How to Write Fast Numerical Code

Spring 2017

Lecture: Architecture/Microarchitecture and Intel Core

Instructor: Markus Püschel

TA: Alen Stojanov, Georg Ofenbeck, Gagandeep Singh

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technicalities

- **Class this Wednesday, no class this Thursday**
- **Midterm: *Wed, April 26th* (during recitation time)**
- **Research project:**
  - Let us know once you have a partner
  - If you have a project idea, talk to me (break, after Wed class, email)
  - Deadline: ***March 6th***
- **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

```
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
```

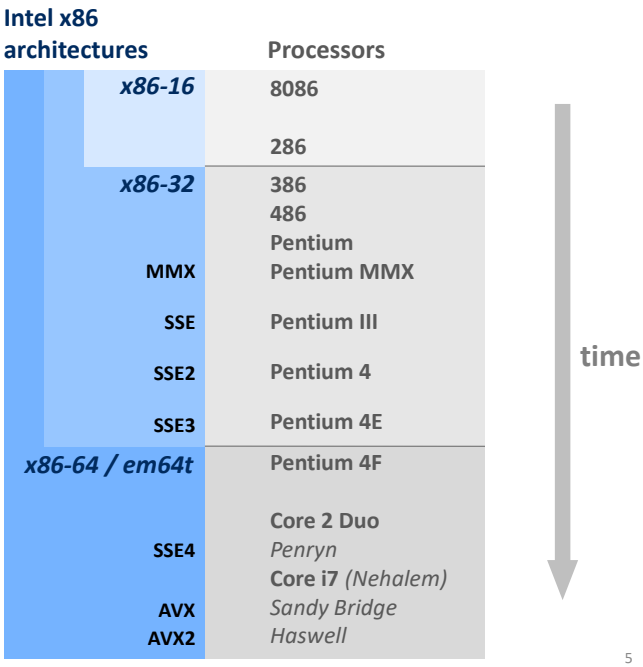
MMX:
Multimedia extension

SSE:
Streaming SIMD extension

AVX:
Advanced vector extensions

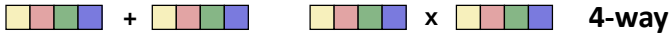
Backward compatible:
Old binary code (≥ 8086) runs on new processors.

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



  - 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

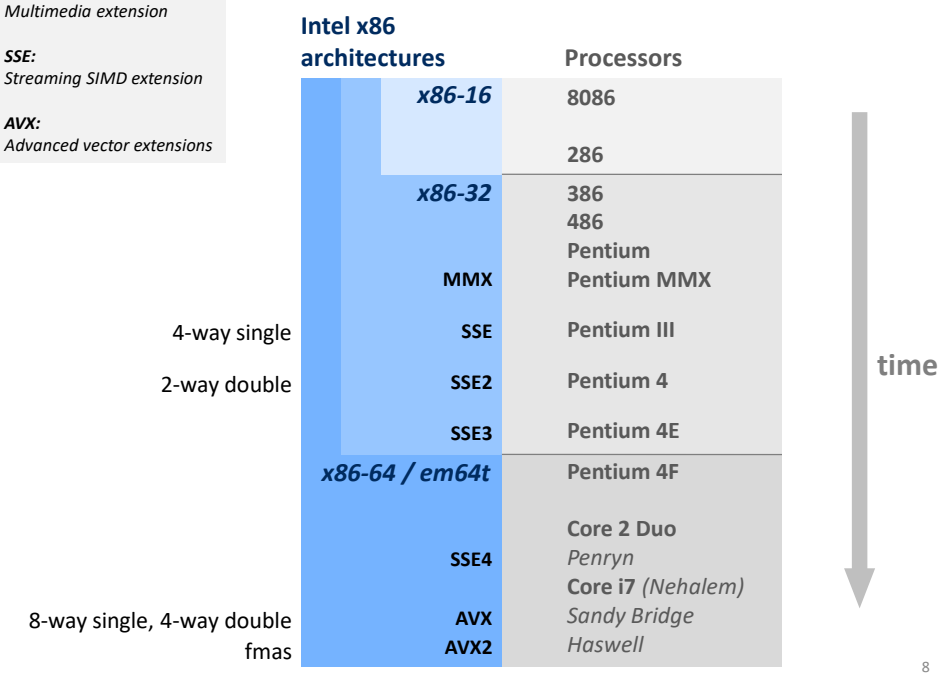
- x = x + y*z
- Done as one operation, i.e., involves only one rounding step
- Better accuracy than sequence of mult and add
- Natural pattern in many algorithms

```
/* 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];
```


