A parallel version of the OSCI SYSTEMC™ kernel

Philippe Combes

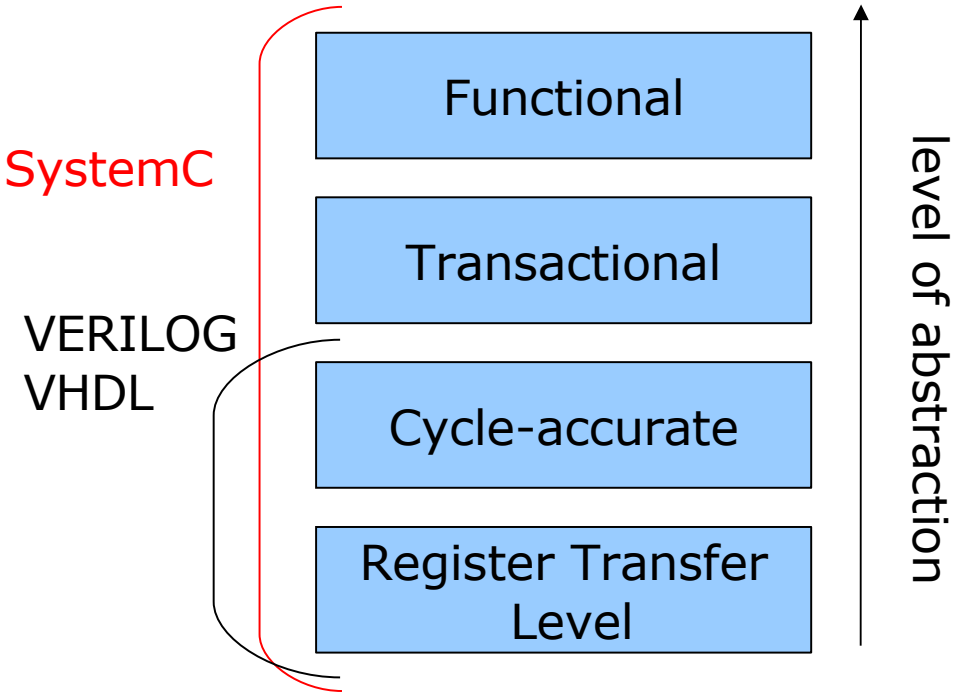
Bastien Chopard

University of Geneva

Julien Zory

STMicroelectronics

- OSCI – year 2000
- C++ class library
  - modules, channels ↔ C++ classes
  - processes ↔ member functions of modules



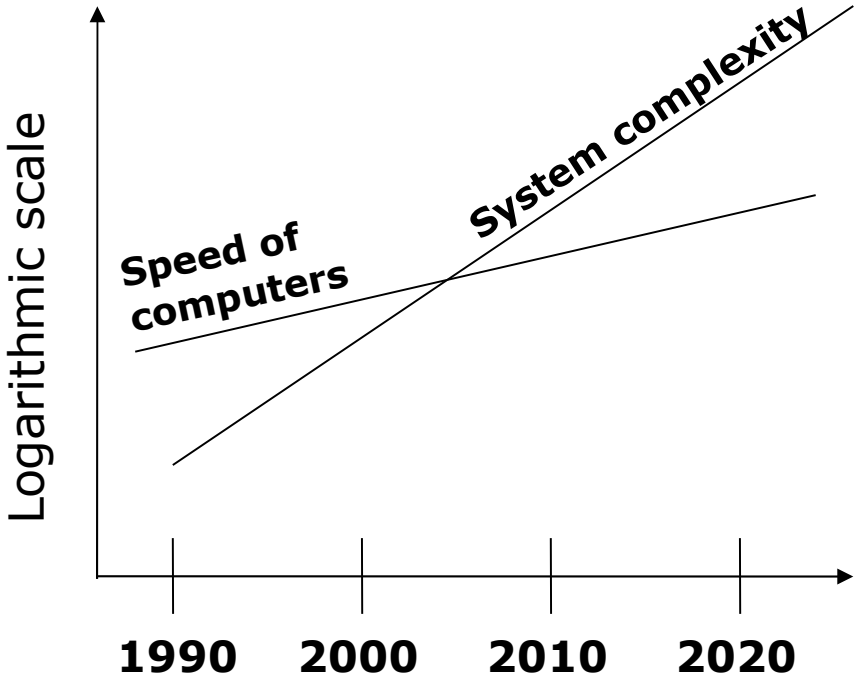
Integrated design flow

starts at high level software

hierarchically refined down to the lowest level of abstraction

Need for efficient simulation

- More and more functions on SoC's
- Algorithms are more and more complex
- Size of SoC's grow by a factor of 4 every 3 years
- Speed of workstations increase by roughly 50% every year



- Sequential scheduler optimizations have limited speedup potential
  - ➔ Parallel simulation

Special thanks to Dr A. Mayer for this information
Source: http://www-ti.informatik.uni-tuebingen.de/~systemc/Documents/Presentation-3-Albrecht-Mayer.pdf

SystemC parallelization attempts

- The quite opaque SimCluster, from Avery Design Systems, may be used with SystemC models but it seems to be dedicated to RTL languages.

  http://www.avery-design.com/

- RITSim, D. Cox, a master thesis of 09/2005: incomplete, dedicated to a specific application and not user-friendly.

