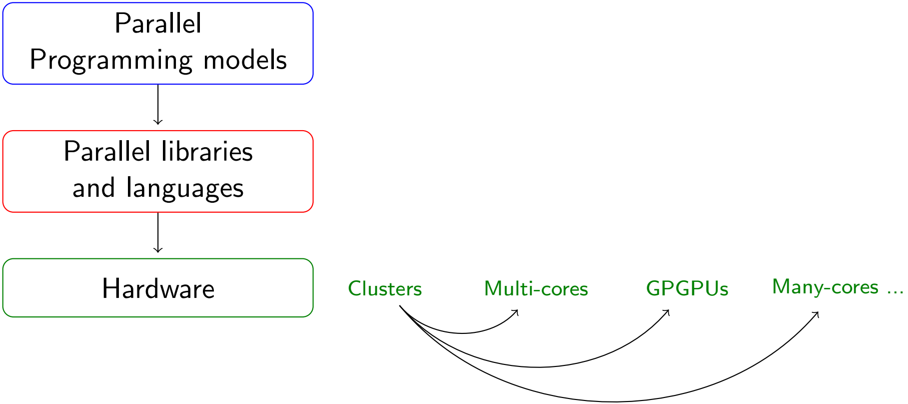
The Multi-Stencil Language

Hélène Coullon, Christian Perez and Julien Bigot

Inria Avalon, LIP Lyon

17^{th} March 2016
Journée Langages du LIP

High performance computing and parallelism in 2015



High performance computing and parallelism in 2015

Parallel Programming models

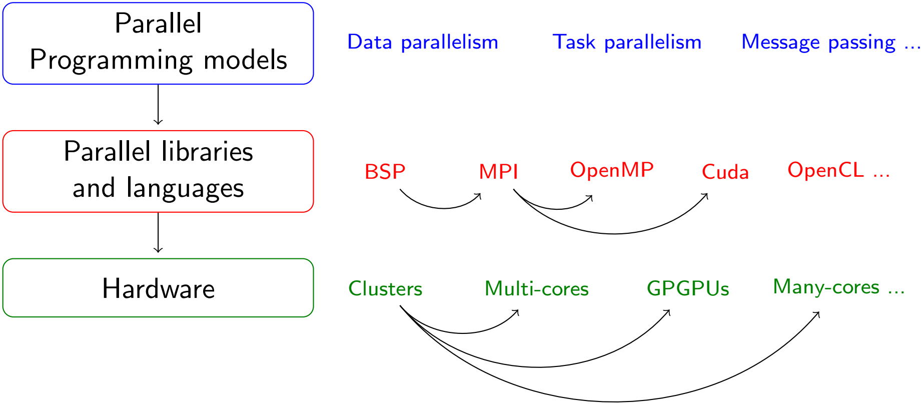
Data parallelism | Task parallelism | Message passing ...

Parallel libraries and languages

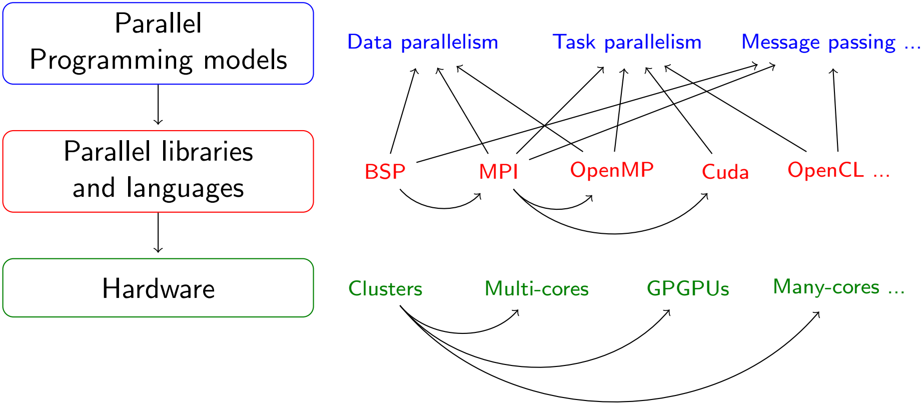
Hardware

Clusters | Multi-cores | GPGPUs | Many-cores ...

