HPC Workshop – University of Kentucky
May 9, 2007 – May 10, 2007

Part 2 – Single Core Optimization (Intel)
Single Core Optimization

- **Execution Environment**
  - CPU Organization
  - Memory Organization
  - Instruction Set

- **Programming Considerations**
  - Memory Hierarchy
  - Loop Optimization
  - SIMD

- **Software Tools**
  - Compilers
  - Debugger
  - Profiling Tools
  - Math Libraries
Basic CPU Functional Blocks

System Bus

Fetch from Main memory

Processor

Local APIC
(Advanced Programmable Interrupt Controller)

On-die Cache
L1, L2, ...

Control Logic

Execution Resources
(ALU, Control Unit)

Interface Unit

Register Array

Store to Main memory

Decode and Execute Operations

Fetch from Main memory

Store to Main memory

Decode and Execute Operations
Basic Processor Execution Flow

Bus Interface Unit

Fetch/Decode

Carry out specific Task and
Output results
Intel Core Microarchitecture - WoodCrest

Source: Intel
Basic Execution Flow in a Super-Scalar Processor

Static Program

Instruction Fetch and Branch prediction

Window of Execution

Instruction Dispatch

Instruction Issue

Instruction Reorder and commit

Intel WoodCrest is 14-stage Super-Scalar pipeline
## Execution Environment - Modes

<table>
<thead>
<tr>
<th>Operating mode</th>
<th>Operating system required</th>
<th>Application rebuild required</th>
<th>Default address size</th>
<th>Default operand size</th>
<th>Register extensions</th>
<th>Typical GPR width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64-bit mode</td>
<td></td>
<td>Yes</td>
<td>64</td>
<td>32</td>
<td>Yes</td>
<td>64</td>
</tr>
<tr>
<td>Compatibility mode</td>
<td>New OS with 64-bit support</td>
<td>No</td>
<td>32</td>
<td>32</td>
<td>No</td>
<td>32</td>
</tr>
<tr>
<td>Legacy mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protected mode</td>
<td>Legacy 16-bit or 32-bit OS</td>
<td>No</td>
<td>32</td>
<td>32</td>
<td>No</td>
<td>32</td>
</tr>
<tr>
<td>Virtual 8086 mode</td>
<td>Legacy 16-bit OS</td>
<td>No</td>
<td>16</td>
<td>16</td>
<td>No</td>
<td>16</td>
</tr>
<tr>
<td>Real mode</td>
<td>Legacy 16-bit OS</td>
<td></td>
<td>16</td>
<td>16</td>
<td></td>
<td>16</td>
</tr>
</tbody>
</table>

*Source: Wikipedia*
Execution Environment – 64-bit mode

Basic Program Execution Registers
- Sixteen 64-bit General Purpose Registers
- Six 16-bit Segment Registers
- 64-bit RFLAGS Register
- 64-bit Instruction Pointer Register (RIP)

FPU Registers
- Eight 80-bit Floating-point Data Registers
- 16-bit Control Register
- 16-bit Status Register
- 16-bit Tag Register
- 64-bit FPU Instruction Pointer Register
- 64-bit FPU Data Pointer Register (RIP)

MMX Registers
- Eight 64-bit Registers

XMM Registers
- Sixteen 128-bit Registers

Address Space

$2^{64}-1$

$0$
Basic Memory Management Model

Flat Model

Linear Address

Linear Address Space

Segmented Model

Segments

Offset (effective address)

Segment Selector

Linear Address Space

Real-Address Mode Model

Offset

Linear Address Space Divided Into Equal Sized Segments

Logical Address

Segment Selector

Logical Address

* The linear address space can be paged when using the flat or segmented model.
Instruction Sets – very very basic

