# (Portable) Task-based programming model for elastodynamics

*Hoscar workshop - Sophia Antipolis, September 21-24 2015*

**Julien Diaz – INRIA Magique3D, Pau**



- Hélène BARUCQ, Lionel BOILLOT – INRIA Magique3D, Pau
- Henri CALANDRA – TOTAL E&P, Houston
- Emmanuel AGULLO – INRIA Hiepacs, Bordeaux
- George BOSILCA – ICL, University of Tennessee

# Heterogeneity is everywhere

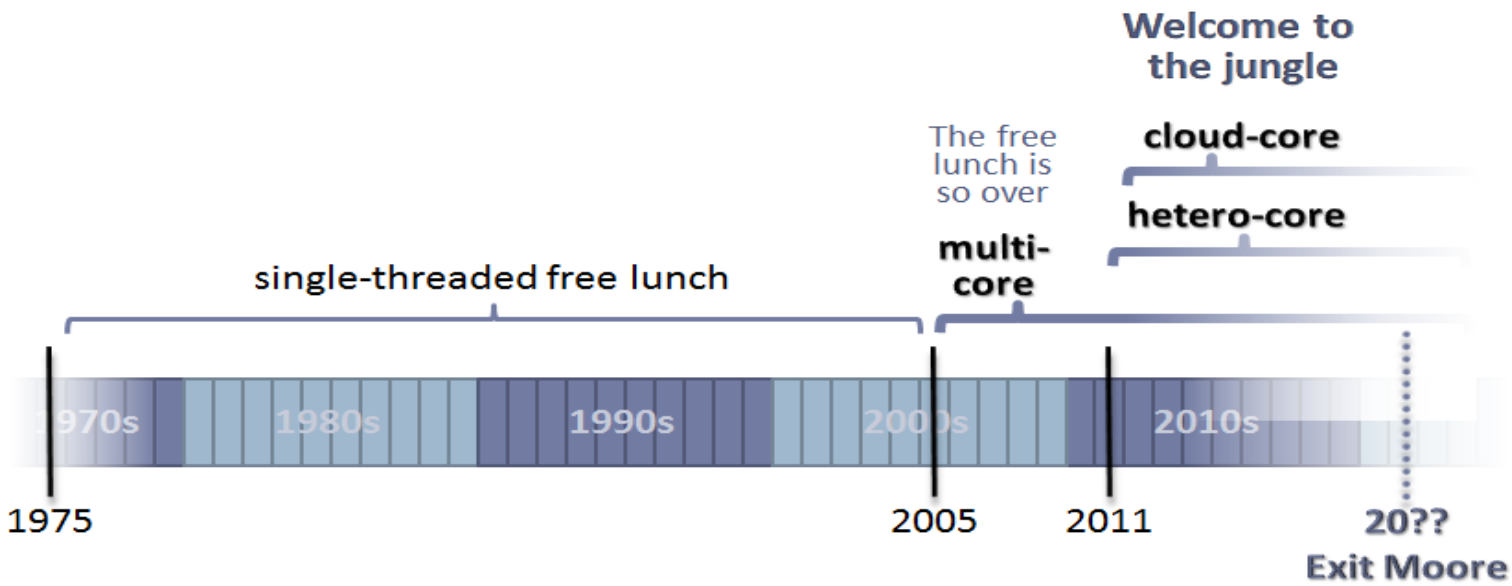



Figure : "*Welcome to the jungle*", Herb SUTTER, 2012

- Architectures: Xeon, Cell, MIC, KALRAY, GPU, FPGA, ...
- Memories: RAM, caches, (cc)NUMA, ...
- Programmation: MPI, CUDA/OpenCL, OpenACC, ...

$\Rightarrow$ need of efficient portable programming model, easy to implement!

# Table of contents

# Table of contents

# Discretization

## Elastic wave equation (first order)

$$\begin{cases} \rho(\mathbf{x})\partial_t v(\mathbf{x},t) &= \nabla.\underline{\underline{\sigma}}(\mathbf{x},t) \\ \partial_t \underline{\underline{\sigma}}(\mathbf{x},t) &= \underline{\underline{C}}(\mathbf{x}) : \underline{\underline{\epsilon}}(v(\mathbf{x},t)) \end{cases} \tag{1}$$

## Discontinuous Galerkin with Leap-frog scheme

Iteration on $n$:

$$\begin{cases} M_v \dfrac{v_h^{n+1} - v_h^n}{\Delta t} + R_{\underline{\underline{\sigma}}} \underline{\underline{\sigma}}_h^{n+1/2} &= 0 \\ M_{\underline{\underline{\sigma}}} \dfrac{\underline{\underline{\sigma}}_h^{n+3/2} - \underline{\underline{\sigma}}_h^{n+1/2}}{\Delta t} + R_v v_h^{n+1} &= 0 \end{cases} \tag{2}$$

$M_v$ and $M_{\underline{\underline{\sigma}}}$ block-diagonal matrices $\Rightarrow$ easily invertible!