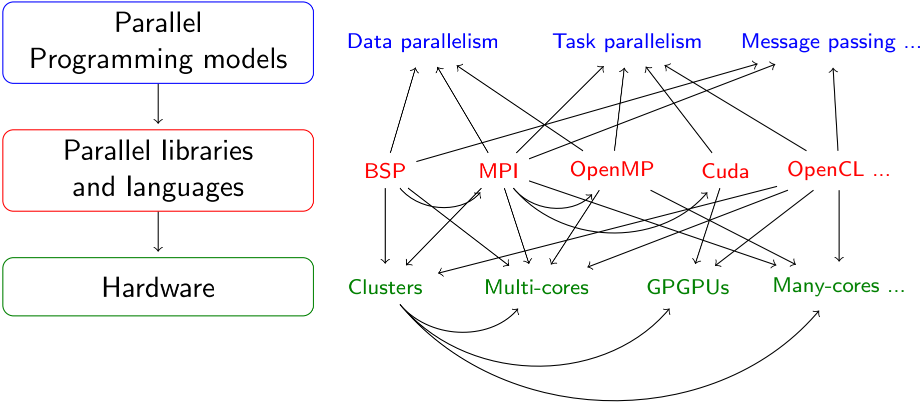
High performance computing and parallelism in 2015



High performance computing and parallelism in 2015



High performance computing and parallelism in 2015



Very complex domain !

"Please Help"

HPC

- Heterogenous and complex parallel hardware
- Many programming models to combine
- Expert programming

Users

Weather, climat, physics, biology etc.

Separation of concerns between domain/parallelization and optimization
Domain specific languages (DSL)

Contribution

From MSL

- Descriptive language,
- for Multi-Stencil simulations,
- without numerical code.

To

- Parallel pattern of the simulation,
- with automatic synchronizations for data and task parallelism,
- with empty functions to fill,
- with good performance.

Separation of concerns between domain/implementation/parallelization

Table of contents

Table of contents

What is a multi-stencil simulation ?

set of PDEs

$$\frac{\partial u(x,y,t)}{\partial t} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2}$$

Behavior of the phenomena

and boundary behaviors

Discretization

Mesh and time iterations

$+$

Numerical methods

Finite difference/volume/elements

Numerical schemes

Explicit = stencil

Quantity $\sigma(x, t)$ computed from quantities at (x, t) if different,

$(x, t-1)$ and $(y, t-1)$, where y is a neighborhood of x

Multi-Stencil algorithm

Create the mesh μ
Create the quantities to simulate mapped onto μ
Initialization of the quantities and parameters
Create the time step δt
Create the maximum time *tmax*
while $criteria(t) == true$ **do**
    for each element of the boundary **do**
        Numerical computations of boundary conditions
    end
    for each $x \in \mu$ **do**
        Compute stencils and auxiliary computations
    end
    $t = t + \delta t$
end

...

Definitions

Mesh

- *Mesh* : graph without bridges
- *Mesh entity* : sub-graph of the mesh
- *Group of mesh entities* : set of mesh entities of the same type
- *Computation domain* : sub-part of a group
- *Neighboorhood* : function from a mesh entity to a set of mesh entities (possibly from a different group than the input)

Quantities

- A *quantity* is mapped onto a group of mesh entities
- A *scalar* is global to the simulation and is typically a numerical value constant or not

Definitions

Time and computations

- *Time* : time step and convergence criteria
- *Computation* :
  - Set of scalar read
  - Set of quantities read, on which neighborhood
  - Quantity written, on which computation domain
  - A numerical expression

Four kinds of computations

- *Stencil computation* : it exists a quantity read with a neighborhood
- *Boundary computation* : it exists a quantity read equal to the quantity written with a neighborhood
- *Local computation* : for all quantity read, there is not neighborhood
- *Reduction* : the quantity written is a scalar

Wait a while

One can notice that

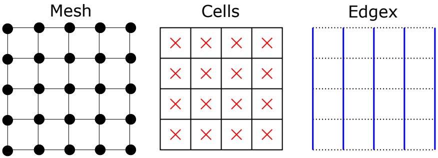
- Definitions independant of the mesh topology
- Definitions independant of the numerical expression of computations

The Multi-Stencil Language

- Descriptive language mesh-agnostic and without numerical code asked to the user
- Separation of concerns : description / implementation / parallelization

Table of contents

The MSL language : the mesh




```
mesh : cart
mesh entities : cell , edgex
computation domains :
  d1 in cell
  d2 in edgex
stencil shapes :
  ncc from cell to cell
  nce from cell to edgex
  nec from edgex to cell
```

- A cartesian mesh is used
- Two kind of mesh entities are declared onto it
- Two computations domains, one for each entity type
- Three stencil shapes (neighborhood from mesh entity to mesh entity)

The MSL language : quantity

```
quantity :
    A, cell
    B, cell
    C, edgex
    D, cell
    E, cell
    F, cell
    G, cell
    H, edgex
    I, cell
    J, cell
scalar : mu, tau
```

- A is a quantity applied onto cell
- C is a quantity applied onto edgex
- mu and tau are scalar values

The MSL language : time and computations

```
time : 500
computations :
  B[d1] = k0({tau},{A})
  C[d2] = k1({},{B[nce]})
  D[d1] = k2({},{C})
  E[d1] = k3({},{C})
  F[d1] = k4({},{D,C[nec]})
  G[d1] = k0({mu,tau},{E})
  H[d2] = k6({},{F})
  I[d1] = k7({},{G,H})
  J[d1] = k8({mu},{I[ncc]})
```

- The time loop is composed of 500 iterations.
- J is written
- onto the computation domain d1,
- by the computation k8,
- which read the scalar mu
- and the quantity I,
- which is accessed with the neighborhood ncc.

