
Performance
Optimization
Intelligent Systems Group
Intel Corporation

Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

  UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

  Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

  The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

  Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

  Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm
- All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel's current plan of record product roadmaps.
- Intel® vPro™ Technology is sophisticated and requires setup and activation. Availabiity of features and results will depend upon the setup and configuration of your hardware, software and IT environment. To learn more visit: http://www.intel.com/technology/vpro.
- Requires activation and a system with a corporate network connection, an Intel® AMT-enabled chipset, network hardware and software. For notebooks, Intel AMT may be unavailable or limited over a host OS-based VPN, when connecting wirelessly, on battery power, sleeping, hibernating or powered off. Results dependent upon hardware, setup and configuration. For more information, visit http://www.intel.com/technology/platform-technology/intel-amt.
- No computer system can provide absolute security under all conditions. Intel® Trusted Execution Technology (Intel® TXT) requires a computer with Intel® Virtualization Technology, an Intel TXT-enabled processor, chipset, BIOS, Authenticated Code Modules and an Intel TXT-compatible measured launched environment (MLE). Intel TXT also requires the system to contain a TPM v1.s. For more information, visit http://www.intel.com/technology/security
- Intel Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, and virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit http://www.intel.com/go/virtualization
-
- Intel, the Intel logo and Inter Atom are trademarks of Intel Corporation in the United States and/or other countries.
- *Other names and brands may be claimed as the property of others.

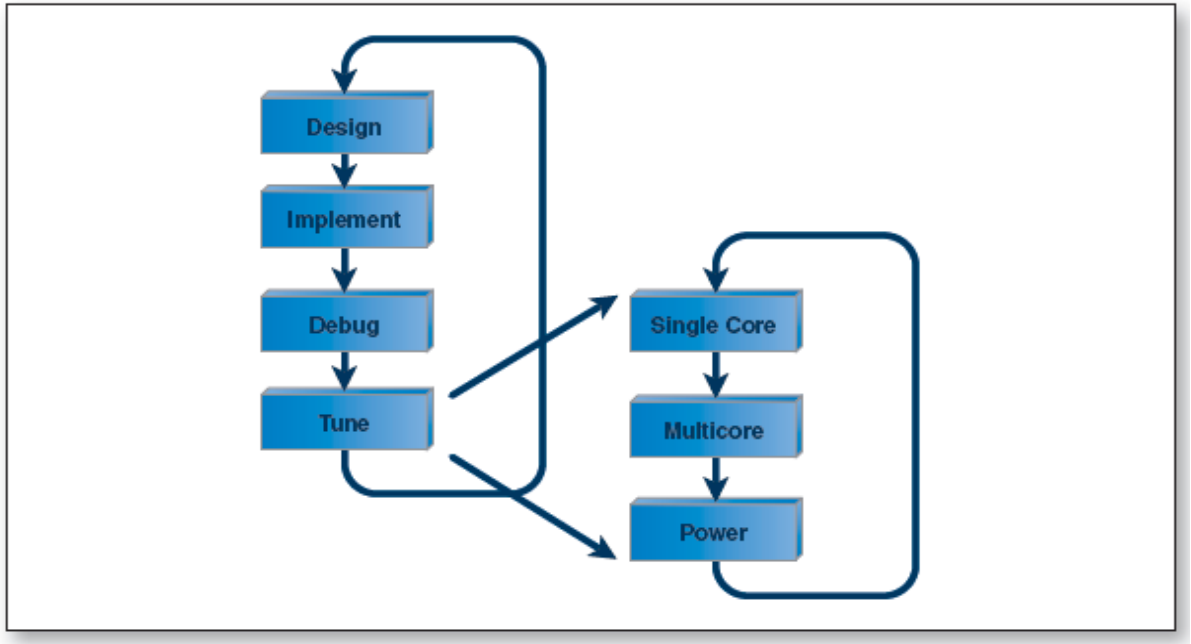
Motivation and Introduction

- Embedded developers targeting an Intel® Atom™ processors require knowledge of a performance optimization process employing the right tools and analysis techniques.
- This optimization process must integrate tuning steps for single core performance, multi-core performance, and power performance.
- The quality of tools support has a direct impact on the effectiveness of the optimization efforts.
- Performance tools that target single processor core performance provide insight into the application and how the application is behaving at the level of the microarchitecture.
- Multi-core performance tools provide insight into how the application is executing in the context of Intel® Hyper-Threading Technology (Intel® HT Technology) and multi-core processing.

