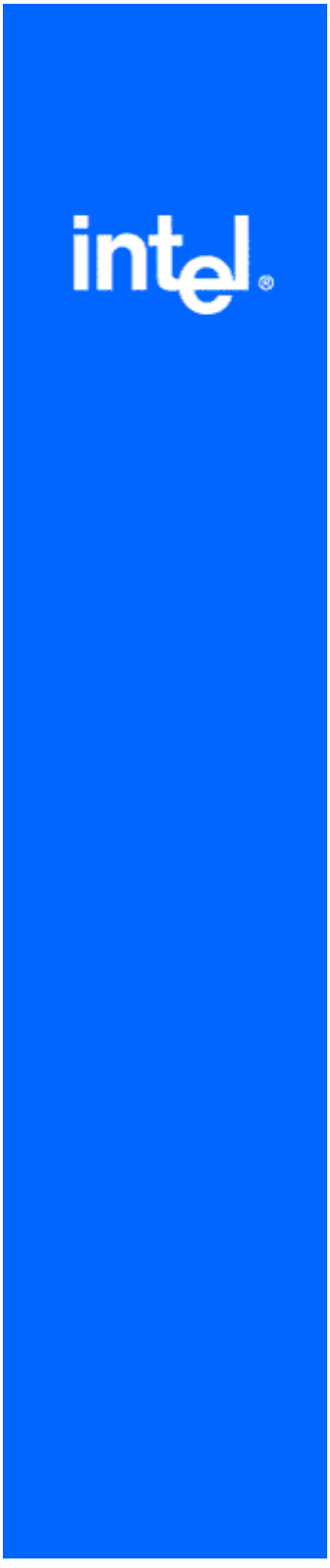

The Power of Multithreading for Embedded Infrastructure Applications

1/24/2005

Abstract

This paper discusses software application threading. It is intended for those who would like to learn the basics of threading as well as how threading can maximize performance of platforms utilizing Intel® architecture processors.

Disclaimers

THE INFORMATION IS FURNISHED FOR INFORMATIONAL USE ONLY, IS SUBJECT TO CHANGE WITHOUT NOTICE, AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY INTEL CORPORATION. INTEL CORPORATION ASSUMES NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR INACCURACIES THAT MAY APPEAR IN THIS DOCUMENT OR ANY SOFTWARE THAT MAY BE PROVIDED IN ASSOCIATION WITH THIS DOCUMENT. THIS INFORMATION IS PROVIDED "AS IS" AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THE USE OF THIS INFORMATION INCLUDING WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, COMPLIANCE WITH A SPECIFICATION OR STANDARD, MERCHANTABILITY OR NONINFRINGEMENT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit http://www.intel.com/performance/resources/limits.htm.

Hyper-Threading Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and a HT Technology enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. See www.intel.com/homepage/land/hyperthreading_more.htm for additional information.

Legal Notices

1 Overview

This paper discusses application threading, summarizing where applications benefit from multithreading as well as tradeoffs to consider when implementing threads. Methods for implementing threads are discussed, including pros and cons of each.

The intent of this paper is to raise software developers' awareness of how their applications can benefit from the evolution of Intel® architecture (IA-32) processors, which are based on multiple logical and physical processor cores.

This paper is not intended as an in depth tutorial for thread development. Instead, it provides a basic understanding of threading, its benefits and important role for maximizing performance of applications using IA-32. Several very helpful threading references are provided for developers who want to learn more about threading.

A brief overview of processor architecture, as they relate to threading, is provided to set a foundation for understanding how these processor platforms affect system performance. This overview becomes more relevant as the information in this paper compares single and multithreaded code across various IA-32 processors.

After reading this paper, application developers may wish to learn more about threading and determine where threads can increase performance of their applications.