Table of contents

Synchronizations

Application = *Quantity* + *Program*
Data parallel Application = *Split quantity* + *Program* + *Synchronizations*
$\rightarrow$ Find where synchronizations are needed

Synchronizations

- For stencil computations (point to point synchronization) if for k_i and k_j, with k_j a stencil, $R_j \cap \{w_i\} \neq \emptyset$
- For reduction computations (global synchronization)

$\Gamma = [k_0, k_1, k_2, k_3, k_4, k_0, k_6, k_7, k_8]$
$\Gamma_{sync} = [k_0, k^{sync}_{0;1}, k_1, k_2, k_3, k^{sync}_{1;4}, k_4, k_0, k_6, k_7, k^{sync}_{7;8}, k_8]$
Γ_{sync} **is applied on each resource !**

Dependencies

Program = sequence of tasks

Task parallel Program = schedule of parallel and sequential tasks

$\rightarrow$ Define dependency relations

Read after write dependency

- for k_i and k_j, with $i < j$
- if $R_j \cap \{w_i\} \neq \emptyset$, and domain computations intersect
- k_i has to be computed before k_j

Write after write dependency

- for k_i and k_j, with $i < j$
- if $w_j = w_i$, and domain computations intersect
- k_i has to be computed before k_j

Dependency graph

From read/write and write dependencies is built Γ_{dep}
Γ_{dep} represents a **single time iteration**

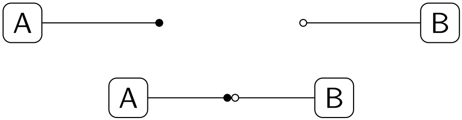
$$
\begin{tikzcd}
 & & & k^{sync}_{1;4} \arrow[dr] & & & & & \\
 & & & k_2 \arrow[r] & k_4 \arrow[r] & k_6 \arrow[dr] & & & \\
k_0 \arrow[r] & k^{sync}_{0;1} \arrow[r] & k_1 \arrow[uur] \arrow[ur] \arrow[dr] & & & & k_7 \arrow[r] & k^{sync}_{7;8} \arrow[r] & k_8 \\
 & & & k_3 \arrow[r] & k_5 \arrow[urr] & & & &
\end{tikzcd}
$$

Γ_{dep} **is applied on each resource !**

Table of contents

Component models

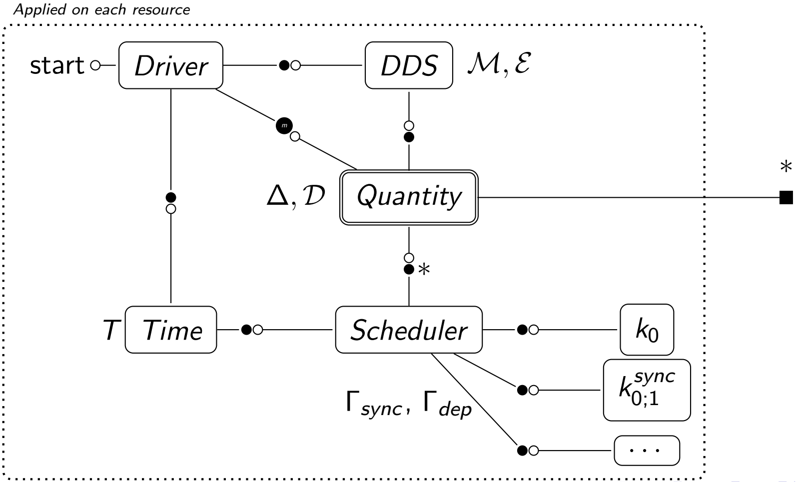
- Black box with code
- Use/provide interfaces
- Improve code reuse (productivity)
- Improve separation of concerns (maintainability and portability)
- Improve composability of applications



Components A and B into a component assembly

What is produced ?

A data parallel/task parallel component assembly



What is produced ?

