GPU Computing

E. Carlinet, J. Chazalon {`firstname.lastname@lrde.epita.fr`}

Oct 21

EPITA Research & Development Laboratory (LRDE)

Slides generated on October 8, 2021

Fifty shades of Parallelism

Fifty shades of Parallelism

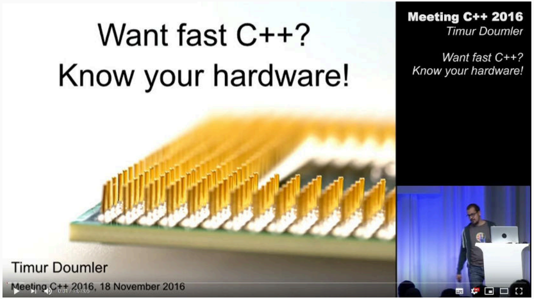
How to get things *done* quicker

1. Do less work
2. Do *some* work better (i.e. the one being the more time-consuming)
3. Do *some* work at the same time
4. Distribute work between different workers

How to get things *done* quicker

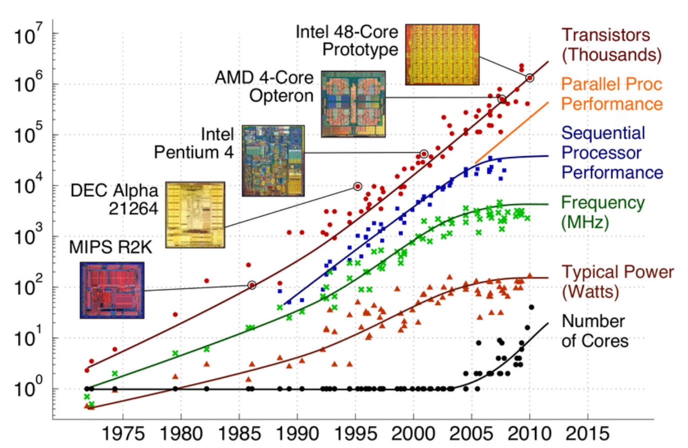
1. Do less work
2. Do *some* work better (i.e. the one being the more time-consuming)
3. Do *some* work at the same time
4. Distribute work between different workers

- (1) Choose the most adapted **algorithms**, and avoid re-computing things
- (2) Choose the most adapted **data structures**
- (3,4) Parallelism



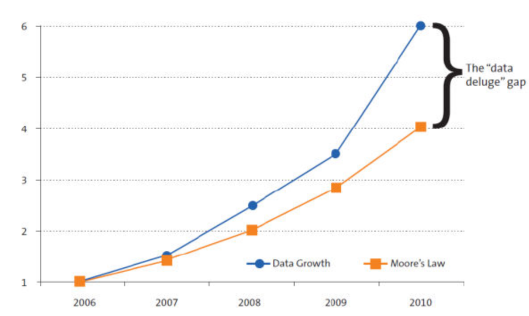
Why parallelism ?

- Moore’s law: processors are **not** getting twice as powerful every 2 years anymore



- So the processor is getting smarter:
  - Out-of-order execution / dynamic register renaming
  - Speculative execution with branch prediction
- And the processor is getting super-scalar:

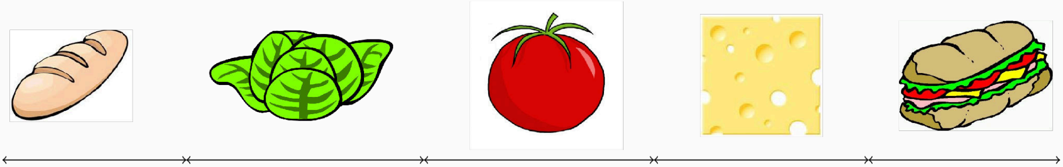
Toward data-oriented programming



- while the CPU clock rate got bounded…
- … the quantity data to process has shot up!