MMX:
Multimedia extension

SSE:
Streaming SIMD extension

AVX:
Advanced vector extensions

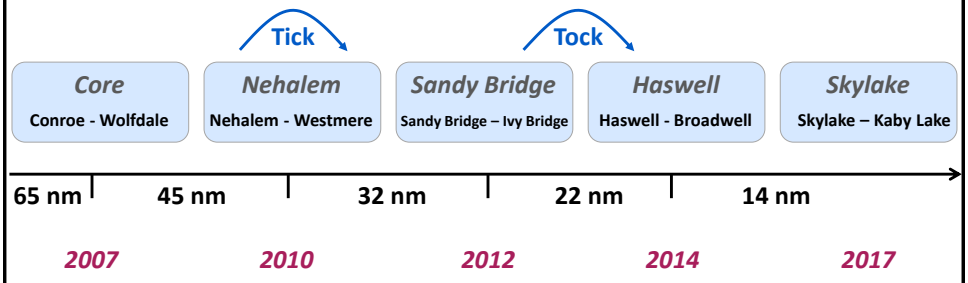


Definitions

- *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

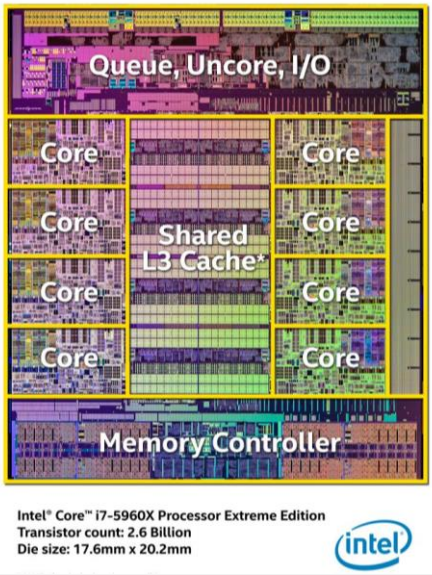
- **Tick: Shrink of process technology**
- **Tock: New microarchitecture**
- **Example: Core and successors**
  Shown: Intel's microarchitecture code names (server/mobile may be different)



In 2016 the Tick-tock model got discontinued
Now: processor – architecture – optimization (since Tick becomes harder)

Pictures: Intel

Intel Processors: Example Haswell



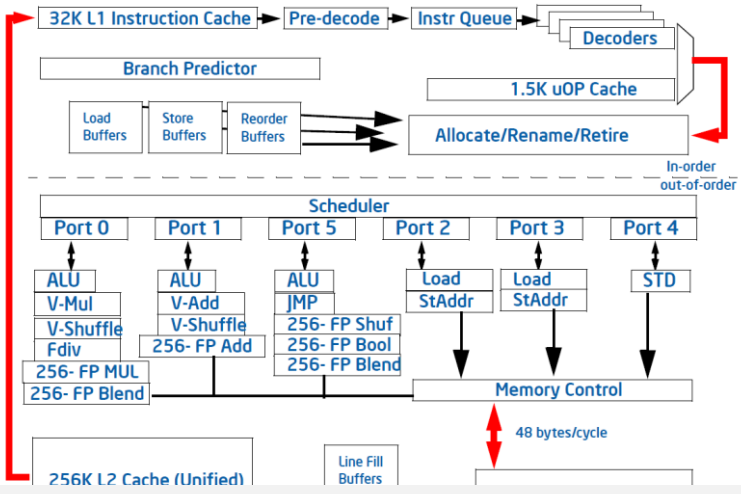
http://www.anandtech.com

Intel® Core™ i7-5960X Processor Extreme Edition
Transistor count: 2.6 Billion
Die size: 17.6mm x 20.2mm