# Parallel algorithm

**Algorithm 1:** DIP parallel

**Data**: $N_p, N_h, \Delta_t, N_t$

**Result**: $v_h$, $\underline{\underline{\sigma}}_h$

$N_{h_{loc}} \leftarrow$ DomainDecomposition($N_p$);

$[v_h^1, \underline{\underline{\sigma}}_h^{3/2}] \leftarrow$Initialization($N_{h_{loc}}, \Delta_t$);

**for** $n = 1..N_t$ **do**

  $\underline{\underline{\sigma}}_{h_{\overline{K}}}^{n+1/2} \leftarrow$ Communication($\underline{\underline{\sigma}}_{h_K}^{n+1/2}$);

  **for** $K = 1..N_{h_{loc}}$ **do**

    $v_{h_K}^{n+1} \leftarrow$ UpdateVelocity($v_{h_K}^n, \underline{\underline{\sigma}}_{h_{\overline{K}}}^{n+1/2}, \Delta_t$);

  **end**

  $v_{h_{\overline{K}}}^{n+1} \leftarrow$ Communication($v_{h_K}^{n+1}$);

  **for** $K = 1..N_{h_{loc}}$ **do**

    $\underline{\underline{\sigma}}_{h_K}^{n+3/2} \leftarrow$ UpdateStress($\underline{\underline{\sigma}}_{h_K}^{n+1/2}, v_{h_{\overline{K}}}^{n+1}, \Delta_t$);

  **end**

**end**

# MPI version = one domain per process

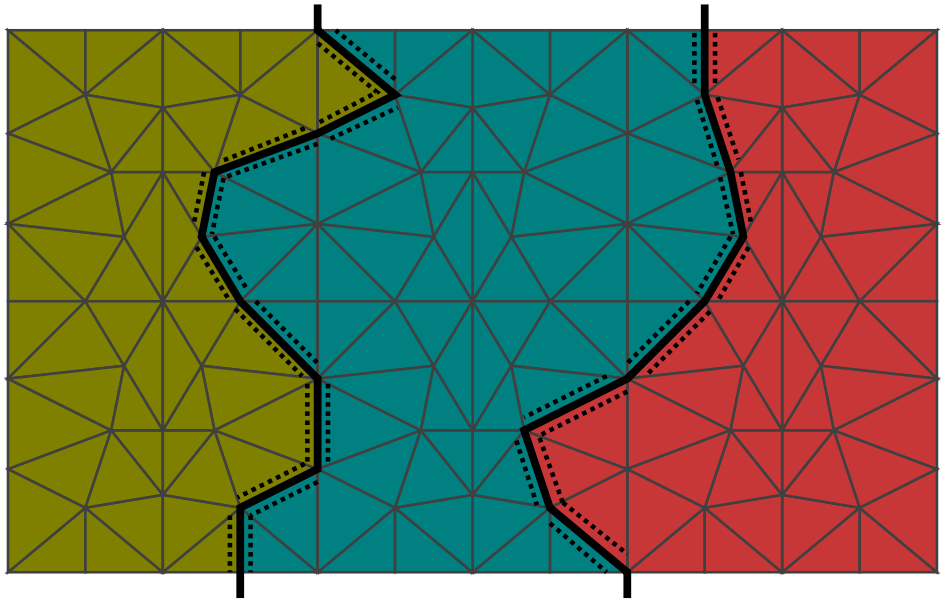


Figure : Domain decomposition example

**Load balancing limitations:**

- order of (discretization of) each cell
- order of the neighbor cells (for the face terms)
- architecture specifications (e.g. SIMD, cache size, . . . )