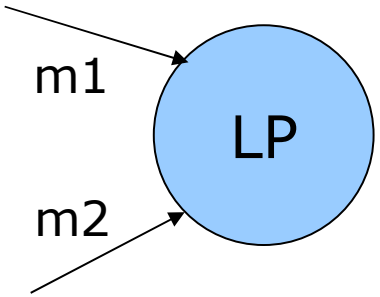
  https://ritdml.rit.edu/dspace/handle/1850/1014

Parallel VHDL/VERILOG: PDES

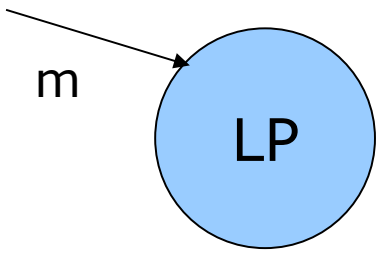
- In the 90's, the Parallel Discrete Event Simulation concept has lead to parallel VHDL and VERILOG simulation.
- Discrete Event System ≡ set of Logical Processes (LP), affecting one another through event messages.
- Each LP has a local clock that stamps its output messages.

➢ Causality constraint:

$$t_{m1} \geq t_{LP} \text{ and } t_{m2} \geq t_{LP}$$



Conservative vs Opimistic PDES



The causality constraint is never violated: $t_m \geq t_{LP}$

- Easy to implement
- Low memory usage
- High synchronization overhead

The causality constraint might be violated: if $t_m < t_{LP}$ then LP rolls back to t_m.

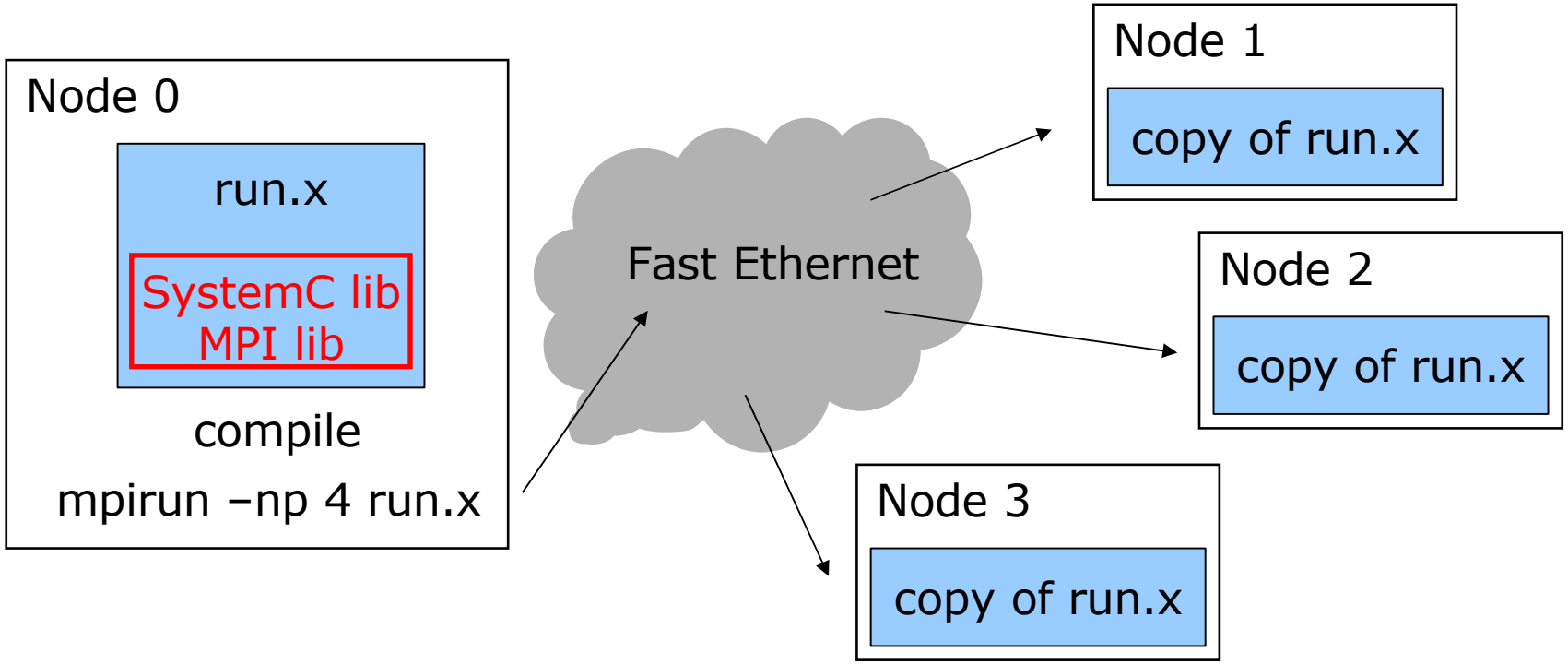
- Low synchronization overhead
- Not user-friendly
- Rollback mechanisms can be cost-effective for RTL models, but untractable for SystemC models with a high level of abstraction.

