Durham University
Department of Computer Science

Core I—Introduction to High Performance Computing (HPC)

Session II: Roofline, caches, and performance measurements

Dr. Weinzierl

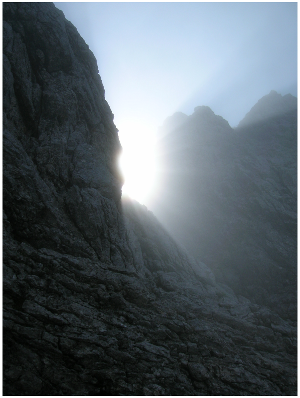
Michaelmas term 2019

Announcements

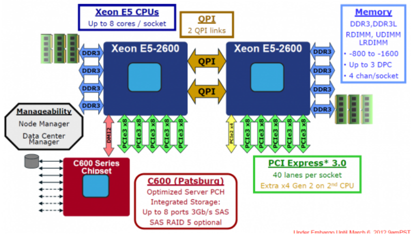
Next week:

- Standard schedule, i.e. Monday/Tuesday
- We will program in C (required)
- You will need access to HPC software (either Intel Parallel Studio or Hamilton)

Outline



Teaser: Which SIMD speedup can we expect?



Ivy Bridge processor:

- Memory can deliver 50GB/s
- CPU cores can handle 400 GB/s through vectorisation
- Algorithm works with double precision, i.e. 8 bytes per value

Trivial observations

1. If a code is not well-suited for vectorisation, we can't expect the code to benefit from full SSE/AVX.

⇒ Vectorisation pays off iff there is enough work to do so we can use SSE/AVX

2. If the memory can't serve the CPUs, we can't expect the code to benefit from full SSE/AVX.

⇒ Vectorisation pays off iff the memory subsystem can deliver enough bytes per second

3. If the code fits brilliantly to vectorisation we still can't expect it to scale better than SSE/AVX.

Arithmetic intensity

Durham University
Department of Computer Science

```
for (int i=0; i<N; i++) {
  load a[i] into reg[0]
  load b[i] into reg[1]
  add reg[0],reg[1]
  store reg[0] into c[i]
}
```

- Per loop iteration we have two loads ($2 \cdot 8$ bytes)
- Per loop iteration we have one store (8 bytes)
- Per loop iteration we have one addition

$\Rightarrow$ On average, we have $\frac{1}{3 \cdot 8}$ operations per byte

Arithmetic intensity: Number of operations per byte/unknown per memory access.

When a data access is not a data access

- Data is already in a register
- Data is already in a cache (to be discussed later)

Arithmetic intensity: Number of operations per byte/unknown squeezing through the memory subsystem.

The original paper by Williams (2009) at all is barely useful, as it gives an abstract AI definition.

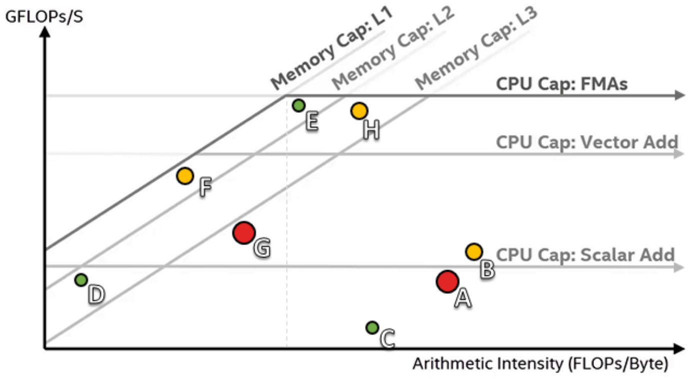
Roadmap

- Identify what our machine would be able to do
- Compute/measure arithmetic intensity
- Action:
  - find out what we could expect from vectorisation, or
  - start to think how to increase arithmetic intensity

The roofline model—a gedankenexperiment (5min)

- Create diagram:
  - x-axis: Arithmetic intensity in flops per byte ($\frac{1}{8} - 4$)
  - y-axis: flops per cycle achived (speed)
- Insert line: Computer works only with floats, i.e. 4 bytes per unknown
  - No vectorisation: 1 Flop per cycle
  - With SSE: 2 Flops per cycle
  - With SSE and brilliant memory layout: 2.5 Flops per cycle
- Insert line:
  - Memory system can deliver one byte per cycle.
- What is theoretical peak?