# Load imbalance example

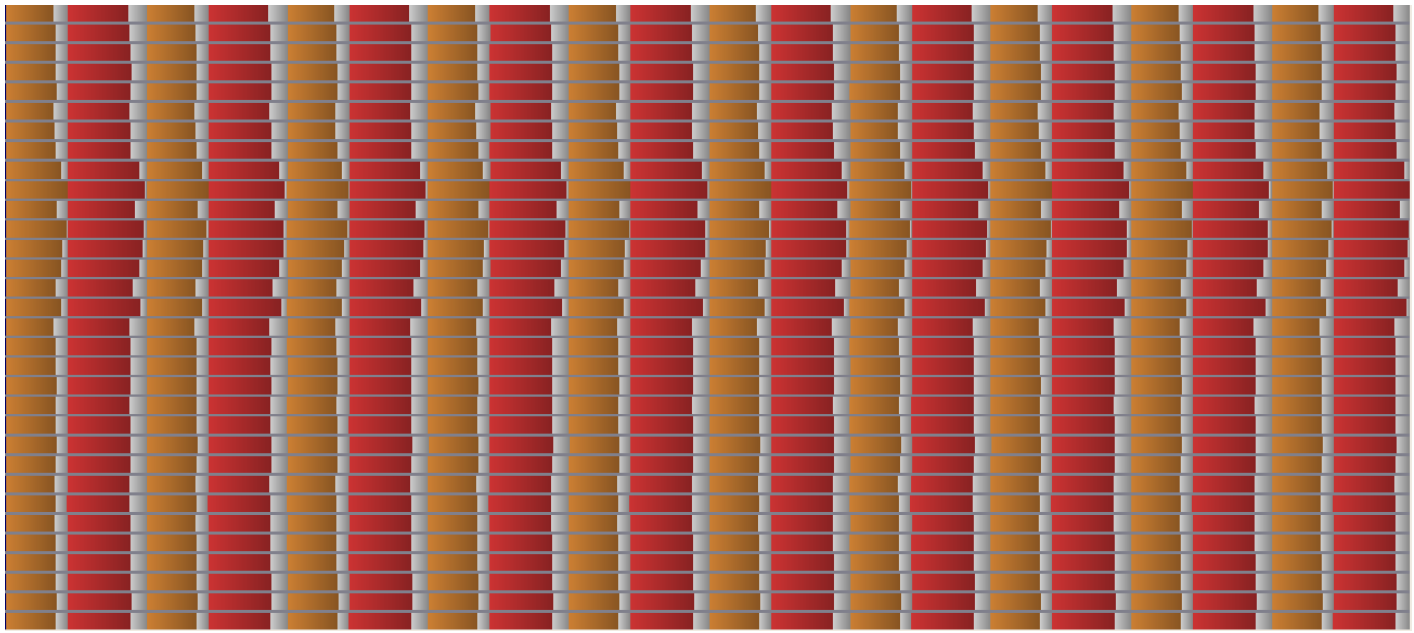


Figure : Trace analysis of 10 timesteps on 32 MPI processes

Each line represents a core activity during the execution time:

- ORANGE – computation of the velocity field
- RED – computation of the stress tensor
- GREY – waiting time (synchronization)

# Table of contents

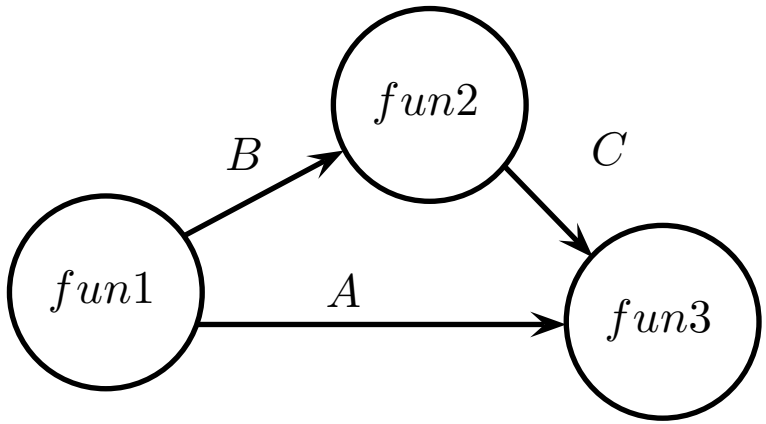
# Task-based programming

## Principles

- Determine dependencies between tasks with the Bernstein's conditions
- Specify the data modes: Read/Write

$\Rightarrow$ creation of a DAG (Direct Acyclic Graph)