SystemC kernel parallelization: constraints and choices

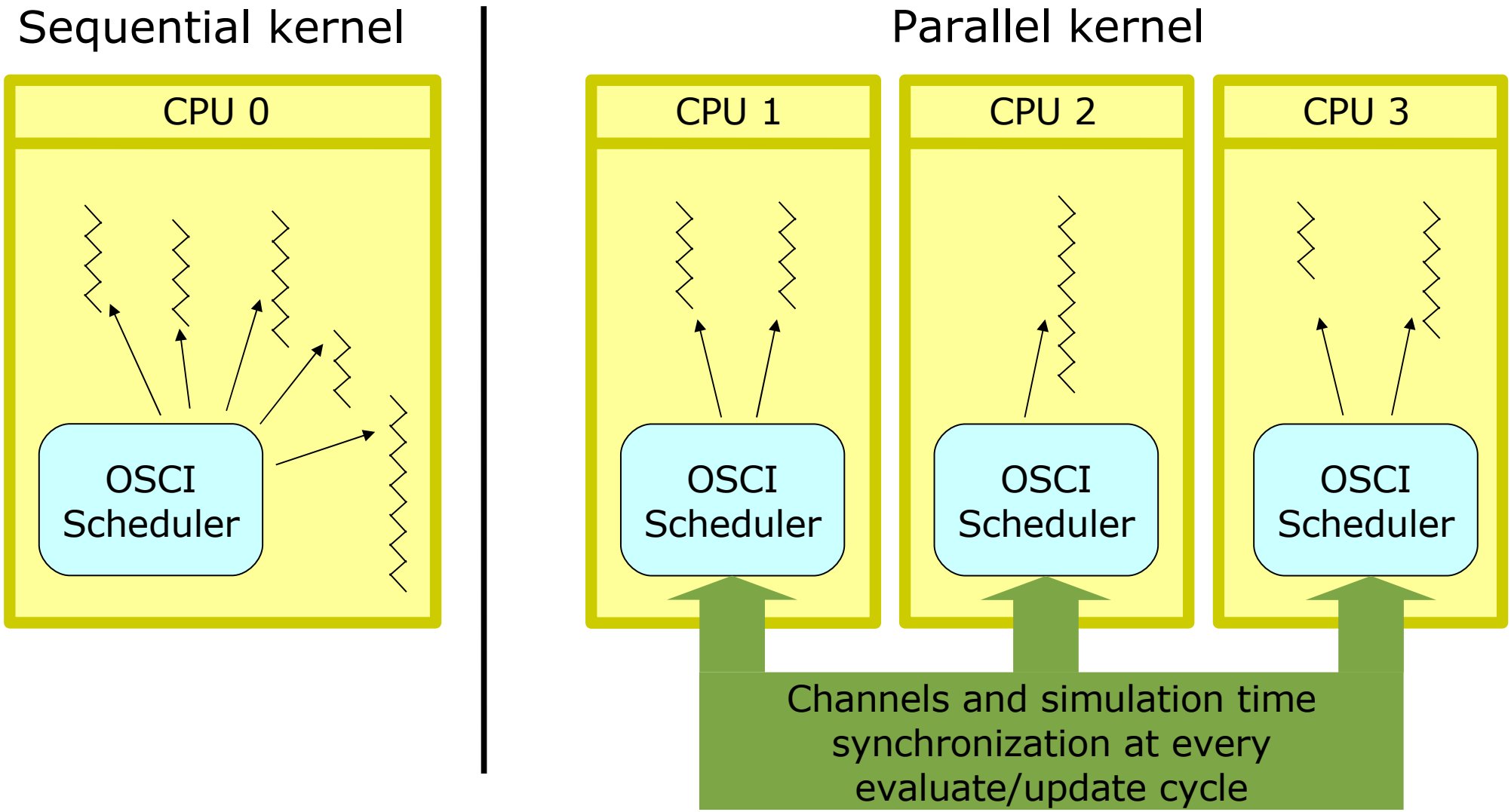
- Conservative PDES approach
- Target: functional level
  - coarse grain, CPU intensive processes
  - reduced synchronization/computation ratio
- USER-FRIENDLY
  - as little changes in the API as possible
  - open source: modify the OSCI kernel

SystemC kernel parallelization: target platform

- Homogeneous clusters of workstations (cheaper than huge SMP's)
- Use MPI standard for communications



