Advanced Systems Lab
Spring 2022
Lecture: Architecture/Microarchitecture and Intel Core

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

Organization

Research project: Deadline *March 11th*
Finding team: fastcode-forum@lists.inf.ethz.ch
Today

Architecture/Microarchitecture: \textit{What is the difference?}

In detail: Intel Skylake

Crucial microarchitectural parameters

Peak performance

Operational intensity

Brief: Apple M1 processor

Definitions

\textit{Architecture (also instruction set architecture = ISA):} The parts of a processor design that one needs to understand to write assembly code

\textit{Examples:} instruction set specification, registers

\textit{Counterexamples:} cache sizes and core frequency

Example ISAs

\begin{itemize}
  \item \textit{x86}
  \item \textit{MIPS}
  \item \textit{POWER}
  \item \textit{SPARC}
  \item \textit{ARM}
\end{itemize}

\begin{center}
\begin{tabular}{|l|l|}
\hline
Some assembly code & \\
\hline
\verb|spf:| & \\
\verb|xorps %xmm1, %xmm1| & \\
\verb|xorl %ecx, %ecx| & \\
\verb|jmp .l8| & \\
\verb|.l8:| & \\
\verb|movslq %ecx,%rax| & \\
\verb|incl %ecx| & \\
\verb|movss (%rsi,%rax,4), %xmm0| & \\
\verb|mull (%rdi,%rax,4), %xmm0| & \\
\verb|addss %xmm0, %xmm1| & \\
\verb|.l8:| & \\
\verb|cmp %edx, %ecx| & \\
\verb|jl .l10| & \\
\verb|movaps %xmm1, %xmm0| & \\
\verb|ret| & \\
\hline
\end{tabular}
\end{center}
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

- 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 (subset)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE: Streaming SIMD extension</td>
<td>x86-16</td>
<td>8086, 286</td>
</tr>
<tr>
<td>AVX: Advanced vector extensions</td>
<td>x86-32</td>
<td>386, 486</td>
</tr>
<tr>
<td></td>
<td>x86-64</td>
<td>Pentium, Pentium MMX, Pentium III, Pentium 4, Pentium 4E</td>
</tr>
<tr>
<td>4-way single</td>
<td>MMX</td>
<td>8086, 286</td>
</tr>
<tr>
<td>2-way double</td>
<td>SSE</td>
<td>386, 486</td>
</tr>
<tr>
<td></td>
<td>SSE2</td>
<td>Pentium, Pentium MMX, Pentium III, Pentium 4, Pentium 4E</td>
</tr>
<tr>
<td></td>
<td>SSE3</td>
<td>386, 486</td>
</tr>
<tr>
<td>8-way single, 4-way double</td>
<td>SSE4</td>
<td>Pentium 4F, Core 2, Penryn, Core i3/5/7</td>
</tr>
<tr>
<td>FMAs</td>
<td>AVX</td>
<td>Sandy Bridge, Haswell</td>
</tr>
<tr>
<td>16-way single, 8-way double</td>
<td>AVX2</td>
<td>Skylake-X</td>
</tr>
<tr>
<td></td>
<td>AVX-512</td>
<td></td>
</tr>
</tbody>
</table>

---

MMX, SSE, AVX: Intel x86 Processors (subset)
Definitions

*Microarchitecture:* Implementation of the architecture

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

Examples
- Intel processors ([Wikipedia](#))
- AMD processors ([Wikipedia](#))

**Intel’s Tick-Tock Model**

Tick: Shrink of process technology

Tock: New microarchitecture

2016: Tick-tocck 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 Skylake

http://www.anandtech.com

Detailed information about Intel processors

Microarchitecture:
The View of the Computer Architect

we take the software developer’s view ...

Distribute microarchitecture abstraction

Abstrated Microarchitecture: Example Core i7 Skylake (2015)

Latency (lat) is measured in cycles.
1 double floating point (FP) = 8 bytes

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

Hard disk

≥ 0.5 TB

Throughput (tp) is measured in doubles/cycle. For example: 4.

Numbers are for loading into registers.