*fun1(A: inout, B: out)*
*fun2(B: in, C:out)*
*fun3(A: inout, C: in)*




Original control-flow becomes useless (!) and an obstacle to efficiency

# DAG of DIP

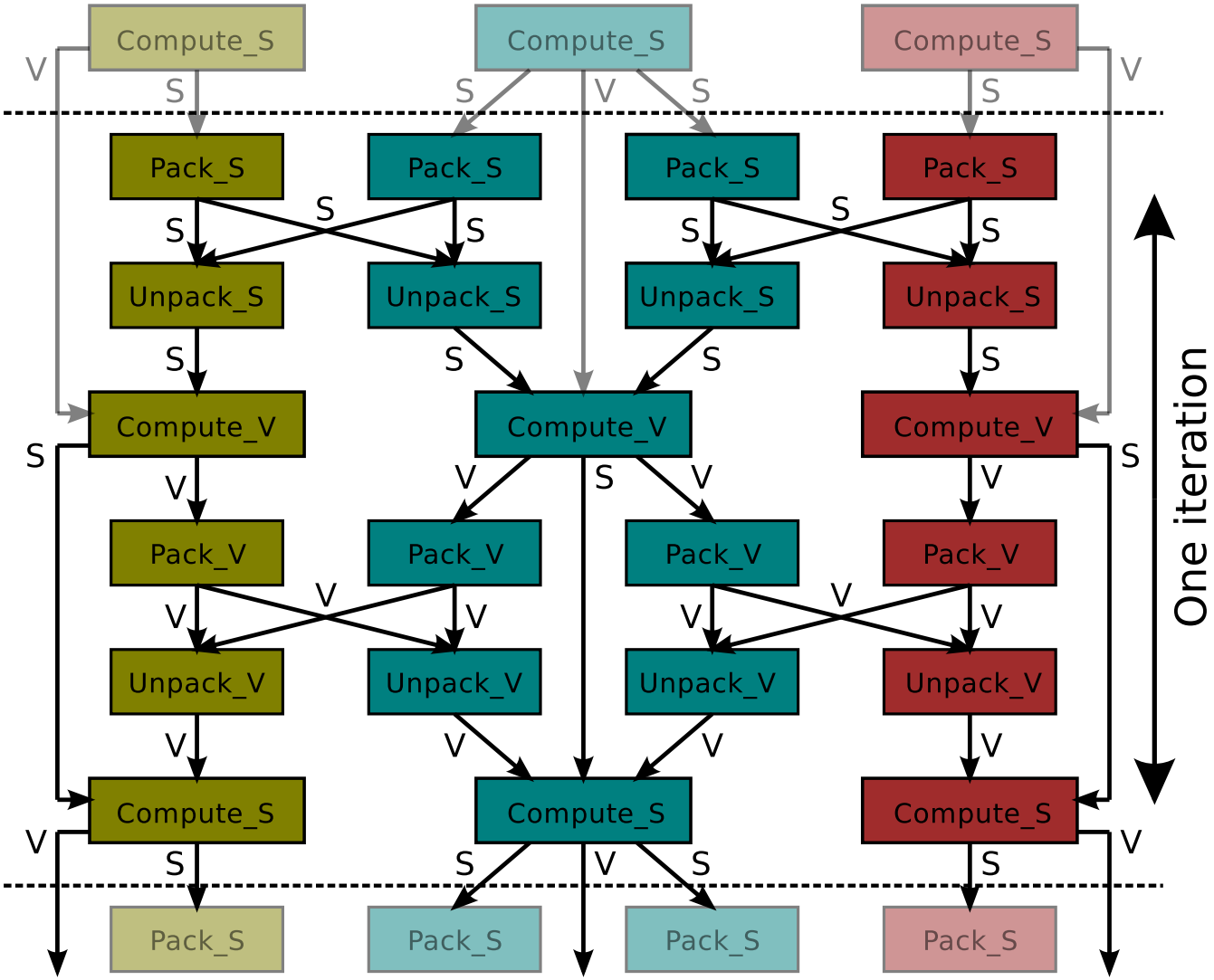



Figure : One iteration of DIP on three sub-domains

# Fine granularity

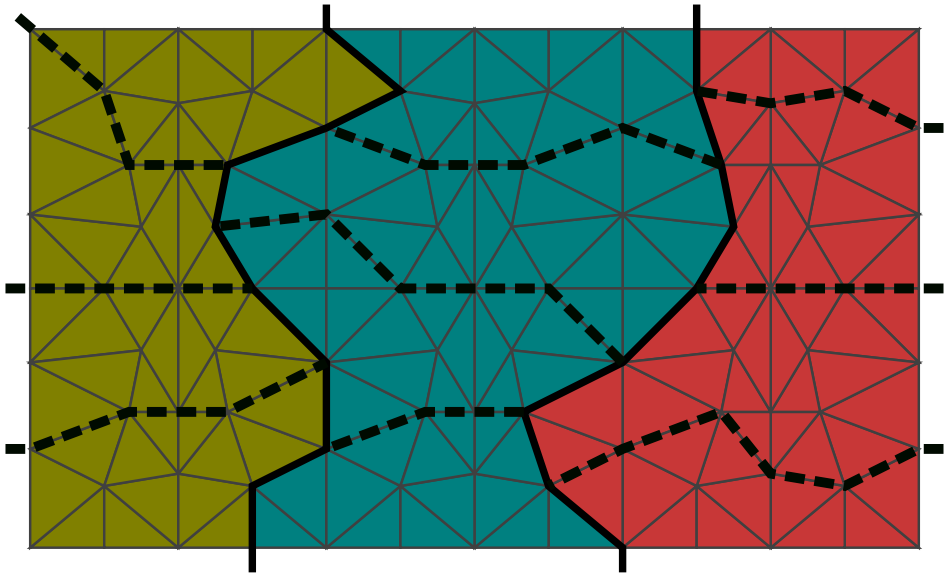


Figure : Subdivision example

## More than one domain per CPU

- exhibit deeper parallelism
- allow dynamic flexibility
- reduce the boundary size

