



Consortium des Equipements  
de Calcul Intensif  
en Fédération Wallonie-Bruxelles

# Introduction to Parallel Computing

damien.francois@uclouvain.be  
October 2017



# Why parallel?



Speed up – Solve a problem faster  
→ more processing power  
(a.k.a. strong scaling)

Scale up – Solve a larger problem  
→ more memory and network capacity  
(a.k.a. weak scaling)

Scale out – Solve many problems  
→ more storage capacity

# Agenda



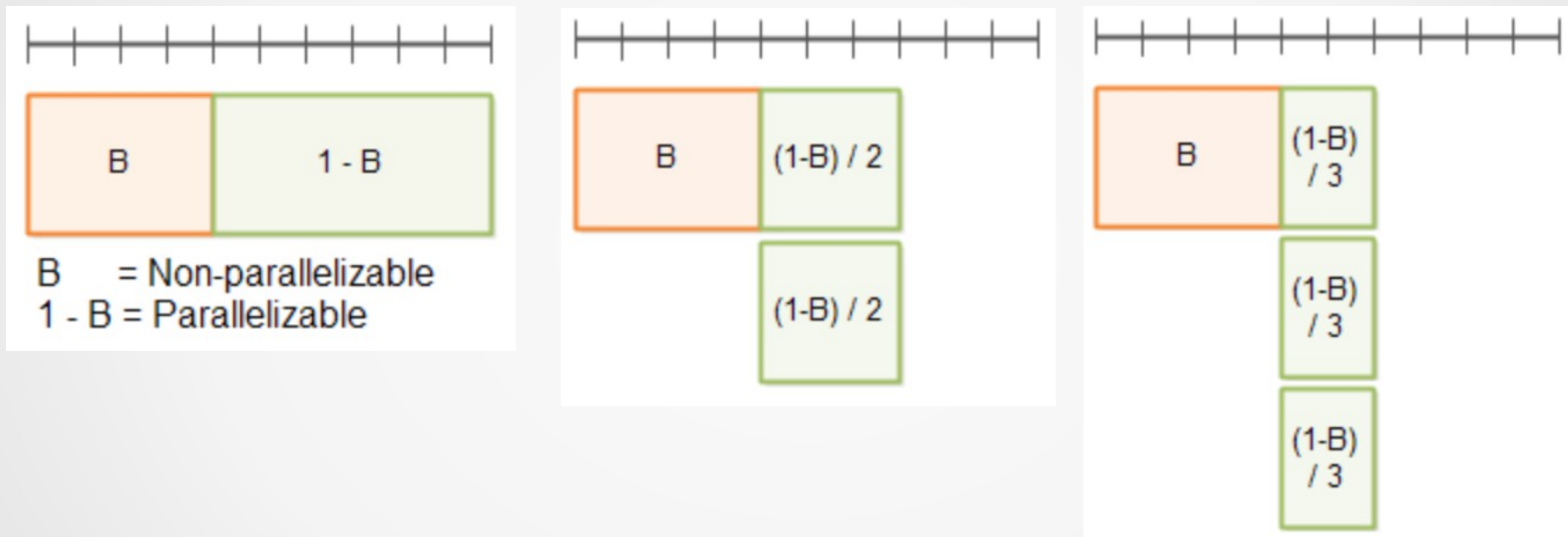
1. General concepts
2. Hardware
3. Programming models
4. User tools

1.