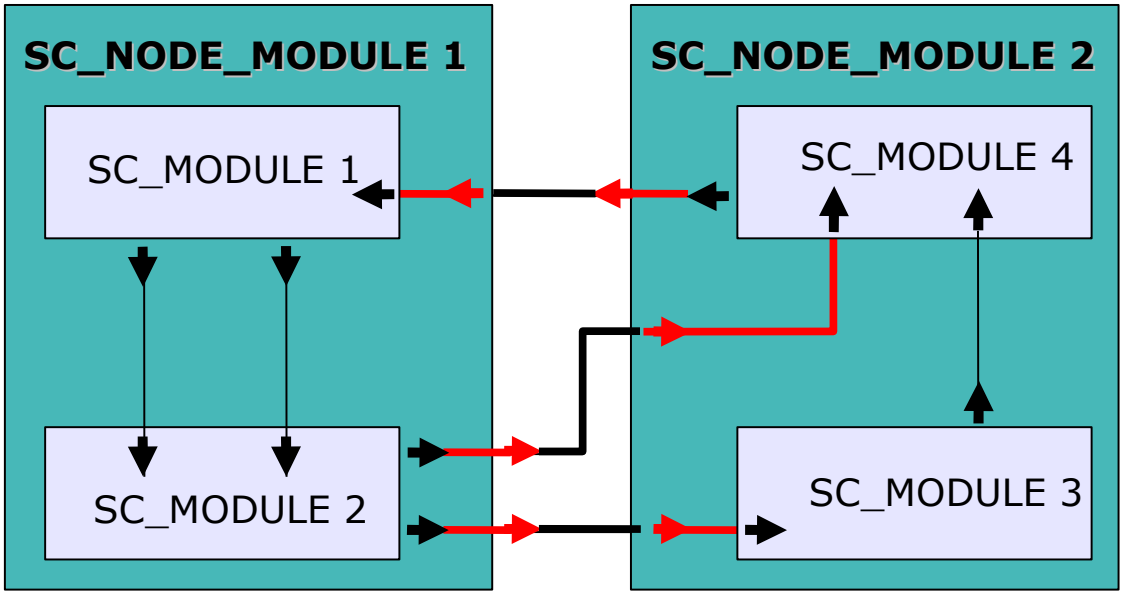
Principle of parallelization



From sequential to parallel

- A new top-level module type: SC_NODE_MODULE

➔ Static partitioning by the model programmer

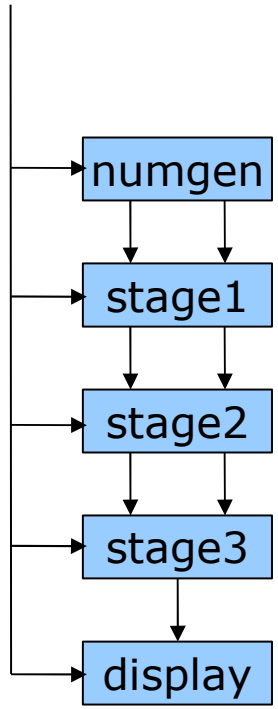


If user defined types are involved in MPI communications, a couple of additional functions are to be implemented.

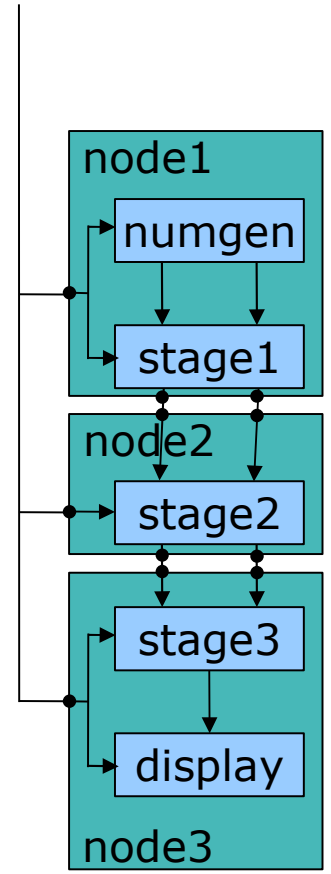
Compilation needs an MPI implementation installed.