Effective Optimization Depends on the Right Process and Tools

Development and Optimization Process

- A typical development cycle consists of four phases: design, implementation, debugging, and tuning.
- The development cycle concludes when performance and stability requirements are met.
- The tuning phase consists of these steps in this order: Single core optimization, multi-core optimization, power optimization.
- The first step is to analyze the system and software, and record benchmark results of the starting point. Analyze again at each tuning step.
- Stop when the required performance is achieved.



Tuning is an Iterative Process

Single Core Tuning Steps

Characterized as: Gain an understanding of the application, tune upon general performance analysis, tune specific to the Intel® Atom™ processor architecture.

Steps:

1. *Benchmark - Develop a benchmark that represents typical application* usage.
2. *Profile - Analyze and understand the architecture of the application.*
3. *Compiler optimization - Use aggressive optimizations if possible.*
4. *General microarchitecture tuning - Tune based upon insight from* general performance analysis statistics. These statistics, such as clock cycles per instruction retired, are generally accepted performance analysis statistics that can be employed regardless of the underlying architecture.
5. *Intel Atom processor tuning -Tune based on insight about known* processor "glass jaws." These include statistics and techniques to isolate performance issues specific to the Intel Atom processor.

Boundaries of Optimization

- Pareto Principle:
  - States that 80 percent of the time spent in an application is in 20 percent of the code, the "80/20 rule."
  - Prioritize optimization efforts to the areas of highest impact, the most frequently executed portions of the code.
- Amdahl's Law:
  - Provides guidance on the limits of optimization.
  - If the optimization can only be applied to 75 percent of the application, the maximum theoretical speedup is 4 times.

Amdahl's Law

Maximum theoretical speedup is defined by Amdahl's Law:

$$S_p^{\max}(n) = \frac{1}{1 - p + \frac{p}{n}}$$

Actual Speedup *n* *n = #threads, p = fraction parallel*

Speedup < #Threads : sublinear speedup

Speedup = #Threads : linear speedup

Speedup > #Threads : superlinear speedup ***

Benefits of Multi-Core

Thread programs for performance

- Execute independent tasks in parallel

Serial applications can still benefit

- Multiple applications can run simultaneously

How can this improve performance?

Improve Turnaround

Process a single task in the shortest time

Strategy:

Divide and Conquer

Improve Throughput

Process many tasks in a fixed amount of time

Strategy:

Parallelism for Performance

Taking Advantage of Multi Core

Optimal performance benefits products that are prepared for multi-core technologies