The roofline model—a gedankenexperiment



- X-axis: flops per byte (AI)
- Y-axis: achieved performance
- Horizontal lines: peak performance for particular instruction sets

Understanding roofline plots

Durham University
Department of Computer Science

With a roofline model and a code analysis we can

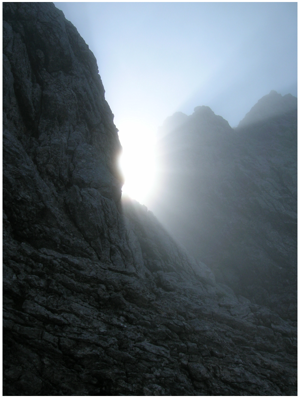
- predict impact of vectorisation
- clarify whether vectorisation is worth the effort

Concept of building block

- Content
  - Definitions/ideas underlying the roofline model
  - Set up roofline model manually
  - Study Intel Advisor
- Expected Learning Outcomes
  - The student knows fundamental definitions of the roofline model
  - The student can explain axes and graphs in the roofline model
  - The student can sketch the roofline model for a given architecture
  - The student can explain, for a given software characterisation, which optimisation strategies are promising

Outline

Durham University
Department of Computer Science



Roofline model
Performance measurements
Caches

Measuring performance is hard

```
#define n 2000
int main() {
  double x[n], y[n];
  double alpha = 2, start, end;
  int i, j;
  for (i = 0; i < n; i++) y[i] = i;
  start = get_time();
  for (j = 0; i < n*n; j++)
     for (i = 0; i < n; i++)
        x[i] = alpha*y[i];
  end = get_time();
  printf("Loop took %g seconds\n", end - start);
  return 0;
}
```

What could go wrong?

I: Clock accuracy

- Timers do not have infinite accuracy
  - All clocks have some *granularity*
  - Error, even if everything else went right, is likely of the same size, the *clock tick*.

$\Rightarrow$ always know what the clock granularity is.

$\Rightarrow$ Ensure measurement is for “long enough” (e.g. $100\times$ clock tick)

II: Cold start

Durham University
Department of Computer Science

- What happens when we execute a binary?
  - Code is loaded from disk into memory
  - Data are in RAM (or on disk?), not cache.
  - Dynamically linked functions may be loaded lazily

$\Rightarrow$ May need multiple tests to mitigate cold start effects

$\Rightarrow$ Need to work hard to ensure data are in the “right” place in memory hierarchy. Use microbenchmarks or real applications?

III: Compiler effects

```
#define n 2000
int main() {
  double x[n], y[n];
  double alpha = 2, start,
        end;
  int i, j;
  for (i = 0; i < n; i++)
    y[i] = i;
  start = get_time();
  for (j = 0; j < n*n; j++)
    for (i = 0; i < n; i++)
      x[i] = alpha*y[i];
  end = get_time();
  printf(
    "Loop took %g seconds\n",
    end - start);
  return 0;
}
```

```
$ cc -O0 time-loop.c -o time-loop
$ ./time-loop
Loop took 19.9509 seconds

$ cc -O1 time-loop.c -o time-loop
$ ./time-loop
Loop took 3e-06 seconds
```

- A sufficiently smart compiler may eliminate unused code
  - Performance will look far too high
  - Or (worse), only some of the code will be eliminated, so it looks plausible.

$\Rightarrow$ Ensure that compiler cannot prove results are unused.