SystemC 2.0.1 example

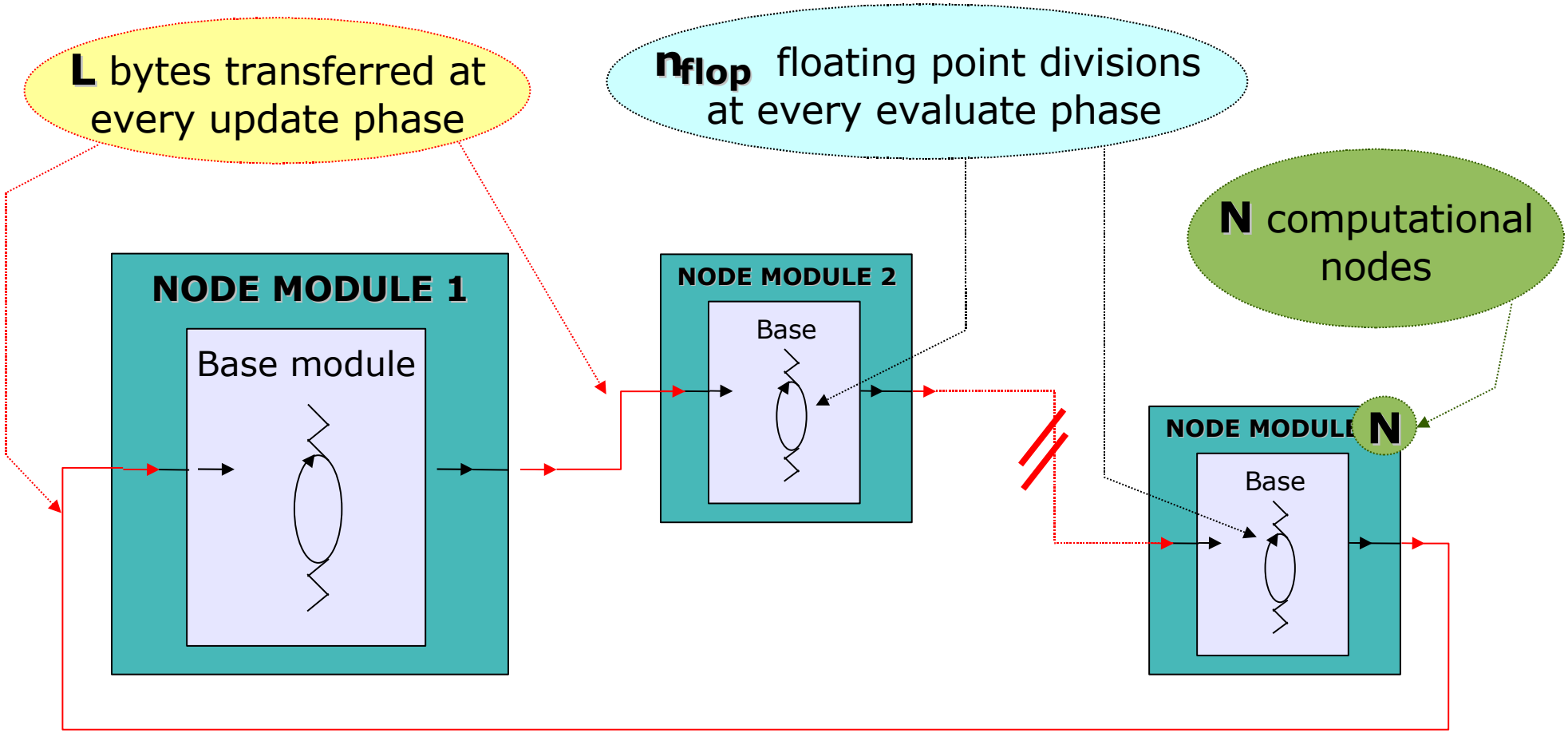
```
int sc_main(int ac, char *av[]) {
  sc_signal<bool>   clk;  //Clock

  sc_signal<double> in1;
  sc_signal<double> in2;
  sc_signal<double> sum;
  sc_signal<double> diff;
  sc_signal<double> prod;
  sc_signal<double> quot;
  sc_signal<double> powr;

  numgen N("numgen");
  N(in1,in2,clk);
  stage1 S1("stage1");
  S1(in1,in2,sum,diff,clk);
  stage2 S2("stage2");
  S2(sum,diff,prod,
     quot,clk);
  stage3 S3("stage3");
  S3(prod,quot,powr,clk);
  display D("display");
  D(powr,clk);

  sc_initialize();
  ... // Clock writes...
  return 0;
}
```




```
int sc_main(int ac, char* av[]) {
  sc_signal<bool>   clk;  //Clock, duplicated

  sc_signal<double> sum;
  sc_signal<double> diff;
  sc_signal<double> prod;
  sc_signal<double> quot;

  node1 N1("node1");
  N1(sum,diff,clk);
  node2 N2("node2");
  N2 << sum,diff,prod,
        quot,clk;
  node3 N3("node3");
  N3 << prod,quot,clk;

  sc_initialize();
  for(int i=0; i<50; i++){
    clk.write(1);
    sc_cycle( 10 NS );
    clk.write(0);
    sc_cycle( 10 NS );
  }
  return 0;
}
```

