Advanced Systems Lab
Spring 2020
Lecture: Architecture/Microarchitecture and Intel Core

Instructor: Markus Püschel, Ce Zhang
TA: Joao Rivera, Bojan Karlas, several more

Organization

- Research project: Deadline March 6th
- Finding team: fastcode-forum@lists.inf.ethz.ch
Today

- Architecture/Microarchitecture: What is the difference?
- In detail: Intel Haswell and Sandybridge
- Crucial microarchitectural parameters
- Peak performance
- Operational intensity

Definitions

- **Architecture** (also instruction set architecture = ISA): The parts of a processor design that one needs to understand to write assembly code
- **Examples**: instruction set specification, registers
- **Counterexamples**: cache sizes and core frequency
- **Example ISAs**
  - x86
  - ia
  - MIPS
  - POWER
  - SPARC
  - ARM

Some assembly code

```assembly
ipf:
 xorps  %xmm1, %xmm1
 xorl  %ecx, %ecx
 jmp .L8
.L8:
 movslq  %ecx,%rax
 incl  %ecx
 movss (%rsi,%rax,4), %xmm0
 mulss (%rdi,%rax,4), %xmm0
 addss %xmm0, %xmm1
 cmpq  %edx, %ecx
 jl .L8
 movaps  %xmm1, %xmm0
 ret
```

### ISA SIMD (Single Instruction Multiple Data) Vector Extensions

**What is it?**
- Extension of the ISA. Data types and instructions for the parallel computation on short (length 2–8) vectors of integers or floats

<table>
<thead>
<tr>
<th>4-way</th>
<th>4-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>🟥</td>
<td>🟥</td>
</tr>
<tr>
<td>🟦</td>
<td>🟦</td>
</tr>
</tbody>
</table>

- Names: MMX, SSE, SSE2, ..., AVX, ...

**Why do they exist?**
- **Useful:** Many applications have the necessary fine-grain parallelism
  - Then: speedup by a factor close to vector length
- **Doable:** Chip designers have enough transistors to play with; easy to build with replication

**We will have an extra lecture on vector instructions**
- What are the problems?
- How to use them efficiently
FMA = Fused Multiply-Add

- \( x = x + yz \)
- Done as one operation, i.e., involves only one rounding step
- Better accuracy than sequence of mult and add
- Natural pattern in many algorithms

```c
/* matrix multiplication; A, B, C are n x n matrices of doubles */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++)
            C[i*n+j] += A[i*n+k]*B[k*n+j];
```

- Exists only recently in Intel processors (Why?)

---

### MMX:
Multimedia extension

### SSE:
Streaming SIMD extension

### AVX:
Advanced vector extensions

<table>
<thead>
<tr>
<th>Intel x86</th>
<th>Processors (subset)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86-16</td>
<td>8086</td>
</tr>
<tr>
<td>x86-32</td>
<td>386, 486, Pentium</td>
</tr>
<tr>
<td>x86-64</td>
<td>MMX, SSE, SSE2, SSE3</td>
</tr>
<tr>
<td></td>
<td>MMX, Pentium MMX,</td>
</tr>
<tr>
<td></td>
<td>Pentium III, Pentium 4</td>
</tr>
<tr>
<td></td>
<td>Pentium 4, Pentium 4E</td>
</tr>
<tr>
<td></td>
<td>SSE4, AVX, AVX2</td>
</tr>
<tr>
<td></td>
<td>Pentium 4F, Core 2, Pentium 4F</td>
</tr>
<tr>
<td></td>
<td>Core i3/5/7, Sandy Bridge</td>
</tr>
<tr>
<td></td>
<td>Haswell, Skylake-X</td>
</tr>
</tbody>
</table>

- 4-way single
- 2-way double
- 8-way single, 4-way double
- FMA
- 16-way single, 8-way double
- AVX
- AVX-512
Definitions

- **Microarchitecture**: Implementation of the architecture

- **Examples**: Caches, cache structure, CPU frequency, details of the virtual memory system