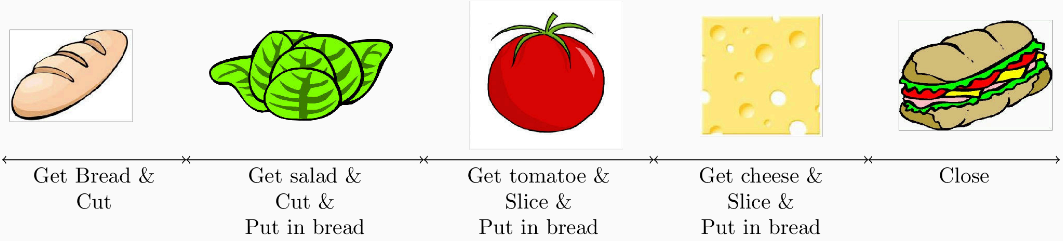
We need another way of thinking "speed"

The burger factory assembly line



How to make several sandwiches as fast as possible ?

The burger factory assembly line



How to make several sandwiches as fast as possible ?

- Optimize for **latency**: time to get 1 sandwich done.
- Optimize for **throughput**: number of sandwiches done during a given *duration*

Data-oriented programming parallelism

Flynn's Taxonomy

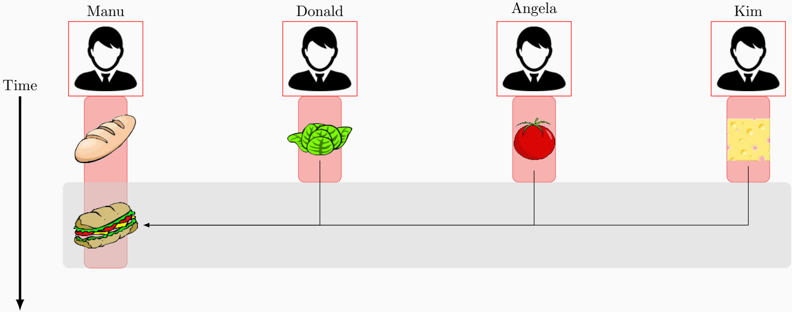
	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

- SISD: no parallelism
- SIMD: same instruction on data group (vector)
- MISD: rare, mostly used for fault tolerant code
- MIMD: usual parallel mode

Optimize for latency (MIMD with collaborative workers)

4 **super-workers** (4 CPU cores) collaborate to make 1 sandwich.

- Manu gets the bread and cuts and waits for the others
- Donald slices the salad
- Angela slices the the tomatoes
- Kim slices the cheeses

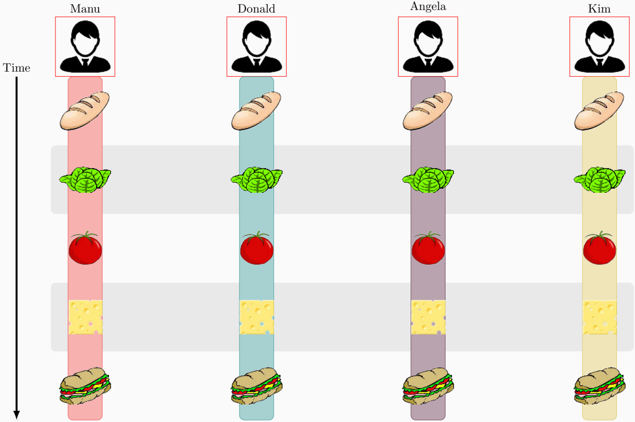


Time to make 1 sandwich: $\frac{s}{4}$ (400% speed-up)

This is optimized for ***latency*** *(CPU are good for that).*

Optimize for throughput (MIMD Horizontal with multiple jobs)

- Manu makes sandwich 1
- Donald makes sandwich 2
- ...

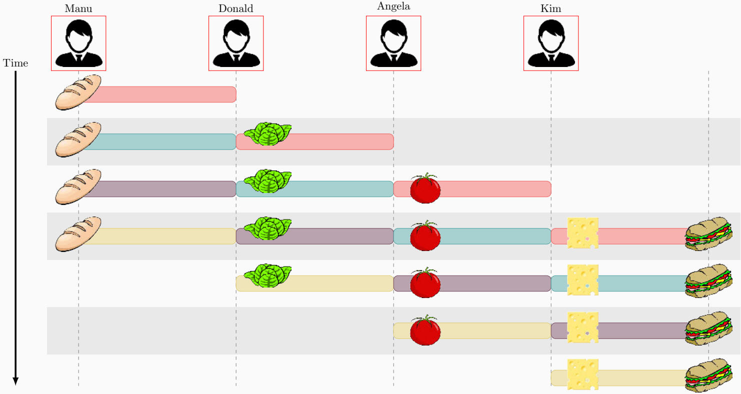


Time to make 4 sandwiches: s (400% speed-up)

This is optimized for ***throughput*** *(GPU are good for that).*

Optimize for throughput (MIMD Vertical Pipelining)

- Manu cuts the bread
- Donald slices the salads
- Angela slices the tomatoes
- ...