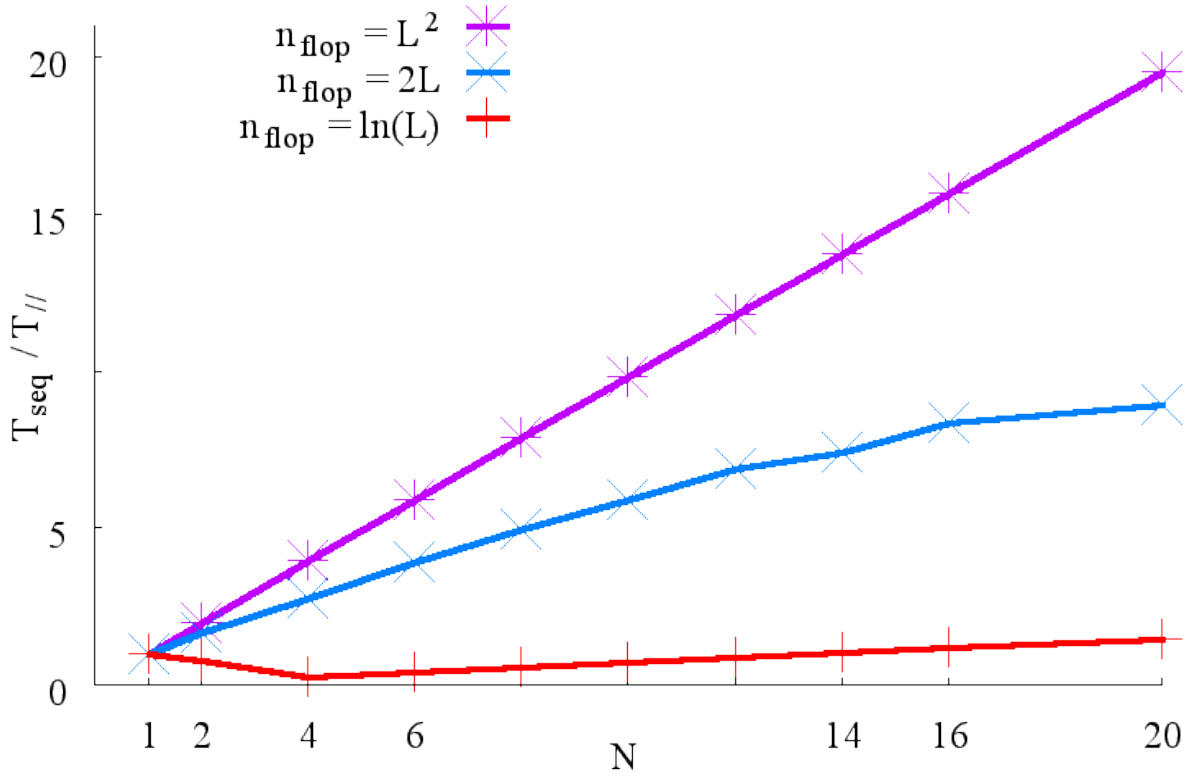


Validation – Test case



$$T_{seq} = n_\delta NP\left(\frac{6}{\gamma}L + \frac{n_{flop}}{\alpha}\right) \qquad T_{//} = n_\delta\left[\frac{6}{\gamma}LP + \frac{L}{\beta} + \frac{n_{flop}}{\alpha}P + t_{st}\right]$$

Speedup



When communication load << CPU load, almost perfect speedups are reached.

Performance on an industrial application

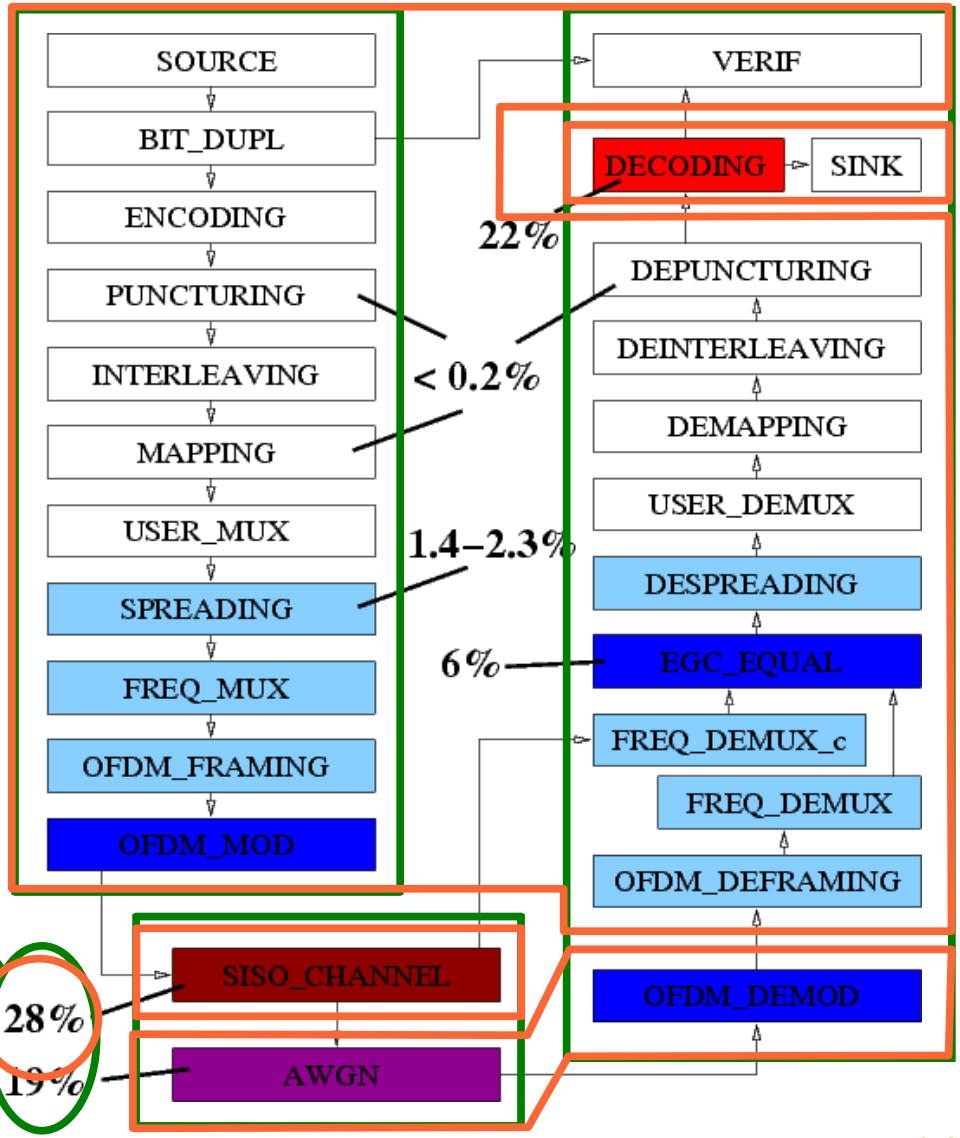
IST MATRICE project

- 26000 lines of C++ code
- 26 modules
- 26 FIFO channels

Speedups:

