CPUs and Memory

Computational Science II (CAAM 520)

Christopher Thiele

Rice University, Spring 2020

Overview

- Why do we need to know about CPU architecture and memory?
- What do typical CPU architectures look like?
- What does this mean for scientific computations?

→ Hager & Wellein, chapter 1

Computational complexity

Computational complexity or computational cost is often assessed by counting floating point operations (FLOP), e.g.,

- Vector dot product: $\mathcal{O}(n)$
- Matrix-vector product: $\mathcal{O}(n^2)$

While such classifications can provide useful insight, they do not always translate directly to modern computer hardware.

CPU architecture

A CPU has several units for integer and floating point arithmetic.

These units operate on (integer or floating point) registers.

```
mov eax, 1
mov ebx, 2
; Add eax and ebx, store result in eax.
add eax, ebx
```

CPU architecture

The CPU can load data from memory into registers and write from registers to memory.

```
; Load value from address given by ebx, increment,
; and write back.
mov eax, [ebx]
add eax, 1
mov [ebx], eax
```

CPU architecture

We can express the ***peak performance*** of a CPU in floating point operations per second (FLOPS).

For a long time, peak performance could be increased using higher and higher clock frequencies.

→ Today, physical limitations make further increases in frequency impossible or undesirable.

CPU architecture

What can be done to futher improve performance?

- Pipelining
- Superscalar architecture
- SIMD
- Out-of-order execution
- RISC instead of CISC
- Caches
- Multi-core architectures
- Hyperthreading

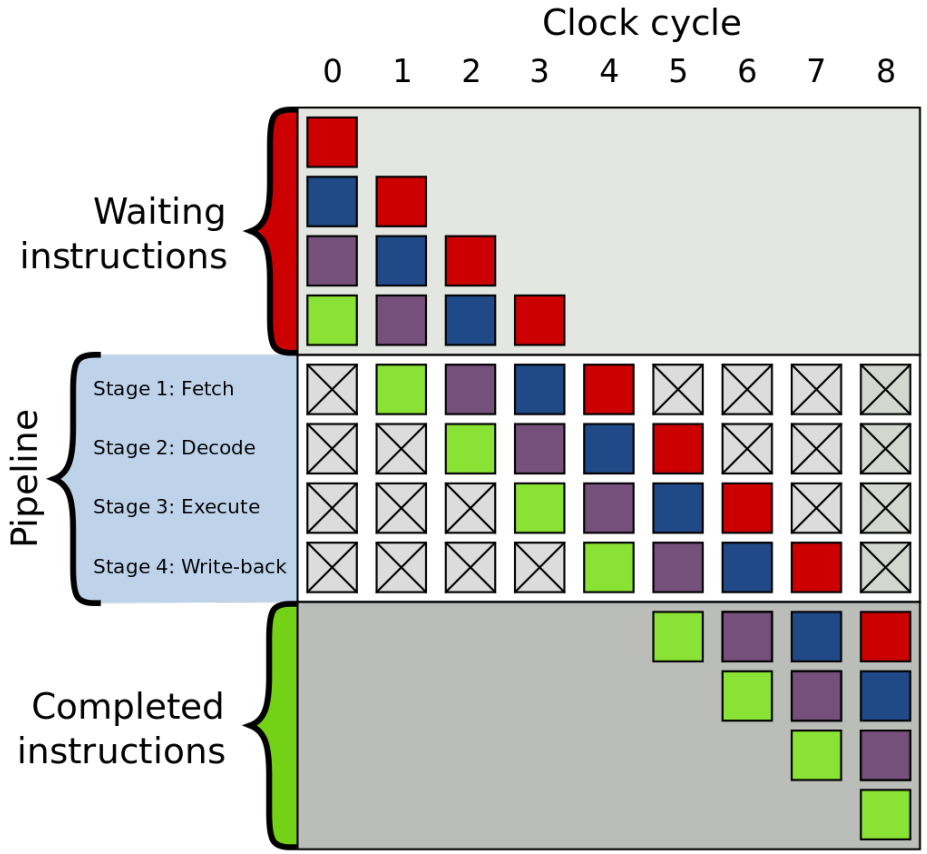
Pipelining

Instead of executing instructions one by one, break them down into simple tasks.

Simple tasks can be performed by separate, simple units on the CPU.

These units can be kept busy using pipelines.