1 double floating point (FP) = 8 bytes

fma = fused multiply-add

Rectangles not to scale

Hard disk

≥ 0.5 TB

Runtime Lower Bounds (Cycles) on Skylake

/* 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];

Consider reads only

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 Lower Bounds (Cycles) on Skylake

/* 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];

Consider reads only

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

\[
\text{Operational intensity: } I(n) = \frac{W(n)}{Q(n)}
\]

Lower bounds for \( Q(n) \) yield upper bounds for \( I(n) \)

Sometimes we only consider reads from memory \( Q_{\text{read}}(n) \leq Q(n) \)
and thus \( I_{\text{read}}(n) \geq I(n) \)

Operational Intensity (Cold Cache)

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

Operational intensity (reads only):

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

/* matrix multiplication; \(A, B, C\) are \(n \times n\) 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 (reads only):
- **Flops**: \(W(n) = 2n^3\)
- **Memory transfers (doubles)**: \(\geq 3n^2\) (just from the reads)
- **Reads (bytes)**: \(Q_{\text{read}}(n) \geq 24n^2\)
- **Operational intensity**: \(l(n) \leq l_{\text{read}}(n) = \frac{W(n)}{Q_{\text{read}}(n)} \leq \frac{n}{12}\)

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
Superscalar Processor

Definition: A superscalar processor can issue and execute *multiple instructions in one cycle*. The instructions are retrieved from a sequential instruction stream and are usually scheduled dynamically.

Benefit: Superscalar processors can take advantage of *instruction level parallelism (ILP)* that many programs have.

Most CPUs since about 1998 are superscalar

Intel: since Pentium Pro

Simple embedded processors are usually not superscalar

---

Execution Units and Ports (Skylake)

<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><strong>fp fma</strong></td>
<td><strong>fp fma</strong></td>
<td><strong>load</strong></td>
<td><strong>load</strong></td>
<td><strong>store</strong></td>
<td><strong>SIMD log</strong></td>
<td><strong>Int ALU</strong></td>
<td><strong>st addr</strong></td>
</tr>
<tr>
<td><strong>fp mul</strong></td>
<td><strong>fp mul</strong></td>
<td><strong>st addr</strong></td>
<td><strong>st addr</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>shuffle</strong></td>
</tr>
<tr>
<td><strong>fp add</strong></td>
<td><strong>fp add</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>fp mov</strong></td>
</tr>
<tr>
<td><strong>fp div</strong></td>
<td><strong>SIMD log</strong></td>
<td><strong>Int ALU</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Int ALU</strong></td>
</tr>
<tr>
<td><strong>SIMD log</strong></td>
<td><strong>Int ALU</strong></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></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fma</td>
<td>4</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>mul</td>
<td>4</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>add</td>
<td>4</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>div (scalar)</td>
<td>14</td>
<td>1/4</td>
<td>4</td>
</tr>
<tr>
<td>div (4-way)</td>
<td>14</td>
<td>1/8</td>
<td>8</td>
</tr>
</tbody>
</table>

- Every port can issue one instruction/cycle
- Gap = 1/throughput
- *Intel says gap for 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

Source: Intel manual (Table C-8. 256-bit AVX Instructions, Table 2-1. Dispatch Port and Execution Stacks of the Skylake Microarchitecture, Figure 2-1. CPU Core Pipeline Functionality of the Skylake Microarchitecture)
Notes on Previous Slide

The availability of more than one port makes the processor superscalar.

Execution units behind different ports can start an operation in the same cycle (superscalar).

Execution units behind the same port cannot start an operation in the same cycle.

Adds, Mults and FMAs have throughput of 2 because they have 2 units behind 2 different ports. Each of these units has a throughput of 1.

An execution unit with throughput 1 is called fully pipelined.

By default the compiler does not use FMAs for single adds or mults.

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
- A function with $n$ add and $n$ mult instructions in the assembly code
- A function with $n$ adds in the C code
- A function with $n$ add instructions in the assembly code
- A function with $n$ adds and $n/2$ mults in the C code

Comments on Intel Skylake Lake \(\mu\)arch

Peak performance 16 double precision 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: 2 adds/cycle (scalar 2 flop/cycle, SIMD AVX 8 flops/cycle)

L1 bandwidth: two 32-byte loads \textit{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

Peak performance of this computer:
4 cores x
2-way SSE x
1 add and 1 mult/cycle
= 16 flops/cycle
= 48 Gflop/s

About M1 Processor

Release: November 2020
8 cores: 4 High-performance and 4 energy-efficient cores
ISA: ARMv8.4 (includes NEON 128-bit vector extension)
Microarchitecture: Firestorm (High-perf) and Icestorm (Energy-efficient)
Technology: 5nm

Successors (released in October 2021):
- **M1 Pro**: Same as M1 but with 10 cores and twice L3 cache.
- **M1 Max**: Same as M1 Pro but with twice L3 cache (also bigger GPU).

