



# Multilevel Programming Paradigms for Exascale Computing

Serge G. Petiton

[serge.petiton@lifl.fr](mailto:serge.petiton@lifl.fr)



MAISON DE LA SIMULATION

# Outline

- Introduction
- Multilevel programming paradigms
- The YML environment and experiments
- YML/XMP/starPu and the Japanese-French FP3C project
- Reusable library, experiment with PETSc, SLEPc and YML
- Conclusion

# Outline

- **Introduction**
- Multilevel programming paradigms
- The YML environment and experiments
- YML/XMP/starPu and the Japanese-French FP3C project
- Reusable library, experiment with PETSc, SLEPc and YML
- Conclusion

# The Future Exaflop barrier : not only another symbolic frontier coming after the Petaflops

- Sustained Petascale applications on an unique computer exist since a few months
- Gordon Bell award : more than 3 sustained petaflops
- Next frontier : **Exascale** computing (and how many MWatts???)
- Nevertheless, many challenges would emerge, probably before the announced 100 Petaflop computers and beyond.
- We have to be able to **anticipate solutions** to be able to educate scientists as soon as possible to the future programming.
- We have to use the existing emerging platforms and prototype to imagine the future language, systems, algorithms,....
- We have to propose **new programming paradigms** (SPMD/MPI for 1 million of cores and 1 billions of threads????), MPI-X or X-MPI or Z-MPI-X???
- We have to propose **new languages**.
- Co-design and **domain application languages and/or high level multi-level language** and frameworks,.....

# New methods for future *hypercomputers*

- We have to imagine new methods for the exascale computers
- Methods would define new architectures (co-design), not the old and present methods....
- Many people propose new systems and language starting from the existing methods and numerical libraries .... but they were developed for MPI-like programming and only SPMD paradigm, and at the “old time” of the Moore law.
- We have to adapt the methods with respect to criteria from the architecture, the arithmetic, the system, the I/O, the latencies,..... **then, auto-tuning (at runtime) is becoming a general approach.**
- We have to **hybridize numerical methods** to solve large scientific applications, asynchronously, and each of them have to be auto-tuned,
- We have to find smarter criteria, some of them at the application and at the mathematical level, for each method : **smart tuning**
- These auto-tuned methods will be correlated : **intelligent numerical methods.** **End-Users have to be able to give expertise.**

***Our research goals are to propose a framework and  
programming paradigm for Xtrem computing and to introduce  
well-adapted modern smart-tuned hybrid (Krylov) methods***

# Outline

- Introduction
- Exascale challenges
- **Multilevel programming paradigms**
- The YML environment and experiments
- YML/XMP/starPu and the Japanese-French FP3C project
- Reusable library, experiment with PETSc, SLEPc and YML
- Conclusion

# Toward graph of tasks/components computing

- Communications have to be minimized : but all communications have not the same costs, in term or energy and time.
- Latencies between farther cores will be very time consuming : global reduction or other synchronized global operations will be really a bottleneck.
- We have to avoid large inner products, global synchronizations, and others operations involving communications along all the cores. Large granularity parallelism is required (cf. CA technics and Hybrid methods).
- Graph or tasks/components programming allows to limit these communications only between the allocated cores to a given task/ components.
- Communications between these tasks and the I/O may be optimized using efficient scheduling and orchestration strategies(asynchronous I/O and others)
- Distributed computing meet parallel computing, as the future super(hyper) computers become very hierarchical and as the communications become more and more important. Scheduling strategies would have to be developed.

# Toward graph of tasks/components computing and other computing levels

- Each task/component may be an existing method/software developed for a large part of the cores, but not all of them (then classical or CA methods may be efficient)
- The computation on each core may use multithread optimizations and runtime libraries
- Accelerator programming may be optimized also at this level.
- Then we have the following levels of programming and computing :
  - Graph of components, already developed or new ones,
  - Each component is run on a large part of the computer, on a large number of cores
  - On each processor, we may program accelerators,
  - On each core, we have a multithread optimization.
- In terms of programming paradigms, we propose : Graph of task (Data flow oriented)/SPMD or PGAS-like or.../Data parallelism
- We have to allow the users to give expertise to the middleware, runtime system and schedulers. Scientific end-users have to be the principal target of the co-design process. Frameworks and languages have to consider them first.

# Outline

- Introduction
- Multilevel programming paradigms
- **The YML environment and experiments**
- YML/XML/starPu and the Japanese-French FP3C project
- Reusable library, experiment with PETSc, SLEPc and YML
- Conclusion

# YML

yml.prism.uvsq.fr

Many people were involved in the development of the open source YML software : from Olivier Delannoë (master internship, Univ. Paris 6, 2000) to the today developers : in particular Miwako Tusji (Tsukuba), Makarem Dandouna (Univ. Versailles) and Maxime Hugues (INRIA).

## Main properties :

- High level graph description language
- Independent of Middleware, hardware and libraries
- A backend for each systems or middleware
- Expertise may be proposed by end-users
- May use existing components / thought eventually libraries