Pipelining



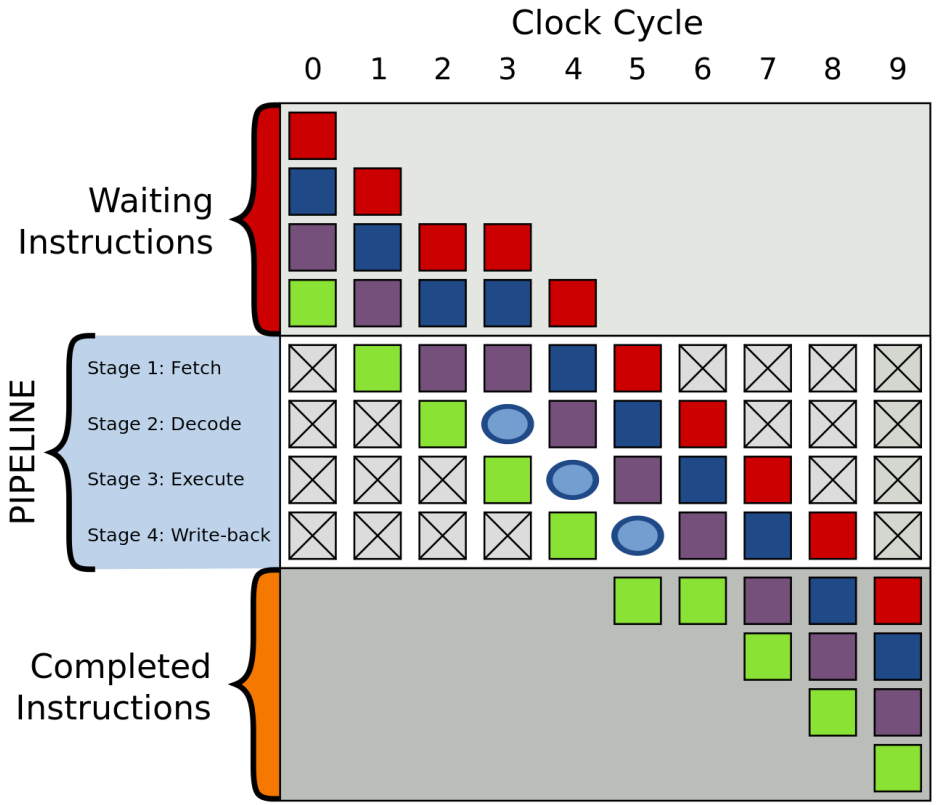
https://en.wikipedia.org/wiki/Instruction_pipelining#/media/File:Pipeline,_4_stage.svg

Pipelining

While pipelines keep CPU units busy, they also add

- latency, and
- the possibility of pipeline bubbles.

Pipelining



https://en.wikipedia.org/wiki/Instruction_pipelining#/media/File:Pipeline,_4_stage_with_bubble.svg

Superscalar architecture

A CPU can have multiple units of any given type (load, store, integer and floating point arithmetic, etc.). Hence, it can perform multiple operations concurrently.

$\rightarrow$ Compare this to pipelining.

SIMD

A CPU can provide instructions that operate on multiple pieces of data at the same time.

$\rightarrow$ Single instruction, multiple data (SIMD)

This is not the same as superscalar architecture!

SIMD instructions are part of SSE, AVX, AVX2, AVX512, etc.

Out-of-order execution

A CPU may reorder instructions if possible.

```
; Load value from address given by ebx
; and add it to eax.
add eax, [ebx]
; Increment ecx.
add ecx, 1
```

→ The second instruction could be executed while the input data for the first instruction is loaded.

RISC instead of CISC

In the past, there were two types of computers/CPUs:

- Complex instruction set computers (CISC)
- Reduced instruction set computers (RISC)

While many contemporary CPUs expose a CISC interface, they are RISC CPUs internally.

Caches

Modern CPUs can perform computations much faster than data can be loaded from main memory.

→ Gap between FLOPS and ***memory bandwidth***

To limit the impact of low memory bandwidth, modern CPUs have ***caches***, which are

- small (KBs to a few MBs) and
- fast (low latency, high bandwidth.)

Caches

In fact, modern CPUs typically have a hierarchy of caches:

- L1 cache: Very small, very fast
- L2 cache: Larger, but slower
- L3 cache: Even larger (a few MB), even slower
- Main memory: Huge, but very slow in comparison

Caches

When the CPU needs data from memory, one or more ***cache lines*** are loaded into the cache.

→ The CPU ***cannot*** load individual bytes from main memory.

Caches are used most efficiently by algorithms with

- ***temporal locality***, i.e., algorithms that reuse date before discarding it, or
- ***spatial locality***, i.e., algorithms that operate on contiguous blocks of data.

Caches

If the CPU requests data which is already present in a cache, we call it a ***cache hit***.

Conversely, if the CPU requests data which must be loaded from main memory, we call it a ***cache miss***.

Note: CPUs also have an ***instruction cache***, i.e., there can be instruction cache misses, too.

Multi-core architectures

The power dissipation of a CPU is proportional to the third power of the clock frequency (Hager & Wellein).

→ Use multiple CPU ***cores*** with lower clock frequency in a single CPU.

Multi-core CPUs are ubiquitous nowadays, and they allow for a variety of CPU designs.

