Intel Compiler
Nehalem-EP CPU Summary

**Performance/Features:**
- 4 cores
- 8M on-chip Shared Cache
- Simultaneous Multi-Threading capability (SMT)
- Intel® QuickPath Interconnect up to 6.4 GT/s, each direct. per link
- Integrated Memory Controller (DDR3)
- New instructions

**Power:**
- 95W, 80W, 60W

**Socket:**
- New LGA 1366 Socket

**Process Technology:**
- 45nm CPU

**Platform Compatibility**
- Tylersburg (TBG)
- ICH9/10

---

Driving performance through Multi-Core Technology and platform enhancements

Source: Intel Corporation
Intel® Xeon™ 5500 Series (Nehalem-EP) Overview

**IT Benefits**
- More application performance
- Improved energy efficiency
- End to end HW assist (virtualization technology improvements)
- Stable IT image
  - Software compatible
  - Live migration compatible with today’s dual and quad-core Intel® Core™ microarchitecture products using enabled virtualization software

**Key Technologies**
- New 45nm Intel® Microarchitecture
- New Intel® QuickPath Interconnect
- Integrated Memory Controller
- Next Generation Memory (DDR3)
- PCI Express Gen 2

Source: Intel Corporation
Energy Efficiency Enhancements
Intel® Intelligent Power Technologies

**Integrated Power Gates**
Enables idle cores to go to near zero power independently

Voltage (cores): Core0, Core1, Core2, Core3
Memory System, Cache, I/O
Voltage (rest of processor)

**Automated Low Power States**

More & Lower CPU Power States
Reduced latency during transitions
Power management now on memory, I/O

**Automatic operation or manual core disable**

Adjusts system power consumption based on real-time load

Source:
1 Integrated power gates (C6) requires OS support
2 Requires BIOS setting change and system reboot

Source: Intel Corporation
More Efficient Chipset and Memory

Memory Power Management

- DIMMs are automatically placed into a lower power state when not utilized\(^1\)
- DIMMs are automatically idled when all CPU cores in the system are idle\(^2\)

Chipset Power Management

- QPI links and PCIe lanes placed in power reduction states when not active\(^3\)
- Capable of placing PCIe* cards in the lowest power state possible\(^4\)

End-to-end platform power management

Source: 1 Using DIMM CKE (Clock Enable)
2 Using DIMM self refresh
3 Using L0s and L1 states
4 Using cards enabled with ASPM (Active State Power Management)

Source: Intel Corporation
Performance Enhancements
Intel Xeon® 5500 Series Processor (Nehalem-EP)

Intel® Turbo Boost Technology

Increases performance by increasing processor frequency and enabling faster speeds when conditions allow

- Normal: All cores operate at rated frequency
- 4C Turbo: All cores operate at higher frequency
- <4C Turbo: Fewer cores may operate at even higher frequencies

Higher performance on demand

Intel® Hyper-Threading Technology

Increases performance for threaded applications delivering greater throughput and responsiveness

Higher performance for threaded workloads

Source: Intel Corporation

† Source: Intel internal measurements, January 2009. For notes and disclaimers, see performance and legal information slides at end of this presentation. Source: Intel Corporation
Nehalem-EP Turbo Mode Frequencies

<table>
<thead>
<tr>
<th>Basic (80W)</th>
<th>Standard (80W)</th>
<th>Perf (95W)</th>
<th>WS (130W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E5502 E5504 E5506</td>
<td>E5520 E5530 E5540</td>
<td>X5550 X5560 X5570</td>
<td></td>
</tr>
</tbody>
</table>

- **1.86 GHz DC**
- **1.93 GHz**
- **2.00 GHz**
- **2.13 GHz**
- **2.26 GHz**
- **2.40 GHz**
- **2.53 GHz**
- **2.66 GHz**
- **2.80 GHz**
- **3.00 GHz**
- **3.13 GHz**
- **3.20 GHz**
- **3.33 GHz**
- **3.46 GHz**