intel

Detailed information about Intel processors

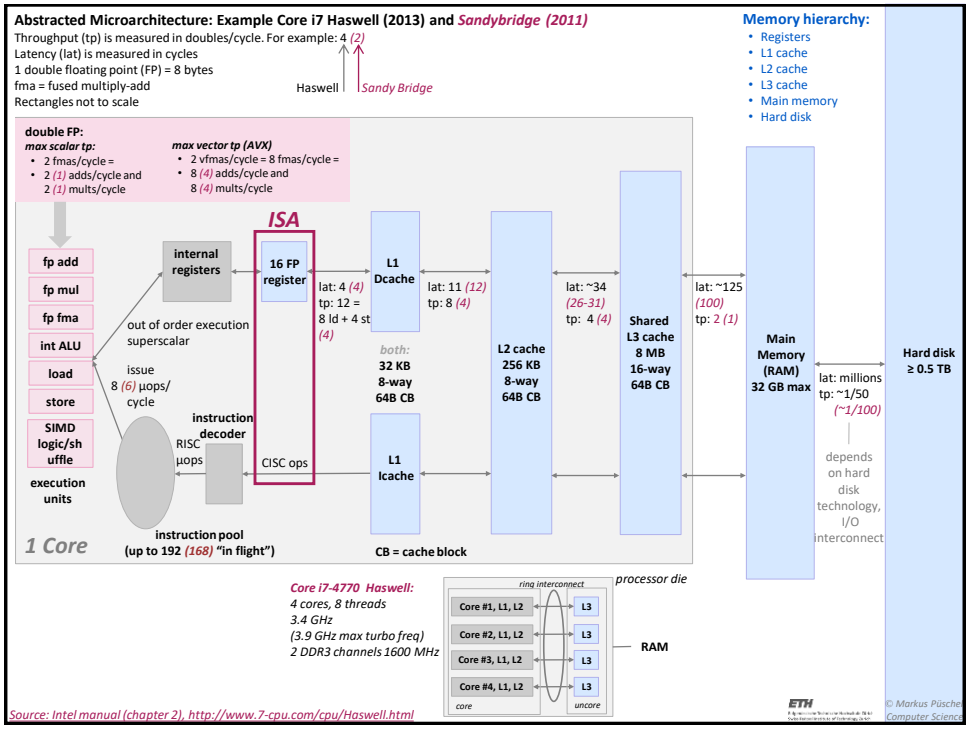
Microarchitecture: The View of the Computer Architect



we take the software developer's view …

Source: Intel Architectures Optimization Reference Manual

- **Distribute microarchitecture abstraction**



Runtime Bounds (Cycles) on Haswell

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

		maximal achievable percentage of (vector) peak performance
■ **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

```
/* 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**

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

W(n): #flops (input size n)

Q(n): #bytes transferred cache ↔ memory (for input size n)

Operational Intensity (Cold Cache)

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

- **Operational intensity:**
  - Flops: W(n) **= 2n**
  - Memory transfers (doubles): **≥ 2n (just from the reads)**
  - Reads (bytes): Q(n) **≥ 16n**
  - Operational intensity: I(n) **= W(n)/Q(n) ≤ 1/8**

Operational Intensity (Cold Cache)

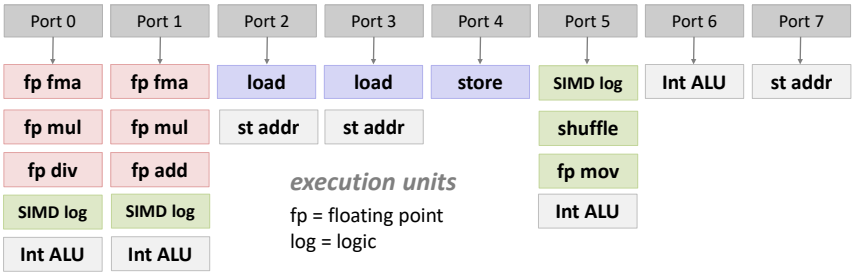
```
/* matrix multiplication; A, B, C are n x n matrices of doubles */
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      C[i*n+j] += A[i*n+k]*B[k*n+j];
```

- **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**

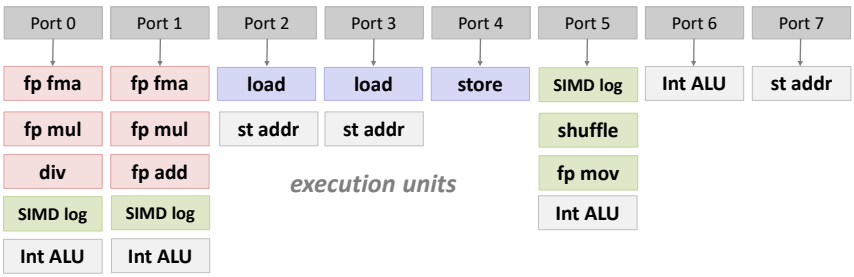
Mapping of execution units to ports



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

- 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),



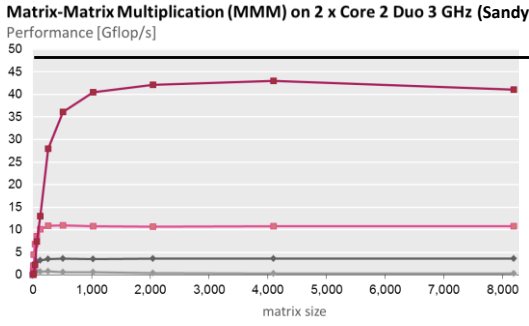
How Many Cycles are at least required?

- A function with n adds and n mults in the C code — ***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 — ***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 — ***n/2***

Comments on Intel Haswell uarch

- **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



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