# Table of contents

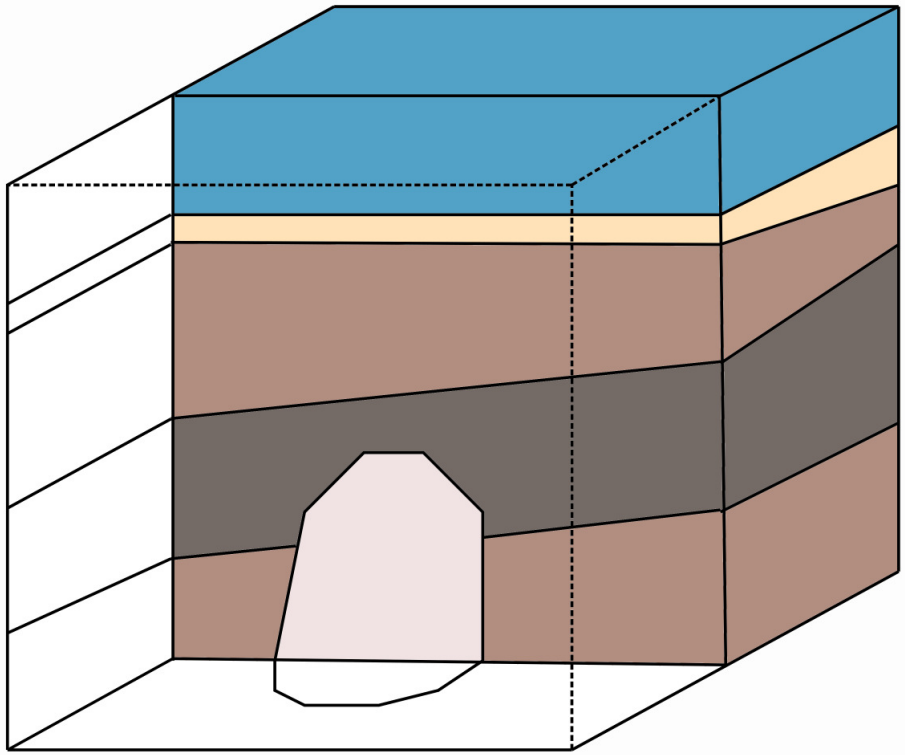
# Geophysics test case

Realistic test case:

- 3D elastic
- TTI (anisotropy)
- multi-layers

Hybrid discretization:

- unstructured tetrahedra
- P1-P2-P3 orders
- boundary conditions



# ccNUMA machine

- 6 processors (previous Intel Xeon E7-8837)
- Total of 48 CPU cores in ccNUMA architecture