**Source:** Intel Corporation
## Optimizing for Memory Performance: General Guidelines and Potential Configurations

- Use identical DIMM types throughout the platform: Same size, speed, and number of ranks
- Use a “balanced” platform configuration: Populate the same for each channel and each socket
- Maximize number of channels populated for highest bandwidth

### IT Requirements (assumes two 4C Nehalem-EP CPUs)

<table>
<thead>
<tr>
<th>GB/core</th>
<th>0.5</th>
<th>1</th>
<th>1.5</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>4.5</th>
<th>6</th>
<th>8</th>
<th>9</th>
<th>12</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Capacity</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>36</td>
<td>48</td>
<td>64</td>
<td>72</td>
<td>96</td>
<td>144</td>
</tr>
</tbody>
</table>

### Potential Configurations

<table>
<thead>
<tr>
<th></th>
<th>2x2</th>
<th>4x2</th>
<th>6x2</th>
<th>8x2</th>
<th>12x2</th>
<th>18x2</th>
<th>2x4</th>
<th>4x4</th>
<th>6x4</th>
<th>8x4</th>
<th>12x4</th>
<th>18x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 GB DIMM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 GB DIMM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 GB DIMM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Green** indicates DDR3 1333, 1066, 800 support
- **Yellow** indicates DDR3 1066, 800 support
- **Red** indicates DDR3 800 only

Indicates memory “sweet spot” that equally populates all 6 memory channels

Source: Intel Corporation
Simultaneous Multi-Threading (SMT)

- Nehalem is a 4 issue, superscalar, out-of-order CPU
  - Scheduler tries to feed 4 instruction/cycle to the execution unit!
  - Wider (execution units) and deeper (more stages) is a challenge
- **Solution:** Symmetric Multi-Threading
  - Run 2 threads at the same time per core
  - Leaving few units unused
  - Duplicate Registers
  - Share 000 logic, Execution Units, Cache
  - Increase reorder buffer from 96 to 126 entries
- Take advantage of 6-wide execution engine
  - Keep it fed with multiple threads
  - Hide latency of a single thread
  - Increase the 000 pressure
- Most **power efficient** performance feature
  - Very low die area cost
  - Can provide significant performance benefit depending on application
- Nehalem advantages
  - Larger caches, bigger buffers
  - Massive memory BW

Note: Each box represents a processor execution unit

*Copyright © 2009, Intel Corporation. All rights reserved.*

*Other brands and names are the property of their respective owners*
Hyper-Threading (HT, SMT)

**The key question:** Enabling or disabling HT?
- Can be disabled in BIOS for all Nehalem systems
- In case it is enabled, the computer user still has the freedom not to make use of it by reducing number of threads created
  - Intel-internal tests covering many HPC applications have show: Running one thread only on each core of a HT-enabled system gives almost identical performance data than switching off HT

**There is no simple answer for all environments but based on experience:**
- At sites where all applications create a similar workload, it is better to disable it
- “Pure” HPC sites typically see no benefit overall; for many applications performance suffers
- Non-HPC sites in general see great benefit and should enable it
Intel® Smart Cache – Core Caches

- **New 3-level Cache Hierarchy**

- **1st level caches**
  - 32kB Instruction cache
  - 32kB, 8-way Data Cache
    - Support more L1 misses in parallel than Intel® Core™2 microarchitecture

- **2nd level Cache**
  - New cache introduced in Intel® Core™ microarchitecture (Nehalem)
  - Unified (holds code and data)
  - 256 kB per core (8-way)
  - **Performance**: Very low latency
    - 10 cycle load-to-use
  - **Scalability**: As core count increases, reduce pressure on shared cache

Source: Intel Corporation
Intel® Smart Cache – 3rd Level Cache

- Shared across all cores
- Size depends on # of cores
  - Quad-core: Up to 8MB (16-ways)
  - **Scalability:**
    - Built to vary size with varied core counts
    - Built to easily increase L3 size in future parts
- Perceived latency depends on frequency ratio between core & uncore ~ 40 clocks
- Inclusive cache policy for best performance
  - Address residing in L1/L2 must be present in 3rd level cache

Source: Intel Corporation
Why Inclusive?

