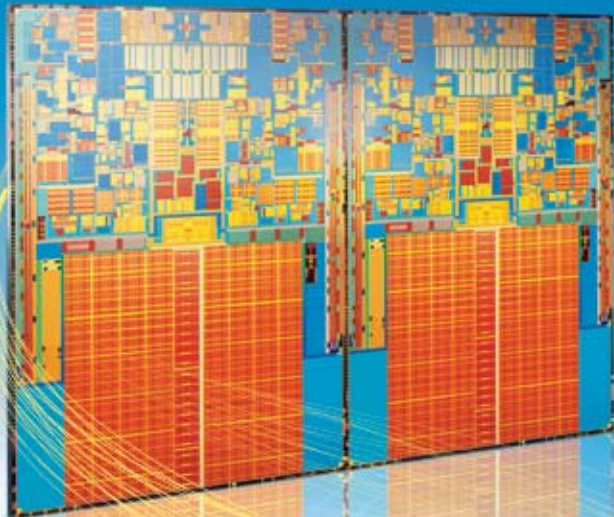




Tuning Your Software for the Next Generation Intel® Microarchitecture (Nehalem) Family



Antonio C. Valles, Software Engineer
Zia Ansari, Compiler Engineer
Pallavi Mehrotra, Software Engineer
Intel Corporation

NGMS002

Agenda

1. Optimizing for Intel® Core™ i7 processor
 - Tuning recommendations
 - Tools
2. Parallelism strategies
 - CPU Enumeration and affinitization
 - Removing parallelism bottlenecks
3. Using Non-Uniform Memory Access (NUMA)
 - Locality Optimizations
 - Operating system differences

Optimization Guidelines For Intel® Core™ i7 Processor

- Many new features introduced that you get for free
 - Better branch prediction + faster mispredict correction
 - Improvements on unaligned loads + cache-line splits
 - Improvements on store forwarding
 - Memory bandwidth increase
 - Reduced memory latency
 - Etc...
- No large differences in tuning guidelines
 - Still use Intel® 64 and IA-32 Architectures Optimization Reference Manual: <http://www.intel.com/products/processor/manuals/>
- This presentation will discuss optimizations/recommendations to further enhance performance on Intel® Core™ i7 processor

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

Opens up many opportunities

Unaligned Loads/Stores - example

```
for (i = 0; i < NUM; i++) {  
    dst[i] = src1[i]*src2[i+1] + src3[i+1];  
}
```

**Code1: no Core i7 optimizations
(16 instructions per loop iteration):**

loop_label:

```
movups  xmm1, XMMWORD PTR [_src2+4+eax*4]  
movsd   xmm4, QWORD PTR [_src2+20+eax*4]  
movss   xmm2, DWORD PTR [_src2+28+eax*4]  
movhps  xmm2, QWORD PTR [_src2+32+eax*4]  
.....  
shufps  xmm4, xmm2, 132  
.....  
add     eax, 8  
cmp     eax, 1024  
jb     loop_label
```

**Code2: generate more 16-bytes loads
(13 insts per iteration):**

loop_label:

```
movups  xmm1, XMMWORD PTR [_src2+4+eax*4]  
movups  xmm3, XMMWORD PTR [_src2+20+eax*4]  
.....  
add     eax, 8  
cmp     eax, 1024  
jb     loop_label
```

*~11% Speedup**

*As measured in Intel labs

Unaligned Loads/Stores

- CPU2000 Estimated Data
 - No performance regressions
 - 168.wupwise +8%
 - 172.mgrid +21%
 - 178.galgel +3%
 - 301.apsi +5%
 - **Overall fp Geomean +2.78%**

- CPU2006 Estimated Data
 - No performance regressions
 - 436.cactusADM +11%
 - 437.leslie3d +9%
 - 454.calculix +8%
 - 459.GemsFDTD +12%
 - **Overall fp Geomean +2.6%**

- Source: Intel. Data estimated based on measurements on preproduction hardware; per SPEC* Run Rules section 4