IV: performance jitter

Durham University
Department of Computer Science

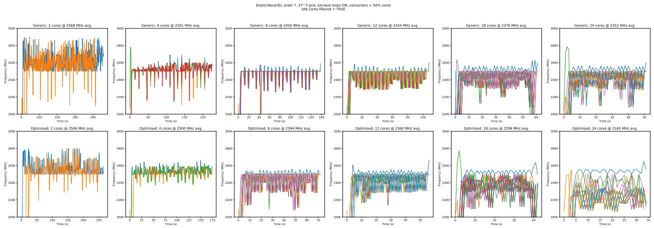
- Your program is not the only thing running
  - Operating system, dæmons, other programs
  - Interference can be very indirect: e.g. modern chips have lower clock rate when more cores are being used

$\Rightarrow$ benchmark on hardware with exclusive access (Hamilton compute node).

$\Rightarrow$ Run multiple tests and report statistics

Guidance: Hoefler and Belli, *Scientific benchmarking of parallel computing systems*, SC15 (2015)

V: frequency



- Effective frequency depends on power consumption
- Effective frequency depends on AVX usage
- Effective frequency depends on core concurrency

VI: reporting

- What is wrong with reporting a compute time as $4.274158 \cdot 10^{-6}$?
  1. No units. Seconds? Years?
  2. Did your measurement *really* have 7 digits of accuracy?
     - 2.1 If units are seconds, in absolute terms you've measured to an accuracy of 1 picosecond
     - 2.2 In relative terms, one part in 10^6

$\Rightarrow$ Store all digits, but *think* before presenting them, does it make sense?

VII: It’s just hard!

- Accurate and reproducible performance measurement is *hard*

⇒ Think before measuring

  - What, precisely, do you want to measure?
  - Is your test code representative of the real code?

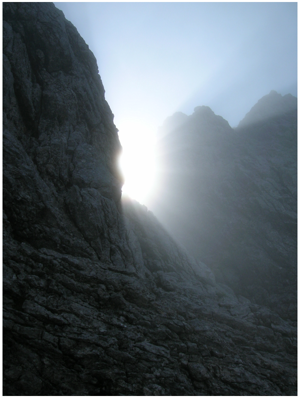
⇒ Check after measuring

  - Do the results seem reasonable?
  - It help tremendously to have a model of how the code should behave (roofline, others later)

Concept of building block

- Content
  - Identify measurement challenges
  - Introduce hardware counters
  - Sketch some tools
- Expected Learning Outcomes
  - The student can explain challenges tied to measurements
  - The student can use at least one analysis tool

Outline



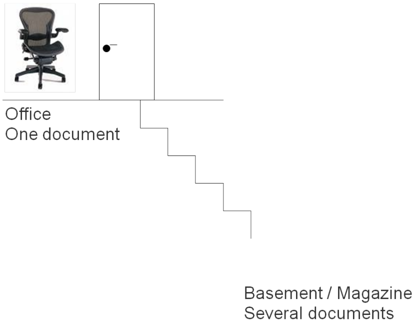
Roofline model
Performance measurements
Caches

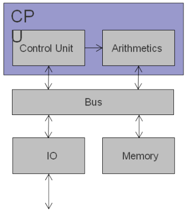
von-Neumann

John von Neumann

- John von Neumann
  - 1903–1957
  - Manhattan Project (Los Alamos)
  - June 30, 1945 (but Turing et. al. published similar ideas)
- Computer Consists of Four Components
- There is a *Von-Neumann bottleneck*

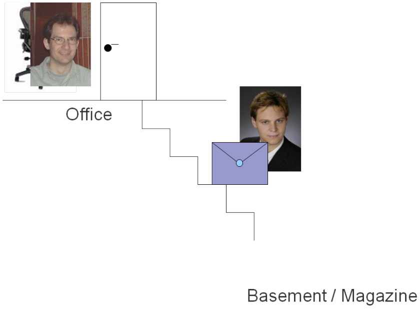
The von Neumann bottleneck

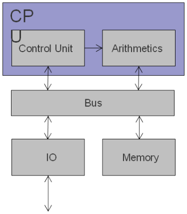




- One or two documents in the office (two registers in the ALU) ain't sufficient.
- Introduce more registers (Itanium e.g. has 128 of them).
- However, number of registers still is limited (though GPGPUs kind of stress increase of registers).