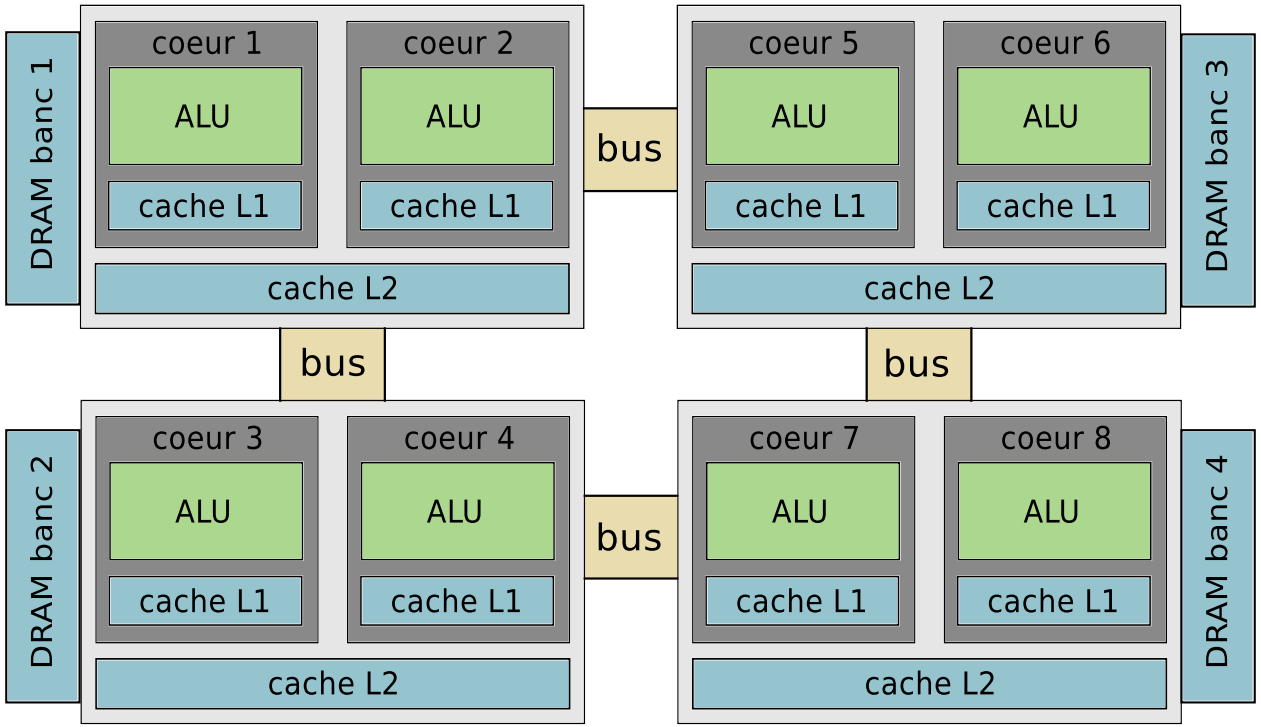



Figure : cache-coherent Non-Uniform Memory Access (ccNUMA) scheme

# ccNUMA results - speedup

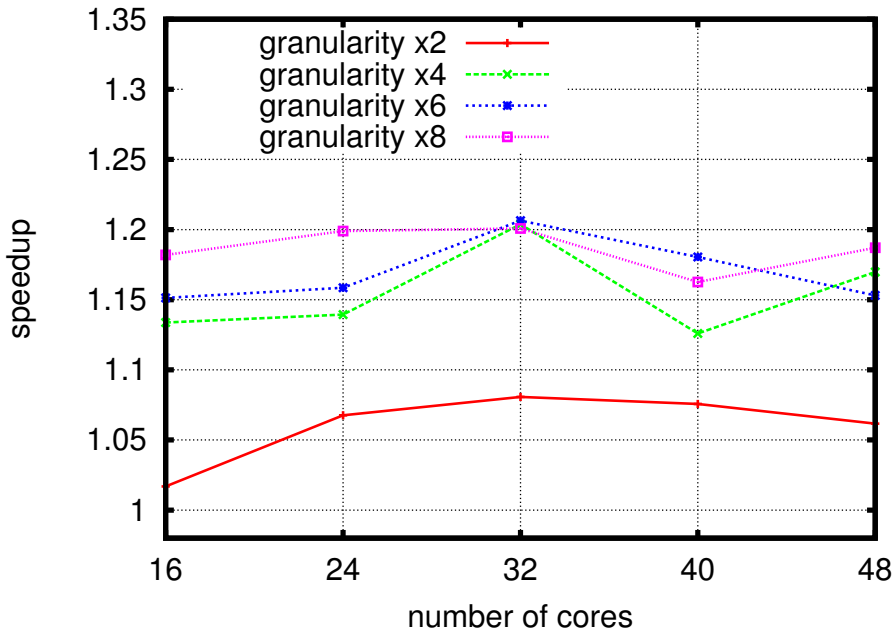



Figure : With multi-virtual processes

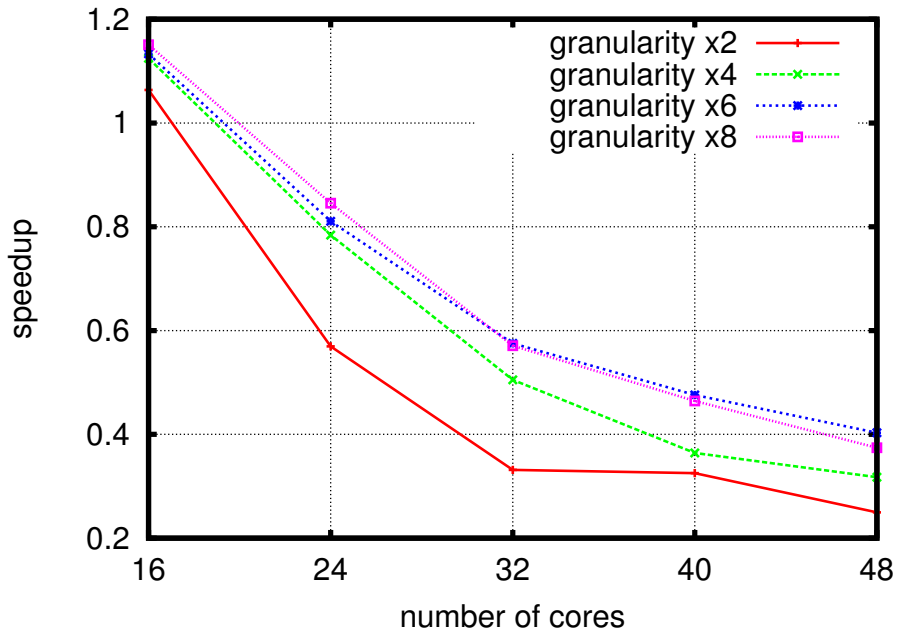