Implementation

- DDS + Quantity + k_*^{sync} = written components using SkelGIS (distributed memory, MPI)
- Scheduler = Serie-Parallel graph of Γ_{dep} (OpenMP 3.0)
- Time + Scheduler = generated components
- k_i = empty components to write

Separation of concerns

- numerician (MSL)
- computer engineer (K)
- HPC engineer (DDS, quantity, k_*^{sync} etc.)
- DSL designer (component assembly + generated components)

Table of contents

Evaluations

Application

- FullSWOF2D : developed at the MAPMO, Université d'Orléans
- Solve the shallow-water equations using a finite volume method
- Cartesian mesh
- 3 mesh entities, 7 computation domains, 48 quantities
- 98 computations (32 stencils, 66 local kernels)

Evaluations

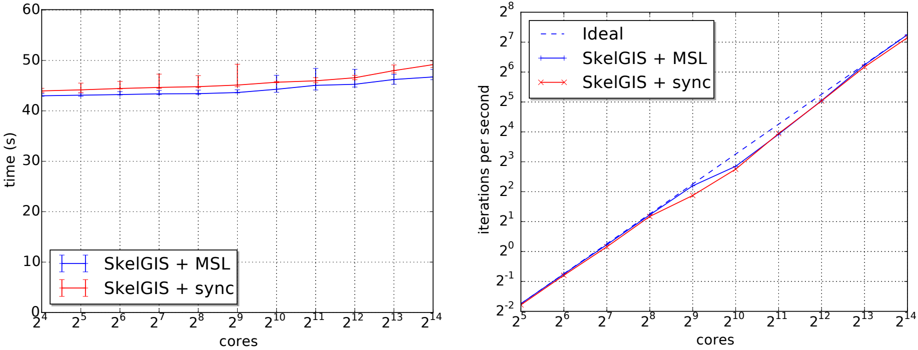
Evaluations

- Data parallelization : no overhead
  - weak scaling, fix size 400x400 (16.384 cores)
  - strong scaling, 10kx10k :1k (16.384 cores)
- Hybrid parallelization : increase performance
  - when data parallelism reaches its limits
  - introduce new parallelism

Machines

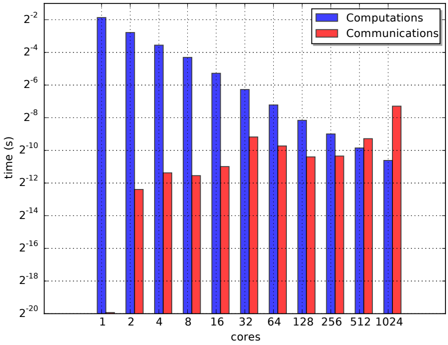
- Thin nodes TGCC Curie
- 2 CPU, 8-cores, Intel Sandy Bridge EP (E5-2680) 2.7 GHz, 64 Go
- OpenMPI and gcc 4.9

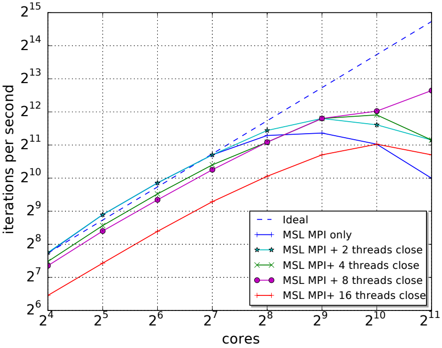
Evaluations



No overhead introduced by components

Evaluations





Good performance with 8 threads

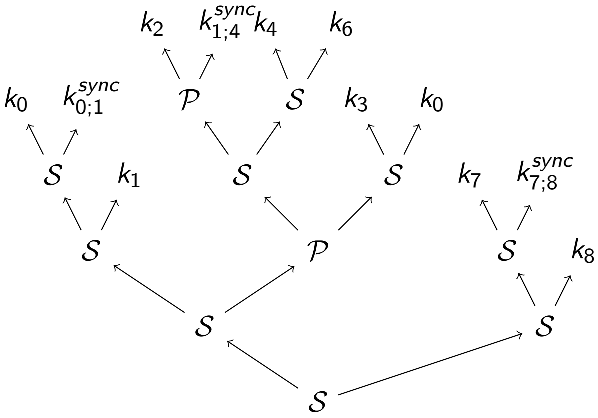
Table of contents

Perspectives

- Other schedulers (OpenMP 4, KStar)
- Entire dependency graph
- Other DDS + quantities + Sync (Global Arrays)
- Other hardware (GPGPUs, MIC)
- More than one mesh, multi-physics simulations
- Unstructured meshes implementation (PamPA LaBRI + application)
- Show interest of components

Thank You !

From the dependency graph to a static scheduling



From the dependency graph to a static scheduling

To build a static scheduling we build a **Serie-Parallel tree decomposition**

1. Transitive reduction
2. Remove the forbidden N-Shapes
3. Tree decomposition