Time to make 4 sandwiches: s (400% speed-up)

Optimize for throughput (SIMD DLP)

A worker has many arms and make 4 sandwiches at a time

Time to make 4 sandwiches: s (400% speed-up)

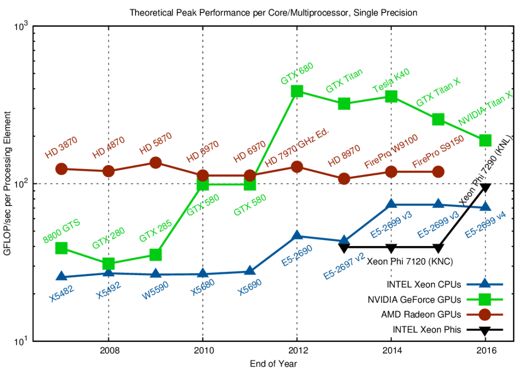
More cores is trendy

Data-oriented design have changed the way we make processors (even CPUs):

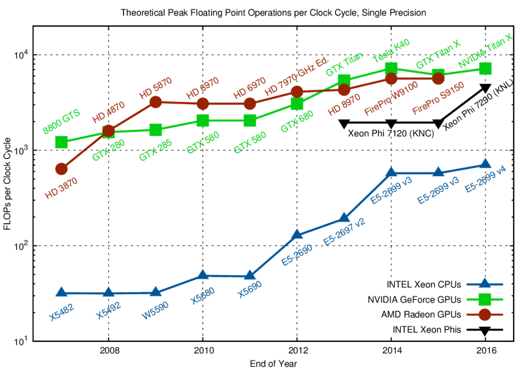
- Lower clock-rate
- Larger vector-size, more vector-oriented ISA
- More cores (processing units)

	64bits Intel Xeon	Xeon 5100 series	Xeon 5500 series	Xeon 5600 series	Xeon E5 2600 series	Xeon Phi 7120P
Freq	3.6 Ghz	3.0 Ghz	3.2 Ghz	3.3 Ghz	**2.7 Ghz**	**1.24 Ghz**
Cores	1	2	4	6	**12**	**61**
Threads	2	2	8	12	**24**	**244**
SIMD Width	128 bits (2 clocks)	128 bits (1 clock)	128 bits (1 clock)	128 bits (1 clock)	**256 bits (1 clock)**	**512 bits (1 clock)**

More cores is trendy



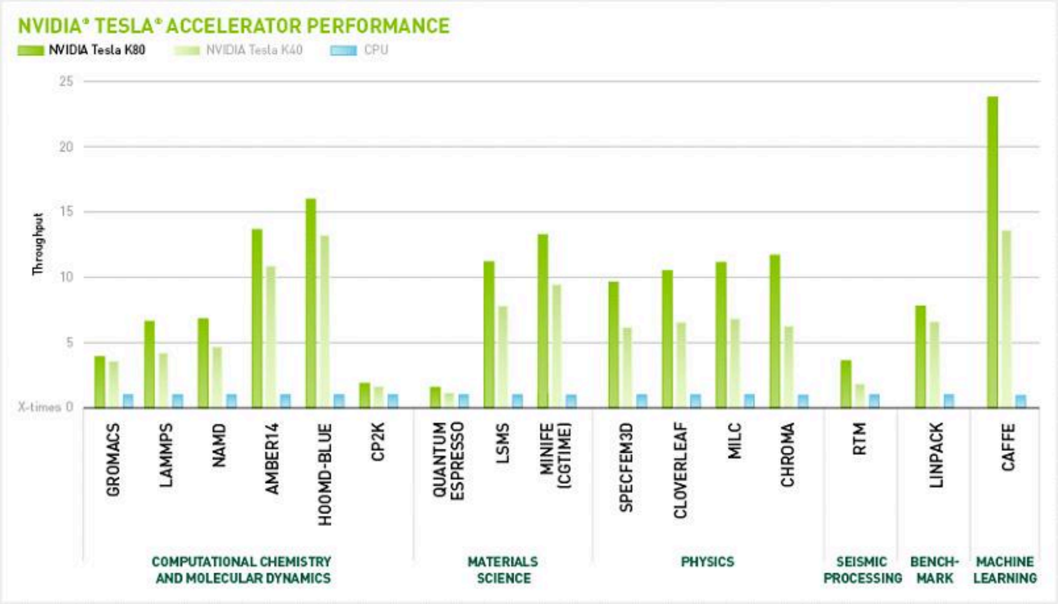
Peak performance / core is getting lower



