Why do we need parallel computing?

Write down as many reasons as you can during 5 minutes!

Introduction

Parallel Computing is a part of Computer Science and Computational Sciences (hardware, software, applications, programming technologies, algorithms, theory and practice) with special emphasis on parallel computing or supercomputing

1 Parallel Computing – motivation

The main questions in parallel computing:

- How is organised interprocessor communication?
- How to organise memory?
- Who is taking care of parallelisation?
  - CPU?
  - compiler?
  - ... or programmer?

1.1 History of computing

Pre-history

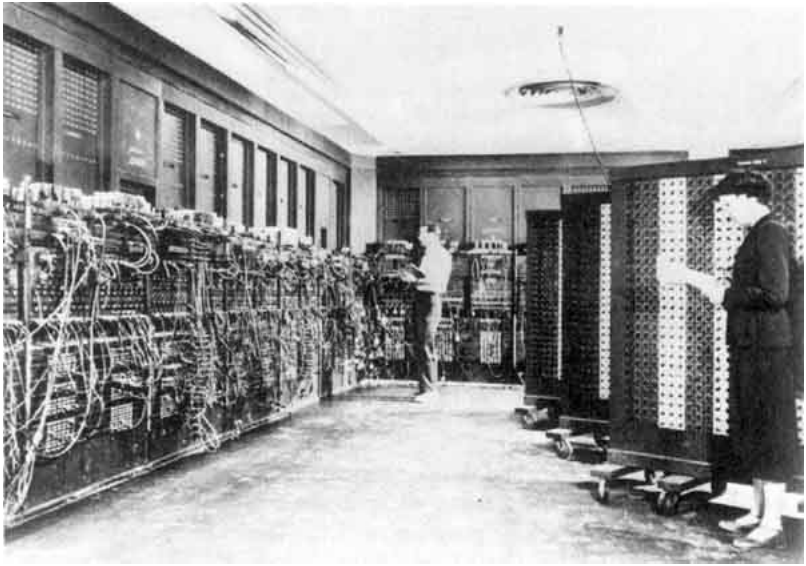
Even before electronic computers parallel computing existed

? . . . – (pyramides.)

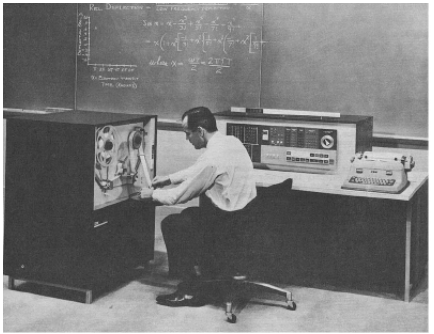
1929 – parallelisation of weather forecasts

≈**1940** – Parallel computations in war industry using Felix-like devices

1950 - Emergence of first electronic computers. Based on lamps. ENIAC and others



1960 - Mainframe era. IBM



1970 - Era of Minis
1980 - Era of PCs
1990 - Era of parallel computers
2000 - Clusters / Grids
2010 - Clouds

1.2 Expert's predictions

Much-cited legend: In 1947 computer engineer Howard Aiken said that USA will need in the future at most 6 computers

1950: Thomas J. Watson as well: 6 computers will be needed in the world

1977: Seymour Cray: The computer Cray-1 will attract only 100 clients in the world

1980: Study by IBM – about 50 Cray-class computers could be sold at most in a year worldwide

Reality: Many Cray-* processing power in many of nowadays homes

Gordon Moore's (founder of Intel) **law**:

(1965: the number of switches doubles every second year)

1975: - refinement of the above: **The number of switches on a CPU doubles every 18 months**

Until 2020 or 2030 we would reach in such a way to the atomic level or quantum computer! – The reason is: light speed limit (see Example 1.2 below!)

Flops:

first computers	10^2	**100 Flops**	*(sada op/s)*
desktop computers today	10^9	Gigaflops (**GFlops**)	*(miljard op/s)*
supercomputers nowadays	10^{12}	Teraflops (**TFlops**)	*(triljon op/s)*
the aim of today	10^{15}	Petaflops (**PFlops**)	*(kvadriljon op/s)*
next step	10^{18}	Exaflops (**EFlops**)	*(kvintiljon op/s)*

http://en.wikipedia.org/wiki/Orders_of_magnitude_(numbers)

