An introduction to OpenMP

How the exploit multi core processors' power, with ease

Gabriele Gaetano Fronzé

INFN Istituto Nazionale di Fisica Nucleare

UNIVERSITA' DEGLI STUDI DI TORINO

ELEMENTO MODULAR CLOUD

What is OpenMP™

What is OpenMP

OpenMP is a set of directives for the compiler assisted by libraries

OpenMP helps offloading the parallel coding tasks to the compiler

OpenMP is compatible with C, C++ and Fortran with almost the same standards

OpenMP is made to make use of multi-thread processors

What's a thread?

A thread is the smallest possible independent instructions sequence.

A thread has private stack space.

A thread has shared heap space (address space).

A thread cannot be standalone: it must belong to a process which has own heap space.

A core in a multi-core CPU can process instructions from (at least) one thread per clock cycle.

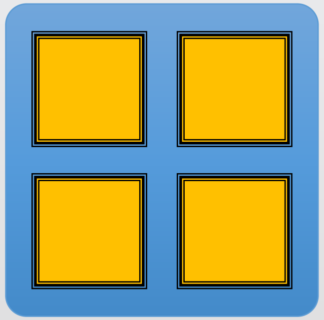
What's a thread?

Is the basic execution unit provided with its own

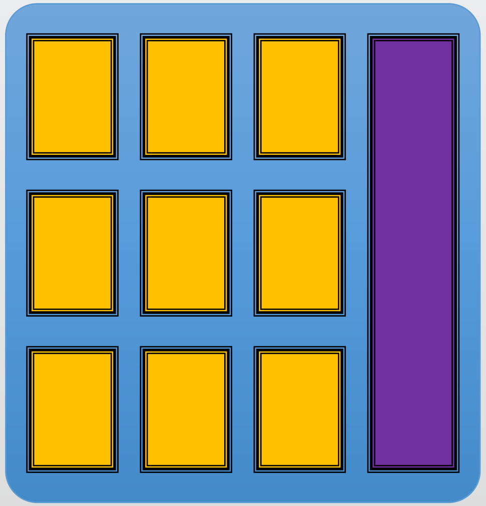
<u>private instruction pointer</u>

What is OpenMP

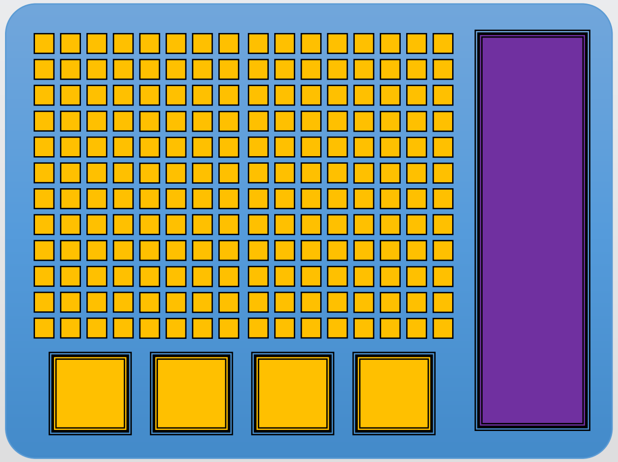
OpenMP is made to use any kind of multi core processor, like:



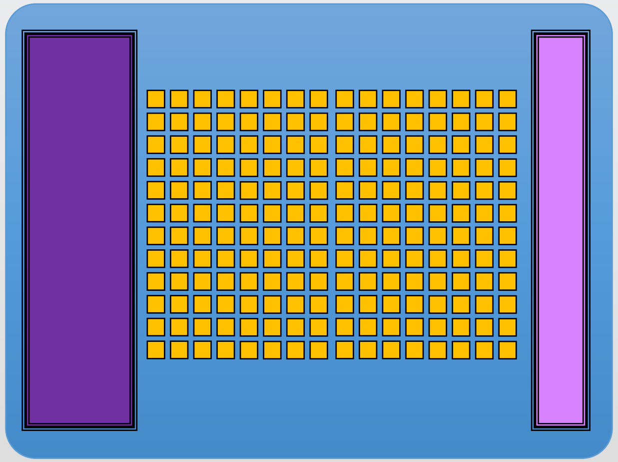
A simple quad core…



A more complex CPU…



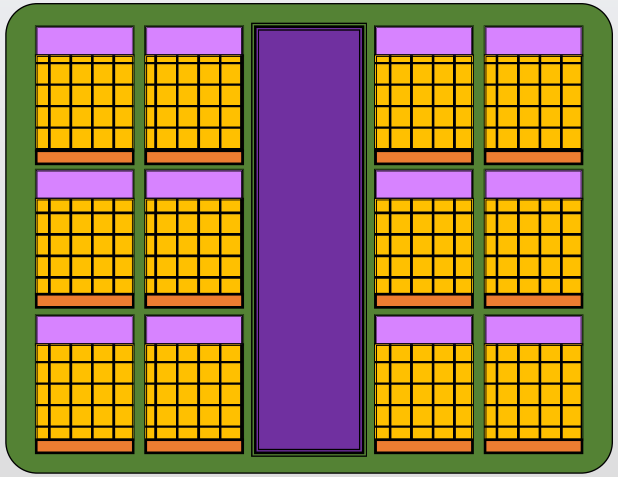
An heterogeneous processor…



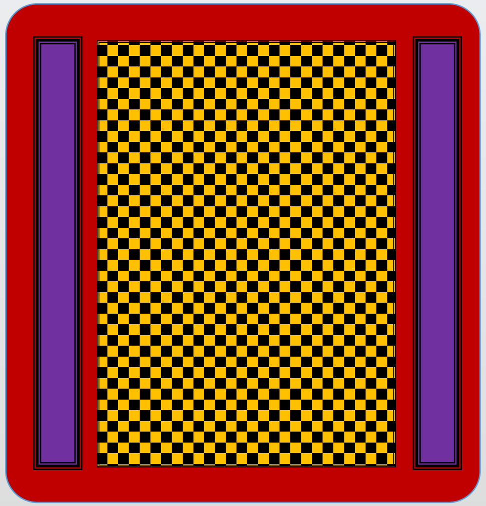
And more…

What is OpenMP

OpenMP and from version 4.5 can also handle:



A modern video card...

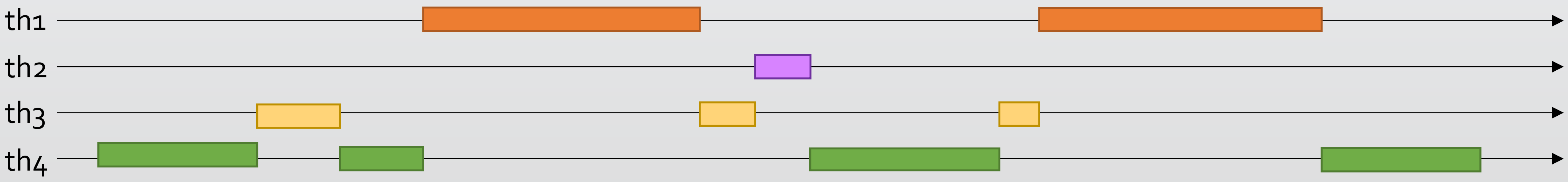


An FPGA...

What is OpenMP

In general OpenMP is a tool to exploit computation parallelism and concurrency, making good use of multi and many core architectures.

