How to Write Fast Numerical Code
Spring 2019

Lecture: Architecture/Microarchitecture and Intel Core

Instructor: Markus Püschel
TA: Tyler Smith, Gagandeep Singh, Alen Stojanov

Technicalities

- Midterm: Mon, April 15th
- Research project:
  - Let us know once you have a partner
  - If you have a project idea, talk to me (break, after class, email)
  - Deadline: March 4th
- Finding partner: 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
.L10:
    movslq %ecx,%rax
    incl %ecx
    movss (%rsi,%rax,4), %xmm0
    mulss (%rdi,%rax,4), %xmm0
    addss %xmm0, %xmm1
.L8:
    cmpl %edx, %ecx
    jl .L10
    movaps %xmm1, %xmm0
    ret
```
**MMX:** Multimedia extension

**SSE:** Streaming SIMD extension

**AVX:** Advanced vector extensions

*Backward compatible:* Old binary code (≥ 8086) runs on new processors.

*New code to run on old processors?* Depends on compiler flags.

---

**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
  - \[+\] \[\times\] 4-way
  - 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

- \( x = x + y \cdot z \)
- 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?*)

---

<table>
<thead>
<tr>
<th>MMX: Multimedia extension</th>
<th>Intel x86</th>
<th>Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x86-16</td>
<td>8086</td>
</tr>
<tr>
<td></td>
<td></td>
<td>286</td>
</tr>
<tr>
<td></td>
<td>x86-32</td>
<td>386</td>
</tr>
<tr>
<td></td>
<td>MMX</td>
<td>Pentium</td>
</tr>
<tr>
<td></td>
<td>SSE</td>
<td>Pentium MMX</td>
</tr>
<tr>
<td></td>
<td>SSE2</td>
<td>Pentium III</td>
</tr>
<tr>
<td></td>
<td>SSE3</td>
<td>Pentium 4</td>
</tr>
<tr>
<td></td>
<td>x86-64</td>
<td>Pentium 4F</td>
</tr>
<tr>
<td></td>
<td>SSE4</td>
<td>Core 2 Duo</td>
</tr>
<tr>
<td></td>
<td>AVX</td>
<td>Penryn</td>
</tr>
<tr>
<td></td>
<td>AVX2</td>
<td>Core i7 (Nehalem)</td>
</tr>
<tr>
<td></td>
<td>AVX-512</td>
<td>Sandy Bridge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Haswell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skylake-X</td>
</tr>
</tbody>
</table>

4-way single
2-way double
8-way single, 4-way double
16-way single, 8-way double

---

© Markus Püschel
Computer Science
ETH Zürich
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_family))
- AMD processors ([Wikipedia](https://en.wikipedia.org/wiki/AMD_processors))

Intel’s Tick-Tock Model

- **Tick**: Shrink of process technology
- **Tock**: New microarchitecture
- **Example**: Core and successors
  Shown: Intel’s microarchitecture code names (server/mobile may be different)