No more split sequences

Unaligned Loads/Stores

- Elimination of local stack alignment (32-bit)
 - Used to ensure 16-byte alignment for fast aligned accesses
 - Used to ensure 8-byte alignment to avoid FP double splits
 - No need for extra prolog / epilog overhead
 - Free up one integer register
- CPU2000fp Estimated Data (32-bit /O2)
 - No performance regressions
 - 168.wupwise +6.5%
 - 172.mgrid +2.4%
 - 187.facerec +4%
 - **Overall fp Geomean +1.42%**
- Source: Intel. Data estimated based on measurements on preproduction hardware; per SPEC* Run Rules section 4

Dynamic alignment overhead less beneficial

Store Forwarding Improvements

- Store forwarding is an optimization enabling loads to get the data before the store updates cache
- Without store forwarding, loads are forced to wait until the store updates cache; loads get the data directly from cache
- Intel® Core™ i7 enables more store forwarding cases
 - Alignment* of the stores is no longer a factor
 - All stores to 1, 2, 4, and 8 byte addresses will forward to lesser or equal sized loads regardless of alignment
 - Most 16-byte stores will forward to equal or lesser sized loads

Example: Case that now forwards on Intel® Core™ i7
32-bit store forwarded to overlapped 16-bit load

```
mov dword ptr [ebp+40h], eax  
movzx edi, word ptr [ebp+41h]
```

Less Optimizations Needed for Store Forwarding

*As before, will not forward if store spans two cache-lines

New Instructions – more vectorization

```
__int64 a[N], b[N], c[N], d[N], e[N];

for (i = 0; i < N; i++) {
    a[i] = (b[i] > c[i]) ? d[i] : e[i];
}
```

Speedups*

Intel® 64: 1.5x

IA-32: 3.0x

Before: No Vectorization Possible	New Instructions allow for vectorization
<pre> xorl %eax, %eax ..B2.2: movq b(,%rax,8), %rdx cmpq c(,%rax,8), %rdx jle ..B2.4 ..B2.3: movq d(,%rax,8), %rdx jmp ..B2.5 ..B2.4: movq e(,%rax,8), %rdx ..B2.5: movq %rdx, a(,%rax,8) incq %rax cmpq \$1024, %rax jl ..B2.2 </pre>	<pre> xorl %eax, %eax ..B2.2: movdqa b(,%rax,8), %xmm0 pcmpgtq c(,%rax,8), %xmm0 movdqa e(,%rax,8), %xmm1 pblendvb %xmm0, d(,%rax,8), %xmm1 movdqa %xmm1, a(,%rax,8) addq \$2, %rax cmpq \$1024, %rax jl ..B2.2 </pre>

*As measured in Intel labs

New Instructions – faster string functions

- `pcmpistri` : Partially inlined `strlen` implementation
 - Avoids call overhead for short strings (common case)
 - Avoids the excessive code bloat from fully inlining