## Some elements on YML

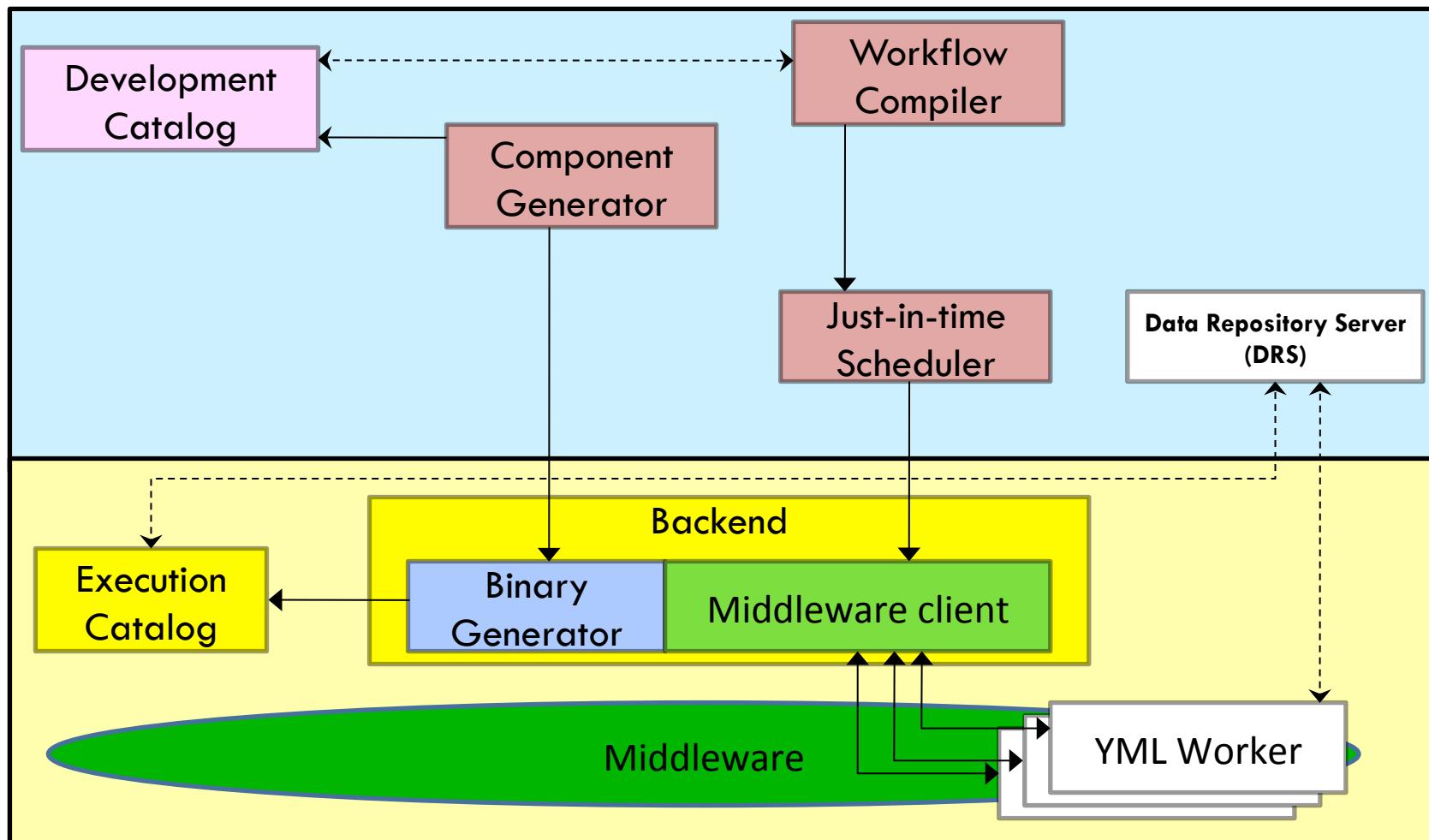
- YML<sup>1</sup> Framework is dedicated to develop and run parallel and distributed applications on Cluster, clusters of clusters, and supercomputers (schedulers and middleware would have to be optimized for more integrated computer – cf. “K” and OmniRPC for example).
- Independent from systems and middlewares
  - The end users can reuse their code using another middleware
  - Actually the main system is OmniRPC<sup>3</sup>
- Components approach
  - Defined in XML
  - Three types : Abstract, Implementation (in FORTRAN, C or C++;XMP,..), Graph (Parallelism)
  - Reuse and Optimized
- The parallelism is expressed through a graph description language, named Yvette (*name of the river in Gif-sur-Yvette where the ASCI lab was*)
- Deployed in France, Belgium, Ireland, Japan, China, Tunisia, USA.

<sup>1</sup> University of Versailles/ PRISM (<http://yml.prism.uvsq.fr/>)

<sup>2</sup> University of Paris XI (<http://www.xtremweb.net/>)

<sup>3</sup> University of Tsukuba (<http://www.omni.hpcc.jp/OmniRPC/>)