1.92 on 3 CPUs
(theoretical max : 2.1)

3.07 on 4 CPUs
(theoretical max. and absolute limit: 3.6)



Current status

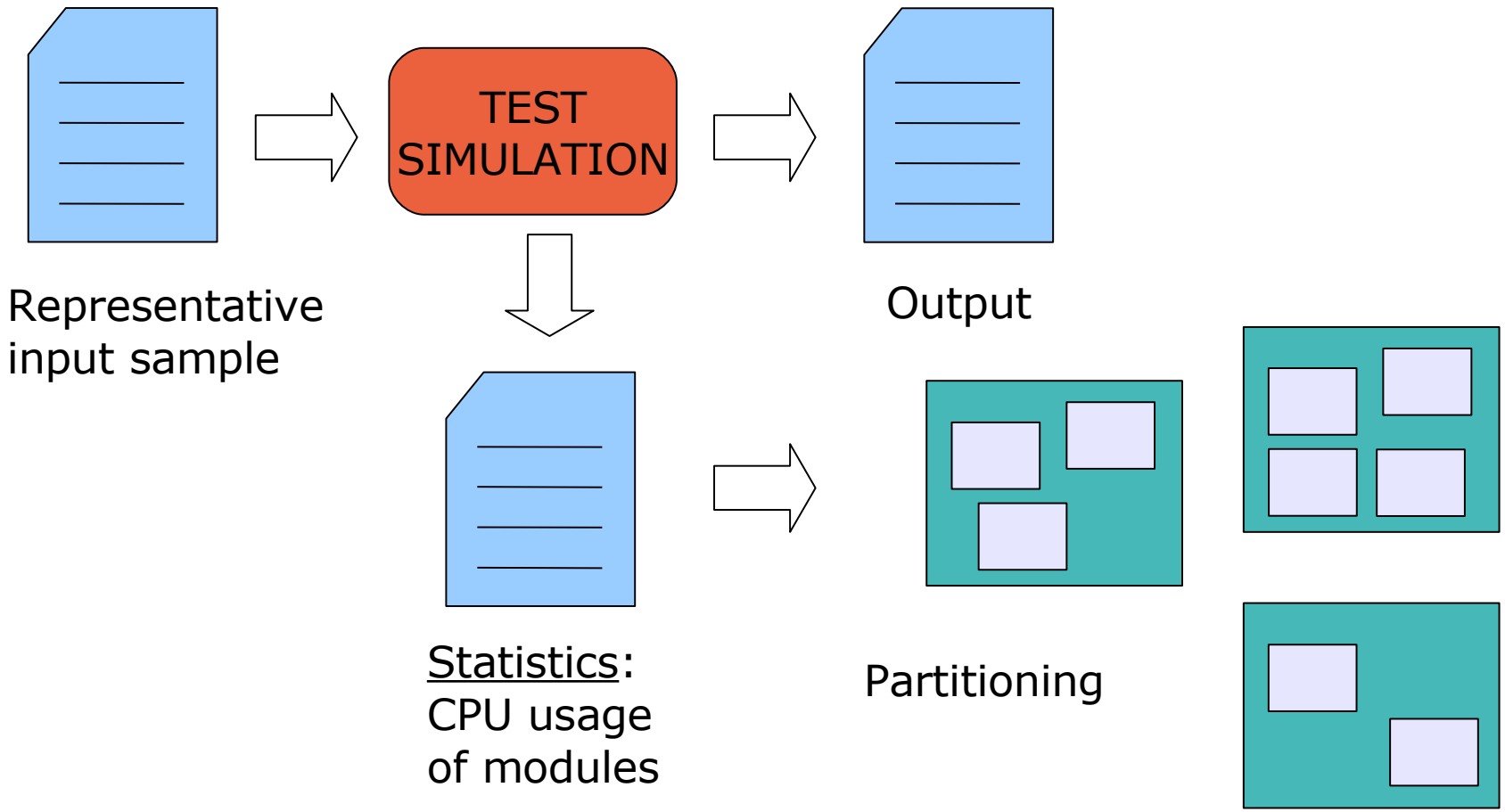
- **Achievements**
  - Open source parallel version of the OSCI SystemC 2.0.1 kernel
    (sequential version available in same source archive)
  - Support for most of SystemC 2.0.1 features
  - Designed for clusters – usable on SMP's
  - Easy parallelization of existing models
  - Reasonable performance results for well-balanced coarse-grained models.