Concurrent execution (single core interleaving)

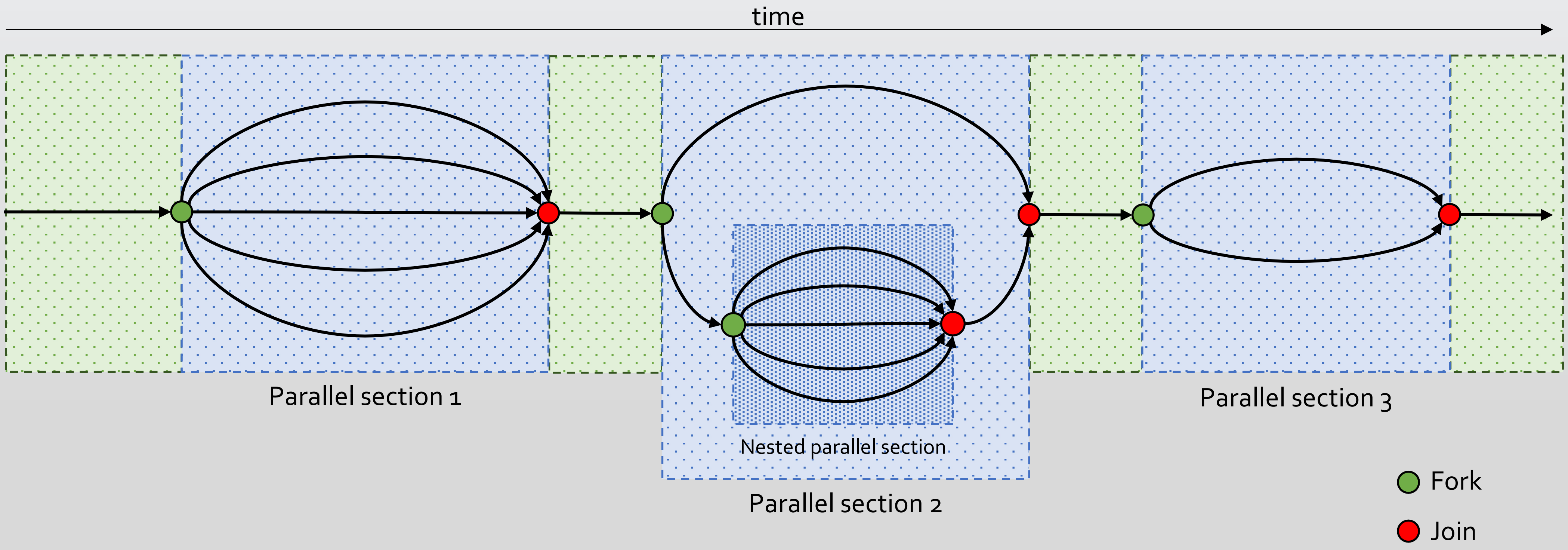


Concurrent parallel execution (independent threads)

What is OpenMP

OpenMP is OS independent: it only requires a supporting compiler.

OpenMP hides the fork-join model to spawn (merge) threads from (into) the main instructions flow:

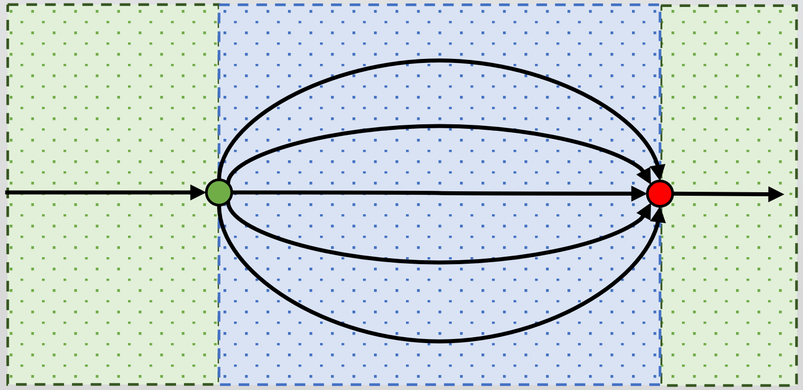


What is OpenMP

The Amdahl's Law defines the maximum speedup factor of a parallel software execution. The speedup is defined with respect to the execution time of the best sequential solution.