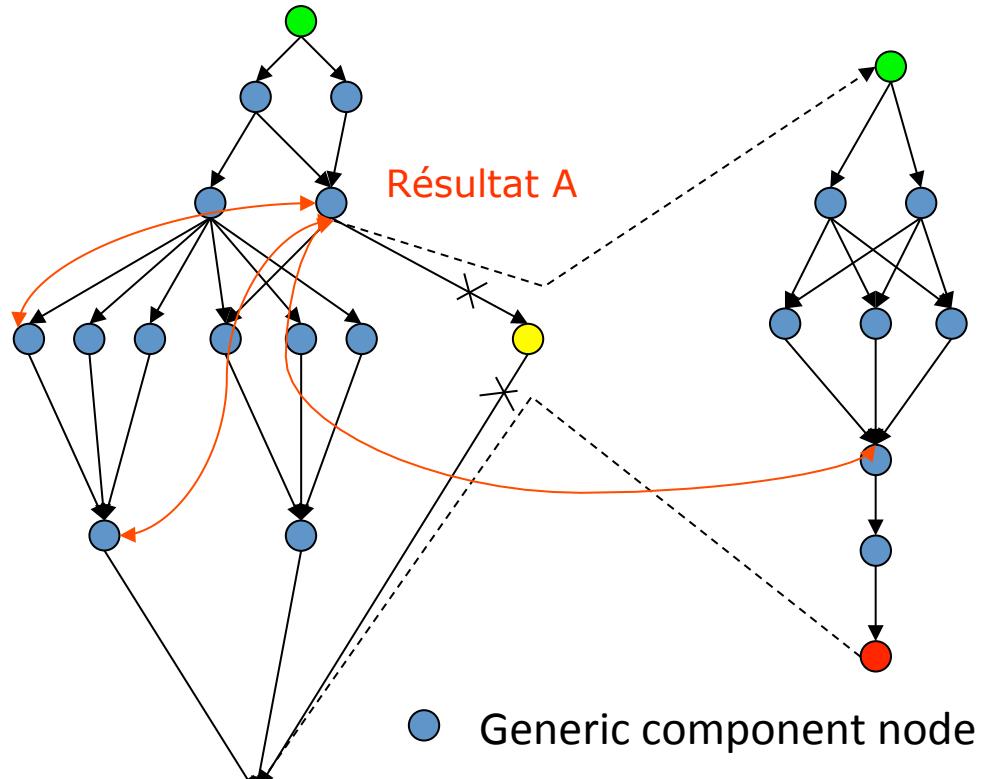
# YML Architecture



# Graph description language: Yvette

- Language keywords
  - Parallel sections: **par section1 // ... // section N endpar**
  - Sequential Loops: **seq (i:=begin;end)do ... enddo**
  - Parallel Loops: **par (i:=begin;end)do ... enddo**
  - Conditionnal structure: **if (condition) then ... else ... endif**
  - Synchronization: **wait(event) / notify(event)**
  - Component call: **compute NameOfComponent(args,...,...)**
- Application examples with a dense matrix inversion method: The block Gauss-jordan,....
- Others experiments (including sparse matrix computation) : Krylov methods, Hybrid Methods,....
- 4 types de components :
  - Abstracts
  - Graphs
  - Implementations
  - Executions

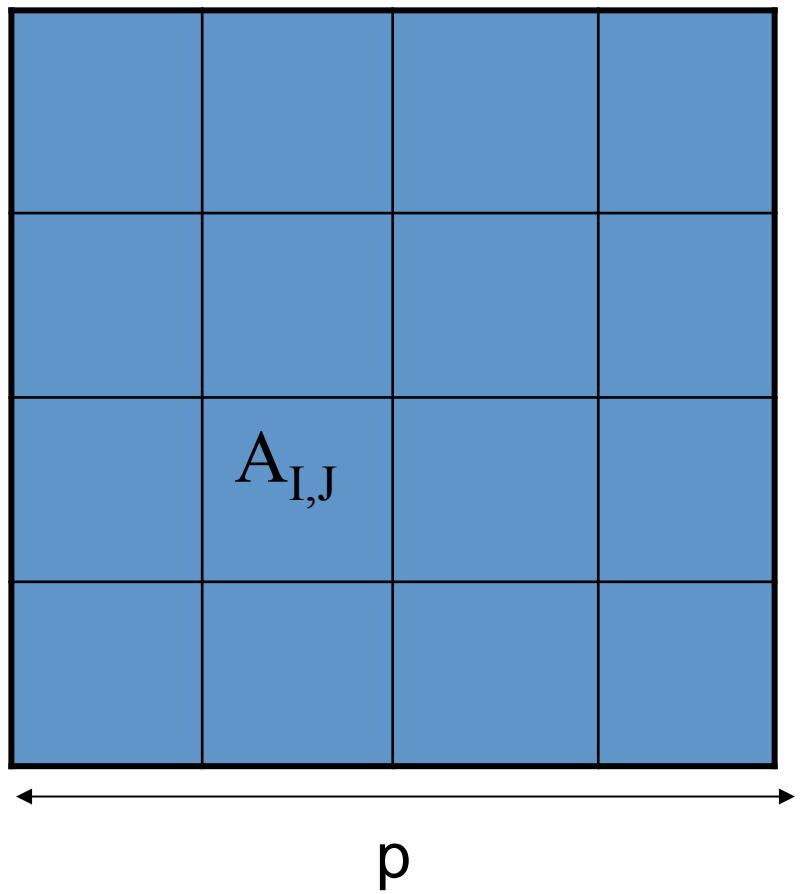
# Components/Tasks Graph Dependency



```

par
  compute tache1(..);
  notify(e1);
 $\text{//}$ 
forall i=1,n
  compute tache2(.i.,.); migrate matrix
  (...) ;
  notify(e2(i));
end forall
 $\text{//}$ 
wait(e1 and e2(1);
Par
  compute tache3(..);
  signal(e3);
//
  compute tache4(..);
  signal(e4);
//
  compute tache5(..); control robot(..);
  signal(e5); visualize mesh(...) ;
end par
 $\text{//}$ 
wait(e3 and e4 and e5);
  compute tache6(..);
  compute tache7(..);
end par

```



$$B = \begin{array}{|c|c|c|c|} \hline I & 0 & 0 & 0 \\ \hline 0 & I & 0 & 0 \\ \hline 0 & 0 & I & 0 \\ \hline 0 & 0 & 0 & I \\ \hline \end{array}$$