Dramatic improvements moving to multi-core hardware

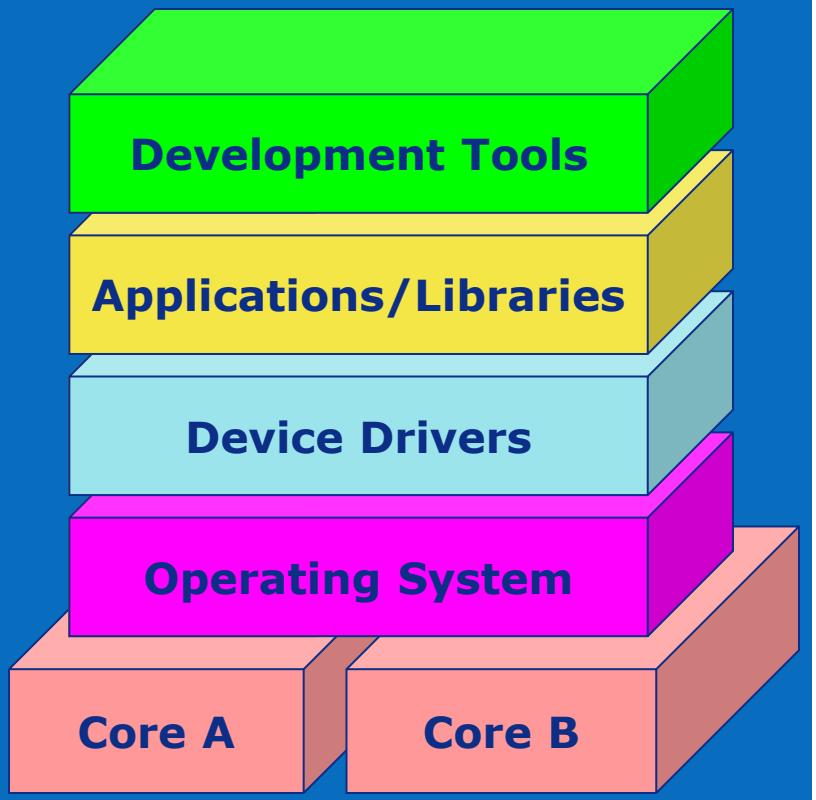
- SMP-enabled operating system and threaded applications
- Consists of multiple applications, well-suited to multitasking

Others may have to apply some elbow grease

- Perhaps the applications haven't been threaded
- Or perhaps the operating system isn't SMP-enabled

Understand your multi-core migration readiness

Multi-core Readiness Evaluation



Component	Description
Development Tools	Ensure tool support is provided on your OS of choice: threading correctness, optimal app threading, threaded libraries and device drivers
Application or Libraries	Applications are thread safe and provide threaded performance Libraries are re-entrant and threaded for performance, may need upgrade to new OS
Device Drivers	Device Drives may need to re-architecture for optimal performance and upgrade to new OS
Operating System	Take measure of the OS roadmap; ensure that your OS of choice has plans to support necessary silicon features to take advantage of Multiple Cores, support of Multi Tasking and Multi-Threading

Survey your software stack to assess readiness

Functional Decomposition

Divide computation based on natural set of independent functions/tasks

- Different tasks operate on the same data
- Assign data for each task as needed

Example:

ATM teller activities distributed among multiple tasks:

- Process encrypted communications for throughput
- Service User Interface (UI) for high interactivity
- Manage equipment interrupts for low latency

Data Decomposition

Divide large data sets whose elements can be computed independently

- Same task operates on different data (opposite of Functional decomposition)
- Distribute data and associated computation among threads

Example:

Photo kiosk with image processing

- Image pixels are divided up and processed by separate threads
- Sometimes "stitch-up" is required to compute the overlapping borders of divided areas

Multi-core Processor Tuning

Focus is on the effective use of parallelism that takes advantage of more than one processor core.

This step pertains to both Intel® Hyper-Threading technology (Intel® HT technology) and true multi-core processing.

Multitasking is the execution of multiple operating system processes on a system.

Multithreading is the execution of multiple threads and by default assumes memory is shared.

Thread Safe Libraries

All routines called concurrently from multiple threads must be thread safe

Two methods are available to ensure thread safety

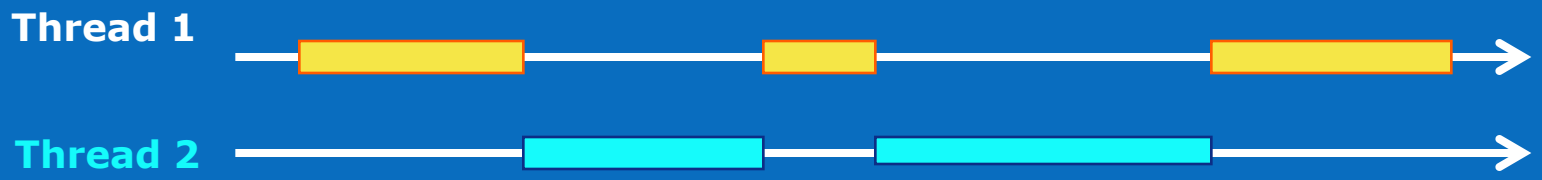
- Routines can be written to be reentrant
- Routines can use mutual exclusion synchronization to avoid conflicts with other threads