- **Data Transfer Instructions**
  - *MOV*, PUSH*, POP*

- **Binary Arithmetic Instructions**
  - ADD, ADC, SUB, MUL, CMP,…

- **Logical Instructions**
  - AND, OR, XOR, NOT

- **Shift and Rotate Instructions**
  - SAR, SAL, ROR, ROL,…

- **Bit and Byte Instructions**
  - SET*, BT*

- **Control Transfer Instructions**
  - JMP, J*, LOOP*,

- **String Instructions**
  - MOV, CMPS, SCAS, LODS,…

- **I/O Instructions**
  - IN, OUT, INS, OUTS

- **x87 FPU Instructions**
  - FADD, FMUL, FLD, FSQRT,…

- **SSE3 Instructions**
  - PHADDD, PHSUBW, ….
Programming Considerations

- Memory Hierarchy
- Optimization
- SIMD
Memory Hierarchy

- CPU
- Registers
- L1
  - ~14 cycles
  - 32 KB data
  - 32 KB Inst.
- L2
  - 4 MB (shared)
- Main Memory
  - ~270 cycles
  - very very large
- Disk
Some Cache Concepts
Cache Line Size and Set-Associativity

- **Line Size** – smallest unit of data that can be transferred to or from Memory
  - 64 bytes (Intel Woodcrest)

- **Set-Associativity** – Mapping Memory to Cache
  - 16-way (Intel Woodcrest)

- **Replacement policies** – Cache line replacement
  - Least Recently Used (Intel Woodcrest)
  - Write-back (Intel Woodcrest)
Virtual Memory and TLB

- Logical Address Space > Physical Address (RAM)
- Use Disk (SWAP area) and Paging
- To access a particular location in Memory:
  - Identify the Page Table for the memory location
  - Perform Virtual to Physical address Translation
  - Done through a special Cache called Traslation Lookaside Buffer (TLB)

Intel Woodcrest
- DTLB0 for Loads
- DTLB1 for Store and Loads that missed DTLB0
- 4KB pages or Large Pages
- 4-way set associative
- DTLB0 – 16 entries for 4KB pages and Large Pages
- DTLB1 – 256 entries for 4KB pages and 32 for Large Pages
- DTLB0 miss and DTLB1 hit causes 2-cylce penalty
SIMD

- MMX → SSE → SSE2 → SSE3 → Supplemental Streaming SIMD Extensions 3 (SSSE3)
- SIMD Integer operations can use the 64-bit MMX or the 128-bit XMM registers
- SIMD Floating-Point operations use the 128-bit XMM registers
- SSSE3 offers 32 instructions to accelerate processing of SIMD data
Typical SIMD Operation

X4  X3  X2  X1

Y4  Y3  Y2  Y1

X4 op Y4  X3 op Y3  X2 op Y2  X1 op Y1
Situations where SIMD is useful

- Applications that are inherently parallel
- Have recurring memory access patterns
- Localized recurring operations performed on the data
- Data-independent control flow
  - 3D-graphics, Speech recognition, Image processing, Scientific Applications
  - Intel compilers have excellent Auto Vectorization capabilities

ifort -O2 -axT -vec-report2 -c source.f90
Compilers and Math Libraries
(Andrew)
DEBUGGING

(some) Reasons:

- **Program doesn’t run**
  - Arguments missing
  - Not compiled properly
  - Libraries missing

- **Program is hung**
  - Dead-lock
  - Infinite Loops

- **Segmentation error**
  - Program is trying to access some memory that is not allowed

- **Bus error**
  - Program is trying to access some memory that is not “Physically” possible
  - Physical problems, Kernel problems

- **Wrong answers**
  - Potentially bad algorithm
  - Potential error in Parallel model
(Some) Debugging Tools

- **gdb (gnu debugger)** –
  - both on Power5+ and x86-64
- **idb (Intel debugger)**
  - only on x86-64
- **TotalView (Etnus)**
  - both on Power5+ and x86-64
- **PurifyPlus (IBM)**
  - Not installed at University of Kentucky
- **Valgrind (Open Source)**
  - Very good for memory debugging (for both Power5+ and x86-64)
For All Debugging:

- Turn off Optimization first
- Take a close look at the “warning” messages during compilation
- Turn Corefilesize to unlimited (ulimit -c unlimited)
  (watch out for large core dump file)
- -g compiler option to include debug instructions
- Understanding some Assembler Code will be useful
- Patience, Patience, Patience – Keep staring at the code
- Art, not Science
gdb – GNU Debugger (basics)

- Compile with -g option
- Various ways of starting gdb session:
  - gdb
    - run a.out command-line-args
  - gdb ./a.out
  - gdb ./a.out processID
  - gdb ./a.out core
- Can debug multi-threaded and MPI programs (in principle but difficult in practice)
Basic gdb commands:

- **run** – run the program
- **break [file:]function** – break point at the function
- **next** – Execute next program line (step over functions)
- **step** – Step into the next instruction (function)
- **continue** – continue execution until next break point
- **bt** – back-trace
- **print expr** – print value of an expressions
- **list** – list source code
- **help**
- **quit**
TotalView Debugger

- Powerful
- GUI or CLI
- Explore complex data
- Multi-platform
- Single and Multi-threaded, MPI, and OpenMP

Recommend attending the webcast by Etnus:
http://www.totalviewtech.com/webinars.php
GDB and TotalView Demo
Profiling - Basics

- **Non-intrusive**
  - No need for recompilation
  - Typically, monitors hardware performance monitors
  - Typically, stops at Function level profiling

- **Intrusive**
  - Need to recompile the code with Profiling/Instrumentation option (typically “-pg”)
  - Can perform instruction level or Source level profiling

- **Cycles, CPU Time, Cache Trashing, TLB Misses, Stalled instructions, etc.,**

(Warning: Intrusive profiling may not provide a true picture)
Profiling Tools

- **gprof (GNU)**
  - Need to recompile code with –pg
  - Good first level view of the code performance
  - Poor for drilling down into the code

- **OProfile (Open Source)**
  - System level profiling
  - Useful but hard to use
  - May require root access (Not sure)

- **VTune (Intel) – only on the Intel architecture**
  - GUI or CLI based
  - Can handle all types applications (multi-threaded, OpenMP, MPI, etc)
  - Easy to Drill down into the code

- **PurifyPlus (IBM)**

- **Etc.,**

  (Intel also has something called Thread Checker and Thread Profiler – may be useful)
gprof - basics

- Compile the code with -pg
- Run the executable a.out (generic name)
- When the code finishes there will be a “gmon.out” (if single threaded) in the Workdir
- gprof [options] a.out > a.gprof.out
- View/Edit the a.gprof.out file
Flat profile:

Each sample counts as 0.01 seconds.

<table>
<thead>
<tr>
<th>%</th>
<th>cumulative</th>
<th>self</th>
<th>calls</th>
<th>self ms/call</th>
<th>total ms/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>time seconds</td>
<td>seconds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33.34</td>
<td>0.02</td>
<td>0.02</td>
<td>7208</td>
<td>0.00</td>
<td>0.00</td>
<td>open</td>
</tr>
<tr>
<td>16.67</td>
<td>0.03</td>
<td>0.01</td>
<td>244</td>
<td>0.04</td>
<td>0.12</td>
<td>offtime</td>
</tr>
<tr>
<td>16.67</td>
<td>0.04</td>
<td>0.01</td>
<td>8</td>
<td>1.25</td>
<td>1.25</td>
<td>memccpy</td>
</tr>
<tr>
<td>16.67</td>
<td>0.05</td>
<td>0.01</td>
<td>7</td>
<td>1.43</td>
<td>1.43</td>
<td>write</td>
</tr>
<tr>
<td>16.67</td>
<td>0.06</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td>mcount</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>236</td>
<td>0.00</td>
<td>0.00</td>
<td>tzset</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>192</td>
<td>0.00</td>
<td>0.00</td>
<td>tolower</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>47</td>
<td>0.00</td>
<td>0.00</td>
<td>strlen</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>45</td>
<td>0.00</td>
<td>0.00</td>
<td>strchr</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>50.00</td>
<td>main</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>memcpy</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>10.11</td>
<td>print</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>profil</td>
</tr>
<tr>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>50.00</td>
<td>report</td>
</tr>
</tbody>
</table>

...
VTune – Drill down

- Start to Finish View
  - Total_Cycles_Completion
    - Issuing_uops
    - Not_Issuing_uops
  - RS View
    - Non_retiring_uops
    - Retiring_uops
    - Stalled
  - Execution View
    - Store Fwd
    - LCP
    - Cache Miss
    - ... (other stalls)
  - Stalls Drill-down
    - Code Layout, Branch Misprediction
    - Vectorize w/ SIMD
    - Identify hot spot code, apply fix
  - Tuning Focus
    - Apply one fix at time; repeat from the top
  - Tuning Consistency

OM19805
VTune Demo
End of First Day