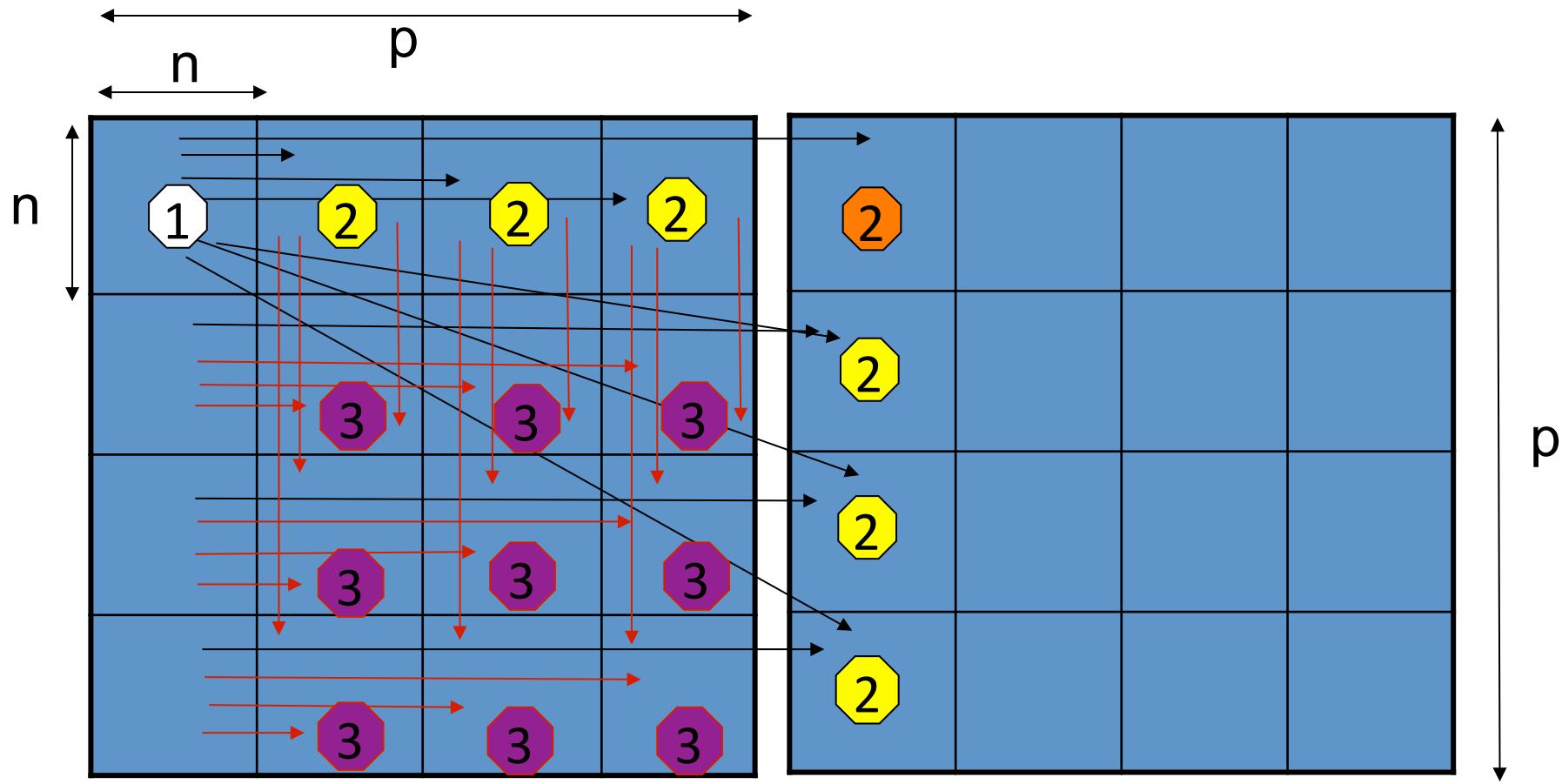
The matrix  $B$  is a  $n \times n$  block diagonal matrix. It consists of four  $1 \times 1$  identity matrices ( $I$ ) on the main diagonal and zeros ( $0$ ) elsewhere. Above the matrix, a double-headed arrow indicates its width is  $n$ .