1.3 Usage areas of a petaflop computer

Engineering applications

- Wind tunnels
- Combustion engine optimisation
- High frequency cirquits
- Buildings and mechanical structures (optimisation, cost, shape, safety etc)
- Effective and safe nuclear power station design
- Simulation of nuclear explosions

Scientific applications

- Bioinformatics
- Computational physics and chemistry (quantum mechanics, macromolecules, composite materials, chemical reactions etc).
- Astrophysics (development of galaxies, thermonuclear reactions, postprocessing of large datasets produced by telescopes)
- Weather modelling
- Climate and environment modelling
- Looking for minerals
- Flood predictions

Commercial applications

- Oil industry
- Market and monetary predictions
- Global economy modelling
- Car industry (like crash-tests)
- Aviation (design of modern aircrafts and space shuttles)

Applications in computer systems

- Computer security (discovery of network attacks)
- Cryptography

- Embedded systems (for example, car)
- *etc*

1.4 Example 1.1

Why speeds of order 10^{12} are not enough?

Weather forecast in Europe for next 48 hours
from sea lever upto 20km
– need to solve an ODE (xyz and t)
The volume of the region: $5000 * 4000 * 20$ km^3.
Stepsize in xy-plane 1km Cartesian mesh z-direction: 0.1km. Timesteps: 1min.
Around 1000 flop per each timestep. As a result, ca

$$5000 * 4000 * 20\,\text{km}^3 \times 10 \text{ rectangles per km}^3 = 4 * 10^9 \text{meshpoints}$$

and

$$4 * 10^9 \text{ meshpoints} * 48 * 60 \text{timesteps} \times 10^3 \text{ flop} \approx 1.15 * 10^{15} \text{ flop.}$$

If a computer can do 10^9 flops (nowadays PC-s), it will take around

$$\begin{array}{rcl} 1.15 * 10^{15} \text{ flop} & / & 10^9 \text{ flop} \\ = 1.15 * 10^6 \text{ seconds} & \approx & 13 \text{ days !!} \end{array}$$

But, with 10^{12} flops,

$$1.15 * 10^3 \text{ seconds} \quad < \quad 20 \text{ min.}$$

Not hard to imagine "small" changes in given scheme such that 10^{12} flops not enough:

- Reduce the mesh stepsize to $0.1km$ in **each** directions and timestep to 10 seconds and the total time would grow from

$$20 \text{ min to} \qquad 8 \text{ days.}$$

- We could replace the Europe with the whole World model (the area: $2 * 10^7 \text{km}^2 \longrightarrow 5 * 10^8 \text{km}^2$) and to combine the model with an Ocean model.

Therefore, we must say: **The needs of science and technology grow faster than available possibilities, need only to change ε and h to get unsatisfied again!**

But again, why do we need a parallel computer to achieve this goal?

1.5 Example 1.2

Have a look at the following piece of code:

```
do i=1,1000000000000
   z(i)=x(i)+y(i)  !  ie. 3*10^12 memory accesses
end do
```

Assuming, that data is traveling with the speed of light $3 * 10^8 \mathrm{m/s}$, for finishing the operation in 1 second, the average distance of a memory cell must be:

$$r = \frac{3*10^8 \mathrm{m/s} * 1\mathrm{s}}{3*10^{12}} = 10^{-4}\mathrm{m}.$$

Typical computer memory is situated on a rectangular mesh. The length of each edge would be

$$\frac{2*10^{-4}\mathrm{m}}{\sqrt{3*10^{12}}} \approx 10^{-10}\mathrm{m},$$

– the size of a small atom!

But why is parallel computing still not predominant?

Three main reasons: **hardware, algorithms and software**
Hardware: speed of

- networking
- peripheral devices
- memory access

do not cope with the speed growth in processor capacity.
Algorithms: An enormous number of different algorithms exist but

- problems start with their implementation on some real life application

Software: development in progress;

- no good enough automatic parallelisers
- everything done as handwork
- no easily portable libraries for different architectures
- does the paralleising effort pay off at all?

