

Hyper threading . The word alone sounds like a marketing tactic, an esoteric feature designed to convince OEMs and end users to upgrade to the latest and greatest Intel-based systems. And to some extent, hyperthreading is exactly that. With returns diminishing on increased clock speeds and memory caches for the average user, But at the same time, hyperthreading solves a real computing problem. And its implementation in the Pentium 4 and succeeding generations of desktop processors could spark a quiet revolution in how software is designed

What Does It Do?

In a nutshell, hyperthreading, or simultaneous multithreading, makes a single physical processor appear to an operating system as two logical processors. Most software today is threaded -- that is, instructions are split into multiple streams so that multiple processors can act on them. With hyperthreading, a single processor can handle those multiple streams, or threads, as if it were two processors. Without hyperthreading, a chip would have to process the two threads sequentially rather than simultaneously. Or it might perform time-slicing, in which a processor rapidly shifts between running various threads at a fixed interval, Intel spokesperson George Alfs said.

In the Chips

The first Intel chips to take advantage of hyperthreading were Xeon server processors. But in November 2002, Intel brought hyperthreading to the desktop with its 3.06 GHz Pentium 4. "We will be providing this technology in additional SKUs over time," Alfs told NewsFactor. "We intend to have hyperthreading in a majority of our desktop Pentium 4 processors." chief research officer Peter Kastner said he expected such a move from the company. "Intel has hinted that it will push hyperthreading technology throughout its Pentium line, making it available to most PC buyers, not just at the top end," he told NewsFactor.

Software Support

Of course, microprocessor improvements mean nothing without software that can take advantage of them. For hyperthreading, software support is in the early stages. "Buying the Pentium 4 with hyperthreading will be an increasingly smart decision over the life of the desktop," Kastner said. "While many applications are not optimized for hyperthreading today, we expect that as new releases come out, hyperthreading will become a standard feature." For software to benefit from hyperthreading, the program must support multithreaded execution -- that is, it must allow two distinct tasks to be executed at the same time, vice president Steve Kleynhans told NewsFactor.

Two Paths

There are two ways to achieve this goal. The first is to write an application that is specifically designed to be multithreaded. The second is to run two independent applications at the same time. "People are running multiple, mixed loads of applications on their desktops," Kleynhans said. "Many of those are background tasks." Both Home and Professional Editions support hyperthreading out of the box. Numerous other multithreaded applications also can get a boost from Intel's hyperthreading feature, particularly content creation applications, such as Photoshop, and video and audio encoding.

INTEL INNOVATION COULD DOUBLE CHIP POWER

Intel showed off a new chip technology will allow one chip to act like two.

Called "hyperthreading," the new technology essentially takes advantage of formerly unused circuitry on the Pentium 4 that lets the chip operate far more efficiently--and almost as well as a dual-processor computer. With it, a desktop can run two different applications simultaneously or run a single application much faster than it would on a standard one-processor box.

"It makes a single processor look like two processors to the operating system," said Shannon Poulin, enterprise launch and disclosure manager at Intel. "It effectively looks like two processors on a chip."

Paul Otellini, general manager of the Intel Architecture Group, demonstrated the hyperthreading technology at the Intel Developer's Forum. They showed off a 3.5GHz Pentium 4 running the computer game "Quake 3" and managing four different video streams simultaneously. The Pentium 4 demonstration didn't depend on Hyper-Threading; instead, it came out as part of Intel's effort to show how consumers and software developers will continue to need faster PCs. "There are a lot of tremendous applications on the horizon that will consume the MIPS (millions of instructions per second)," Otellini said. "Gigahertz are necessary for the evolution and improvement of computing."

Technically, hyperthreading takes advantage of additional registers--circuits that help manage data inside a chip--that come on existing Pentium 4's but aren't used. Through these registers, the processor can handle more tasks at once by taking better advantage of its own resources. The chip can direct instructions from one application on its floating-point unit, which is where the heavy math is done, and run parts of another application through its integer unit. A chip with hyperthreading won't equal the computing power of two Pentium 4's, but the performance boost is substantial, Poulin said. A workstation with hyperthreaded Xeon chips running Alias-Wavefront, a graphics application, has achieved a 30 percent

improvement in tests, he said. Servers with hyperthreaded chips can manage 30 percent more users.

Will developers climb aboard?

The open question is whether software developers will latch onto the idea. Software applications will need to be rewritten to take advantage of hyperthreading, and getting developers to tweak their products can take an enormous amount of time. Intel, for instance, has been working for well over a year to get developers to rewrite their programs to take full advantage of the features of the Pentium 4, which has been out for approximately nine months. The company even changed the migration program to speed the process of optimizing Pentium III applications for the Pentium 4.

Still, to date, only 30 applications have been enhanced to take full advantage of the Pentium 4, according to Louis Burns, vice president and general manager of the Desktop Platforms Group at Intel. But more are on the way, he said. Otellini acknowledged that recruiting developers will take time.

"The real key is going to be to get the applications threaded, and that takes a lot of work," he said. Nonetheless, adopting the technology to server and workstations applications should be fairly easy if the application already runs on dual-processor systems, other Intel officials said. "Thread your applications and drivers and OSes to take advantage of this relatively free performance," Otellini asked developers during his speech.

Hyperthreading, which will appear in servers and workstations in 2002 and desktops in 2003, is part of an overall Intel strategy to find new ways to squeeze more performance out of silicon. For years, the company has largely relied on boosting the clock speed and tweaking parts of the chip's architecture to eke out gains. The performance gains to be achieved from boosting the clock speed, however, are limited. In all practicality, most users won't experience that much realistic difference between a 1GHz computer and one that contains a 2GHz chip, according to, among others, Dean McCarron, an analyst at Mercury Research.

Ideally, hyperthreading, which has been under development for four and a half years, will show meatier benefits. An individual could play games while simultaneously downloading multimedia files from the Internet with a computer containing the technology, Poulin predicted. Hyperthreaded chips would also be cheaper than dual-processor computers. "You only need one heat sink, one fan, one cooling solution," he said, along with, of course, one chip. Chips running hyperthreading have been produced, and both Microsoft's Windows XP and Linux can take advantage of the technology, according to Poulin. Computers containing a single hyperthreaded chip differ from dual-processor computers in that two applications can't take advantage of the same processor substructure at the same time. "Only one gets to use the floating point at a single time," Poulin said.

On other fronts, Intel on Tuesday also unveiled Machine Check Architecture, which allows servers to catch data errors more efficiently. The company will also demonstrate McKinley for the first time. McKinley is the code name for the next version of Itanium, Intel's 64-bit chip that competes against Sun's UltraSparc. McKinley is due in demonstration systems by the end of this year.

CONCLUSION

Intel Xeon Hyper-Threading is definitely having a positive impact on Linux kernel and multithreaded applications. The speed-up from Hyper-Threading could be as high as 30% in stock kernel 2.4.19, to 51% in kernel 2.5.32 due to drastic changes in the scheduler run queue's support and Hyper-Threading awareness. Today with Hyper-Threading Technology, processor-level threading can be utilized which offers more efficient use of processor resources for greater parallelism and improved performance on today's multi-threaded software.

REFERENCES

- www.google.com
- www.wikipedia.com
- www.studymafia.org