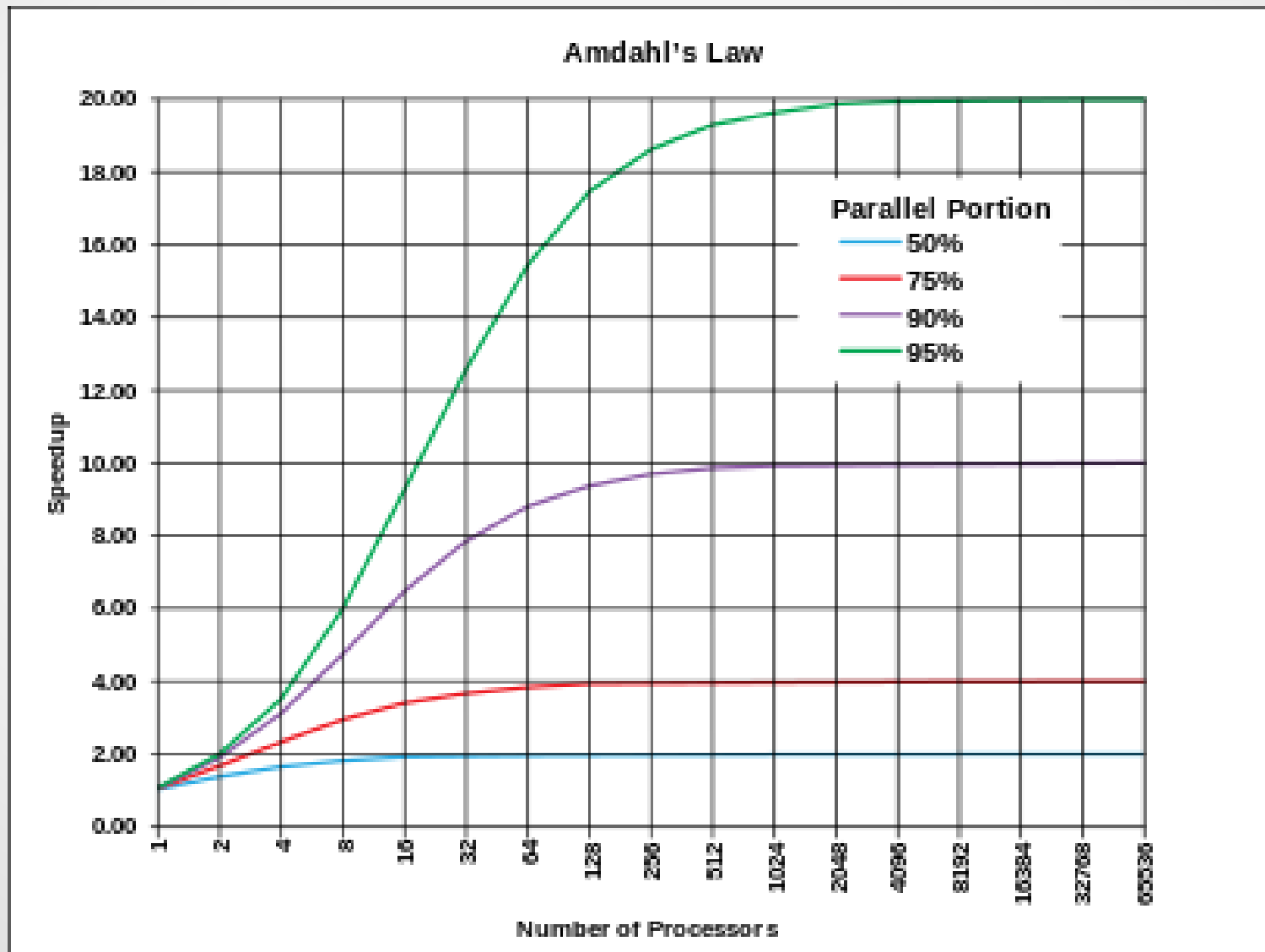
# General concepts

# Amdahl's Law

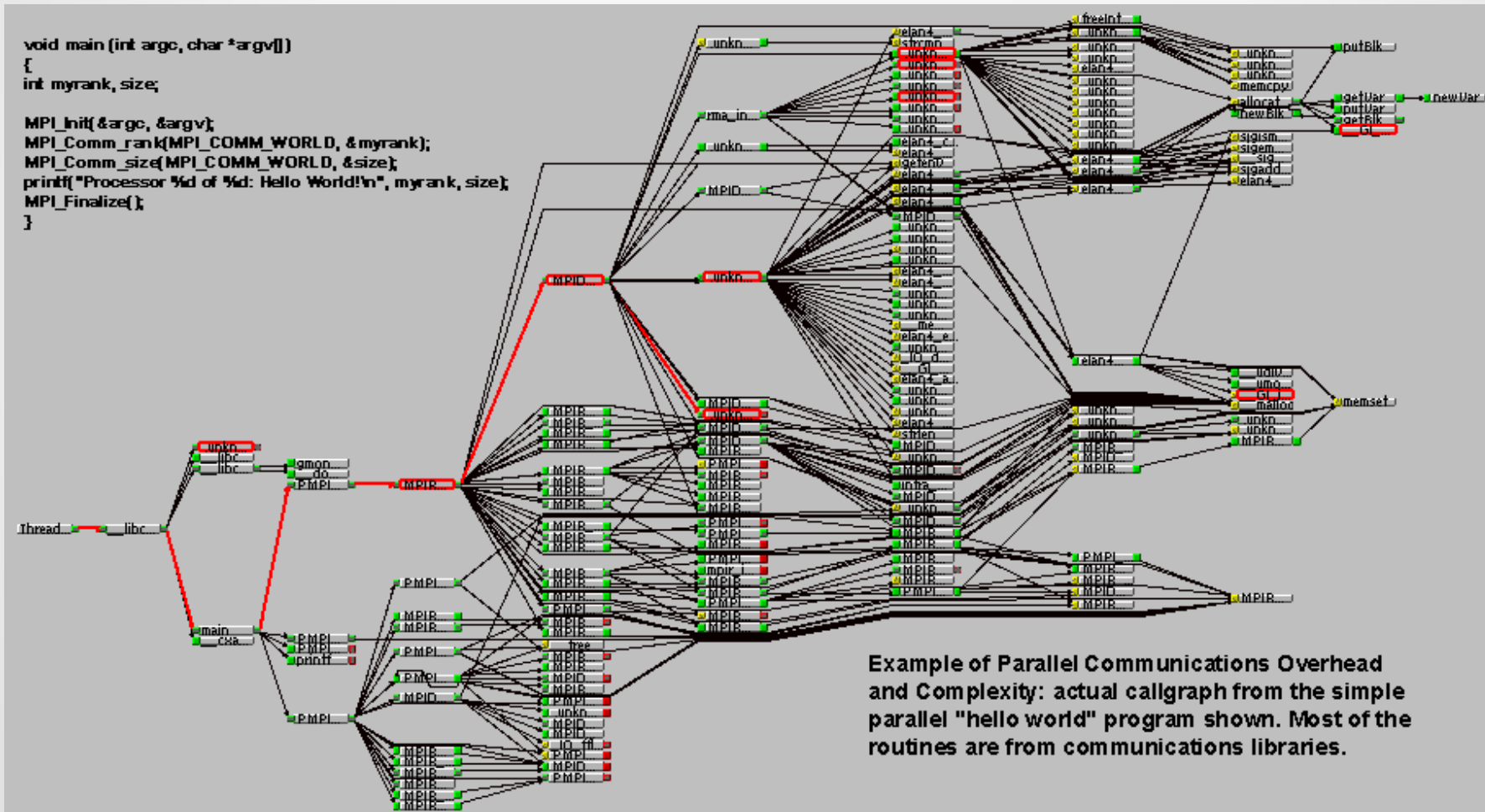


In parallel computing, Amdahl's law is mainly used to predict the theoretical maximum speedup for programs using multiple processors.