- Inclusive cache provides benefit of an on-die snoop filter
- Core Valid Bits
  - 1 bit per core per cache line
    - If line may be in a core, set core valid bit
    - Snoop only needed if line is in L3 and *core valid* bit is set
    - Guaranteed that line is not modified if multiple bits set
- Scalability
  - Addition of cores/sockets does not increase snoop traffic seen by cores
- Latency
  - Minimize effective cache latency by eliminating cross-core snoops in the common case
  - Minimize snoop response time for cross-socket cases

Source: Intel Corporation
Hardware Prefetching (HWP)

- **HW Prefetching critical to hiding memory latency**
- **Structure of HWPs similar as in Intel® Core™ 2 microarchitecture**
  - Algorithmic improvements in Intel® Core™ microarchitecture (Nehalem) for higher performance
- **L1 Prefetchers**
  - Based on instruction history and/or load address pattern
- **L2 Prefetchers**
  - Prefetches loads/RFOs/code fetches based on address pattern
  - Intel Core microarchitecture (Nehalem) changes:
    - **Efficient Prefetch** mechanism
      - Remove the need for Intel® Xeon® processors to disable HWP
    - Increase prefetcher **aggressiveness**
      - Locks on address streams quicker, adapts to change faster, issues more prefetchers more aggressively (when appropriate)

Source: Intel Corporation
Extending Performance and Energy Efficiency
- Intel® SSE4.2 Instruction Set Architecture (ISA)

**Accelerated String and Text Processing**
- Faster XML parsing
- Faster search and pattern matching
- Novel parallel data matching and comparison operations

**Accelerated Searching & Pattern Recognition of Large Data Sets**
- Improved performance for Genome Mining, Handwriting recognition. Fast Hamming distance / Population count

**New Communications Capabilities**
- Hardware based CRC instruction
- Accelerated Network attached storage
- Improved power efficiency for Software I-SCSI, RDMA, and SCTP

**SSE4.2 (Nehalem Core)**
- STTNI
- ATA

**SSE4.1 (Penryn Core)**
- STTNI (e.g. XML acceleration)

**ATA (Application Targeted Accelerators)**
- POPCNT (e.g. Genome Mining)
- CRC32 (e.g. iSCSI Application)

Projected 3.8x kernel speedup on XML parsing & 2.7x savings on instruction cycles

*Source: Intel Corporation*
Tools Support for Intel® Core™ Microarchitecture (Nehalem)

- **Intel® Compiler 10.x supports the new instructions**
  - Nehalem specific compiler optimizations
  - SSE4.2 supported via vectorization and intrinsics
  - Inline assembly supported on both IA-32 and Intel® 64 architecture targets
  - Necessary to include required header files in order to access intrinsics

- **Intel® XML Software Suite**
  - High performance C++ and Java runtime libraries
  - Version 1.0 (C++), version 1.01 (Java) available now
  - Version 1.1 w/SSE4.2 optimizations planned for September 2008

- **Microsoft Visual Studio® 2008 VC++**
  - SSE4.2 supported via intrinsics
  - Inline assembly supported on IA-32 only
  - Necessary to include required header files in order to access intrinsics
  - VC++ 2008 tools masm, msdis, and debuggers recognize the new instructions

- **GCC® 4.3.1**
  - Support Intel Core microarchitecture (Merom), 45nm next generation Intel Core microarchitecture (Penryn), Intel Core microarchitecture (Nehalem)
  - via –mtune=generic.
  - Support SSE4.1 and SSE4.2 through vectorizer and intrinsics
Optimization Guidelines with Intel Compiler for Intel Core i7 processor

- Many new features introduced that you get for free
  - Better branch prediction + faster miss-prediction correction
  - Improvements on unaligned loads
  - Improvements on store forwarding
  - Memory bandwidth increase
  - Improvement on cache-line splits
    - Data is being loaded across cache-line boundaries which causes instructions to run two to four times slower

- No large differences in tuning guidelines relevant for Intel® Core™ 2 processor architecture

Source: Intel Developer Forum, “Tuning your Software for the Next Generation Intel Microarchitecture (Nehalem)”
What are Cache Line Splits?