It is better to make a routine reentrant than to add synchronization by avoiding potential overhead

Concurrency vs. Parallelism

Concurrency: two or more threads are in progress at the same time

Emulation of Parallelism by Software threads

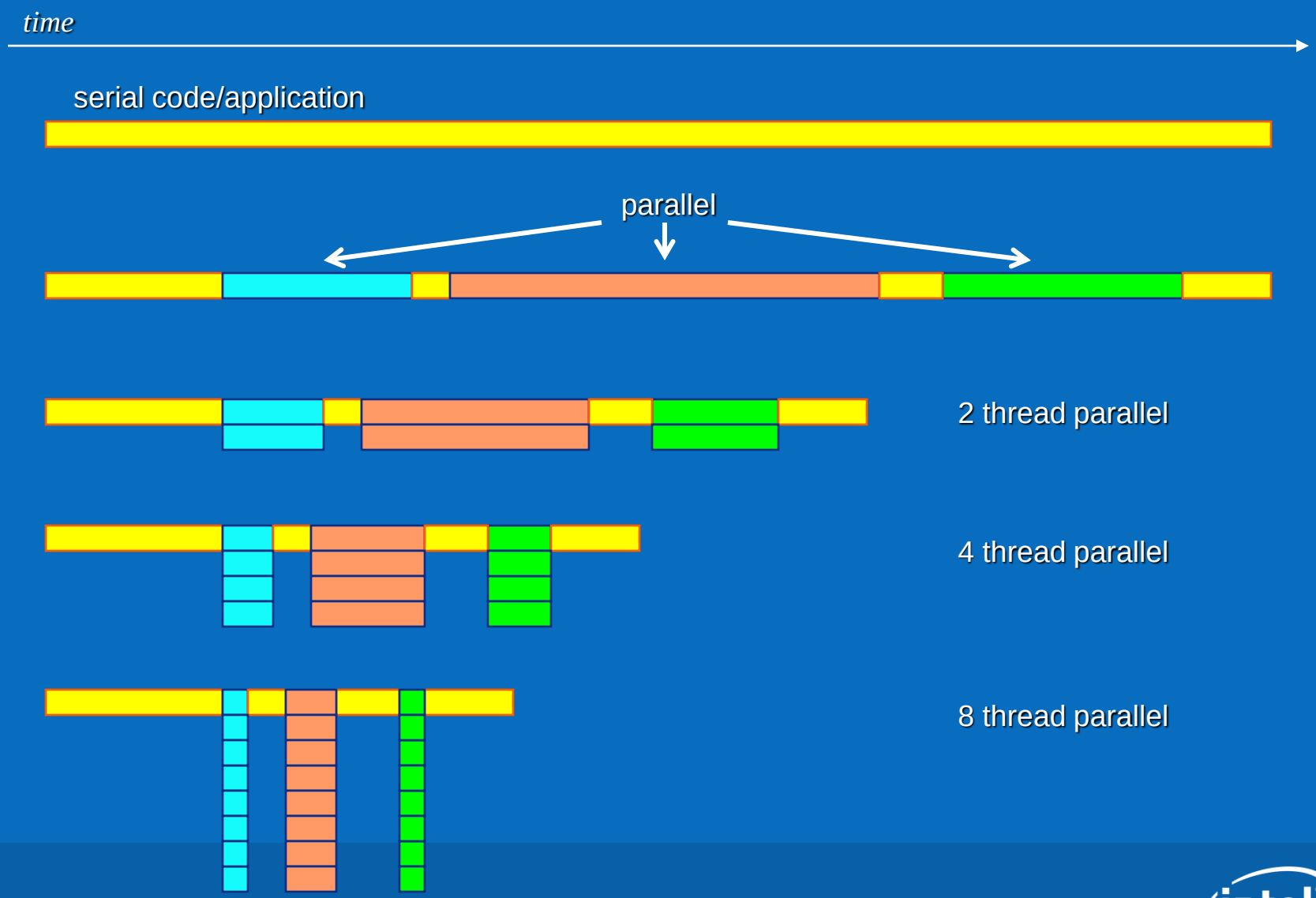


Parallelism: two or more threads are executing at the same time

True Parallelism

Multiple processors needed for parallelism

Threading = Parallelization

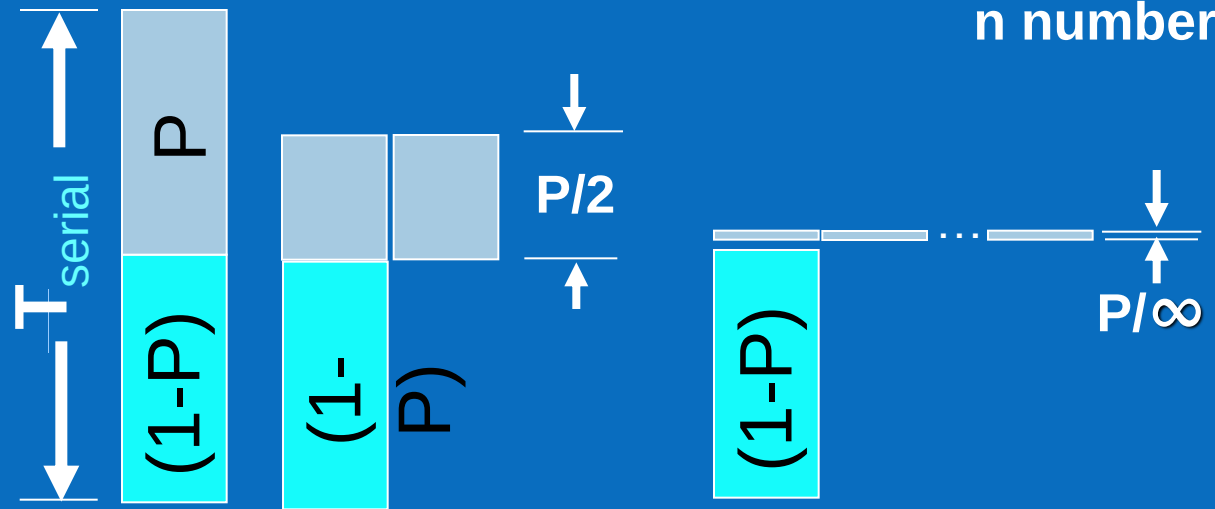


Amdahl's Law for Parallelism

Describes the upper bound of parallel speedup (scaling)

P parallel part

n number of threads



	n = 1	2	∞
$T_{parallel}$		0.5 + 0.25	0.5 + 0.0
Speedup		1.0/0.75 = 1.33	1.0/0.5 = 2.0

Serial code limits scaling

Computing Speedup

Example: Painting a picket fence – 300 pickets

- 30 min of preparation (serial)
- 30 min of cleanup (serial)
- 1 min to paint a single picket

Number of painters	Time	Speedup
1	360 = 30 + 300 + 30	1.0X
2	210 = 30 + 150 + 30	1.7X
4	135 = 30 + 75 + 30	2.7X
8	98 = 30 + 38 + 30	3.7X
16	78 = 30 + 18 + 30	4.6X
Infinite	60 = 30 + 0 + 30	6.0X