$$AB = BA = I$$

Block Gauss-Jordan

Matrix size =  $N = p n$

To invert a matrix  
 $2N^3$  operations

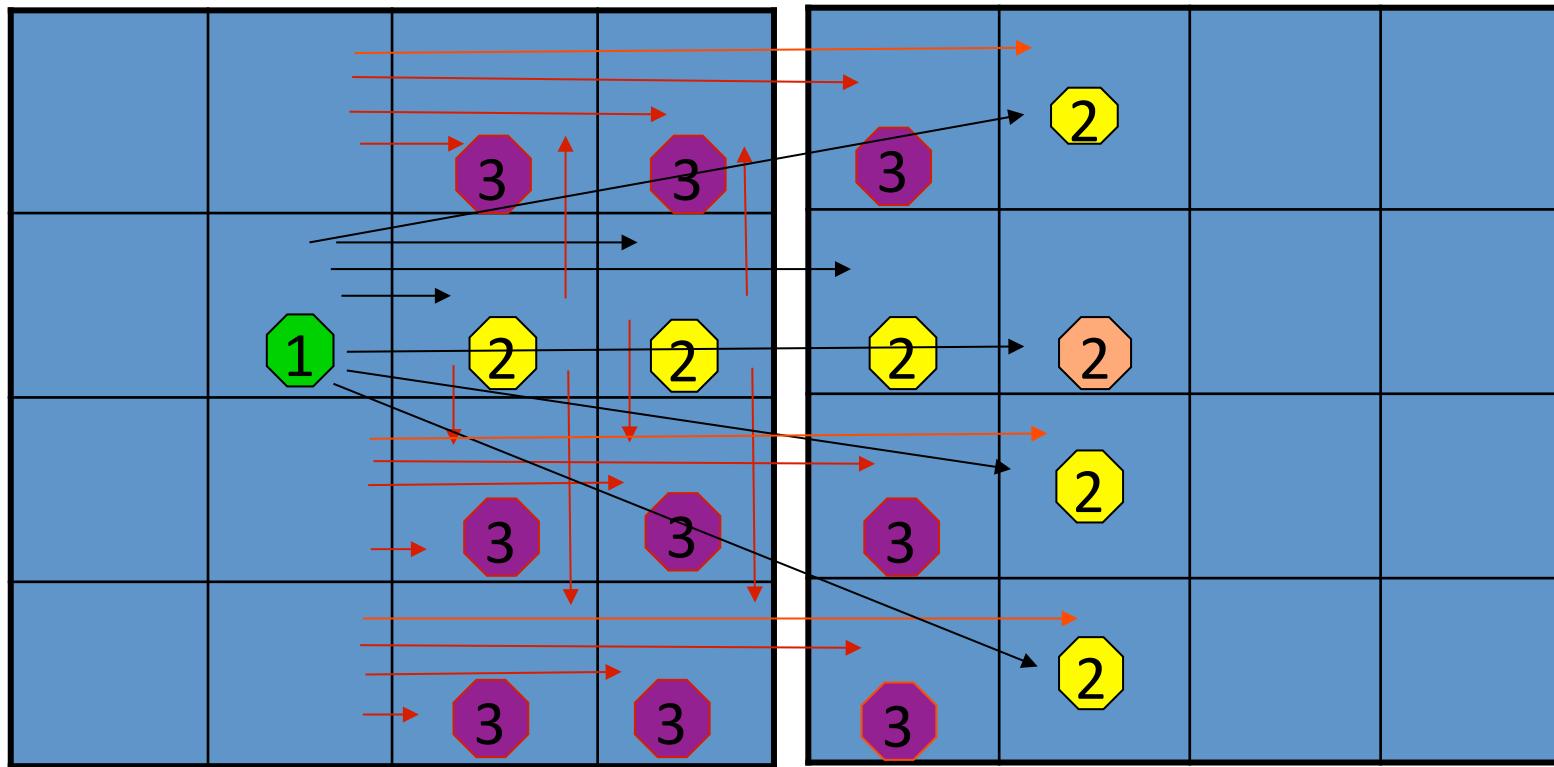


① Element Gauss-Jordan, LAPACK,  $\text{cx} = 2n^3 + O(n^2)$

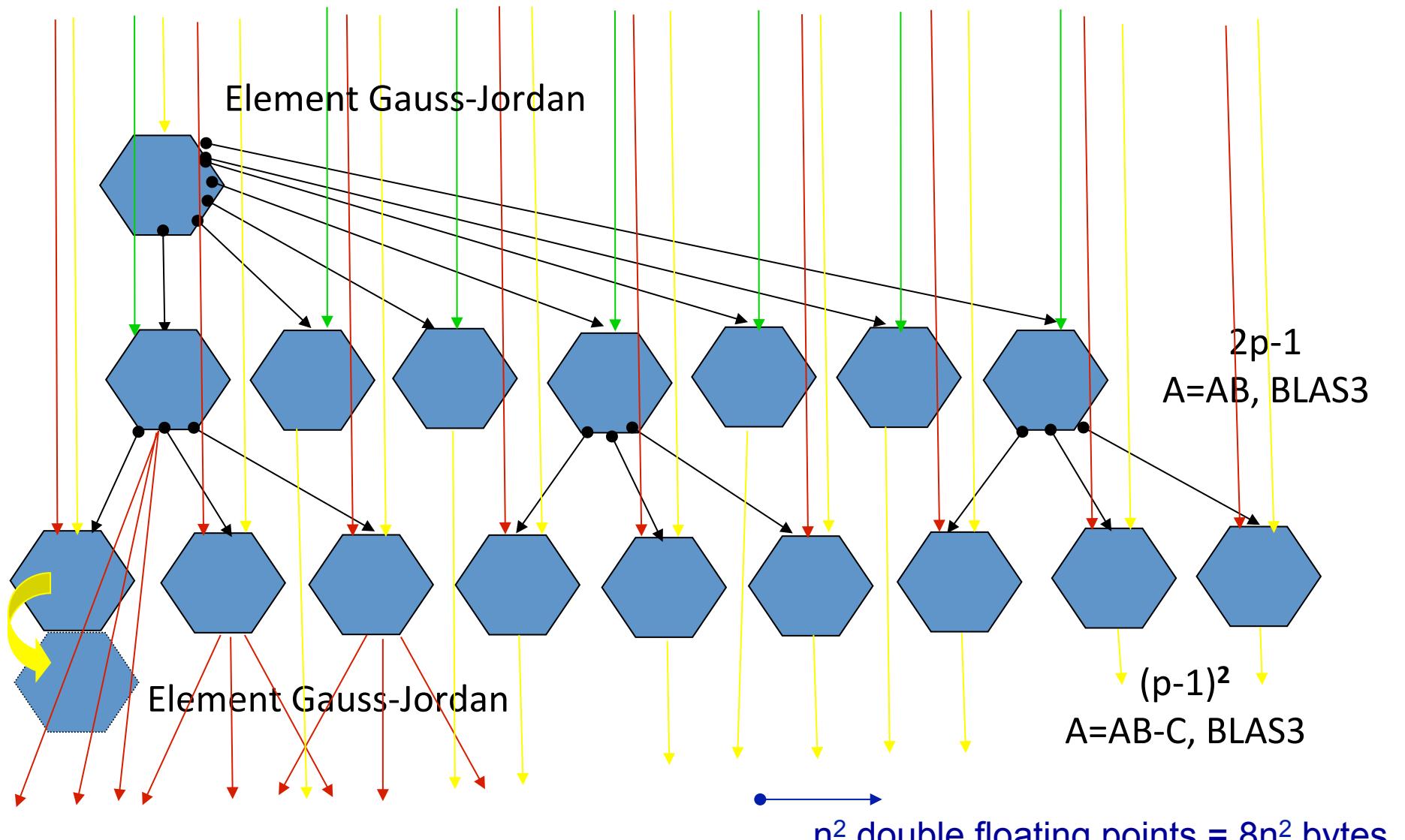
②  $A = +/- A B$  ; BLAS3,  $\text{cx} = 2 n^3 - n^2$ ,      ②  $A = B$