Global peak performance is getting higher (with more cores!)

CPU vs GPU performance

And you see it with HPC apps:



But don't forget, you may need to optimize both **latency** and **throughput**.

What is the bounds speedup attainable on a parallel machine with a program which is parallelizable at **P** % (i.e. must run sequentially for **(1 - P)**).

But don't forget, you may need to optimize both **latency** and **throughput**.

What is the bounds speedup attainable on a parallel machine with a program which is parallelizable at **P** % (i.e. must run sequentially for **(1 - P)**).

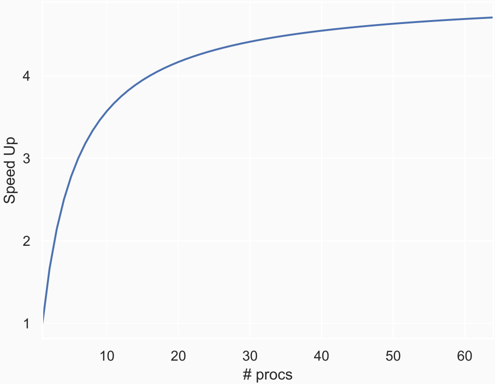
Sequential=20%	Parallelizable=80%

If you have *N* processors, the speed-up is:

$$S = \frac{t\_old}{t\_new} = \frac{1}{(1 - P) + P/N}$$

- Time to run the sequential part
- Time to run the parallel part

P = 80%, max speed-up = 5

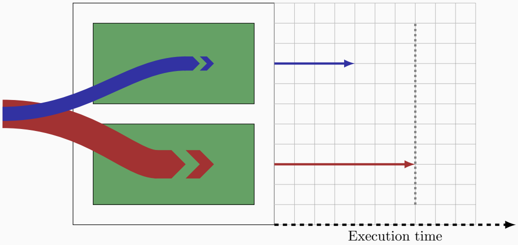


Toward Heterogeneous Architectures (1/2)

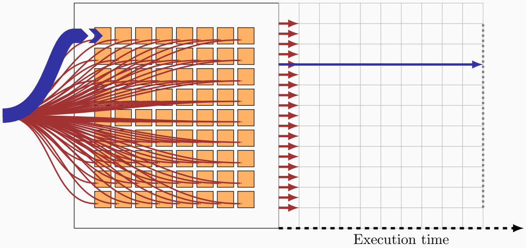
$$S = \frac{t\_old}{t\_new} = \frac{1}{(1 - P) + P/N}$$

- Time to run the sequential part
- Time to run the parallel part

Latency-optimized (multi-core CPU)
👎 Poor perfs on parallel portions



Throughput-optimized (GPU)
👎 Poor perfs on sequential portions



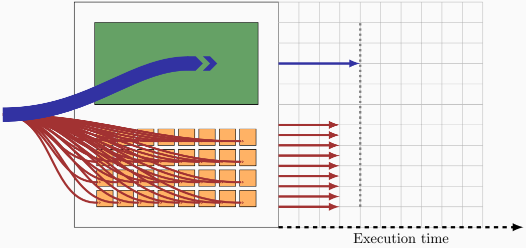
$$S = \frac{t\_old}{t\_new} = \frac{1}{(1 - P) + P/N}$$

- Time to run the sequential part
- Time to run the parallel part

Heterogeneous (CPU+GPU)

👍 Use the right tool for the right job

👍 Allows aggressive optimization for latency or for throughput



Toward Heterogeneous Architectures