- Data is being loaded or stored across cache line boundaries
- Intel® Core™ microarchitecture (Nehalem) has optimized accesses that span two cache-lines
Can Performance get Worse?

Really only for “exotic” cases

- Hyper-Thread might cause a minimal performance regression
  - In case threads execute same instruction type (like floating point arithmetic), overhead might not pay off
  - Solution: Switch HT off (BIOS option)

- Test applications making perfect use of Penryn cache hierarchy
  - E.g. working set of test code fits into L2 cache of Hapertown processor but does not fit into L2 cache of Nehalem processor (256KB < size < 6MB)
    - Most data will be in L3 cache of Nehalem
    - Hit by latency difference (about 40 versus 15 cycles)
  - No “real world” scenario – no sample production code we know about
  - But artificial codes (e.g. 20 year old synthetic benchmark like drystone/whetstone might be candidates)
Unaligned Loads / Stores

- Unaligned loads are as fast as aligned loads
- Optimized accesses that span two cache-lines
- Generating misaligned references less of a concern
  - One instruction can replace sequences of up to 7
  - Fewer instructions
  - Less register pressure
- Increased opportunities for several optimizations
  - Vectorization
  - memcpy / memset
  - Dynamic Stack alignment less necessary for 32-bit stacks
Compiler Pro and Nehalem

• Since 11.0 release, compiler support “SSE4_2” extension to specifically tune for Nehalem
  Linux: -xsse4_2, -axsse4_2, -msse4_2

• Specific compiler changes
  – Automatic vectorization makes use of SSE4.2 set
  – Support for new intrinsics for manual coding
    – File nmmintrinsic.h contains declarations
  – Support for manual CPU dispatching
    – __declspec(cpu_specific(core_i7_sse4_2))
  – Code generation exploits benefits of architectural changes

• Libraries Intel® MKL and IPP
  – Performance critical routines specifically tuned for Nehalem architecture
    (instruction set, memory hierarchy incl. NUMA, SMT, u-architecture)
Compatibility with GNU GCC

Source and binary compatible

- Mixing and matching binary files created by g++, including third-party libraries
- Generating C++ code compatible with gcc/g++ 3.2 or higher (up to 4.2)
- Improved support for command-line options offered in the GNU compilers
- Support of most GNU C and C++ language extensions

Limitations

- Intel Fortran Compiler for Linux is not binary compatible with GNU g77 or GNU gfortran compiler
Compatibility with Linux (cont)

GNU gcc/g++ language extensions

- We support most of the GNU gcc language extensions (47 out of 56)

Limitations

- No support for:
  - Nested functions
  - Constructing function calls
  - Looser Rules for Escaped Newlines
  - Prototype and Old-Style Function Definitions
  - Using Vector Instructions Through Built-in Functions
  - Built-in Functions Specific to Particular Target Machines
  - J++) Exceptions
  - Deprecated Features
  - Backward Compatibility

- We can successfully compile the Linux kernel 2.4.21 and 2.6.9 with Intel C++ Compiler on IA-32, Intel® 64 and IA-64, with a small wrapper script and patches
Common Intel Compiler Features

- General optimization settings
- Cache-management features
- Interprocedural optimization (IPO) methods
- Profile-guided optimization (PGO) methods
- Multithreading support
- Floating-point arithmetic precision and consistency
- Compiler optimization and vectorization report