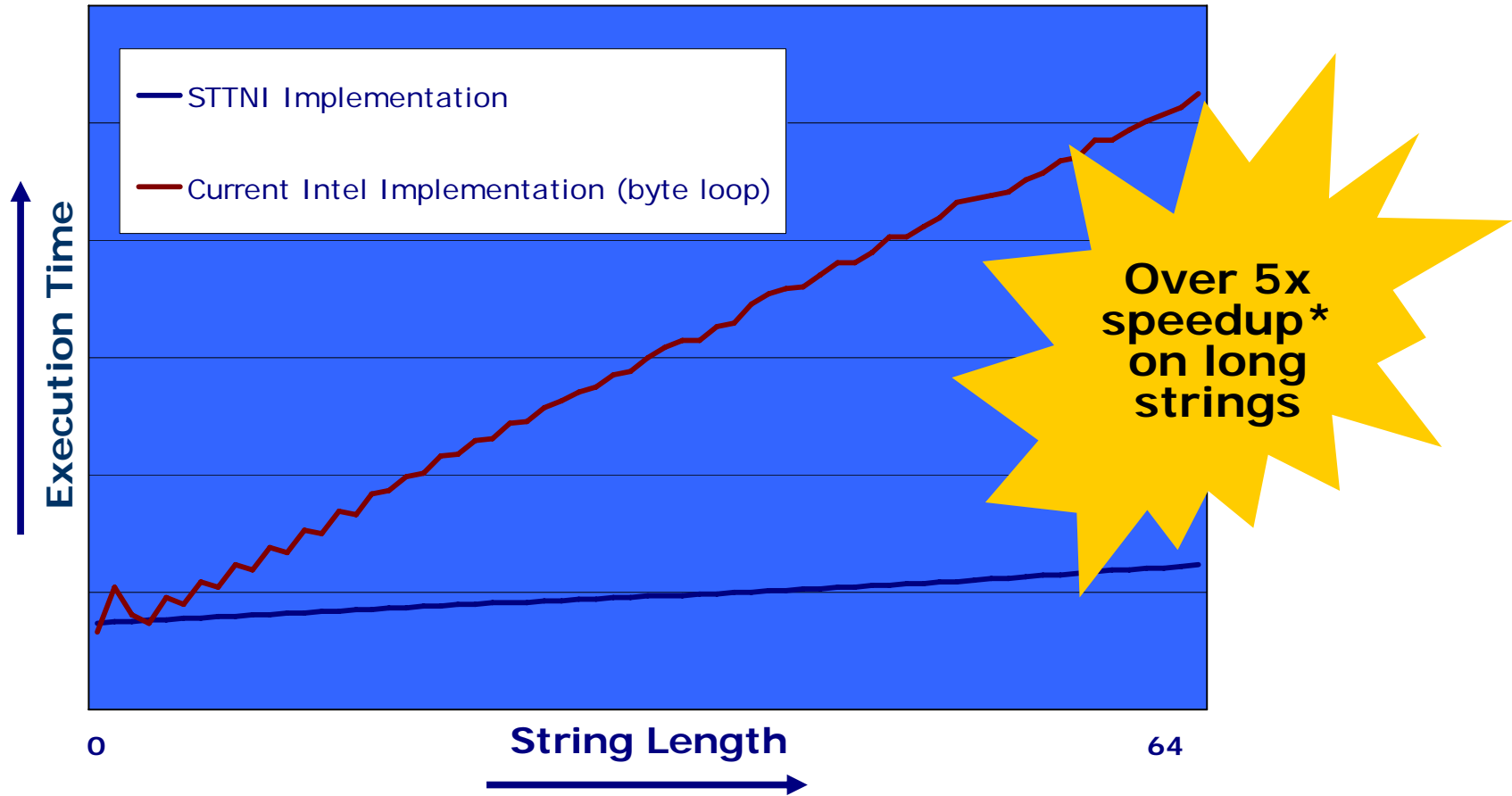
```
mov     ecx, edx
and     edx, 0xFFFFFFFF
pxor   xmm0, xmm0
pcmpeqb xmm0, XMMWORD PTR [edx]
pmovmskb eax, xmm0
and     ecx, 0xF
shr    eax, cl
bsf    eax, eax
jne    ..L1

mov     eax, edx
add     edx, ecx
call   __intel_sse4_strlen
..L1:
```

```
__intel_sse4_strlen:
    add     eax, 16
    movdqa  xmm0, XMMWORD PTR [eax]
    pcmpistri xmm0, xmm0, 58
    jae    __intel_sse4_strlen

    sub     ecx, edx
    add     eax, ecx
    ret
```

New Instructions – faster string functions



*As measured in Intel labs

Tool Support for New Performance Features and Instructions

- Intel® 10.1 + Compilers
 - “-QxH” or “-QxSSE4.2” to vectorize + processor specific compiler optimizations
 - Inlined Assembly support (32 & 64bit)
 - For SSE4.2 intrinsics: include <nmmintrin.h>
- Sun Studio* Express (July 08 build)
 - SSE4.2 supported through intrinsics + processor specific compiler optimizations
- Microsoft Visual Studio* 2008 VC++
 - SSE4.2 supported via intrinsics
 - Inline Assembly supported on IA-32 only
- GCC* 4.3.1
 - Support SSE4.2 through vectorizer and intrinsics + processor specific compiler optimizations
- Intel® Integrated Performance Primitives version 6.0