2 Multithreaded System Architectures

Processors are designed for purposes of specific system architectures ranging from single processor to multiple processor architectures. Multithreaded system architectures are capable of executing multiple processes or threads simultaneously, and can be designed with multiple processors or processors that contain multiprocessing capabilities. In general, the more threads that a system can execute simultaneously, the higher the system performance. Figure 1 shows a graphic of different multithreading technologies.

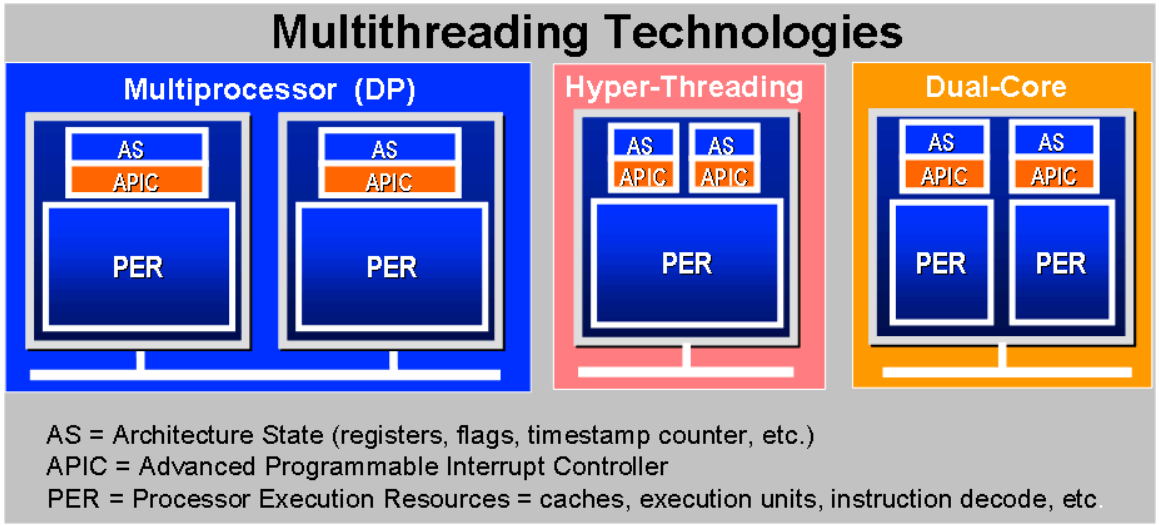

Figure 1 Multithreading Technologies

The leftmost diagram depicts a dual-processor system consisting of two physical processors sharing a common processor side bus, connected to a memory controller. The center diagram

shows a single processor capable of maintaining two processor states and serving two program flows with a single processor execution unit. This concept is the basis for Intel Hyper-Threading Technology (HT Technology). The rightmost diagram shows a dual core implementation which is basically two processor systems on a single silicon die.

2.1 Uni-Processor (UP)

Only one physical processor exists on the system bus. The system bus connects the processor to the chipset, typically containing a memory controller and interfaces.

2.2 Dual-Processor (DP)

Two physical processors share the same system bus. Threads that need the same processor resources will run better on separate processors. The processes run completely independent from each other without requiring a context switch to get at the resources of the processor.
Note: In IA terms a DP system runs in SMP (symmetric multiprocessing) mode.

2.3 Multiprocessor (MP)

Multiple physical processors (four or more) share the same system bus. MP extends benefits of DP with increased scaling.

2.4 Hyper-Threading Technology (HT Technology)

HT Technology, based on Intel Netburst® microarchitecture, is two logical processors contained within one physical processor core. HT Technology supports Simultaneous Multithreading Technology (SMT), which allows different threads to run simultaneously on different execution units within one physical processor. This is accomplished by sharing, partitioning, and replicating certain processor resources.