③  $A = A - B C$  ; BLAS3,  $\text{cx} = 2n^3$

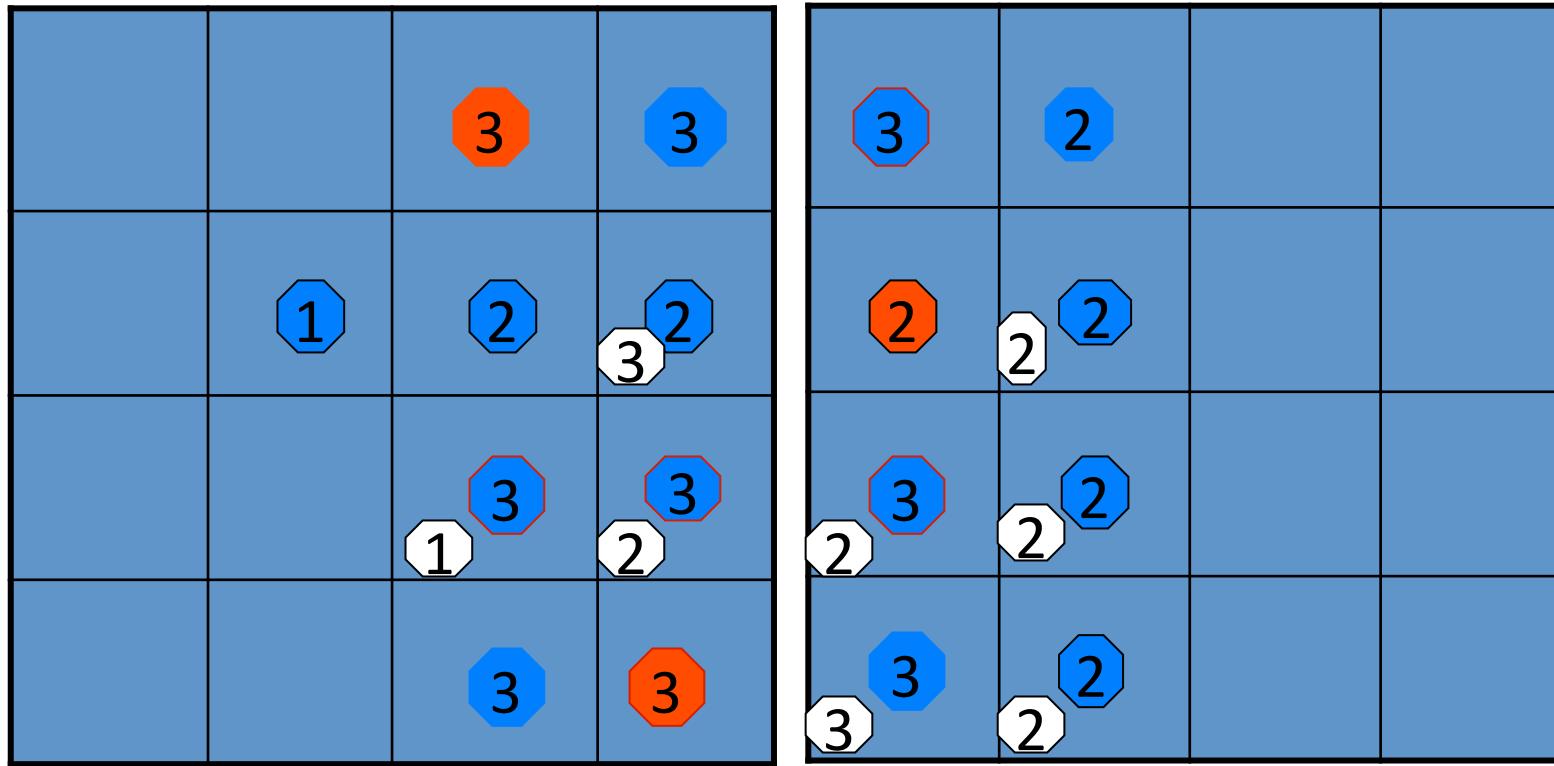
→  $n^2$  64 bit floating point numbers



Each computing task : 1 up to 3 blocks  
 $\text{maximum } n < (\text{memory size of one pair}) / 3$   
 Up to  $(p-1)^2$  computing units (core, node, cluster, peers...)  
**We have to use data persistence and to anticipate data migrations**



**One step of the Block Gauss-Jordan method ;  $p=4$**



Nevertheless, we can have in parallel computing from several steps of the method.

We have to use an inter and intra steps dependency graph (3D for Block Gauss-Jordan).

# Abstract Component

```
<?xml version="1.0" ?>
<component type="abstract"  name="prodMat"
    description="Matrix Matrix Product" >
    <params>
        <param  name="matrixBkk"  type="Matrix" mode="in" />
        <param  name="matrixAki"  type="Matrix" mode="inout" />
        <param  name="blocksize"   type="integer" mode="in" />
    </params>
</component>
```

# Implementation Component

```
<?xml version="1.0"?>
<component type="impl" name="prodMat" abstract="prodMat" description="Implementation
  component of a Matrix Product">
  <impl lang="CXX">
    <header />
    <source>
      <![CDATA[
int i,j,k;
double ** tempMat;
//Allocation
for(k = 0 ; k< blocksize ; k++)
  for (i = 0 ;i <blocksize ; i++)
    for (j = 0 ;j <blocksize ; j++)
      tempMat[i][j] = tempMat[i][j] + matrixBkk.data[i][k] * matrixAki.data[k][j];

    for (i = 0 ;i < blocksize ; i++)
      for (j = 0 ;j < blocksize ; j++)
        matrixAki.data[i][j] = tempMat[i][j];
//Desallocation
  ]]>
    </source>
    <footer />
  </impl>
</component>
```