Illustrates Amdahl's Law

Potential speedup is restricted by serial portion

Efficiency

Measure of how effectively computation resources (threads) are kept busy

Efficiency = Speedup / Number_of_Threads

- Expressed as average percentage of non-idle time

Number of painters	Time	Speedup	Efficiency
1	360 = 30 + 300 + 30	1.0X	100%
2	210 = 30 + 150 + 30	1.7X	85%
4	135 = 30 + 75 + 30	2.7X	68%
8	98 = 30 + 38 + 30	3.7X	46%
Infinite	60 = 30 + 0 + 30	6.0X	very low

Efficiency, like other measurements, ties to the amount of serial code

Tuning Multithreaded Applications

- Requires ensuring good performance when the application is executing on both, logical and physical processor cores.
- Intel® Hyper-Threading technology (Intel® HT technology) is on the shared resources of the processor core. For example, the caches are effectively shared between two concurrently executing threads.
  - It is possible for one thread to cause the other to miss in the cache on every access.
- Thread interactions and cache behavior on multi-core processors can be even more complicated.
- It is possible for two threads to cause false sharing,

Synchronization

- OS APIs (mutex, semaphores) are effective but may be costly
  - Choose the optimal OS calls based on the circumstance
  - Ex. spin-wait, spin-locks, critical-sections
- Evaluate the need for synchronization; only use when necessary
  - Intel® Thread Checker (Intel® Inspector XE) can help determine if synchronization is needed
  - Intel® Thread Profiler (Thread analysis capability now included with Intel® VTune™ Amplifier XE) is helpful in deciding which method to use
    - Ex. Use to profile the execution timeline for each thread
  - Intel® VTune™ Performance Analyzer (Intel VTune Amplifier XE) to measure system performance and effectiveness of a given software technique
    - Ex. Use to track cache hit rate

Optimizing Synchronization

- Manage lock contention for large and small critical sections.
  - Critical sections execute serially. Threads should spend as little time inside a critical section as possible to reduce the amount of time other threads sit idle waiting to acquire the lock, a state known as lock contention.
  - In other words, it is best to keep critical sections small. Use synchronization routines provided by the threading API rather than hand-coding synchronization.
- Choose appropriate synchronization primitives to minimize overhead.
  - Understand the available synchronization objects and their tradeoffs.
  - Examples include simple operations on variable, inter-process synchronization and timed waits, and controlled spin counts for critical sections. Spin counts can greatly affect SMP performance on processors employing Intel® Hyper-Threading technology (Intel® HT technology).
- Use non-blocking locks when possible.
  - Use non-blocking threading calls to avoid context-switch overheads. The non-blocking synchronization calls usually start with the try keyword in C++.

Alternatives to Synchronization

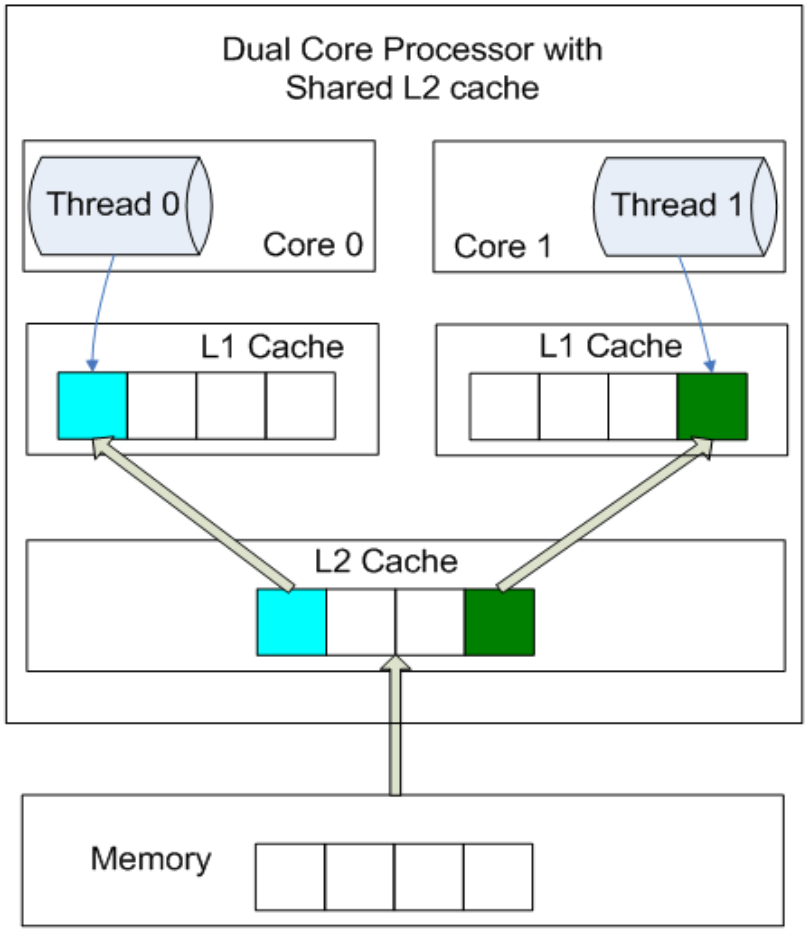
- OS can use light-weight objects such as MONITOR/MWAIT to reduce power consumption