# Amdahl's Law



# Parallel overhead



# Parallelization involves

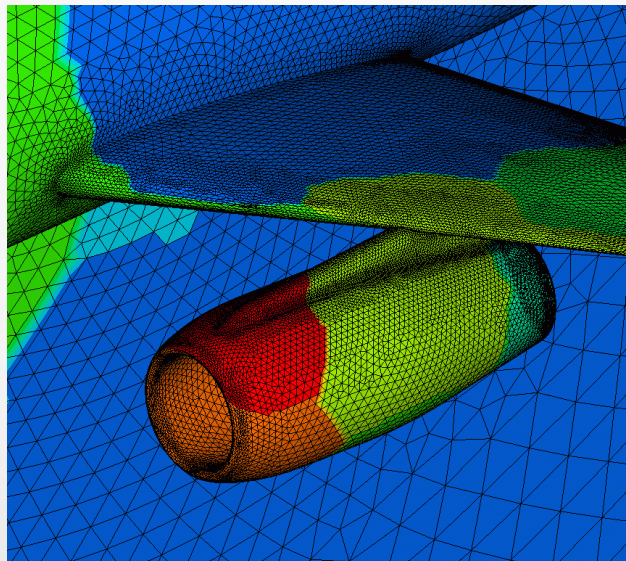


- *decomposition* of the work
  - **distributing instructions** to processors
  - **distributing data** to memories
- *collaboration* of the workers
  - **synchronization** of the distributed work
  - **communication** of data



# Decomposition

- **Work decomposition** : task-level parallelism
- **Data decomposition** : data-level parallelism
- **Domain decomposition** : decomposition of work and data is done in a higher model, e.g. in the reality



- **Synchronous** (SIMD) at the processor level
- **Fine-grained** parallelism if subtasks must communicate many times per second (instruction level); loosely synchronous
- **Coarse-grained** parallelism if they do not communicate many times per second (function-call level)
- **Embarrassingly parallel** if they rarely or never have to communicate (asynchronous)

# Parallel programming paradigms



- **Task-farming** (master/slave or work stealing)
- **SPMD** (Single program multiple data)
- **Pipelining** (A->B->C, one process per task concurrently)
- **Divide and Conquer** (processes spawned at need and report their result to the parent)
- **Speculative parallelism** (processes spawned and result possibly discarded)

2.



# Hardware

# At the processor level



- Instruction-level parallelism (ILP)
  - Instruction pipelining
  - Superscalar execution
  - Out-of-order execution
  - Speculative execution
- Single Instruction Multiple Data (SIMD)

