Introduction to modern hardware

Lukas Einkemmer
University of Innsbruck

Link to slides: `http://www.einkemmer.net/training.html`

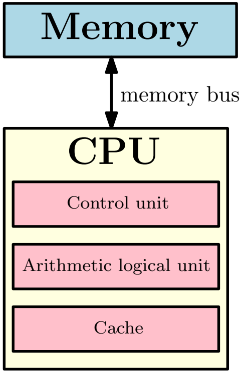
Introduction

Understanding hardware is important to understand performance.

Components of a computer

CPU (central processing unit) performs arithmetic operations, conditionals, loops.

Memory stores data used for processing (main memory, caches, disk).



CPU and main memory is connected via the memory bus.

CPU

The CPU executes a sequence of instructions (referred to as machine code).

Example of a vastly simplified assembly/machine code

```
mov 0x38AF2 r1
mov 0xA03DD r2
add r1 r2
mov r2 0x38AF2
```

Instructions can be grouped as follows

- memory instructions (write or read from main memory)
- arithmetic operations on registers
- control instructions (comparisons, jumps)

CPU performance

Performance is usually measured in floating point operations per second (FLOPS)

- amount of arithmetic operations that can (theoretically) be performed per second

A 3 GHz CPU that can perform one floating point operation per cycle equals 3 GFLOPS.

To attain this level of performance might not be possible in practice

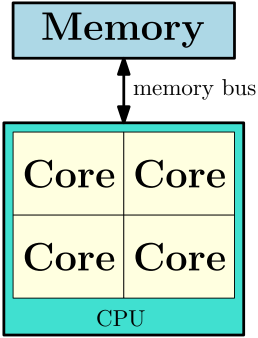
- algorithm dependent
- implementation dependent

Modern CPUs

Increasing clock frequency has not been viable for a while

- power dissipation scales as the frequency squared
- but transistors still get smaller

Modern CPUs look more like this.

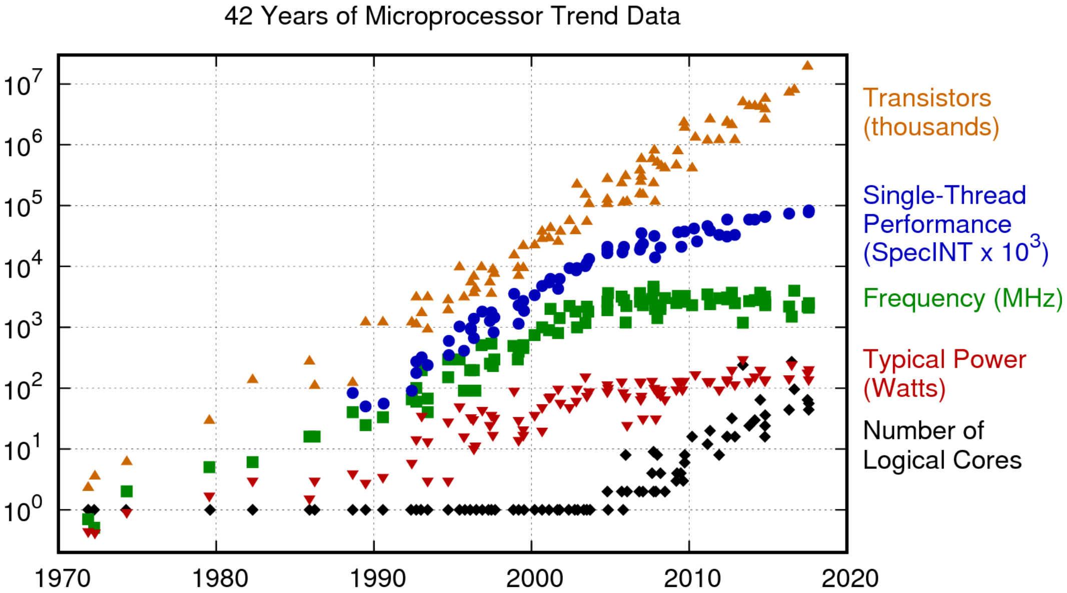


Multiple CPUs on the same chip

- usually the sequential execution units are referred to as cores
- CPUs with 16 cores are quite common now

Beware: hyperthreads are not cores.

Modern CPUs


42 Years of Microprocessor Trend Data
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1970
1980
1990
2000
2010
2020
Transistors (thousands)
Single-Thread Performance (SpecINT x 10^3)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores

Vectorization

Each core can perform vector operations in a single clock cycle

- 256 bit registers (4 double, 8 floats)
- fused multiply add

Vectorization is only possible if the CPU supports that specific instruction

- ideally handled by the compiler
- OpenMP includes support for vectorization

Types of parallelism

- Vectorization is single instruction multiple data (SIMD)
- Core level parallelism is multiple instruction multiple data (MIMD)

Performance of a modern CPU

The theoretically achieved FLOPS are calculated as follows

$$\begin{aligned}&(3\text{ GHz})\cdot(16\text{ cores})\cdot(4\text{ SIMD})\cdot(2\text{ fused multiply add})\cdot(2\text{ ALUs})\\&=768\text{ GFLOPS}\end{aligned}$$

Let us consider multiplying a vector by a scalar

```
for(int i=0;i<n;i++)
    y[i] = 3*x[i]
```

Requires one memory read and one memory write per floating point operation.