# Graph component of Block Gauss-Jordan Method

```

<?xml version="1.0"?>
<application name="Gauss-Jordan">
<description>produit matriciel pour deux matrice carree
</description>
<graph>
blocksize:=4;
blockcount:=4;

    par (k:=0;blockcount - 1)
    do
        #inversion
        if (k neq 0) then
            wait(prodDiffA[k][k][k - 1]);
        endif
        compute inversion(A[k][k],B[k][k],blocksize,blocksize);
        notify(bInversed[k][k]);

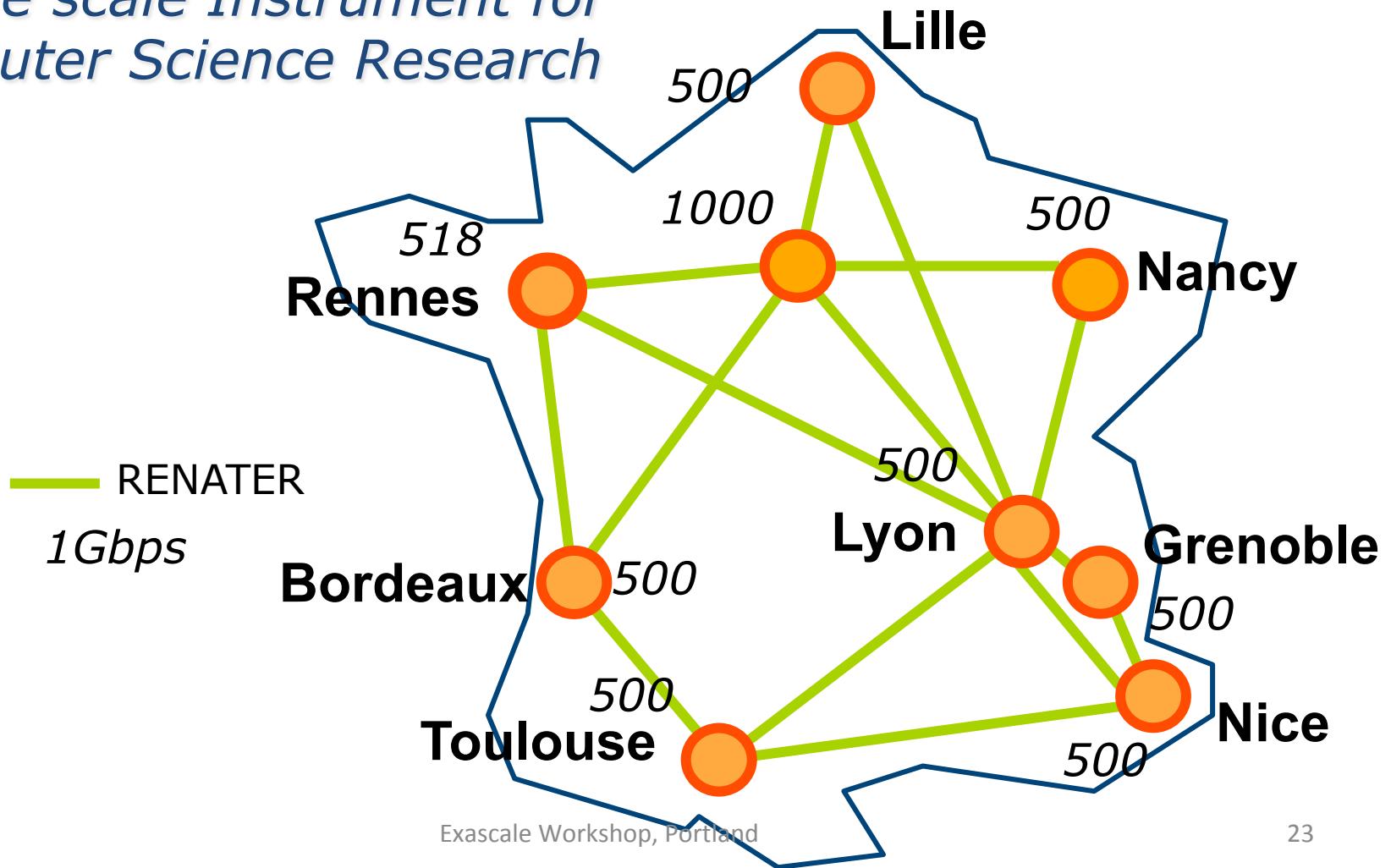
        #step 1
        par (i:=k + 1; blockcount - 1)
        do
            wait(bInversed[k][k]);
            compute prodMat(B[k][k],A[k][i],blocksize);
            notify(prodA[k][i]);
        enddo

        par(i:=0;blockcount - 1)
        do
            #step 2.1
            if(i neq k) then
                wait(bInversed[k][k]);
                compute mProdMat(A[i][k],B[k][k],B[i][k],blocksize);
                notify(mProdB[k][i][k]);
            endif
            #step 2.2
            if(k gt i) then
                wait(bInversed[k][k]);
                compute prodMat(B[k][k],B[k][i],blocksize);
                notify(prodB[k][i]);
            endif
        enddo
    enddo
#Step3
    par( i:= 0;blockcount - 1)
    do
        if (i neq k) then
            if (k neq blockcount - 1) then
                #step 3.1
                par (j:=k + 1;blockcount - 1)
                do
                    wait(prodA[k][j]);
                    compute
prodDiff(A[i][k],A[k][j],A[i][j],blocksize);
                    notify(prodDiffA[i][j][k]);
                enddo
            endif
            #step 3.2
            if (k neq 0) then
                par(j:=0;k - 1)
                do
                    wait(prodB[k][j]);
                    compute
prodDiff(A[i][k],B[k][j],B[i][j],blocksize);
                enddo
            endif
        endif
    enddo
</graph>
</application>