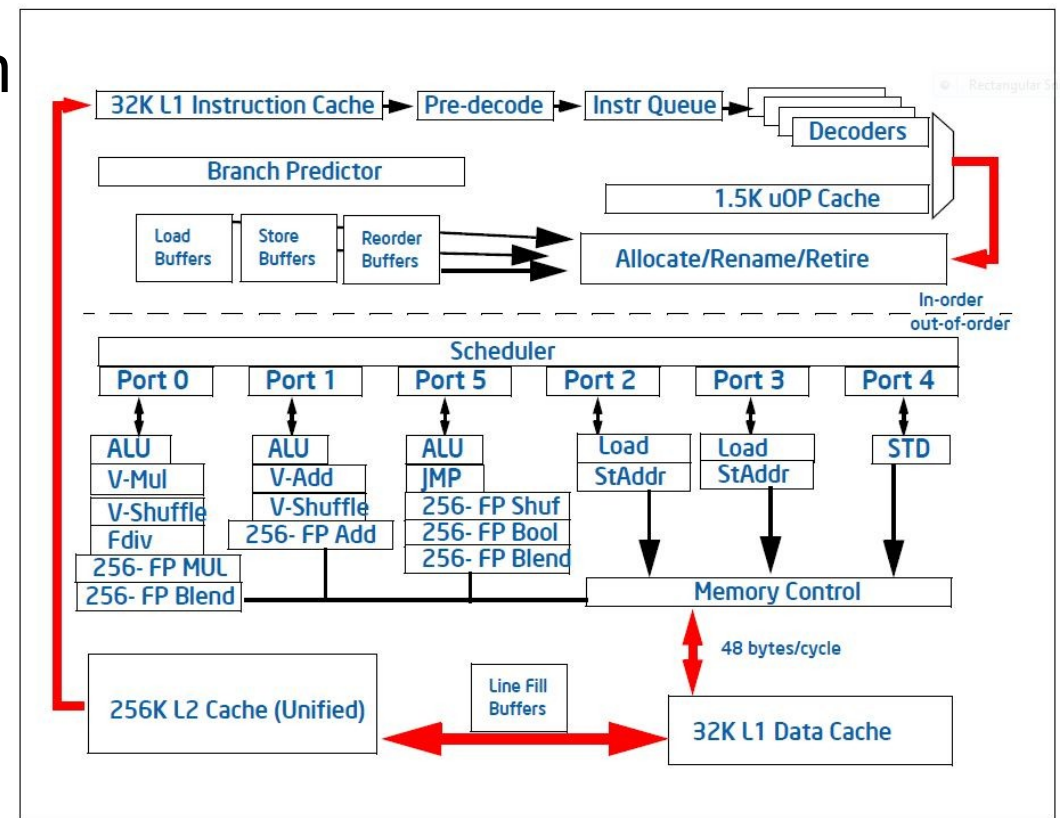
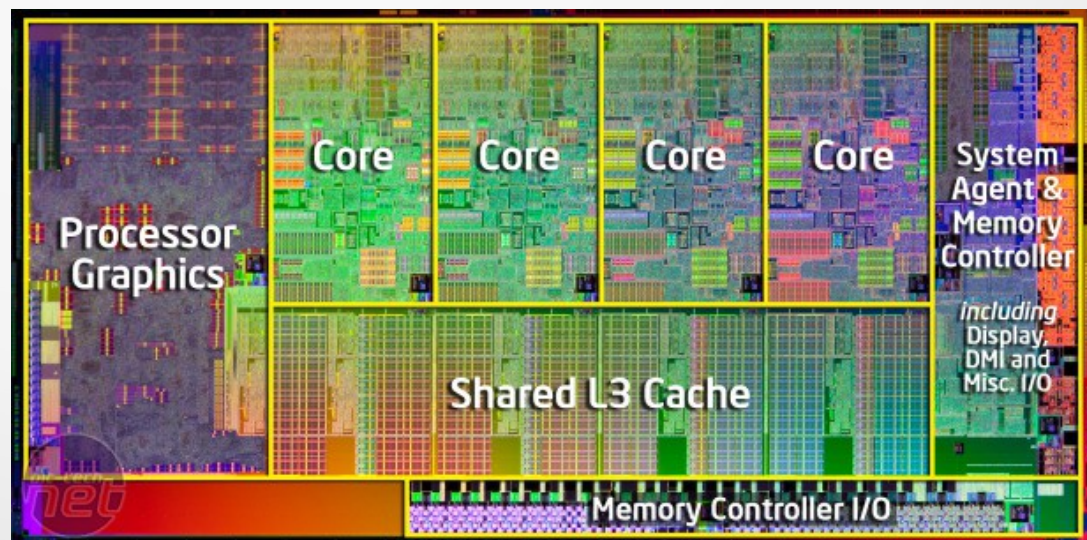
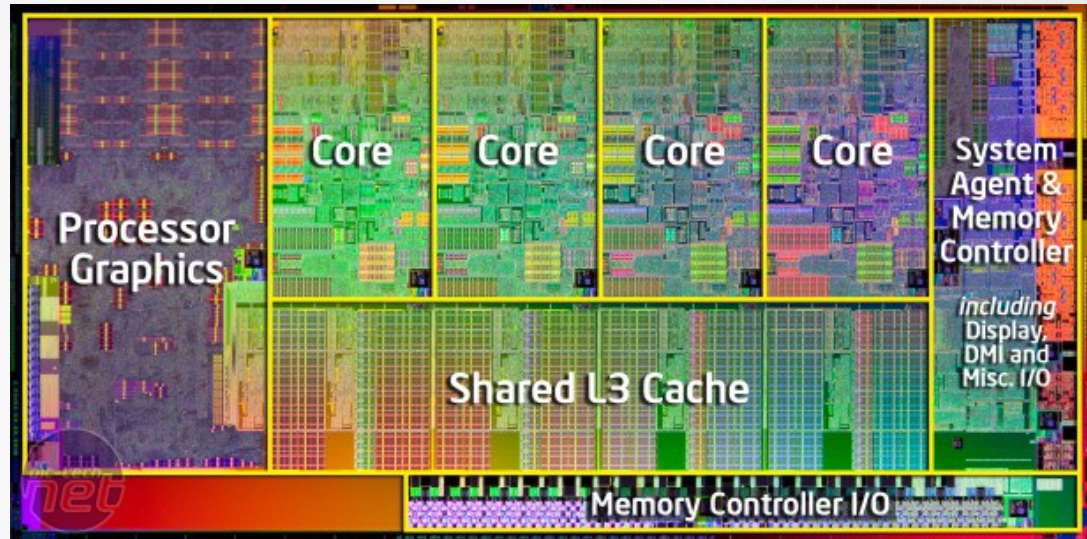