Provides:

- Simultaneous multi-tasking on the single core
- Appears as two independent processors to the software
- Increases utilization of the execution unit of the processor.
- Accelerates performance of multithreaded applications

Processors resources:

- Shared – cache, out-of-order execution engine, branch predictors, control logic, and system bus
- Partitioned – Registers, Advanced Programmable Interrupt Controller, Timestamp Counter, Instruction re-order buffer, load/store buffer, queues
- Replicated – Architecture states, instruction pointers, renaming logic, ITLB, return stack

Performance:
Speedup depends on how logical processors are utilized (coding and resource dependencies). See section: Performance Estimations.

In general, multithreaded applications do not require modification to run on processors with HT Technology enabled. However, software that is specifically optimized for HT Technology should increase processor performance. See section: HT Technology Programming.

Note. If it is suspected that multiple threads running on an HT Technology system will degrade over running a single thread, thread spawning should be prevented. This can happen if there is thread contention of processor resources.

2.5 Multi-Core (MC or CMP)

Multi-core is also referred to as Chip Multiprocessing (CMP). Multiple independent execution cores and pipelines contained within one packaged processor assembly.

In general, software optimized for MP or HT Technology also works well with CMP. However, software that is specifically optimized for CMP technology should increase processor performance and decrease processor power consumption.

2.6 Dual Core (DC)

Dual core processors are multi-core processors that contain only two execution cores.

3 Multiprocessing

Multiple processes run at the same time, allowing different programs or different threads of any program to run on different processors concurrently. This is referred to as symmetric multiprocessing (SMP)

3.1 Asymmetrical

One or more processors are exclusively dedicated to specific tasks, such as running the OS. This configuration may not be performance optimized because on some machines the OS processors were found to be running at 100% capacity while the user-assigned processors were practically idle.

3.2 Symmetrical

Symmetrical multiprocessing (SMP) is the favored architecture for IA32 systems, and is considered the norm today for better balancing of the processing load. The “symmetry” refers to symmetric access to memory. SMP uses at least two processors. Each additional processor adds incrementally to the system management administration of the OS, thus each additional processor will contribute less and less to the overall system performance. Shared-memory multiprocessor architectures are divided into two categories: SMP and Non-Uniform Memory Access (NUMA). For purposes related to this paper, only SMP is discussed.

4 Multithreading

Applications can be multithreaded for turnaround, which is maximized with Symmetric Multiprocessing (SMP). When a single-threaded program is waiting for something to happen, such as waiting for user input or an I/O device to become ready, the processor is being underutilized and turnaround is decreased. Programs can be written so that there are several running software routines, called threads, which are processed independently of each other.

4.1 Threads and the Operating System

When an application executes, the OS creates a process for the application. Multithreaded applications create additional threads, which run within the same process. All of the threads share the code and data segments, but have separate instruction pointers and stacks. Figure 2 depicts the process threads environment.

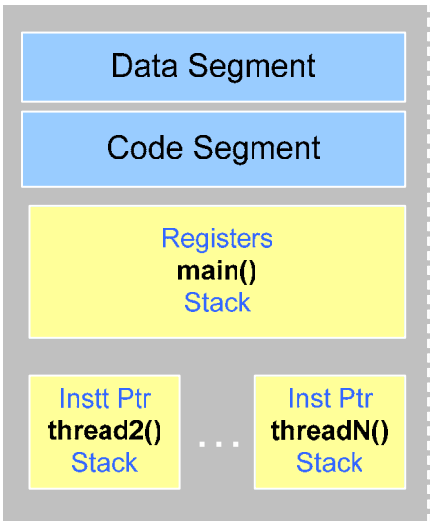

Figure 2 – Process Threads

5 BIOS and Operating System Requirements

Operating systems are written with specific support for threading, as well as multithreaded system architectures, such as MP, HT Technology, and MC.