$$S = \frac{t_{sequential\ solution}}{t_{parallel\ solution}} = \frac{t_{sequential\ solution}}{t_{sequential\ part} + t_{parallel\ part}}$$

$$\begin{cases} t_{sequential\ part} = t_{sequential\ solution} \cdot f_{sequential} \\ f_{sequential} = 1 - f_{parallel} \\ t_{parallel\ part} = f_{parallel} \cdot \dfrac{t_{sequential\ solution}}{n_{threads}} \end{cases}$$



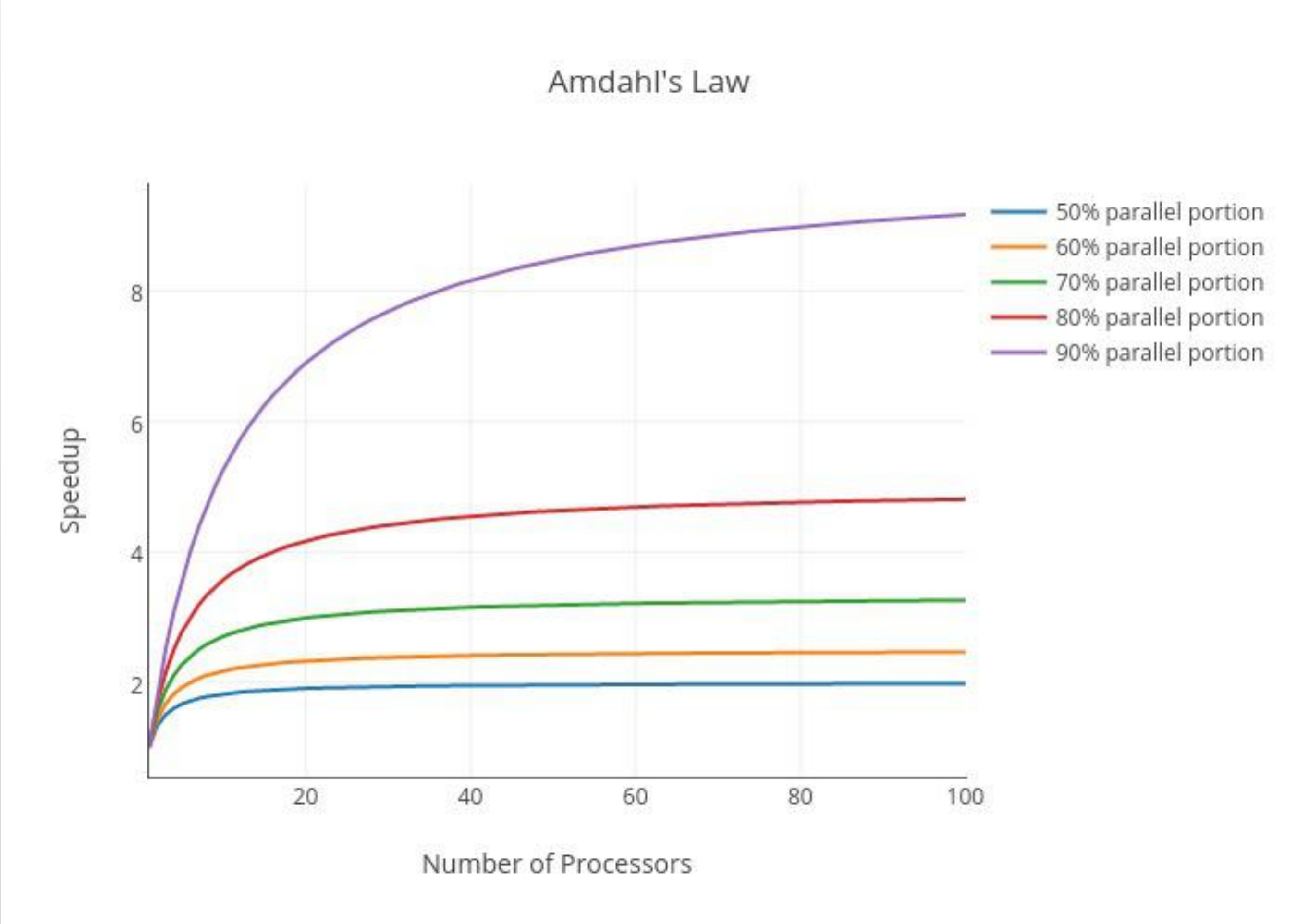
Parallel section 1

$$S = \frac{1}{(1 - f_{parallel}) + \dfrac{f_{parallel}}{n_{threads}}}$$

What is OpenMP

The Amdahl's Law highlights how difficult is to get good speedups even with a ton of cores!

$$S = \frac{1}{1 - f_{parallel} + \frac{f_{parallel}}{n_{threads}}}$$



OpenMP™

Everything to know, in a nutshell

Into the nutshell

OpenMP provides some constructs in the form of compiler

```
#pragmas
```

The necessary definitions are contained in a header file included using

```
#include <omp.h>
```

The only requirement is to use a omp-compliant compiler, such as:

GCC | Cl[#] | Clang* | ICC

*Clang does support OpenMP 3.1 thanks to an Intel Engineer work. Now LLVM is supporting OpenMP 4.5+.
[#] Cl (Windows C/C++ compiler) is not offering an updated OpenMP support, sticking to version 2.0

What's a `pragma`?

A pragma is a compiler instruction able to force the compiler to behave in a specific way.

In other words it's a way to tell the compiler:

"whichever interpretation of my code you'd like to give,

just trust this pragma and compile the code adhering to that"