Figure : With no virtual process

⇒ virtual processes are key-point for NUMA-awareness

# Trace comparison

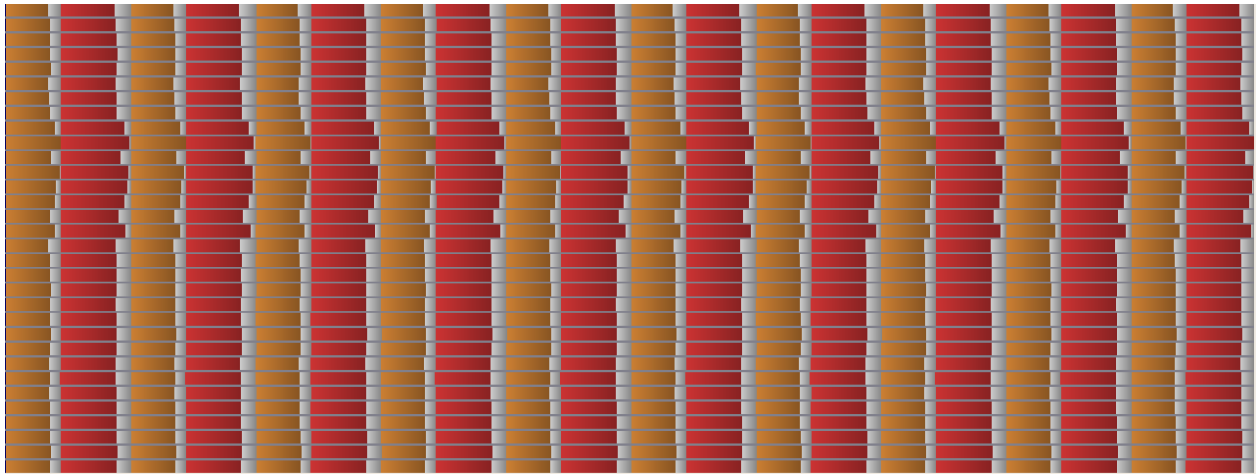


Figure : MPI-based $t = 2.517s$

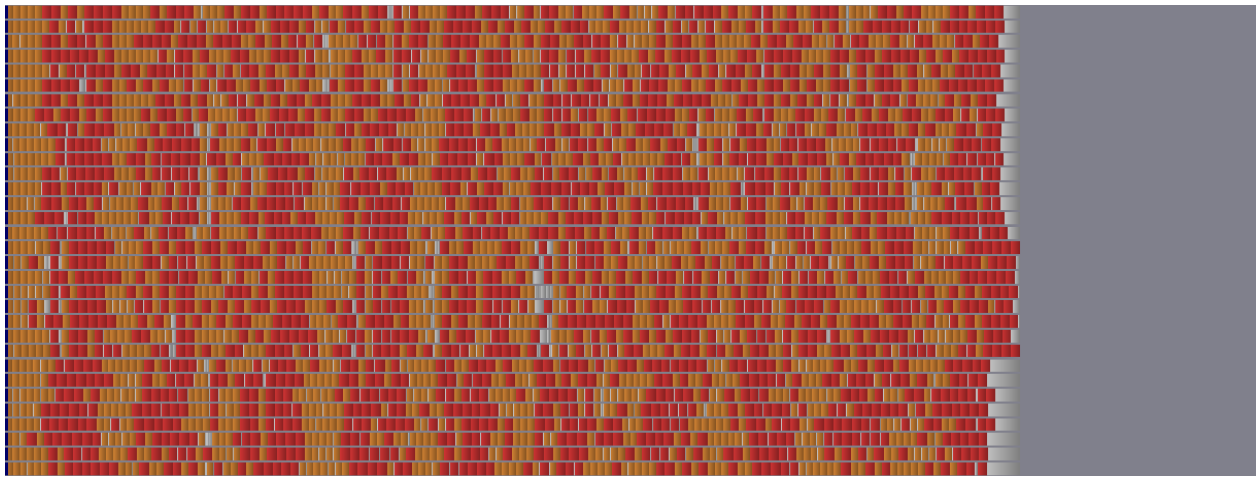
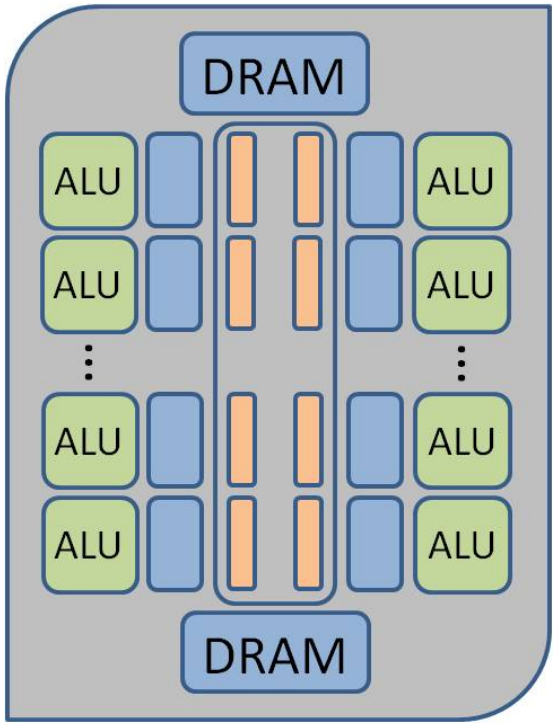