Source: Intel white paper “Optimization Applications with Intel C++ and Fortran Compilers for Windows, Linux and Mac Os X”
<table>
<thead>
<tr>
<th>Linux/Mac OS* X Equivalent</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>-00</td>
<td>No optimization. Used during the early stages of application development and debugging. Use a higher setting when the application is working correctly.</td>
</tr>
<tr>
<td>-01</td>
<td>Optimize for size. Omits optimizations that tend to increase object size. Creates the smallest code in most cases. This option is useful in many large server/database applications where memory paging due to larger code size is an issue.</td>
</tr>
<tr>
<td>-02</td>
<td>Maximize speed. Default setting. Enables many optimizations, including vectorization. Creates faster code than /O1 (-01) in most cases.</td>
</tr>
<tr>
<td>-03</td>
<td>Enables /O2 (-02) optimizations plus more aggressive loop and memory-access optimizations, such as scalar replacement, loop unrolling, code replication to eliminate branches, loop blocking to allow more efficient use of cache and additional data prefetching. The /O3 (-03) option is particularly recommended for applications that have loops that heavily use floating-point calculations or process large data sets. These aggressive optimizations may occasionally slow down other types of applications compared to /O2 (-02).</td>
</tr>
<tr>
<td>-g</td>
<td>Generates debug information for use with any of the common platform debuggers. This option turns off /O2 (-02) and makes /Od (-00) the default unless /O2 (-02) (or another option) is specified.</td>
</tr>
<tr>
<td>-debug full</td>
<td>Produces full debugging information including symbol table information needed for full symbolic debugging of unoptimized code and global symbol information needed for linking. It produces the largest size object modules. If this option is specified for an application that makes calls to C library routines that will be debugged, the option /dbglibs must also be specified to link the appropriate C debug library.</td>
</tr>
</tbody>
</table>

If this option is used with optimized code, full symbol information will be generated including the local symbol table information, regardless of the optimization level. This may result in minor performance degradation.

Source: Intel Corporation
<table>
<thead>
<tr>
<th><strong>Linux/Mac OS X Equivalent</strong></th>
<th><strong>Comment</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-lp</code></td>
<td>Single file interprocedural optimizations, including selective inlining, within the current source file. Caution: For large files, this option may sometimes significantly increase compile time and code size.</td>
</tr>
<tr>
<td><code>-lpo[\text{value}]</code></td>
<td>Permits inlining and other interprocedural optimizations among multiple source files. The optional \text{value} argument controls the maximum number of link-time compilations (or number of object files) spawned. Default \text{value} is 0 (the compiler chooses). Caution: This option can in some cases significantly increase compile time and code size.</td>
</tr>
<tr>
<td><code>-lpo-jobs[n]</code></td>
<td>Specifies the number of commands (jobs) to be executed simultaneously during the link phase of interprocedural optimization (IPO). The default is 1 job.</td>
</tr>
<tr>
<td><code>-finline-functions</code></td>
<td>This option enables function inlining at the compiler's discretion. This option is enabled by default at /O2 and /O3 (-O2 and -O3). Caution: For large files, this option may sometimes significantly increase compile time and code size. It can be disabled by /Oo0 (-fno-inline-functions on Linux and Mac OS X).</td>
</tr>
<tr>
<td><code>-finline-level=2</code></td>
<td></td>
</tr>
<tr>
<td><code>-finline-factor=n</code></td>
<td>This option scales the total and maximum sizes of functions that can be inlined. The default value of n is 100, i.e., 100% or a scale factor of one.</td>
</tr>
<tr>
<td><code>-prof-gen</code></td>
<td>Instruments a program for profile generation.</td>
</tr>
<tr>
<td><code>-prof-use</code></td>
<td>Enables the use of profiling information during optimization.</td>
</tr>
<tr>
<td><code>-prof-dir \text{dir}</code></td>
<td>Specifies a directory for the profiling output files, *.	ext{dyn} and *.	ext{dpl}.</td>
</tr>
</tbody>
</table>

Source: Intel Corporation
Common Intel Compiler Features

- **Target systems with processor-specific options:**
  - `-xsse4.2` generate optimized code specialized for the Intel Core (Intel Core i7) processor family that executes the program
  - `-xHOST` optimize for and use the most advanced instruction set for the processor on which you compile
  - `-axsse4.2` Generate multiple processor-specific auto-dispatch code paths for Intel processors if there is performance benefit. Executable will run on Intel processor architecture other than Intel Core i7