BIOS and Operating systems support:

- The BIOS and OS must support multithreading.
- The BIOS must detect the presence of multiple physical and logical processors, such as with HT Technology.
- The OS must retrieve the information of system capabilities from the BIOS.
- The OS must differentiate between physical and logical processors in order for the kernel to correctly schedule the processes and threads. Since logical processors share some common processor resources, they are not as efficient as multiple separated physical processors. The OS should therefore attempt to use all physical processors before resorting to the available logical processors.
- Operating systems that support multithreaded system architectures require affinity support for binding processes to a physical processor, which reduces cache thrashing. Most modern operating systems support affinity and it has been supported in Linux since the 2.2 kernel.
- The OS requires HT Technology or MC support for simultaneous execution of instructions (threads) on each processor.

6 Application Benefits of MP, HT Technology, MC

Processors that are designed with multiple cores (logical or physical) have certain benefits depending on how many applications run at the same time on the system and whether the applications are single or multithreaded.

6.1 Serial Applications

Although serial (single threaded) applications restrict the performance potential of the multithreaded processor architectures, serial applications can still benefit from these architectures. Here are some expectations of serial application performance:

- Multithreaded technologies are not expected to make a given single-threaded application execute faster when executing alone. For example, if HT Technology is enabled and there is only one process in the run state, then the effective result is the same throughput as if HT

Technology is not enabled, which is the same as executing on processors without multithreading capabilities.

- When two or more unrelated applications are multitasked under HT Technology, the overall system throughput may improve as a result of HT Technology because up to two applications can execute simultaneously, one on each logical processor. Note, the same theory exists for MC and MP architectures. However, the performance potential of the system increases as fewer processor resources are shared.

6.2 Multithreaded Applications

Developers design multithreaded applications to maximize processor performance capabilities.

- Multiple threads on UP can make an application more responsive. When multiple threads are executed on UP, performance is increased because the OS can execute one thread while another thread is idle waiting for data or other input. However, only one thread can execute at the same instant in time.

- With HT Technology capable processors, the OS can dispatch two threads from a multithreaded application to run concurrently, one on each of the two logical processors. This is accomplished by the HT Technology processor using out-of-order instruction execution and resource sharing.

- Multithreaded applications gain even more performance on MC processors than on HT Technology processors because MC processors use duplicated processor execution resources for each core.

- Multithread applications gain the most performance advantage in MP environments because processes and threads can be dispatched by the OS to run on a pool of several physical processors simultaneously.

- Multithreaded applications written for UP will run without changes on any of the multithreaded technologies.

6.3 Performance Estimations

Duplication of processor resources increases the performance potential of the processor. HT Technology benchmark tests[1] show some server applications can experience a 30 percent gain in performance if both logical units are 100% utilized. The greatest gain comes when a thread may stall, such as when a thread waits for data from memory or from a disk drive. In this case, the second thread can fully utilize the execution unit. For MC architecture, the performance boost of applications can approach a theoretical limit of 100% over a single processor when all physical cores are 100% utilized. Lastly, MP architecture can increase application performance asymptotically with the number of processors.

Applications that are written to scale well on SMP architectures are not required to be changed to see the benefits from HT Technology or MC.

7 HT Technology Programming

HT Technology is intended to help applications that have multiple threads and/or processes. When HT Technology is used, applications must inquire to the operating system to obtain the number of logical and physical processors in the system. This is not only important for

[1] Hyper-Threading Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and a HT Technology enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. See www.intel.com/homepage/land/hyperthreading_more.htm for additional information.

determining how the code should run, but it could be important for licensed software where the number of physical processors determines the cost of the application, such as the operating system seat licenses.

There isn't a significant performance difference between a multithreaded application and multiple single threaded processes as far as HT Technology is concerned, so long as the CPU utilization isn't bunched up within a single process. If the application realizes benefits from running on a true SMP system (two or more physical CPU's) then with HT Technology the same workload should see a performance improvement over a non-HT Technology enabled SMP system.

Spin Wait Loops are efficient in an MP system but compete for resources in an HT Technology environment. The main problem with spin-waits on HT Technology is that the “spinner” ties-up a CPU without doing any useful work.