Optimize by using Intel and Industry Tools

Summary of Optimization Tips

1. Keep following Intel® 64 and IA-32 Architectures Optimization Reference Manual
<http://www.intel.com/products/processor/manuals/>
2. Take advantage of faster unaligned accesses
3. Many new cases will now forward; As before, find and fix cases that don't forward
4. Take advantage of the new SSE4.2 instructions
 - Use Intel and Industry tools

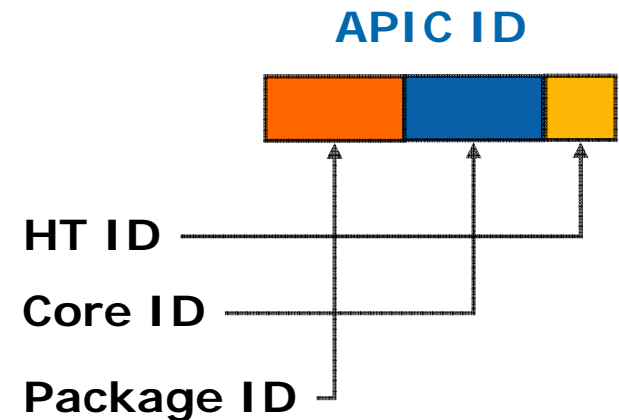
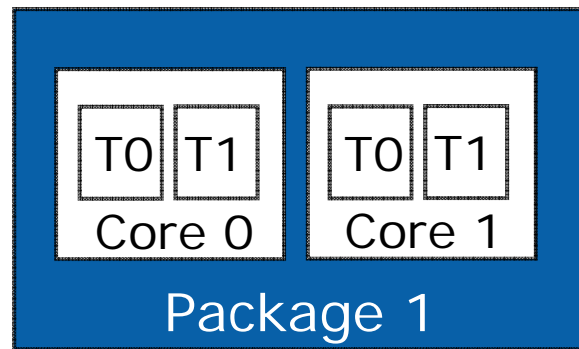
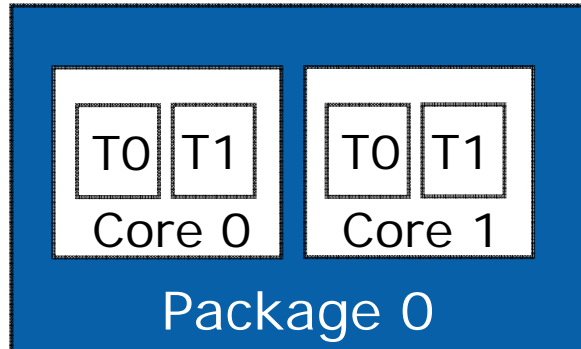
Agenda

1. Optimizing for Intel® Core™ i7 processor
 - Tuning recommendations
 - Tools
2. Parallelism strategies
 - CPU Enumeration and affinitization
 - Removing parallelism bottlenecks
3. Using Non-Uniform Memory Access (NUMA)
 - Locality Optimizations
 - Operating system differences

Multi-Core and Hyper-Threading Optimizations

- Hyper-threading + multi-core + microarchitecture enables more parallelization opportunities
 - Maximize performance by using the same number of software threads or processes as logical processors
- If your app is still single-threaded/single-process consider updating with parallelization strategies
 - Fast/Easy: Consider Cluster OpenMP* for loop-level parallelism
 - Use Intel® Threading Building Blocks to load balance workload
 - For more Intel threading tools:
<http://www3.intel.com/cd/software/products/asmo-na/eng/219785.htm>