Examples
- Intel processors ([Wikipedia](https://en.wikipedia.org/wiki/Intel_processor))
- AMD processors ([Wikipedia](https://en.wikipedia.org/wiki/AMD))

**Intel’s Tick-Tock Model**

- Tick: Shrink of process technology
- Tock: New microarchitecture
- 2016: Tick-tock model got discontinued
  - **Now**: process (tick) architecture (tock) optimization (opt)

- Example: Core and successors
  - Shown: Intel’s microarchitecture code names (server/mobile may be different)
Intel Processors: Example Haswell

http://www.anandtech.com

Microarchitecture:
The View of the Computer Architect

Distribute microarchitecture abstraction

Abstracted Microarchitecture: Example Core i7 Haswell (2013) and Sandybridge (2011)

Throughput (tp) is measured in doubles/cycle. For example: 4 doubles/cycle. Numbers are for loading into registers.
Latency (lat) is measured in cycles.
1 double floating point (FP) = 8 bytes
fma = fused multiply-add

Rectangles not to scale

Core i7-4770 Haswell:
4 cores, 8 threads
3.4 GHz (3.9 GHz max turbo freq)
2 DDR3 channels 1600 MHz

Memory hierarchy:
- Registers
- L1 cache
- L2 cache
- L3 cache
- Main memory
- Hard disk

ISA

Processor die

1 Core

ISA

16 FP register

Lat: 4
Tp: 4

Lat: 12
Tp: 8

Lat: 34
Tp: 4

Lat: ~125
Tp: 2

L1 Dcache

Shared L3 cache 8 MB 16-way 64B CB

Main Memory (RAM) 12 GB max

lat: millions
tp: ~1/50 (~2/200)

Hard disk ≥ 0.5 TB

depends on hard disk technology, I/O interconnect

### Runtime Bounds (Cycles) on Haswell

```c
/* x, y are vectors of doubles of length n, alpha is a double */
for (i = 0; i < n; i++)
    x[i] = x[i] + alpha*y[i];
```

- **Number flops?** 2n
- **Runtime bound no vector ops:** $\frac{n}{2}$
- **Runtime bound vector ops:** $\frac{n}{8}$
- **Runtime bound data in L1:** $\frac{n}{4}$ 50
- **Runtime bound data in L2:** $\frac{n}{4}$ 50
- **Runtime bound data in L3:** $\frac{n}{2}$ 25
- **Runtime bound data in main memory:** n 12.5

---

**Runtime dominated by data movement:**

**Memory-bound**

---

### Runtime Bounds (Cycles) on Haswell

```c
/* matrix multiplication; A, B, C are n x n matrices of doubles */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++)
            C[i*n+j] += A[i*n+k]*B[k*n+j];
```

- **Number flops?** $2n^3$
- **Runtime bound no vector ops:** $\frac{n^3}{2}$
- **Runtime bound vector ops:** $\frac{n^3}{8}$
- **Runtime bound data in L1:** $(3/8) n^2$
- **...**
- **Runtime bound data in main memory:** $(3/2) n^2$

---

**Runtime dominated by data operations (except very small n):**

**Compute-bound**
Operational Intensity

- Definition: Given a program P, assume cold (empty) cache

\[
I(n) = \frac{W(n)}{Q(n)}
\]

Operational Intensity (Cold Cache)

- Flops: \( W(n) = 2n \)
- Memory transfers (doubles): \( \geq 2n \) (just from the reads)
- Reads (bytes): \( Q(n) \geq 16n \)
- Operational intensity: \( I(n) = \frac{W(n)}{Q(n)} \leq 1/8 \)
Operational Intensity (Cold Cache)

```c
/* matrix multiplication; A, B, C are n x n matrices of doubles */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++)
            C[i*n+j] += A[i*n+k]*B[k*n+j];
```

- **Operational intensity:**
  - Flops: \( W(n) = 2n^3 \)
  - Memory transfers (doubles): \( \geq 3n^2 \) (just from the reads)
  - Reads (bytes): \( Q(n) \geq 24n^2 \)
  - Operational intensity: \( I(n) = W(n)/Q(n) \leq n/12 \)