- In multithreaded applications the spin wait loops are normally executed in the thread API functions.
- When a spin wait loop is used outside of a thread API and it is suspected that a thread will release a lock within an OS quantum of time, use the “PAUSE” instruction inside the spin wait loop. If longer than OS quanta of time, use OS synchronization techniques

For more information see Intel’s posting of this online paper: Methods to Utilize Intel’s HyperThreading Technology with Linux*. This paper includes information as to how to determine HT Technology support, how to determine the number of logical processors per physical processor and determining the APIC ID.

8 Threading Basics

8.1 Threading Benefits (When to thread)

Multithreading allows for more efficient use of system resources. When threading an application, it helps to characterize the problem according to a data or functional decomposition model:

8.1.1 Data Decomposition

In data parallelization, the same instructions or operations are applied repeatedly to different data. Compute-intensive loops are good candidates for data parallelism. Also, problems that scale with the amount of data are often good candidates for data parallelism..

8.1.2 Functional Decomposition

In functional decomposition, independent work encapsulated in functions is mapped to threads that execute simultaneously. Problems that scale with the number of independent tasks are good candidates for functional decomposition.

8.2 Threading Methods

The most common threading approaches are library-based and compiler based methods.

8.2.1 Library-Based

Library-based methods are best suited for *functional decomposition*. Win32 and POSIX thread API’s are examples of library based methods. Library-based methods require the programmer to create and synchronize threads explicitly.

8.2.1.1 Explicit

Explicit threading requires the developer to manually write all required code to manage threads that interface to a specific library. This code is responsible for creating and freeing the resources

associated with each thread, as well as synchronizing and managing shared thread resources. Explicit threading requires in-depth threading knowledge of thread management and the thread API functions.

8.2.1.2 Strengths of Explicit Threading

- Explicit threading allows for fine control of threads, processing based on thread function or status of specific variables.
- The priority of individual threads can be changed.
- Explicit threading allows developers to write their own scheduler for fine control over threading operations.

8.2.1.3 Limitations of Explicit Threading

- Explicit threading requires more code modification than compiler-based threading methods.
- Explicit threading alters single threaded implementation forever. Once multithreaded, always multithreaded. Can't be turned on/off.
- Explicit threading requires the developer to guess at the optimum number of threads to use, and then test performance to narrow to the best number.
- Explicit threading requires much more time to write than a compiler-base implementation.
- Explicit threading uses a more complex implementation, which makes it more difficult to maintain and could potentially lead to more bugs.
- Explicit threading is not as portable as compiler-based implementation.

8.2.2 Compiler-Based

Compiler-base methods are best suited for *data parallelization*. OpenMP is an example of a compiler based method.

8.2.2.1 OpenMP

OpenMP is a standardized threading interface in which the programmer uses pragma's to describe parallelism to the compiler. The compiler is responsible for creating the threads. OpenMP is supported by the Intel® compilers. OpenMP is a powerful, portable, and simple means of threading programs. The OpenMP specification standardizes an extensive set of pragma's and directives for controlling key aspects of parallel processing.

8.2.2.2 Strengths of OpenMP

- OpenMP does not require single-threaded code to be changed for threading. Compiler directives (pragma's) are added, keeping serial code intact.
- Code can be compiled as multithreaded or single-threaded by simply enabling or disabling the compiler OpenMP switch respectively.
- OpenMP does a lot of the threading work that would require much more time to write in an explicit threading implementation.
- The simpler implementation of OpenMP provides for easier maintainability and fewer bugs in the compiled code.
- OpenMP code is portable to any system with an OpenMP-compliant compiler.

8.2.2.3 Limitations of OpenMP

- Loops that are flow dependent (results are used by other iterations of the loop) will not work correctly with OpenMP. OpenMP does not determine the correctness of code. Thus, this situation cannot be detected. Therefore, the developer must understand the dependencies of a loop before assigning OpenMP to parallelize it.
- OpenMP does not provide the fine control mechanisms (e.g., thread priority) of explicit threading methods like Pthreads.