Figure : PaRSEC version (multi-VP, granularity x6) $t = 2.060s$

# Intel Xeon Phi configuration




MIC architecture
(Many-Integrated Cores)

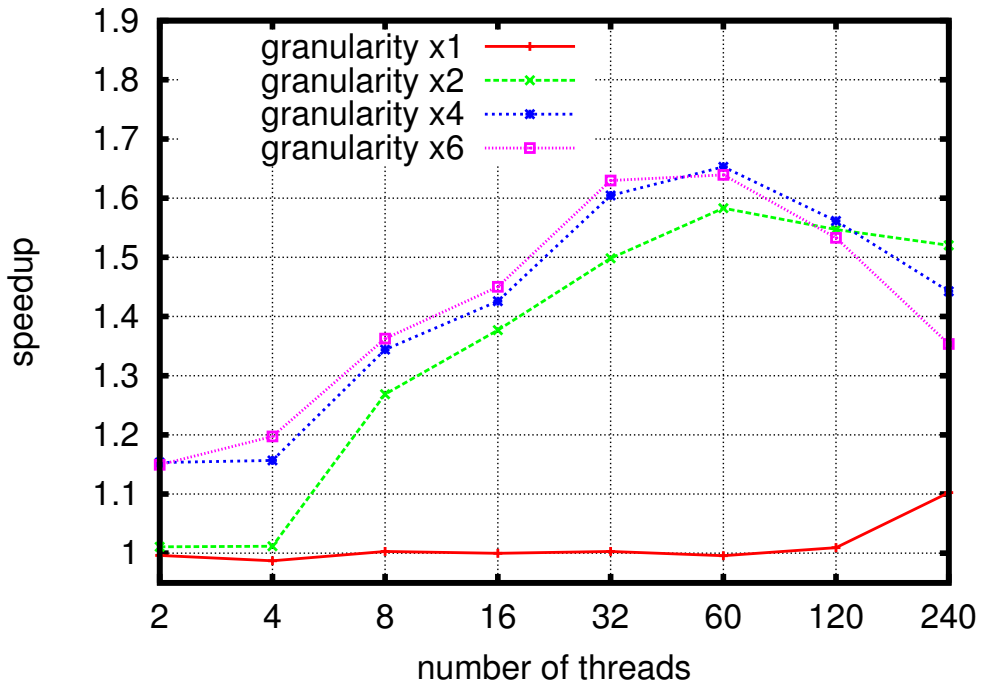
General specifications:

- AVX instruction set
- hypertheading by 4
- no L3 cache memory
- shared L2 (ring bus)

Co-processor 7120P:

- 61 CPUs at 1.238 GHz (244 threads)
- 64 KB of L1 cache (per core)
- 512 KB of L2 cache (per core)
- 16 GB of DRAM (scrappy)

# Intel Xeon Phi results - speedup




On the 60 cores:

- very good speedup
- **fine** granularity
- **good** efficiency

Hyperthreading enabled:

- quite good speedup
- **coarse** granularity
- **bad** efficiency

Idea $\rightarrow$ use hypertheading to manage load/store

# Conclusion and perspectives

## Context

- compute & exchange algorithm
- domain decomposition parallelism

## Key-points

- fine granularity and work-strealing
- virtual process capability of PaRSEC

## Results

Shared memory architectures (ccNUMA machine, Xeon Phi co-processor)
$\Rightarrow$ Next coming: distributed memory machines

Support by INRIA-TOTAL strategic action DIP (`http://dip.inria.fr`)