The von Neumann bottleneck

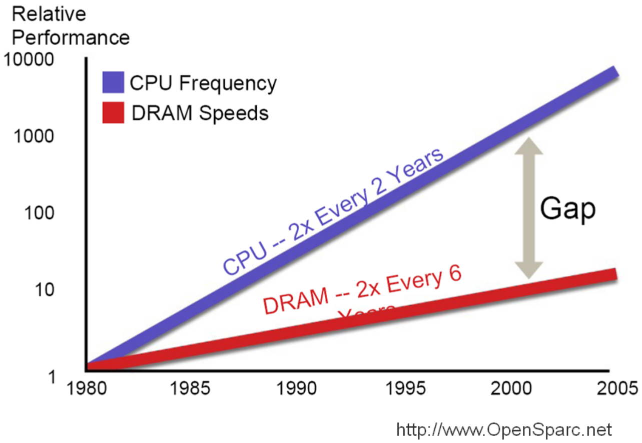




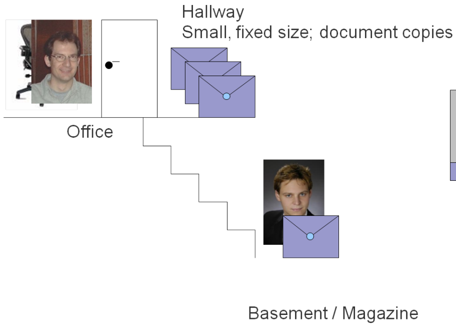
- Running into the basement is time consuming, and
- The bigger the basement (memory), the slower the search becomes.
- The faster the processor, the more annoying the slow search in the memory is.
- Can we study this effect?

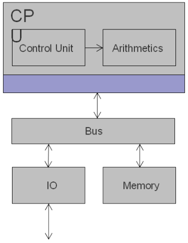
The Memory Gap

Durham University
Department of Computer Science

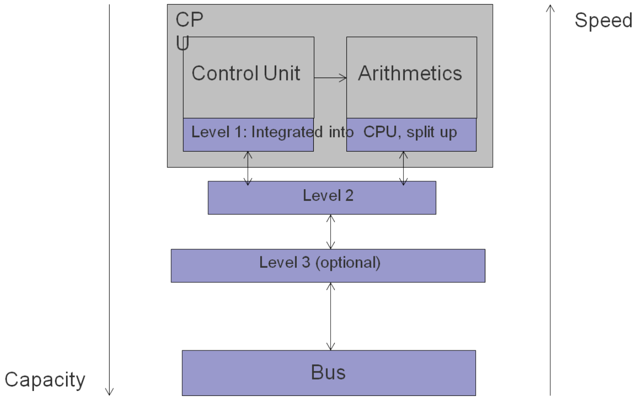


Idea of a Cache





Cache Levels



Cache hits and misses

Cache miss: Required data is not contained within the cache.
Reasons:

- Compulsory miss: first access
- Capacity miss: data has been removed due to another data request (finite size of cache)
- Conflict miss: data has been removed for another reason (cf. MeSI)

- Computers have a hierarchy of caches and lots of registers.
- The time to finish one operation depends significantly on where the data is located right now (see memory stalls (bubbles) when we ran Amplifier, e.g).
- It is thus important for many algorithms to exhibit *spatial locality* and *temporal locality*.
- However, we also have to avoid *cache conflicts*.

Cache lines

Durham University
Department of Computer Science

Cache line: Modern CPUs manage their caches by means of cache lines, i.e. whole blocks of memory.