```
<table>
<thead>
<tr>
<th>Year</th>
<th>Technology</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>65 nm</td>
<td>Core Conroe - Wolfdale</td>
</tr>
<tr>
<td>2010</td>
<td>45 nm</td>
<td>Nehalem Nehalem - Westmere</td>
</tr>
<tr>
<td>2012</td>
<td>32 nm</td>
<td>Sandy Bridge Sandy Bridge – Ivy Bridge</td>
</tr>
<tr>
<td>2014</td>
<td>22 nm</td>
<td>Haswell Haswell - Broadwell</td>
</tr>
<tr>
<td>2017</td>
<td>14 nm</td>
<td>Skylake Skylake – Kaby Lake</td>
</tr>
</tbody>
</table>
```

In 2016 the Tick-tock model got discontinued
Now: process – architecture – optimization (since Tick becomes harder)
Intel Processors: Example Haswell

Detailed information about Intel processors

Microarchitecture:
The View of the Computer Architect

we take the software developer’s view ...

Distribute microarchitecture abstraction

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

Throughput (tp) is measured in doubles/cycle. For example: 4 (2).
Latency (lat) is measured in cycles.
1 double floating point (FP) = 8 bytes
fma = fused multiply add

Rectangles not to scale

Hard disk ≥ 0.5 TB

ISA

16 FP register

L1 Dcache

Lat: 4

Lat: 12 = 8 + 11 + 4

Lat: ~134

both:

L2 cache 256 KB 8-way 64 KB

L3 cache 8 MB 16-way 64 KB

Shared L3 cache 8 MB 16-way 64 KB

Main Memory (RAM) 12 GB max

Lat: ~125

Lat: ~1/50

32 KB

8-way

64 KB

L2 cache

L1 Dcache

L1 icache

OISC ops

Instruction decoder

Instruction pool

(upto 192 (168) “in flight”)

Internal registers

Out of order execution

Superscalar

Load

Store

SIMD logic/shuffle

Execution units

fp add

fp mul

fp fma

int ALU

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


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

Core i7-4790 Haswell:
- 4 cores, 8 threads
- 3.6 GHz
- 3.9 GHz turbo freq
- 2 DDR3 channels 1600 MHz

Core i7-5775K Haswell:
- 4 cores, 8 threads
- 3.5 GHz
- 4.0 GHz turbo freq
- 2 DDR3 channels 1600 MHz

Core i7-6800K Haswell:
- 8 cores, 16 threads
- 3.4 GHz
- 3.8 GHz turbo freq
- 2 DDR3 channels 1600 MHz

Core i7-8700K Haswell:
- 8 cores, 16 threads
- 3.7 GHz
- 4.7 GHz turbo freq
- 2 DDR3 channels 1600 MHz

Core i7-9700K Haswell:
- 8 cores, 16 threads
- 3.6 GHz
- 4.9 GHz turbo freq
- 2 DDR3 channels 1600 MHz

Core i7-10700K Haswell:
- 8 cores, 16 threads
- 3.8 GHz
- 5.1 GHz turbo freq
- 2 DDR3 channels 1600 MHz

Core i7-11700K Haswell:
- 8 cores, 16 threads
- 3.6 GHz
- 5.0 GHz turbo freq
- 2 DDR3 channels 1600 MHz

Core i7-12700K Haswell:
- 8 cores, 16 threads
- 3.6 GHz
- 5.0 GHz turbo freq
- 2 DDR3 channels 1600 MHz
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:** n/2
- **Runtime bound vector ops:** n/8
- **Runtime bound data in L1:** n/4 50
- **Runtime bound data in L2:** n/4 50
- **Runtime bound data in L3:** n/2 25
- **Runtime bound data in main memory:** n 12.5

**Runtime dominated by data movement:** Memory-bound

Runtime Bounds (Cycles) on Core 2

```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:** n^3/2
- **Runtime bound vector ops:** 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: \( I(n) = \frac{W(n)}{Q(n)} \)

*Operational intensity: \( I(n) = \frac{W(n)}{Q(n)} \)*

- 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: \( I(n) = W(n)/Q(n) \leq 1/12 n \)

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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int ALU</td>
<td>Int ALU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Execution Units**
- fp = floating point
- log = logic
- fp units do scalar and vector flops
- SIMD log: other, non-fp SIMD ops

<table>
<thead>
<tr>
<th>Execution Unit</th>
<th>Latency (cycles)</th>
<th>Throughput (ops/cycle)</th>
<th>Gap (cycles/issue)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fma</td>
<td>5</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>mul</td>
<td>5</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>add</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>div (scalar)</td>
<td>14-20</td>
<td>1/13</td>
<td>13</td>
</tr>
<tr>
<td>div (4-way)</td>
<td>25-35</td>
<td>1/27</td>
<td>27</td>
</tr>
</tbody>
</table>

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

Source: Intel manual, Table C.8. 256-bit AVX instructions, Table 2-6. Dispatch Port and Execution Stacks of the Haswell Microarchitecture, Figure 2-2. CPU Core Pipeline Functionality of the Haswell Microarchitecture.

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 \( \frac{n}{2} \) 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** (*Sandy Bridge, either one 16-byte load and one 16-byte store, or one 32-byte load*)

- **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

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz (Sandy Bridge)

Performance [Gflop/s]

Peak performance of this computer:
- 4 cores
- 2-way SSE
- 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