- **A good start:**
  - `-O2 -xsse4.2` or `-O2 -xHost`
  - `-O3 -xsse4.2` or `-O3 -xHost`
  - `-fast :` `-O3 -ipo -no-prec-div -static`
Parallel programming with MPI
Parallel Scalability

- **Amdahl’s Law** – “the law of diminishing returns”
  \[ T_n = S + \frac{P}{n} \]

- Assumption: Parallel content scales inversely with number of parallel tasks, ignoring computation and communication load imbalance

- The maximum parallel speedup is:
  \[ \frac{P}{S} + 1 \]
Strong vs Weak Scaling

- **Strong scaling**
  - Scalability of a *fixed total problem size* with the number of processors

- **Weak Scaling**
  - Scalability of a *fixed problem size per processor* with the number of processors
- **Thread:**
  - An independent flow of control, may operate within a process with other threads.
  - An schedulable entity
  - Has its own stack, thread-specific data, and own registers
  - Set of pending and blocked signals

- **Process**
  - Can not share memory directly
  - Can not share file descriptors
  - A process can own multiple threads

- An OpenMP job is a process. It creates and owns one or more SMP threads. All the SMP threads share the same PID

- An MPI job is a set of concurrent processes (or tasks). Each process has its own PID and communicates with other processes via MPI calls
Shared Memory

- **Characteristics**
  - Single address space
  - Single operating system

- **Limitations**
  - Memory
    - Contention
    - Bandwidth
  - Cache coherency snoop traffic gets very expensive as more L2 caches in node

- **Benefits**
  - Memory size
  - Programming models
Distributed Memory

- **Characteristics**
  - Multiple address spaces
  - Multiple operating systems

- **Limitations**
  - Switch
  - Contention
  - Bandwidth
  - Local memory size

- **Benefits**
  - Economically scale to large processor counts
  - Cache coherency not needed between nodes
Comparison: Shared Memory Programming vs. Distributed Memory Programming

- **Shared memory**
  - Single process ID for all threads
  - List threads
    - `ps -eLf`

- **Distributed memory**
  - Each “task” has own process ID
  - List tasks:
    - `ps`
Parallel Programming on HPC cluster

Message Passing (MPI) between Nodes

High Performance Interconnect

Cluster of Shared Memory Nodes

OpenMP / Multi-threading within SMP node
Parallel Programming choices

- **MPI**
  - Good for tightly coupled computations
  - Exploits all networks and all OS
  - Significant programming effort; debugging can be difficult
  - Master/Slave paradigm is supported.

- **OpenMP**
  - Easier to get parallel speed up
  - Limited to SMP (single node)
  - Typically applied at loop level ← limited scalability

- **Automatic parallelization by compiler**
  - Need clean programming to get advantage

- **pthreads = Posix threads**
  - Good for loosely coupled computations
  - User controlled instantiation and locks

- **fork/execl**
  - Standard Unix/Linux technique
Schematic Flow of an SMP Code

Program ParallelWork
start the program
read input
set up comp. parameters
initialize variable
...
DO i=1,imax
...
...
End do
...
serial work
...
DO i=1,imax
...
...
End do
...
serial work
...
DO i=1,imax
...
...
End do
...
Output
End program ParallelWork
Schematic Flow of an MPI code

Program ParallelWork

start the program

Call MPI_Init(ierr)
read input
set up comp. parameters

... DO i=1,imax
... End do

... serial work

... DO i=1,imax
... End do
serial work

... DO i=1,imax
... End do
serial work

Output

Call MPI_Finalize

End program ParallelWork
MPI Basics

- MPI = Message Passing Interface
- A message passing library standard based on the consensus of MPI Forum, with participants from vendors (hardware & software), academics, software developers
- Initially developed to support distributed memory program architecture
- Not an IEEE or ISO standard, now the “defacto” industry standard
- NOT a library -- specification on what the library should be, but not on the implementation
- MPI-1.0 , released 1994
- MPI-2.2 (latest) , released 2009
- MPI-3.0 standard, ongoing
MPI tutorials

- https://computing.llnl.gov/tutorials/mpi