- Main memory is clustered along cache line size. Only whole cluster/blocks of main memory can be transferred.
- SIMD loads and stores work if and only if they refer to one cache line.

Cache associativity:

- *Direct Mapped*. Each block in the main memory can go into only one particular cache line.
- *k-Way Set Associative*. Each cache set (fragment) can host *k* blocks from the main memory. Usually use least recently used strategy for replacement.
- *Fully Associative*. Each memory block can go into any cache line.

4-Way Set Associative seems to be the predominant strategy at the moment.

Cache coherence

Caches are *copies* of main memory data. If multiple cores have different caches, these copies have to be kept consistent:

> Cache coherence protocol: Mechanism to keep multiple caches consistent.

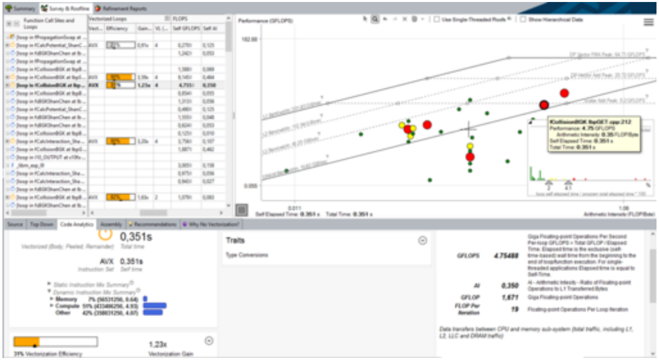
MeSI: Flag each cache entry

- M Modified: we have changed the value, but the value is not yet written back to main memory (cmp. write-through policy on BlueGene/P, e.g.)
- E Exclusive: data is held exclusively here, i.e. noone else holds a copy
- S Shared
- I Invalid

GPGPUs:

- $\Rightarrow$ CUDA offers cache only to read-only data
- $\Rightarrow$ CUDA architectures do not need such a protocol
- $\Rightarrow$ CUDA codes benefit significantly from const variables

Roofline reloaded



- Try to keep data in cache
- Effective BW available to code increases

Cache-aware programming

Cache aware programming: Make code fit to used cache architecture (tuning).
Cache oblivious programming: Make code fit to any cache architecture.

Techniques: (add remark why this improved cache usage)

- Reuse data ⇒
- Prefetch ⇒
- Tile data ⇒
- Do not share data ⇒
- Don't jump around in memory ⇒

Cache-aware programming

Cache aware programming: Make code fit to used cache architecture (tuning).
Cache oblivious programming: Make code fit to any cache architecture.

Techniques: (add remark why this improved cache usage)

- Reuse data ⇒ increase temporal locality, avoid capacity misses
- Prefetch ⇒ eliminate compulsory misses
- Tile data ⇒ increase temporal and spatial locality (capacity and conflict misses)
- Do not share data ⇒ avoid multicore cache ping-pong
- Don’t jump around in memory ⇒ exploit cache lining

```
#pragma noprefetch b

#pragma prefetch a

for(i=0; i<m; i++) {
  a[i]=b[i]+1;
}
```

Concept of building block: Caches

- Content
  - Introduce general concept of a cache
  - Cache levels and GPGPU caches (for read-only data only)
  - Cache miss and cache line as well as cache miss categories
  - MeSI protocol
  - Five ingredients how to write applications with good cache usage characteristics
- Expected Learning Outcomes
  - The student knows definition of cache, cache line, cache miss (*)
  - The student can explain how a cache works (*)
  - The student can explain how given cache optimisations (such as loop tiling) work (*)
  - The student can explain what the difference between SoA and AoS is (*)
  - The student can derive for a given code optimisation how it might affect the cache behaviour (*)
  - The student can interpret cache information as given by KCachegrind (**)

(*) Classic exam topics.

(**) Not assessed via exam.