8.2.3 Which Method is Right for Your Application?

For many applications, OpenMP is sufficient. These applications have the characteristic of easy data decomposition.

Explicit threading is preferred for processing that scales with the amount of data or the number of independent tasks. Clean functional decomposition makes a program better suited to explicit threading.

The good news is that this is not necessarily a one or the other choice. Applications can use both OpenMP and explicit threading (native threading APIs), allowing the best of both worlds to be realized.

8.3 Approach to Threading (How to thread)

Converting a serial application to take advantage of multi-threading requires an approach which uses the generic development cycle, consisting of these six phases: Analysis, Design, Implementation, Debug, Test, and Tune. There are threading tools that help with code analysis, debug, and tune. See appendix: Software Development Tools.

- Analysis – Use the Intel® VTune™ Performance Analyzer to identify the performance hotspots in the critical path. Then determine the appropriate threading model.
- Design – Determine changes required to accommodate a threading paradigm (data restructuring, code restructuring) by characterizing the application threading model (data or task-level parallelization) Identify which variables must be shared and if the current design structure is a good candidate for sharing.
- Implementation – Convert the design into code based on the selected threading model. Consider coding guidelines based on the processor architecture, such as the use of the PAUSE instruction within spin-wait loops. Make use of the of the available software development tools.
- Debug – Use dynamic analysis and the Intel® Thread Checker.
- Test – Compare performance against the serial application performance. Ensure correctness of operation with Thread Checker.
- Tune – Do not begin tuning until correct parallel design is ensured. Use Thread Profiler to determine thread related performance issues. Modify thread implementation as necessary.

9 Conclusion

The latest processor architectures are designed to support simultaneous execution of processor instructions. Intel expects to continue support of HT Technology and CMP architectures for the foreseeable future. Serial applications benefit from these architectures when the applications can be multitasked. Application performance is maximized when multiple execution streams can be processed simultaneously. Multithreading allows applications to execute multiple execution streams, taking advantage of the processor multithreading features and capturing performance that would otherwise be unrealized. Although multithread programming adds a level of complexity to program design and implementation, fortunately there are plenty of resources (services, tools, and training) available to ease the effort.

10 Appendix - Intel® Software Development Tools

10.1 Intel® Compilers

Accelerate software performance using Intel® compilers. Compatible with the tools developers use, Intel compilers plug into popular development environments and feature source and binary compatibility with widely-used compilers. Every compiler purchase includes one year of Intel® Premier Support, providing updates, technical support and expertise for the Intel® architecture.

10.2 Intel® Performance Libraries

Increase your application performance and spend less time coding by using high-performance libraries from Intel. These libraries provide highly optimized functions that take full advantage of Intel® processors so you can achieve maximum application performance and reduce development time.

10.2.1 Intel® Integrated Performance Primitives

The Intel® Integrated Performance Primitives (IPP) is the highly optimized Intel software library for audio, video, imaging, cryptography, speech recognition, and signal processing functions and codec's.

10.2.2 Intel® Math Kernel Library

The Intel® Math Kernel Library is the flagship Intel product for high-performance math software. This library contains highly optimized, thread-safe, mathematical functions for engineering, scientific and financial applications

10.3 Intel® VTune™ Performance Analyzers

Intel® VTune™ Analyzers help locate and remove software performance bottlenecks by collecting, analyzing, and displaying performance data from the system-wide level down to the source level.

10.4 Threading Tools

Intel® Threading Tools simplify the development and maintenance of threaded applications. Adding threading to software enables you to take advantage of the performance benefits of Hyper-Threading Technology included in Intel® Pentium 4 and Intel® Xeon™ processors. Intel Threading Tools help you to quickly find and fix threading errors, and to tune the performance of threaded code.