Hyperthreading

Recall that pipeline bubbles can cause CPU units, e.g., arithmetic units, to idle.

Hyperthreading attempts to avoid pipeline bubbles by duplicating the CPUs ***architectural state*** (registers, control, etc.). The CPU can then execute two applications concurrently and feed instructs from both applications into the same pipeline.

→ It depends on the (scientific) application whether hyperthreading is useful.

Conclusions

What does all of this mean in practice?

- Computational performance is a complex issue.
- Few applications can achieve peak performance.
- FLOP counting does not tell the entire story.

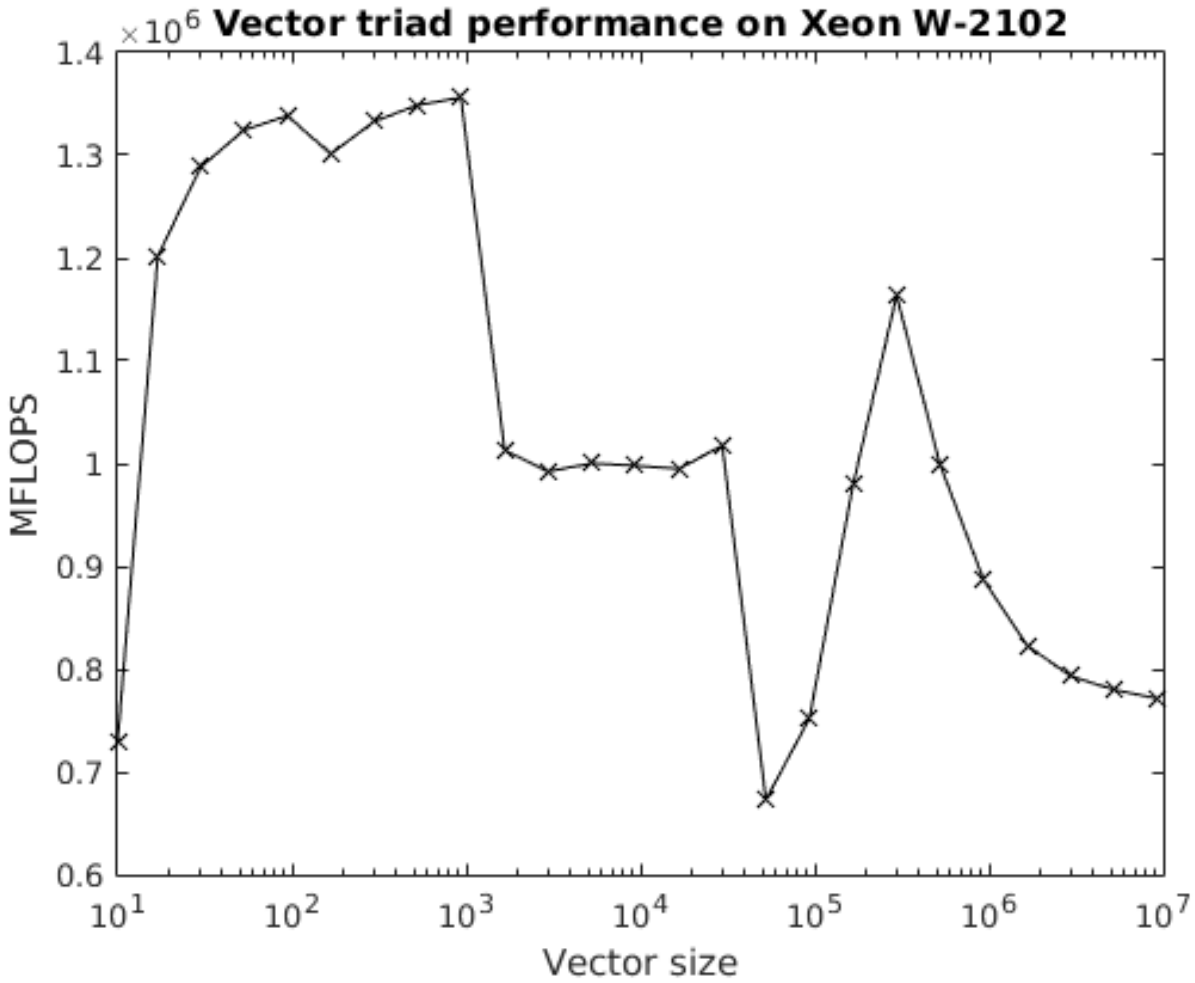
An example

Consider the ***vector triad***

```
for (int r = 0; r < repetitions; r++) {
  for (int i = 0; i < n; i++) {
    a[i] = b[i] + c[i]*d[i];
  }
}
```

Clearly, its computational complexity is $O(n)$.

An example


×10^6 Vector triad performance on Xeon W-2102
MFLOPS
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
10^1
10^2
10^3
10^4
10^5
10^6
10^7
Vector size