Create the Right Number of SW Threads: Use CPU Enumeration



- **Wrong way to get number of cores per package:**
 - Only use CPUID leaf4 CPUID.4.EAX[31:26]
 - This used to work by accident
 - Getting wrong enumeration and affinizing is a double whammy!
- **Right way to get number of cores per package:**
 - **Copy** CPU Topology Enumeration whitepaper **exactly**
<http://softwarecommunity.intel.com/articles/eng/3887.htm>
 - Easy to get it wrong – copy exactly

*Shortcuts may work for one OS/Platform,
but not work on other OS's/Platforms*

Affinitization Recommendations

- Avoid affinitization for non-NUMA optimizations
 - Error prone and not forward-looking strategy
 - Many future products with variable number of cores/threads
 - Affinitization masks can vary per OS and on 32-bit or 64-bit OS versions
 - Does not consider software environment (example: co-existing with other affinitized applications)
 - Short term gains vs. many headaches later...
- When to consider affinitization
 - To improve accesses to local memory vs. remote memory
 - Non-Uniform Memory Accesses (NUMA)...discussed later
 - Rely on quality middle-ware to do this for you correctly
 - Example: Intel Cluster OpenMP* or MPI environment variables

Remove Parallelism Bottlenecks

- Common barriers
 - False sharing, too many locks/synchronization, small parallel region compared to serial region, etc...
- What is false sharing?
 - Two processors sharing the same cache-line
 - Global or static data variables/structures
 - Sometimes: Objects allocated dynamically
 - Ping-pong effect if both processors are writing to L1D cache-line
 - P1: RFO, P2: WB to L2 and INV
 - P2: RFO, P1: WB to L2 and INV
 - ...

Cache-line



False sharing Example:

```
int sum[THREAD_NUM];  
/* each thread uses its  
thread number as index  
to global sum array */  
int inc_sum () {  
  
    sum[my_thr_num] ++;  
    return  
    sum[my_thr_num];  
}
```

Find and Fix False Sharing

- Find:
 - Use Performance Tool Utility's memory access analysis
 - Uses Intel® Core™ i7 processor precise HITM and store events to identify contested cache-line accesses
- Fix:
 - Only fix if significant performance degradation
 - Pad variables to align on cache-line boundary
 - Example using Microsoft Compiler*:
 - Before:

```
int sum1; int sum2;
```
 - Fix:

```
__declspec(align(64)) int sum1;  
__declspec(align(64)) int sum2;
```

Summary of Parallelization Tips

1. Thread your application
2. Maximize performance by using the same number of software threads as logical processors
3. Do CPU Enumeration correctly
 - Get correct number of software threads
4. Avoid affinitization for non-NUMA optimizations
5. Use Intel tools to find and remove parallelism bottlenecks