10.4.1 Intel® Thread Checker

This tool automatically locates bugs in threaded software that might otherwise go undetected. Traditional debugging tools require you to guess where to place traps in the code in hopes of finding useful information about a bug. Intel Thread Checker eliminates this guesswork and pinpoints the location of errors to help quickly analyze and correct them.

10.4.2 Thread Profiler

This tool monitors your application's execution to detect threading performance issues, including thread overhead and synchronization impact. Thread Profiler provides graphical displays to help analyze and correct threading bottlenecks for Win32* or OpenMP* threaded software.

11 Appendix - Intel® Threading Resources

11.1 Threading Services

For those who wish to use and learn from the experience of software developers and for actual hands on help with threading their applications, Intel offers the Intel® Parallel Application Center (PAC). Located in Champaign, Illinois, the Intel® PAC is a state-of-the-art lab equipped with the latest Intel hardware, advanced parallel performance tools, and a staff of highly-skilled applications engineers. The PAC offers a lab environment where Independent Software Vendors (ISVs) can enhance Intel® architecture (IA)-based applications for parallel execution on multiple processors.

11.2 Thread Training

11.2.1 Developer Centers

Hyper-Threading Technology
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/hyperthreading/index.htm

Threading Developer Center
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/index.htm

Threading Knowledge Base
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/knowledgebase/index.htm

11.2.2 Community Forums

Threading on Intel Parallel Architectures
http://softwareforums.intel.com/ids/board?board.id=42

11.2.3 Documents

Threading Methodology: Principles and Practices (Document)
http://www.intel.com/software/products/threading/downloads/ThreadingMethodology.pdf

11.2.4 Online Articles

Adjusting Thread Stack Address to Improve Performance on Intel® Xeon™ Processors (Threading Development Center)
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/knowledgebase/index.htm

Developing Multithreaded Applications: A Platform Consistent Approach (Threading Development Center)
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/hyperthreading/53797.htm

Choosing Between OpenMP* and Explicit Threading Methods (Threading Development Center)
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/167238.htm

Getting Started With OpenMP* (Threading Development Center)
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/20365.htm

More Work-Sharing with OpenMP* (Threading Development Center)

http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/43669.htm

Advanced OpenMP* Programming (Threading Development Center)
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/48795.htm

Intro to Threading (Threading Development Center)
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/applying/intro/index.htm

How to Associate Logical Processors to Physical Processors (Threading Knowledge Base)
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/knowledgebase/43838.htm

How to Determine the Effectiveness of Hyper-Threading Technology with an Application
http://www.intel.com/cd/ids/developer/asmo-na/eng/20470.htm

Methods to Utilize Intel's Hyper-Threading Technology with Linux*
http://www.intel.com/cd/ids/developer/asmo-na/eng/20354.htm?prn=Y%20

Multiple Approaches to Multithreading (Threading Development Center)
http://www.intel.com/cd/ids/developer/asmo-na/eng/technologies/threading/151201.htm

11.2.5 Online Courses

Detecting Hyper-Threading Technology Enabled Processors
https://shale.intel.com/SoftwareCollege/CourseDetails.asp?courseID=62

Introduction to Hyper-Threading Technology (Threading Knowledge Base)
https://shale.intel.com/SoftwareCollege/CourseDetails.asp?courseID=65

Optimizing Performance of Multithreaded Computations
https://shale.intel.com/SoftwareCollege/CourseDetails.asp?courseID=72

Porting Solaris* Applications to Linux: Threads
https://shale.intel.com/SoftwareCollege/CourseDetails.asp?courseID=80

11.2.6 Classroom Training

Thread Programming and Hyper-Threading Technology
https://shale.intel.com/SoftwareCollege/CourseDetails.asp?courseID=1

11.3 Online Demos

11.3.1 Hyper-Threading Technology and Threading Short Demos

http://www.intel.com/business/bss/products/hyperthreading/server/demo/index.htm#software

11.3.2 Intel® Thread Checker Short Demo

http://www.intel.com/software/products/threading/downloads/ThreadCheckerDemo.htm

11.3.3 Intel® VTune™ Performance Analyzers Short Demo

http://www.intel.com/software/products/vtune/downloads/VTune_V7.htm

12 Appendix - Definitions

Affinity – There are two types of affinity, system and process affinity. System affinity indicates the number of processors that the operating system recognizes and can utilize. Process affinity indicates the processor in a multiprocessor or Hyper-Threading Technology system that a process is selected to run on.