<table>
<thead>
<tr>
<th>Cache</th>
<th>M1 (Firestorm)</th>
<th>M1 (Icestorm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1-I</td>
<td>192K</td>
<td>128K</td>
</tr>
<tr>
<td>L1-D</td>
<td>128K</td>
<td>64K</td>
</tr>
<tr>
<td>L2</td>
<td>12M Shared by four Firestorm cores</td>
<td>4M Shared by Icestorm cores</td>
</tr>
<tr>
<td>L3 (SLC)</td>
<td>16M, 24M (Pro) or 48M (Max). Shared by all cores and GPU.</td>
<td></td>
</tr>
</tbody>
</table>
Firestorm Microarchitecture

**Integer ports:**
1: alu + flags + branch + addr + mrs/nzcv + mrs
2: alu + flags + branch + addr + mrs/nzcv + ptrauth
3: alu + flags + mov-from-simd/fp?
4: alu + mov-from-simd/fp
5: alu + mul + div
6: alu + mul + madd + crc + bfm/extr

**Load and store ports:**
7: store + amx
8: load/store + amx
9: load
10: load

**FP/SIMD ports:**
11: fp/simd
12: fp/simd
13: fp/simd + fcsel + to-gpr
14: fp/simd + fcsel + to-gpr + fcmpeq/e + fdiv + ...

This information is based on black-box reverse engineering.
https://dougallj.github.io/applecpu/firestorm.html

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Latency [cycles]</th>
<th>Gap [cycles/issue]</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>3</td>
<td>0.25</td>
</tr>
<tr>
<td>mul</td>
<td>4</td>
<td>0.25</td>
</tr>
<tr>
<td>div</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>load</td>
<td></td>
<td>0.33</td>
</tr>
<tr>
<td>store</td>
<td></td>
<td>0.5</td>
</tr>
</tbody>
</table>

Latency and gap of FP instructions in double precision. The numbers are the same for scalar and vector instructions.

Icestorm Microarchitecture

**Integer ports:**
1: alu + br + mrs
2: alu + br + div + ptrauth
3: alu + mul + bfm + crc

**Load and store ports:**
4: load/store + amx
5: load

**FP/SIMD ports:**
6: fp/simd
7: fp/simd + fcsel + to-gpr + fcmpeq/e + fdiv + ...

This information is based on black-box reverse engineering.
https://dougallj.github.io/applecpu/icestorm.html

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Latency [cycles]</th>
<th>Gap [cycles/issue]</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>3</td>
<td>0.5</td>
</tr>
<tr>
<td>mul</td>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td>div (scalar)</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>div (2-way)</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>load</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>store</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Latency and gap of FP instructions in double precision. The numbers are the same for scalar and vector instructions except for div.
Apple’s Microarchitectures

Example: A series (used in iPhones and iPads)

<table>
<thead>
<tr>
<th>Year</th>
<th>Microarchitecture</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>Cyclone</td>
<td>28 nm</td>
</tr>
<tr>
<td>2014</td>
<td>Typhoon</td>
<td>20 nm</td>
</tr>
<tr>
<td>2015</td>
<td>Twister</td>
<td>16 nm</td>
</tr>
<tr>
<td>2016</td>
<td>Hurricane, Zephyr</td>
<td>10 nm</td>
</tr>
<tr>
<td>2017</td>
<td>Monsoon, Mistral</td>
<td>10 nm</td>
</tr>
<tr>
<td>2018</td>
<td>Vortex, Tempest</td>
<td>7 nm</td>
</tr>
<tr>
<td>2019</td>
<td>Lightning, Thunder</td>
<td>7 nm</td>
</tr>
<tr>
<td>2020</td>
<td>Firestorm, Icestorm</td>
<td>5 nm</td>
</tr>
<tr>
<td>2021</td>
<td>Avalanche, Blizzard</td>
<td>5 nm</td>
</tr>
</tbody>
</table>

Firestorm, Icestorm are the only ones currently used in M series processors (used for MacBook, iMacs and iPad Pro)

https://en.wikipedia.org/wiki/Apple_silicon

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