- Reduce power and increase performance by using PAUSE or HLT instructions in fast spin loops (reduce missed branch prediction):

```
do{
        _asm PAUSE
} while (sync_var != constant_value);
```

- If you do implement a semaphore or MUTEX, place each synchronization variable in a separate cache line to avoid false sharing

Cache Coherency and False Sharing



- False sharing occurs when two or more threads from different processors access different address ranges on the same cache line
- Shared L2 cache architecture eliminates false sharing at the L2 cache level
- However:
  - False sharing caused by private L1 caches still exists, although with less penalty (compared to private L2 cache)
  - False sharing exists between cores that are on different processors that are not sharing the L2 cache

Optimizing Performance with Shared Cache (cont.)

- Avoid cache thrashing:
  - Use a performance analyzer (Intel® VTune) to identify excessive cache misses.
- Perform all processing on that data while it's in cache instead of having to read it into cache several times for different steps of the processing to the same data.
- Pin threads that share data to cores that share cache. This technique benefits from data locality. Thus, it reduces cache misses.
  - Use OS processor affinity functions
- Assign non-shared data to different cache lines.
  - Use structured types to group variables together to save space

Optimizing Performance with Shared Cache

- Use private thread copy
  - OpenMP* threadprivate modifier
  - MS compiler: __declspec(thread)
- Use thread optimized memory allocation routines that allocate data from a separate 64-byte aligned pool for each thread
  - The malloc function in the Intel compiler provides this feature
- In managed environments that provide automatic object allocation, the object allocators and garbage collectors should be responsible for the avoidance of false sharing

Optimizing Performance with Shared Cache (cont.)

- Align data to cache line size
  - Organize all global and static data variables that are accessed by one thread in blocks that are cache line size aligned

Example in Microsoft* Visual C++:

```
#define CAHE_LINE 64
__declspec(align(CACHE_LINE)) unsigned _int64
sum;
```

Group the data structure together to save space:

```
__declspec(align(16) struct  { int i, j; float k;}
sub;
```

Performance Optimization Tips

Multicore
Use multi-core software development tools
Use multi-core enabled performance libraries
Multithread for parallelism
Pin threads that share data on cores that share cache

Tools
Intel® Software Development Products
Analysis tools
Threading Tools and Libraries
Debuggers

Compiler Performance Features
Use an Intel® Atom™ processor targeting option (-xSSE3_ATOM)
Use automatic vectorization (-vec)
Use the -O3 option for aggressive loop and memory optimization
Use interprocedural optimization (IPO)
Use profile guided optimization (PGO)

Example:

-O3 -ipo -no-prec-div -xSSE3_Atom -prof_gen -prof_use

Questions?