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality



# At the computer level

- Multithreading
  - SMP
  - NUMA
- Accelerators

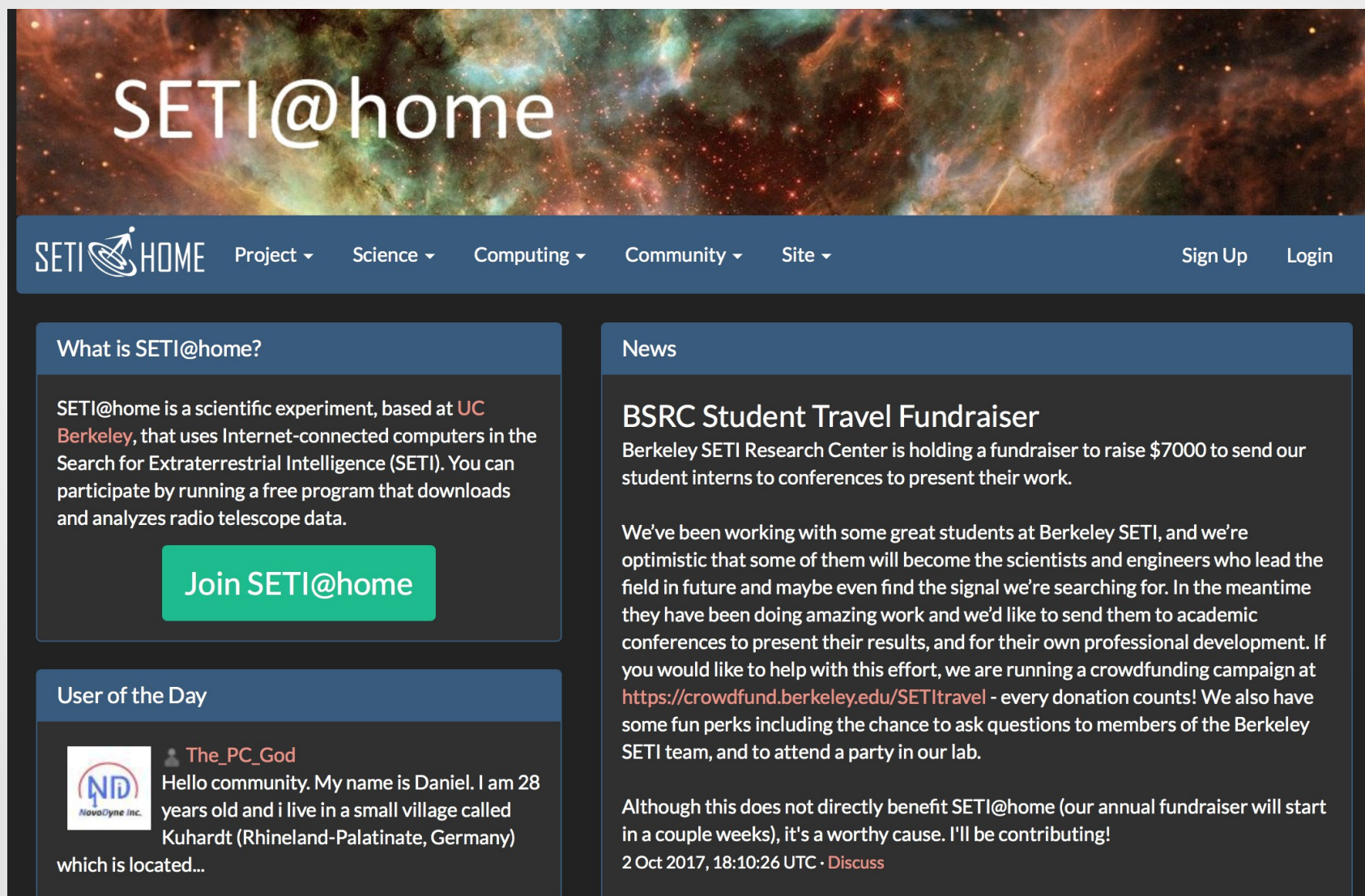


# At the system level





# Distributed computing

A screenshot of the SETI@home website. The header features the 'SETI@home' logo and a navigation menu with items: Project, Science, Computing, Community, and Site. There are also 'Sign Up' and 'Login' links. The main content area is divided into two columns. The left column has a section titled 'What is SETI@home?' with a description of the project and a green 'Join SETI@home' button. Below this is a 'User of the Day' section featuring a profile for 'The\_PC\_God' with a profile picture and a bio. The right column has a 'News' section with a headline 'BSRC Student Travel Fundraiser' and a detailed paragraph about the fundraiser, including a link to the crowdfunding campaign and a date/time stamp at the bottom.

## What is SETI@home?

SETI@home is a scientific experiment, based at **UC Berkeley**, that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You can participate by running a free program that downloads and analyzes radio telescope data.

[Join SETI@home](#)

## User of the Day



**The\_PC\_God**

Hello community. My name is Daniel. I am 28 years old and i live in a small village called Kuhardt (Rhineland-Palatinate, Germany) which is located...

## News

### BSRC Student Travel Fundraiser

Berkeley SETI Research Center is holding a fundraiser to raise \$7000 to send our student interns to conferences to present their work.

We've been working with some great students at Berkeley SETI, and we're optimistic that some of them will become the scientists and engineers who lead the field in future and maybe even find the signal we're searching for. In the meantime they have been doing amazing work and we'd like to send them to academic conferences to present their results, and for their own professional development. If you would like to help with this effort, we are running a crowdfunding campaign at <https://crowdfund.berkeley.edu/SETItravel> - every donation counts! We also have some fun perks including the chance to ask questions to members of the Berkeley SETI team, and to attend a party in our lab.

Although this does not directly benefit SETI@home (our annual fundraiser will start in a couple weeks), it's a worthy cause. I'll be contributing!