In practice that's the pattern Open**MP** adopts to "automatically" rework your code.

Into the nutshell

OpenMP fundamental constructs can be summarized in half a page

OpenMP pragma, function of clause	Concepts
`#pragma omp parallel`	Interleaved execution and teams of threads
`int omp_get_thread_num()` `int omp_get_num_threads()`	Get thread ID and thread number
`double omp_get_wtime()`	Get current wall time
`#pragma omp barrier` `#pragma omp critical`	Race condition and synchronization
`#pragma omp for`	Work-sharing, parallel loops, carried dependencies
`reduction(op:list)`	Reduction of values across teams of threads
`schedule(dynamic[,chunk])` `schedule(static[,chunk])`	Loop balance and scheduling
`private(list), firstprivate(list), shared(list)`	Access modifiers
`#pragma omp single`	Single thread section
`#pragma omp task` `#pragma omp taskwait`	Single tasks definition

Into the nutshell

Three OpenMP functions are all you need to know who is executing what and how long:

```
int omp_get_thread_num()

int omp_get_num_threads()

double omp_get_wtime()
```

The first two functions allow to get a unique ID of the thread and the total number of spawned threads.

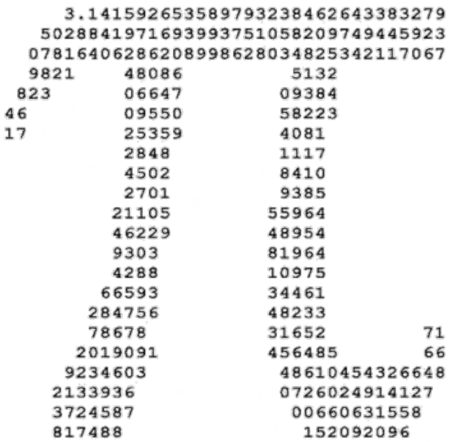
Note that threads IDs are following an array-like numbering:

N threads will be identified by IDs between 0 and N-1

OpenMP wants you!

In the next exercises session we will use a typical problem:

the numerical computation of Pi



$$= \int_0^1 \frac{4}{1 - x^2} dx \cong \frac{1}{n} \cdot \sum_{i=0}^{n} \frac{4}{1 - \left(\frac{i - \frac{1}{2}}{n}\right)^2}$$