<http://softwarecommunity.intel.com/articles/eng/3887.htm>

<http://www3.intel.com/cd/software/products/asmo-na/eng/219785.htm>

<http://whatif.intel.com>

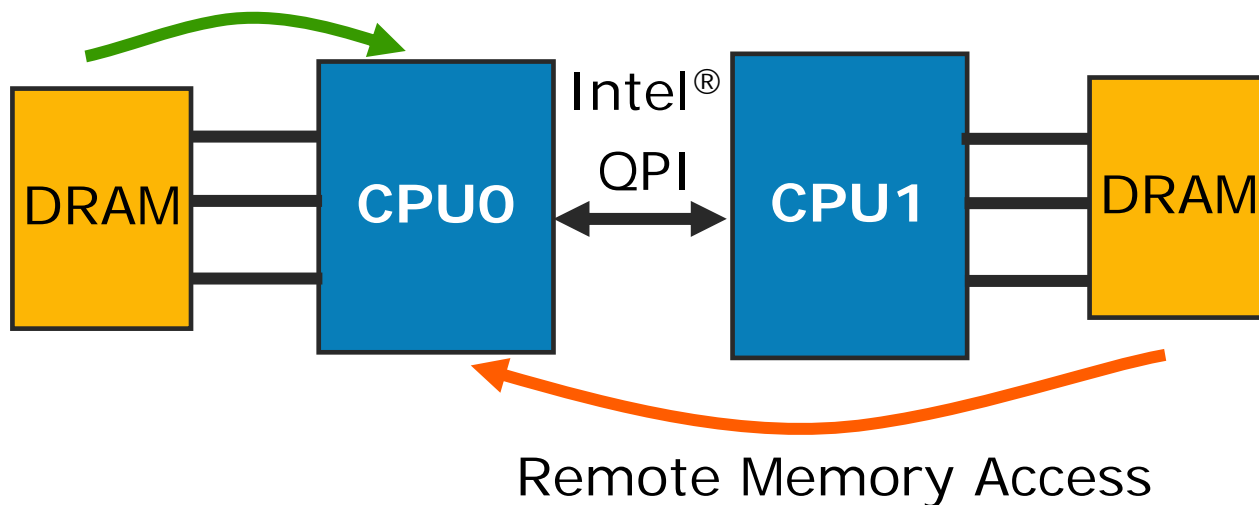
Agenda

1. Optimizing for Intel® Core™ i7 processor
 - Tuning recommendations
 - Tools
2. Parallelism strategies
 - CPU Enumeration and affinitization
 - Removing parallelism bottlenecks
3. Using Non-Uniform Memory Access (NUMA)
 - Locality Optimizations
 - Operating system differences

Non-Uniform Memory Access (NUMA)

- Expect most Multi-socket Intel® Core™ i7 processor platforms to use NUMA
- Locality matters
 - Remote memory access latency ~1.7x than local memory
 - local memory bandwidth can be up to 2x greater than remote
- Operating systems differ in allocation strategies + APIs

Local Memory Access



Operating System Differences

- Operating systems allocate data differently
- Linux*
 - Malloc reserves the memory
 - Assigns the physical page when data touched
- Microsoft Windows*
 - Malloc assigns the physical page on allocation
 - This default allocation policy is not NUMA friendly
- Microsoft Windows has NUMA Friendly API's
 - VirtualAlloc reserves memory (like malloc on Linux*)
 - Physical pages assigned at first use
- For more details:

<http://kernel.org/pub/linux/kernel/people/christoph/pmig/numamemory.pdf>

<http://www.opensolaris.org/os/community/performance/numa/>

<http://msdn.microsoft.com/en-us/library/aa363804.aspx>

Shared Memory Threading Example: TRIAD

- Parallelized time consuming hotspot “TRIAD” using OpenMP

```
main() {  
...  
  #pragma omp parallel  
  {  
    //Parallelized TRIAD loop...  
    #pragma omp parallel for private(j)  
      for (j=0; j<N; j++)  
        a[j] = b[j]+scalar*c[j];  
  } //end omp parallel  
...  
} //end main
```

Parallelizing hotspots may not be sufficient for NUMA

Shared Memory Threading Example – II

Linux*

```
KMP_AFFINITY=compact,0,verbose
```

Environment variable
to pin affinity

```
main() {
```

```
...
```

```
#pragma omp parallel  
{
```

```
#pragma omp for private(i)
```

```
for(i=0; i<N; i++)
```

```
{ a[i] = 10.0; b[i] = 10.0; c[i] = 10.0; }
```

```
...
```

```
//Parallelized TRIAD loop...
```

```
#pragma omp parallel for private(j)
```

```
for (j=0; j<N; j++)
```

```
a[j] = b[j]+scalar*c[j];
```

```
} //end omp parallel ...
```

```
} //end main
```

Each thread initializes its data
pinning the pages to local memory

Same thread that initialized
data uses data

Shared Memory Threading Example - III

Microsoft Windows*

```
KMP_AFFINITY=compact,0,verbose
```

Environment Variable
to pin affinity

```
main() {
```

```
...
```

```
    a* = (char *) VirtualAlloc(NULL, //same for b* and c*  
                                N*(sizeof(double))+1024,  
                                MEM_RESERVE | MEM_COMMIT,  
                                PAGE_READWRITE);
```