2 Oct 2017, 18:10:26 UTC · [Discuss](#)



# Cluster computing



## Consortium des Équipements de Calcul Intensif

6 clusters, 10k cores, 1 login, 1 home directory

### About

CÉCI is the 'Consortium des Équipements de Calcul Intensif'; a consortium of high-performance computing centers of [UCL](#), [ULB](#), [ULg](#), [UMons](#), and [UNamur](#). [Read more.](#)



### The common storage is functional!

Have you tried it yet? [More info...](#)

### Latest News

SATURDAY, 23 SEPTEMBER 2017

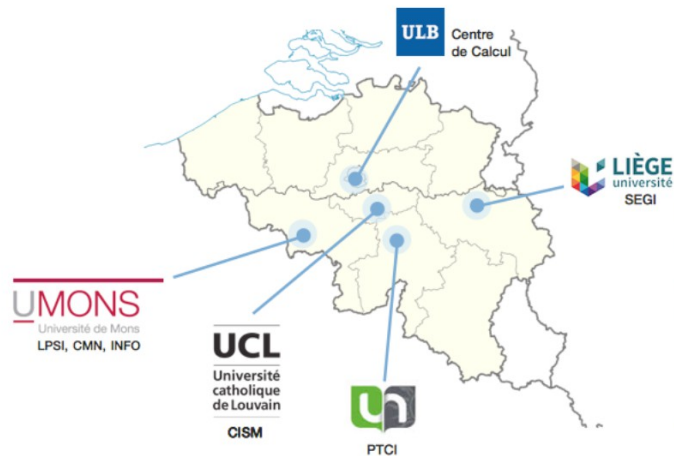
#### A CECI user pictured in the ULiège news!

The ULiège website [published a story](#) (in French) about the work of Denis Baurain and his collaborators on the Tier-1 cluster Zenobe that lead to a publication in [Nature Ecology & Evolution](#).

TUESDAY, 01 AUGUST 2017


#### Ariel Lozano is the new CÉCI logisticien

We are happy to announce the hire of a new CECI logisticien: Ariel Lozano. Welcome Ariel!



# Grid computing






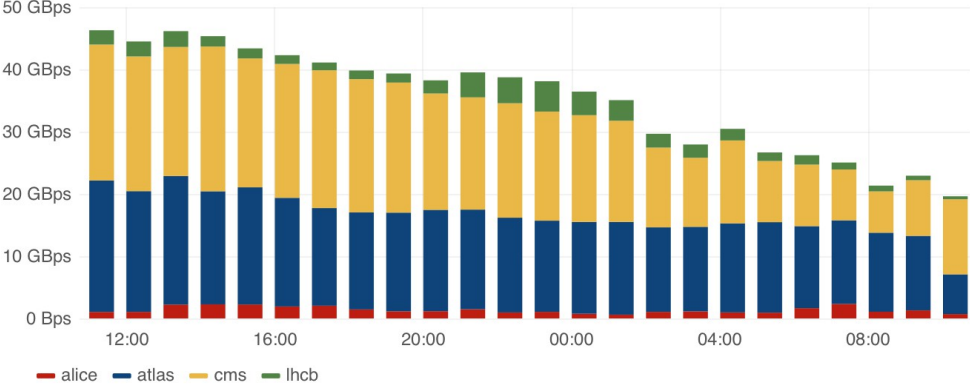
[Home](#) [Collaboration](#) [Meetings](#) [Grid Operations](#) [Security](#) [Tools](#) [Docs & Ref](#) [Getting Started](#) [Public site](#)

Home

## Welcome to the Worldwide LHC Computing Grid

### Last 24 hours

 **Transfer Throughput**



Time	alice	atlas	cms	lhcb
12:00	2	20	20	2
14:00	2	20	20	2
16:00	2	18	20	2
18:00	2	16	18	2
20:00	2	14	16	2
22:00	2	12	14	2
00:00	2	10	12	2
02:00	2	8	10	2
04:00	2	6	8	2
06:00	2	4	6	2
08:00	2	2	4	2

The Worldwide LHC Computing Grid (**WLCG**) project is a global collaboration of more than 170 computing centres in 42 countries, linking up national and international grid infrastructures.

#### About WLCG

[What is WLCG? See our public site...](#)


[Hangout with CERN: LHC and Grid - the world is our calculator](#)

#### Jobs

No jobs currently published.

#### News

- [TEG Reports](#)
- [Computing Model Update available](#)
- [Technology Market Cost Trends](#)



# Cloud computing



3.



# Programming models

# Programming models



- Single computer:
  - multicore: PThreads, OpenMP, TBB, OpenCL
  - GPUs: CUDA, OpenCL, OpenAcc
- Multi-computer:
  - Distributed memory:
    - Shared storage: MPI (clusters)
    - Distributed storage: MapReduce (clouds)
    - No storage: BOINC (distributed computing)
  - Shared memory: CoArray, UPC

4.



# User tools

that GNU/Linux offers

# Parallel processes in Bash



```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ cat lower.sh
#!/bin/bash
#
# Usage:
#   ./lower.sh [input_file [output_file]]
#
# Make ACTG chars lower case with extra processing.
#
# If output_file is not defined, stdout is used
# If input_file and output_file are not defined, stdin and stdout are used.

while read line; do
sleep 1
echo $line | tr ACTG actg >> ${2-/dev/stdout}
done < ${1-/dev/stdin}

dfr@hmem00:~/parcomp $ cat d.txt
G
C
A
G
dfr@hmem00:~/parcomp $ ./lower.sh d.txt
g
c
a
g
dfr@hmem00:~/parcomp $
```