Compute/Memory Bound

- A function/piece of code is:
  - **Compute bound** if it has high operational intensity
  - **Memory bound** if it has low operational intensity

- A more exact definition depends on the given platform
- More details later: Roofline model
Mapping of execution units to ports

<table>
<thead>
<tr>
<th>Port 0</th>
<th>Port 1</th>
<th>Port 2</th>
<th>Port 3</th>
<th>Port 4</th>
<th>Port 5</th>
<th>Port 6</th>
<th>Port 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>fp fma</td>
<td>fp fma</td>
<td>load</td>
<td>load</td>
<td>store</td>
<td>SIMD log</td>
<td>Int ALU</td>
<td>st addr</td>
</tr>
<tr>
<td>fp mul</td>
<td>fp mul</td>
<td>st addr</td>
<td>st addr</td>
<td>shuffle</td>
<td>fp mov</td>
<td>Int ALU</td>
<td></td>
</tr>
<tr>
<td>fp div</td>
<td>fp add</td>
<td>SIMD log</td>
<td>SIMD log</td>
<td>Int ALU</td>
<td>SIMD log</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*execution units*

- Every port can issue one instruction/cycle
- Gap = 1/throughput
- Intel calls gap the throughput!
- Same exec units for scalar and vector flops
- Same latency/throughput for scalar (one double) and AVX vector (four doubles) flops, except for div

**Execution Unit (fp)** | **Latency (cycles)** | **Throughput (ops/cycle)** | **Gap (cycles/issue)**
---|---|---|---
fp fma | 5 | 2 | 0.5
mul | 5 | 2 | 0.5
add | 3 | 1 | 1
div (scalar) | 14-20 | 1/13 | 13
div (4-way) | 25-35 | 1/27 | 27

***SIMD log:*** other, non-fp SIMD ops

Floating Point Registers

- 16 ymm (AVX)
- 16 xmm (SSE)
- Scalar (single precision)

Each register: 256 bits = 4 doubles = 8 singles

- Same 16 registers for scalar FP, SSE and AVX
- Scalar (non-vector) single precision FP code uses the bottom eighth
- Explains why throughput and latency is usually the same for vector and scalar operations
How many cycles are at least required (no vector ops)?

- A function with n adds and n mults in the C code \( \frac{n}{2} \)
- A function with n add and n mult instructions in the assembly code \( n \)
- A function with n adds in the C code \( \frac{n}{2} \)
- A function with n add instructions in the assembly code \( n \)
- A function with n adds and n/2 mults in the C code \( \frac{n}{2} \)

Comments on Intel Haswell μarch

- Peak performance 16 DP flops/cycle (only reached if SIMD FMA)
  - Peak performance mults: 2 mults/cycle (scalar 2 flops/cycle, SIMD AVX 8 flops/cycle)
  - Peak performance adds: 1 add/cycle (scalar 1 flop/cycle, SIMD AVX 4 flops/cycle)
  - FMA in port 0 can be used for add, but longer latency

- L1 bandwidth: two 32-byte loads and one 32-byte store per cycle
- Shared L3 cache organized as multiple cache slices for better scalability with number of cores, thus access time is non-uniform
- Shared L3 cache in a different clock domain (uncore)
**Example: Peak Performance**

![Graph showing matrix-matrix multiplication performance on a 2x2 Core 2 Duo 3 GHz Sandy Bridge system.]

**Peak performance of this computer:**

- 4 cores x 2-way SSE x 1 add and 1 mult/cycle  
  = 16 flops/cycle  
  = 48 Gflop/s

---

**Summary**

- **Architecture vs. microarchitecture**
- To optimize code one needs to understand a suitable abstraction of the microarchitecture and its key quantitative characteristics
  - Memory hierarchy with throughput and latency info
  - Execution units with port, throughput, and latency info
- **Operational intensity:**
  - High = compute bound = runtime dominated by data operations
  - Low = memory bound = runtime dominated by data movement