Reserves
Memory

```
...
```

```
#pragma omp parallel
```

```
{
```

```
    #pragma omp for private(i)
```

```
        for(i=0; i<N; i++)
```

```
            { a[i] = 10.0; b[i] = 10.0; c[i] = 10.0; }
```

```
...
```

```
    //OpenMP on TRIAD loop...
```

```
    #pragma omp parallel for private(j)
```

```
        for (j=0; j<N; j++)
```

```
            a[j] = b[j]+scalar*c[j];
```

```
    } //end omp parallel
```

```
...
```

```
} //end main
```

Each thread initializes its data
pinning pages to local memory

Same thread that initialized
data uses data

Summary

- Intel® Core™ i7 processor has many new performance features
 - Most you get for free...
 - Some you need to recompile
- More parallelization opportunities
 - Use CPU enumeration white paper
- Optimize for NUMA on multi-socket systems
 - Locality, Locality, Locality
 - Be aware of OS differences

Additional Sources of Information on This Topic:

- **Other Sessions / Chalk Talks / Labs:**

- **TCHS001:** Next Generation Intel® Core™ Microarchitecture (Nehalem) Family of Processors: Screaming Performance, Efficient Power (8/19, 3:00 – 3:50)
- **DPTS001:** High End Desktop Platform Design Overview for the Next Generation Intel® Microarchitecture (Nehalem) Processor (8/20, 2:40 – 3:30)
- **NGMS001:** Next Generation Intel® Microarchitecture (Nehalem) Family: Architectural Insights and Power Management (8/19, 4:00 – 5:50)
- **NGMC001:** Chalk Talk: Next Generation Intel® Microarchitecture (Nehalem) Family (8/19, 5:50 – 6:30)
- **NGMS002:** Tuning Your Software for the Next Generation Intel® Microarchitecture (Nehalem) Family (8/20, 11:10 – 12:00)
- **PWRS003:** Power Managing the Virtual Data Center with Windows Server* 2008 / Hyper-V and Next Generation Processor-based Intel® Servers Featuring Intel® Dynamic Power Technology (8/19, 3:00 – 3:50)
- **PWRS005:** Platform Power Management Options for Intel® Next Generation Server Processor Technology (Tylersburg-EP) (8/21, 1:40 – 2:30)
- **SVRS002:** Overview of the Intel® QuickPath Interconnect (8/21, 11:10 – 12:00)

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. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards 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.
- Nehalem, and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user
- 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.
- Intel, Intel Inside, Intel Core, and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
- Copyright © 2008 Intel Corporation.

Risk Factors

This presentation contains forward-looking statements that involve a number of risks and uncertainties. These statements do not reflect the potential impact of any mergers, acquisitions, divestitures, investments or other similar transactions that may be completed in the future. The information presented is accurate only as of today's date and will not be updated. In addition to any factors discussed in the presentation, the important factors that could cause actual results to differ materially include the following: Demand could be different from Intel's expectations due to factors including changes in business and economic conditions, including conditions in the credit market that could affect consumer confidence; customer acceptance of Intel's and competitors' products; changes in customer order patterns, including order cancellations; and changes in the level of inventory at customers. Intel's results could be affected by the timing of closing of acquisitions and divestitures. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of new Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel's response to such actions; Intel's ability to respond quickly to technological developments and to incorporate new features into its products; and the availability of sufficient supply of components from suppliers to meet demand. The gross margin percentage could vary significantly from expectations based on changes in revenue levels; product mix and pricing; capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; excess or obsolete inventory; manufacturing yields; changes in unit costs; impairments of long-lived assets, including manufacturing, assembly/test and intangible assets; and the timing and execution of the manufacturing ramp and associated costs, including start-up costs. Expenses, particularly certain marketing and compensation expenses, vary depending on the level of demand for Intel's products, the level of revenue and profits, and impairments of long-lived assets. Intel is in the midst of a structure and efficiency program that is resulting in several actions that could have an impact on expected expense levels and gross margin. Intel's results could be impacted by adverse economic, social, political and physical/infrastructure conditions in the countries in which Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. A detailed discussion of these and other factors that could affect Intel's results is included in Intel's SEC filings, including the report on Form 10-Q for the quarter ended June 28, 2008.