Current status (cont'd)

- **Limitations**
  - Synchronization overhead
  - No dynamic load-balancing/partitioning
  - Two processes from two different node modules cannot interact via immediate events
    - ⊗ Node modules cannot be connected through most hierarchical channels, sc_semaphore and sc_mutex primitive channels
  - So far, sc_fx* types (only) cannot be used for "networked" channels
  - sc_stop might not behave as in sequential

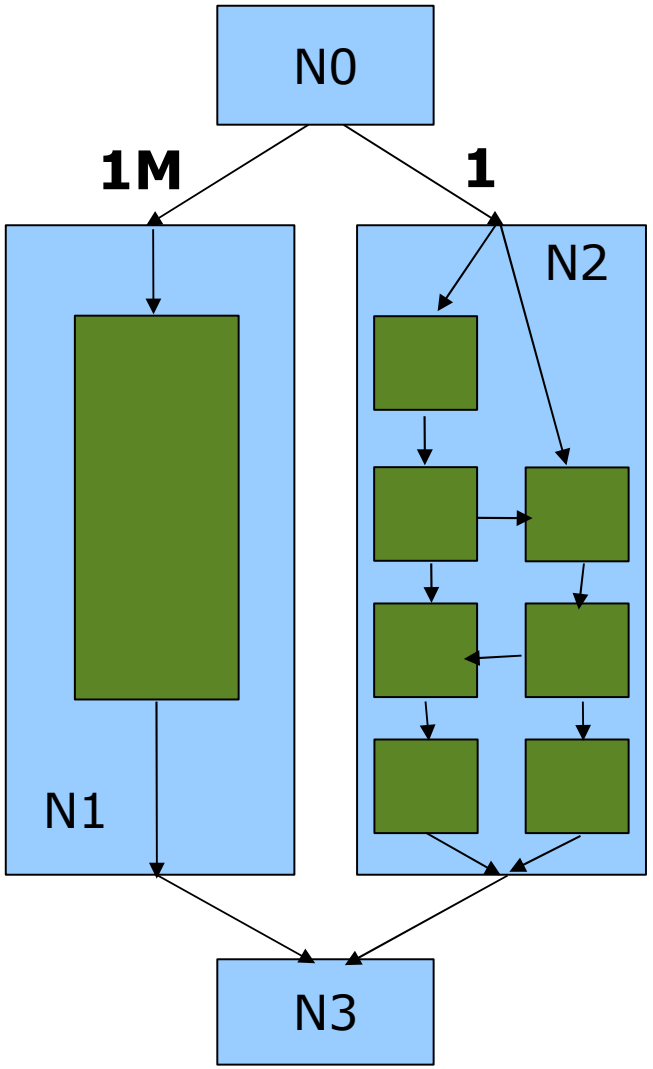
Future work

- Automating the static partitioning



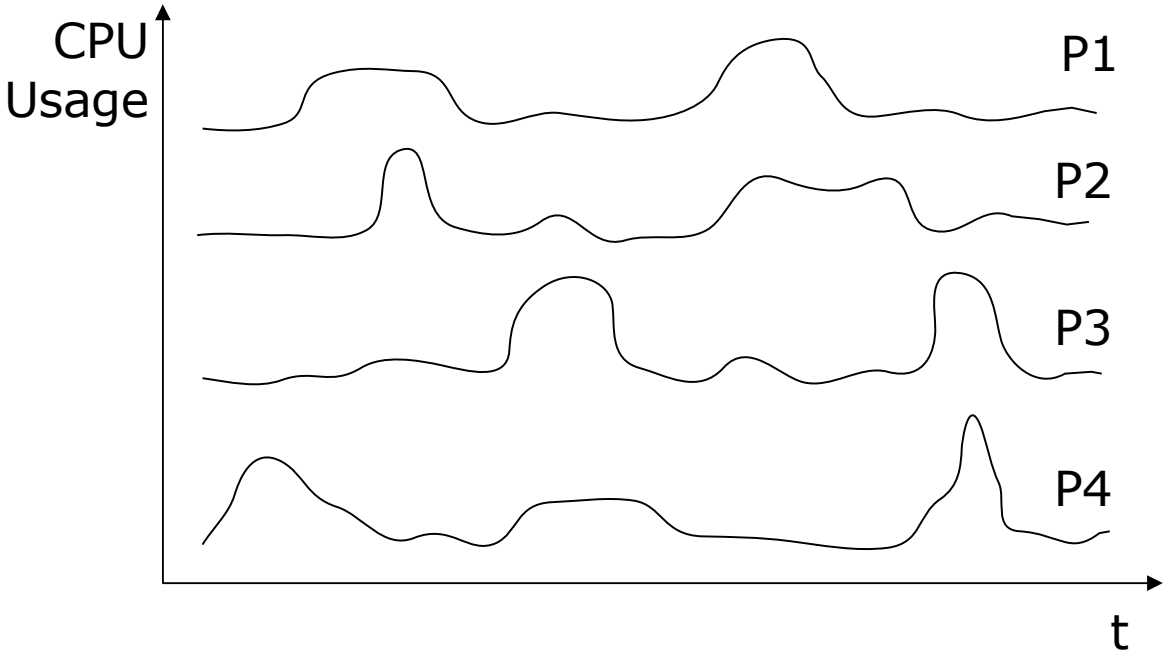
Future work (2)

- N1 and N2 process same data with different granularities. They are independent
- So far: synchronization at every delta-cycle to check for emptiness of all runnable stacks
- Future: change end detection algorithm so that N2 can process 1M delta-cycles ahead.



Future work (3)

- Dynamic load balancing
  - P1,...,P4 have roughly the same CPU usage average
  - but {P1,P3}{P2,P4} is a better partition than {P1,P2}{P3,P4}
  - need dynamic detection



➢ Migrating coarse grain processes can be VERY costly with regards to implementation and performance

Thank you
for your attention !

Contact me: **Philippe.Combes@cui.unige.ch**