# Parallel processes in Bash

```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ # Foreground: commands end with ';'
dfr@hmem00:~/parcomp $ time { ./lower.sh d1.txt r1.txt ; ./lower.sh d1.txt r2.txt ; };

real    0m8.033s
user    0m0.004s
sys     0m0.019s
dfr@hmem00:~/parcomp $ # Background, in parallel: commands end with '&' and 'wait' necessary
dfr@hmem00:~/parcomp $ time { ./lower.sh d2.txt r1.txt & ./lower.sh d2.txt r2.txt & wait ; };
[1] 49722
[2] 49723
[1]-  Done                ./lower.sh d2.txt r1.txt
[2]+  Done                ./lower.sh d2.txt r2.txt

real    0m4.011s
user    0m0.004s
sys     0m0.005s
dfr@hmem00:~/parcomp $ █
```



# One program and many files



```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ ls d?.txt
d1.txt d2.txt d3.txt d4.txt
dfr@hmem00:~/parcomp $ ls d?.txt | xargs -n 1 echo "File: "
File: d1.txt
File: d2.txt
File: d3.txt
File: d4.txt
dfr@hmem00:~/parcomp $ time { ls d?.txt | xargs -n 1 ./lower.sh > /dev/null ; }

real    0m16.041s
user    0m0.010s
sys     0m0.006s
dfr@hmem00:~/parcomp $ time { ls d?.txt | xargs -n 1 -P 4 ./lower.sh > /dev/null ; }

real    0m4.014s
user    0m0.008s
sys     0m0.016s
dfr@hmem00:~/parcomp $ █
```

# Several programs and one file

```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ # Using an intermediary file
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmp.txt ; ./upper.sh tmp.txt res.txt ; }

real    0m8.033s
user    0m0.005s
sys     0m0.017s
dfr@hmem00:~/parcomp $ # Using pipes (as our programs can handle stdin and stdout)
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt | ./upper.sh > res.txt ; }

real    0m5.014s
user    0m0.006s
sys     0m0.009s
dfr@hmem00:~/parcomp $ mkfifo tmpfifo
dfr@hmem00:~/parcomp $ ls -l tmpfifo
prw-rw-r-- 1 dfr dfr 0 Oct  7 10:27 tmpfifo
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmpfifo & ./upper.sh tmpfifo res.txt ; }
[1] 65343
[1]+  Done                  ./lower.sh d.txt tmpfifo

real    0m5.013s
user    0m0.002s
sys     0m0.007s
dfr@hmem00:~/parcomp $ █
```

# One program and one large file



```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ # One process to process the whole file
dfr@hmem00:~/parcomp $ time { cat d.txt | ./lower.sh > res.txt ; }

real    0m4.014s
user    0m0.003s
sys     0m0.009s
dfr@hmem00:~/parcomp $ # Four processes handling one line in round robin fashion
dfr@hmem00:~/parcomp $ time { cat d.txt | split --unbuffered --number r/4 --filter ./lower.sh >res.txt ; }

real    0m1.011s
user    0m0.009s
sys     0m0.021s
dfr@hmem00:~/parcomp $ !! & top -u dfr -bn1 | grep lower
time { cat d.txt | split --unbuffered --number r/4 --filter ./lower.sh >res.txt ; } & top -u dfr -bn1 | gr
ep lower
[1] 12817
12822 dfr      20   0  103m 1252 1052 S   0.0  0.0   0:00.00 lower.sh
12823 dfr      20   0  103m 1252 1052 S   0.0  0.0   0:00.00 lower.sh
12824 dfr      20   0  103m 1252 1052 S   0.0  0.0   0:00.00 lower.sh
12825 dfr      20   0  103m 1252 1052 S   0.0  0.0   0:00.00 lower.sh
dfr@hmem00:~/parcomp $
real    0m1.011s
user    0m0.011s
sys     0m0.019s

[1]+  Done                  time { cat d.txt | split --unbuffered --number r/4 --filter ./lower.sh > res
.txt; }
dfr@hmem00:~/parcomp $ █
```

Need recent version of Coreutils/8.22-goolf-1.4.10



# Several programs and many files



```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ time make
./lower.sh d1.txt d1.tmp
./upper.sh d1.tmp d1.res
./lower.sh d2.txt d2.tmp
./upper.sh d2.tmp d2.res
./lower.sh d3.txt d3.tmp
./upper.sh d3.tmp d3.res
./lower.sh d4.txt d4.tmp
./upper.sh d4.tmp d4.res
rm d1.tmp d2.tmp d4.tmp d3.tmp

real    0m32.260s
user    0m0.028s
sys     0m0.099s
dfr@hmem00:~/parcomp $ rm *res
dfr@hmem00:~/parcomp $ time make -j 4
./lower.sh d1.txt d1.tmp
./lower.sh d2.txt d2.tmp
./lower.sh d3.txt d3.tmp
./lower.sh d4.txt d4.tmp
./upper.sh d1.tmp d1.res
./upper.sh d2.tmp d2.res
./upper.sh d4.tmp d4.res
./upper.sh d3.tmp d3.res
rm d1.tmp d2.tmp d4.tmp d3.tmp

real    0m8.163s
user    0m0.025s
```

# Summary



- You have either
  - one very large file to process
    - with one program: `split`
    - with several programs: `pipes`, `fifo`
  - many files to process
    - with one program `xargs`
    - with many programs `make`

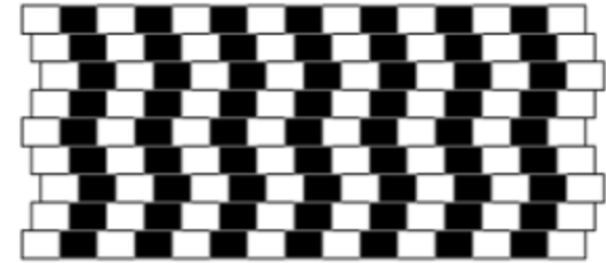
## GNU Parallel

GNU **parallel** is a shell tool for executing jobs in parallel using one or more computers. A job can be a single command or a small script that has to be run for each of the lines in the input. The typical input is a list of files, a list of hosts, a list of users, a list of URLs, or a list of tables. A job can also be a command that reads from a pipe. GNU **parallel** can then split the input and pipe it into commands in parallel.