Concurrency – Simultaneous execution of multiple structurally different application activities, such as multitasking on multiple activities. For example: computation, disk access, and network access. It reduces latency (see definition below) and improves throughput, thus reduces the time an application spends idle.

Context Switch – The current running thread is suspended and its execution state is saved. The state information belonging to the thread being switched to (next thread) is loaded and the CPU resumes execution of the next thread.

Intel Netburst® microarchitecture - Adds significant enhancements to the core architecture:

- Change in execution stages: Instruction fetch -> trace cache -> queue -> register rename -> queue -> schedule -> register read -> execute -> L1 cache store -> register write -> reorder buffer retire
- Changes in L1 code cache
- Changes to system bus speed
- Changes to maximum core speed

Latency - In general, the period of time that one component in a system is spinning its wheels waiting for another component. Latency, therefore, is wasted time.

Multitasking - The ability to execute more than one *task* at the same time, a task being a program. The terms *multitasking* and *Multiprocessing* are often used interchangeably, although Multiprocessing implies that more than one CPU is involved.

Quanta – OS Quanta are defined as discrete units of time in the operation of the OS.

Thread – A single stream of instructions. Processes themselves are therefore threads. Processes can also create threads within the process.

Throughput - The amount of data transferred from one place to another or processed in a specified amount of time. Bandwidth is also used as a basis for throughput. Throughput is also used to reference the number of jobs finished in a given time interval.

Turnaround – The time required to finish a job

Appendix - Linux 2.4.x Kernel Hyper-Threading Support Known Issues

There are known HT Technology performance issues in the v2.4 SMP Linux kernels. The issues are related to the kernel thread scheduler that can degrade performance when HT Technology is enabled. Most improvements to these issues were initially made available in the v2.5.32 kernel. There are several sources that discuss HT Technology performance implications on these Linux kernels. A couple of these resources are listed below:

- Intel® has posted "Methods to Utilize Intel's Hyper-Threading Technology with Linux*". Linux issues are discussed in section "Linux* Issues with Hyper-Threaded Technology". See this link: http://www.intel.com/cd/ids/developer/asmo-na/eng/20354.htm?prn=Y

- The v2.6 Linux kernel will address specific issues with HT Technology. Information about the v2.6 Linux kernel changes can be found at the Open Source Development Labs (OSDL) web site.

13 Appendix - Additional References

Binstock, A., & Gerber, R. (2004). *Programming with Hyper-Threading Technology,* Intel Press.

Gerber, R. (2002). *The Software Optimization Cookbook,* Intel Press.

Intel Corporation. (2004). *IA-32 Intel® Architecture Optimization Reference Manual*

Intel Corporation. *Hyper-Threading Technology.*
http://www.intel.com/business/bss/products/hyperthreading/overview.htm

Intel Corporation. *Multithreading, Hyper-threading, Multiprocessing: Now, What's the Difference?*
http://www.intel.com/cd/ids/developer/asmo-na/eng/20456.htm?page=1

OpenMP Organization. *Specifications.*
http://www.openmp.org/drupal/node/view/8?PHPSESSID=34639b10121747cff6fbcb463ebf624a

Open Source Development Labs, Inc. *Linux 2.6.0: What's New.*
http://www.osdl.org/newsroom/press_releases/2003/2003_12_18_beaverton_2_6_new.html