- Achieving 768 GFLOPS would require a memory transfer rate (bandwidth) of 12 TB/s
- State of the art hardware achieves 50-150 GB/s

Compute bound vs memory bound

A problem is **compute bound** if the performance is dictated by how many arithmetic operation the CPU can perform.

- numerical quadrature, solving dense linear system (LU), Monte Carlo methods, ...

A problem is **memory bound** if the performance is dictated by the bandwidth of main memory.

- stencil codes, solving sparse linear systems, FFT, ...
- performance measured in achieved GB/s

How many memory instructions?

```
for(int i=0;i<n-1;i++)
    y[i] = x[i+1] - x[i];
```

Memory hierarchy

FLOPS have increased dramatically but memory speed has lagged behind.

- there is a cost, capacity, speed trade-off involved

The result is a **memory hierarchy**

- Caches on the CPU (fast, tens of megabytes)
- Main memory (medium speed, tens of gigabytes)
- Disk storage (slow, terabytes)

Caches are usually further divided

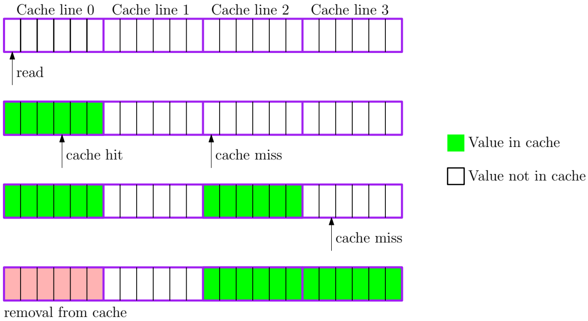
- Modern CPUs usually have L1, L2, L3 cache
- Caches are completely transparent to the programmer

Caches

Knowledge of how caches work is important for performance.

Caches transfer data in chunks of fixed size (so-called cache lines)

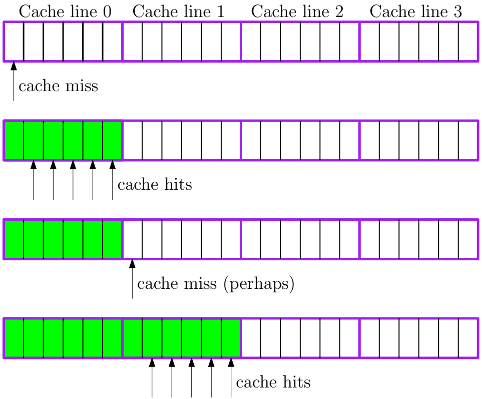
- usually 64-256 bytes in size (8-32 doubles)



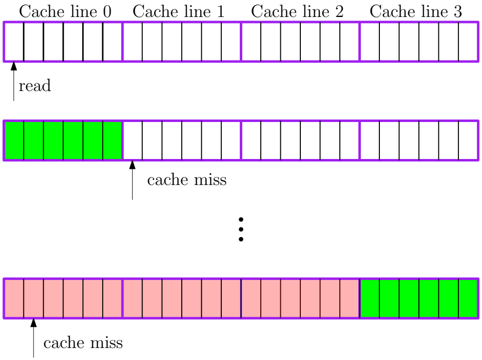
First read of any byte in a cache line transfers the entire cache line.

Memory access pattern

```
// Access with stride 1
for(int i=0;i<n;i++)
    out[i] = in[i];
```



```
// Access with stride 6
for(int j=0;j<6;j++)
    for(int i=0;i<n/6;i++)
        out[j+6*i] = in[j+6*i];
```



Latency

Latency – how long the CPU has to wait between issuing a memory request and receiving the first byte – is an important performance consideration.

- There are physical limitations when reducing latency

Modern CPUs employ a range of techniques to hide latency.

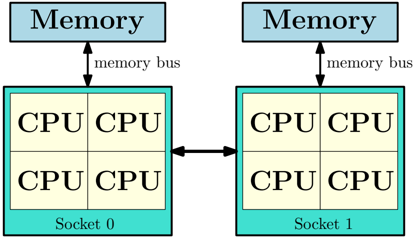
Prefetching tries to load data that is likely used next into the cache (before the actual memory instruction is issued)

- Particularly efficient for problems where memory locations close together are accessed in sequence

Instruction Level Parallelism (ILP) tries to skip instructions that still wait for memory.

NUMA domains

But many systems are dual or quad socket now.



All cores can access the entire memory but speed might differ depending on which memory modules are accessed.

- Non uniform memory access (NUMA)
- Cores are grouped into NUMA domains

Determine NUMA domains: `numactl --hardware`

First touch

For best performance memory has to be placed 'close' to where it is used.

- Neither C++ nor OpenMP provides a way to directly do that
- This might not necessarily be desirable anyhow

First touch principle: A memory location is mapped close to the core that first touches (reads or writes) it.

Different parts of an array can be placed on different NUMA domains.

A note on optimization

Modern CPUs are complicated.

- A basic understanding is vital but often measurement is necessary.

As a rule of thumb we pay the following penalty (in clock cycles)

Operation	cost in cycles
arithmetics	1
L1 hit	1-10
function call	10-20
L3 hit	40
sin/cos	100
memory	200
disk	10^5

Exact numbers depend on the specific architecture.

Further reading

Ulrich Drepper
What every programmer should know about memory.
`https://lwn.net/Articles/250967/?rss=1`.