BUT (as explained above) parallel computing will take over

1.6 Example 1.3

Solving a sparse system of linear equations using MUMPS4.3 (***Mu****ltifrontal* ***M****assively* ***P****arallel* ***S****olver*):

SUN computer class procesors for solution of a medium size problem (262144 unknowns, 1308672 nonzeros)

#procs.	time (s)	speedup
1	84.3	
2	63.0	1.34
4	53.6	1.57
8	27.0	3.12
12	59.3	1.47
16	29.5	2.86
20	57.5	1.47
23	74.0	1.14

1.7 Example 1.4

Approximation of π using Monte-Carlo method

#procs.	time (s)	speedup
1	107.8	
2	53.5	2.01
4	26.9	4.00
8	13.5	7.98
12	9.3	11.59
16	6.8	15.85
20	5.5	19.59
21	9.0	11.97
23	29.2	3.69

2 Computer architectures and parallelism

Various ways to classify computers:

- Architecture
  - Single processor computer
  - Multicore processor
  - distributed system
  - shared memory system
- operating system
  - UNIX,
  - LINUX,
  - (OpenMosix)
  - Plan 9

  - WIN*
  - *etc*

- usage area
  - supercomputing
  - distributed computing
  - real time sytems
  - mobile systems
  - neurological networks
  - *etc*

But impossible to ignore (implicit or explicit) parallelism in a computer or a set of computers

Osa I

Parallel Architectures

2.1 Processor development

Instruction pipelining (similar to car production lines) - performing different sub-operations with moving objects

Instruction throughput (the number of instructions that can be executed in a unit of time) increase

operation overlap

RISC processor pipelines:

- Instruction fetch
- Instruction decode and register fetch
- Execute
- Memory access
- Register write back

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Basic five-stage pipeline in a RISC machine (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back). In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.

Pipeline length?

Limitations:

- pipeline speed defined by the slowest component
- Usually, each 5-6 operation - branch
- Cost of false prediction grows with the length of pipeline (larger number of subcommands get waisted)

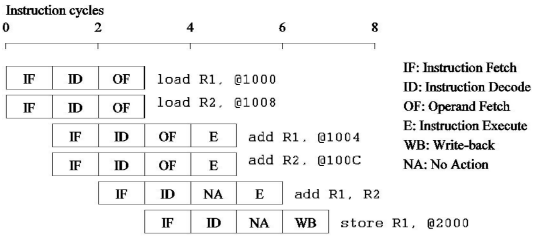
One possibility: Multiple pipelines (*super-pipelined processors, superscalar execution)* - in effect, execution of multiple commands in a single cycle

Example 2.1: Two pipelines for adding 4 numbers:

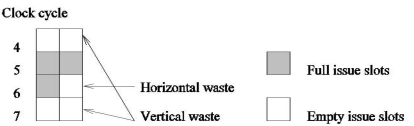
```
1. load R1, @1000      1. load R1, @1000      1. load R1, @1000
2. load R2, @1008      2. add R1, @1004       2. add R1, @1004
3. add R1, @1004       3. add R1, @1008       3. load R2, @1008
4. add R2, @100C       4. add R1, @100C       4. add R2, @100C
5. add R1, R2          5. store R1, @2000     5. add R1, R2
6. store R1, @2000                            6. store R1, @2000
```

(i) (ii) (iii)

(a) Three different code fragments for adding a list of four numbers.



(b) Execution schedule for code fragment (i) above.



(c) Hardware utilization trace for schedule in (b).

True Data Dependency - result of one operation being an input to another

Resource Dependency - two operations competing for the same resource (e.g. processor floating point unit)

Branch Dependency (or Procedural Dependency) - scheduling impossible in case of if-directive

Instruction Level Parallelism (ILP)

- possibility for **out-of-order instruction execution**
  - factor: **size of instruction window**
- ILP is limited by:
  - parallelism present in the instruction window
  - hardware parameters
    - existence of different functional units

- number of pipelines
- pipeline lengths
- pipeline properties
- etc

- not possible to always utilise all **Functional Units (FU)**
  - if at certain timestep none of FUs is utilised – ***vertical waste***
  - if only some FUs utilised – ***horisontal waste***

- Typically, processors support 4-level superscalar execution

Very Long Instruction Word (VLIW) Processors

- Main design concern with superscalar processors – complexity and high price
- VLIW – based on compile-time analysis - which commands to bind together for ILP
- These commands get packed together into one (long) instruction (giving the name)
- First used in *Multiflow Trace machine (ca 1984)*
- Intel IA64 - part of the concept being implemented

VLIW properties

- Simpler hardware side
- Compiler has more context to find ILP
- But compiler lacks all the run-time information (e.g. data-misses in cache), therefore, only quite conservative approach possible (just-in-time compilers might have benefit here!)
- More difficult prediction of branching and memory usage
- VLIW performance depends highly on compiler abilities, like
  - loop unrolling
  - speculative execution
  - branch prediction etc.
- 4-way to 8-way parallelism in VLIW processors