If you use `xargs` and `tee` today you will find GNU **parallel** very easy to use as GNU **parallel** is written to have the same options as `xargs`. If you write loops in shell, you will find GNU **parallel** may be able to replace most of the loops and make them run faster by running several jobs in parallel.

GNU **parallel** makes sure output from the commands is the same output as you would get had you run the commands sequentially. This makes it possible to use output from GNU **parallel** as input for other programs.

For each line of input GNU **parallel** will execute *command* with the line as arguments. If no *command* is given, the line of input is executed. Several lines will be run in parallel. GNU **parallel** can often be used as a substitute for `xargs` or `cat | bash`.



## GNUparallel

For people who live life in the parallel lane.

More complicated to use but very powerful



- Syntax: `parallel command ::: argument list`

```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ parallel echo ::: 1 2 3 4
1
2
3
4
dfr@hmem00:~/parcomp $ parallel echo ::: {1..10}
1
2
3
4
5
6
7
8
9
10
dfr@hmem00:~/parcomp $ time parallel sleep ::: {1..10}

real    0m11.200s
user    0m0.206s
sys     0m0.129s
dfr@hmem00:~/parcomp $ parallel echo ::: d?.txt
d1.txt
d2.txt
d3.txt
d4.txt
dfr@hmem00:~/parcomp $
```



# GNU Parallel



- Syntax: {} as argument placeholder. Can be modified

```
dfr@hmem00:~/parcomp $ parallel echo {} ::: d?.txt
d1.txt
d2.txt
d3.txt
d4.txt
dfr@hmem00:~/parcomp $ parallel echo {} {.}.res ::: d?.txt
d1.txt d1.res
d2.txt d2.res
d3.txt d3.res
d4.txt d4.res
dfr@hmem00:~/parcomp $ parallel echo {} ::: ../parcomp/d?.txt
../parcomp/d1.txt
../parcomp/d2.txt
../parcomp/d3.txt
../parcomp/d4.txt
dfr@hmem00:~/parcomp $ parallel echo {/} ::: ../parcomp/d?.txt
d1.txt
d2.txt
d3.txt
d4.txt
dfr@hmem00:~/parcomp $
dfr@hmem00:~/parcomp $
dfr@hmem00:~/parcomp $
```

- Syntax: `--xapply`

```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ parallel echo ::: 1 2 3 4 ::: A B
1 A
1 B
2 A
2 B
3 A
3 B
4 A
4 B
dfr@hmem00:~/parcomp $ parallel --xapply echo ::: 1 2 3 4 ::: A B C D
1 A
2 B
3 C
4 D
dfr@hmem00:~/parcomp $ parallel echo {1} and {2} ::: 1 2 3 4 ::: A B C D
1 and A
1 and B
1 and C
1 and D
2 and A
2 and B
2 and C
2 and D
3 and A
3 and B
3 and C
3 and D
4 and A
```

- Syntax: `:::` arguments in file

```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ cat experiments.csv
Number,Letter
1,A
2,B
3,B
3,A
4,C
5,C
5,A
dfr@hmem00:~/parcomp $ parallel --colsep ',' --header '\n' echo {Number} {Letter} ::: experiments.csv
1 A
2 B
3 B
3 A
4 C
5 C
5 A
dfr@hmem00:~/parcomp $
```

- Syntax: --pipe

```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ cat *.csv
Number,Letter
1,A
2,B
3,B
3,A
4,C
5,C
5,A
dfr@hmem00:~/parcomp $ tail -n +2 *.csv | parallel -kN1 --recend '\n' --pipe echo -n "JOB{#} : ;cat;"
JOB1 :1,A
JOB2 :2,B
JOB3 :3,B
JOB4 :3,A
JOB5 :4,C
JOB6 :5,C
JOB7 :5,A
dfr@hmem00:~/parcomp $
```

# Other interesting options



- S Use remote servers through SSH
- j n Run n jobs in parallel
- k Keep same order
- delay n Ensure there are n seconds between each start
- timeout n Kill task after n seconds if still running

Author asks to be cited: O. Tange (2011): *GNU Parallel - The Command-Line Power Tool*, The USENIX Magazine, February 2011:42-47.

# Exercise



Reproduce the examples with `./lower` and `./upper.sh`  
using GNU Parallel

- One program and many files

```
$ time parallel -k ./lower.sh {} > res.txt ::: d?.txt
```

- One program and one large file

```
$ time cat d.txt | parallel -k -N1 --pipe ./lower.sh {} > res.txt
```

- Several programs and several files

```
$ time { parallel ./lower.sh {} {}.tmp ::: d?.txt ; \  
> parallel ./upper.sh {} {}.res ::: d?.tmp ; }
```