```

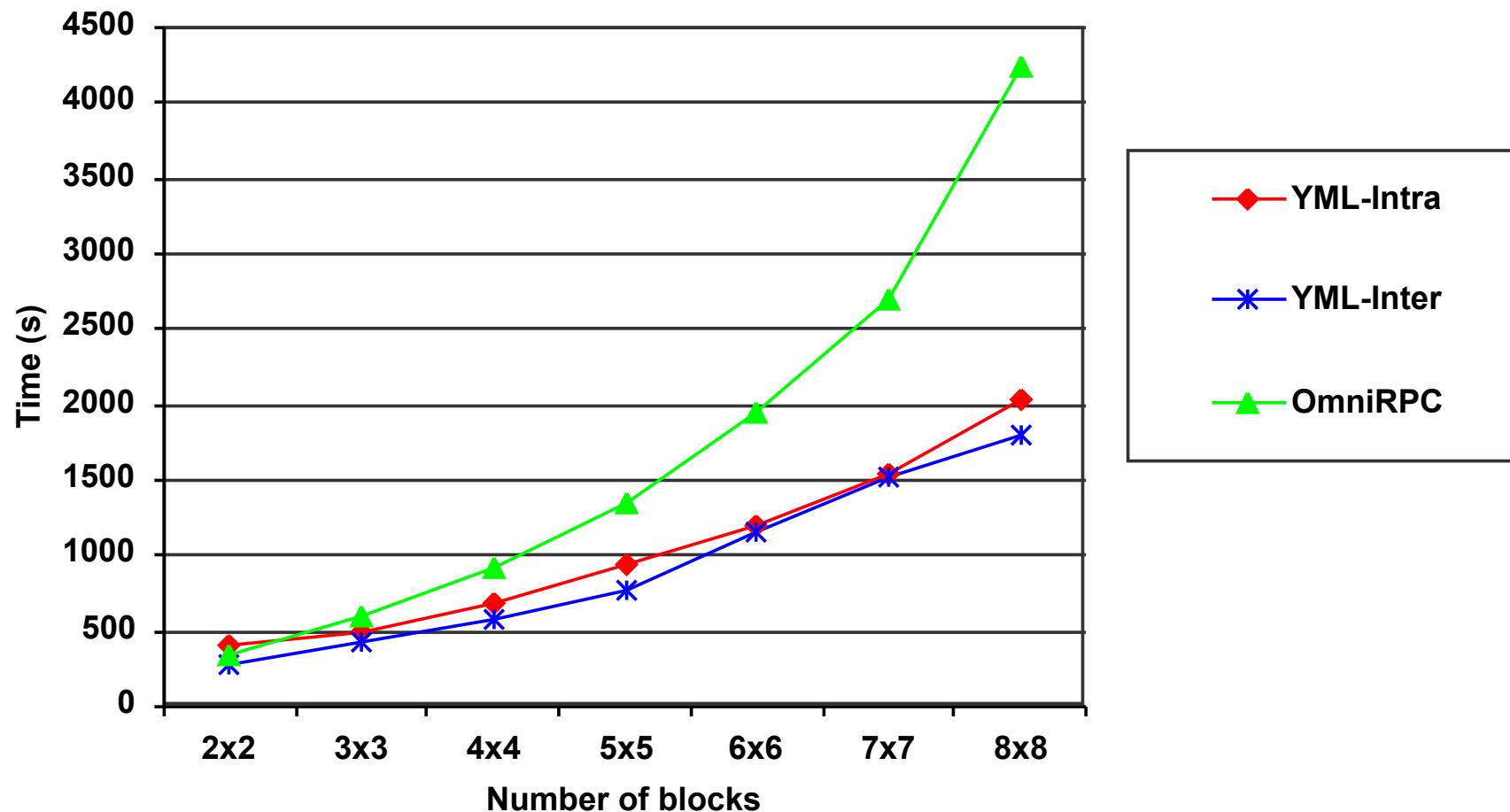
## YML experimentations on Grid'5000, seen as a cluster of heterogeneous clusters

*A large scale Instrument for  
Computer Science Research*



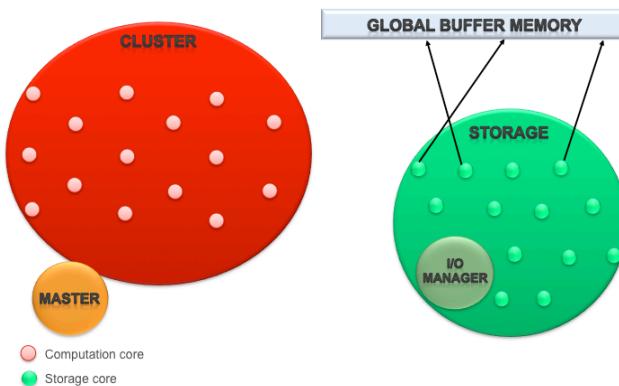
BGJ on a Cluster of clusters composed of 200 processors distributed over 4 clusters (without data migration optimizations)

Time of execution for Intra and Inter step implementation, block size = 1500

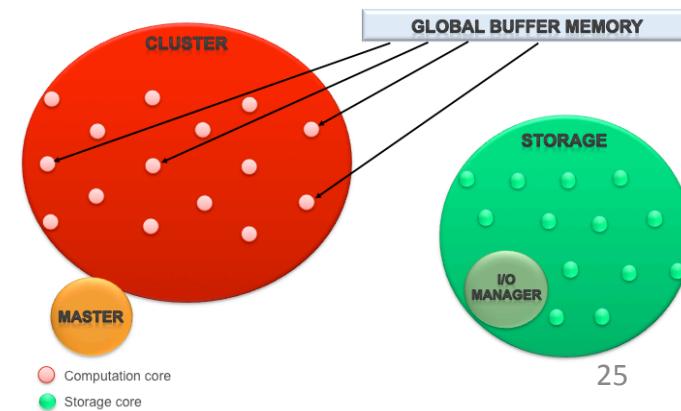


# Data persistence, data anticipation migration and asynchronous I/O

- Data persistence and data migration anticipation are not proposed yet by the YML scheduler
  - In consequence, the data persistence mechanism is "emulated" by regenerating blocks on nodes
  - An overhead is to add for data management and anticipation migration
- Henri Calandra (TOTAL), Maxime Hugues (then in TOTAL), Serge Petiton (Univ. Lille/CNRS) and Data Direct Network (DDN), propose a system named ASIODS which combines graph and delegation in order to avoid disk contention and have a better caching : these technics would be very well-adapted to YML as the graph is available.

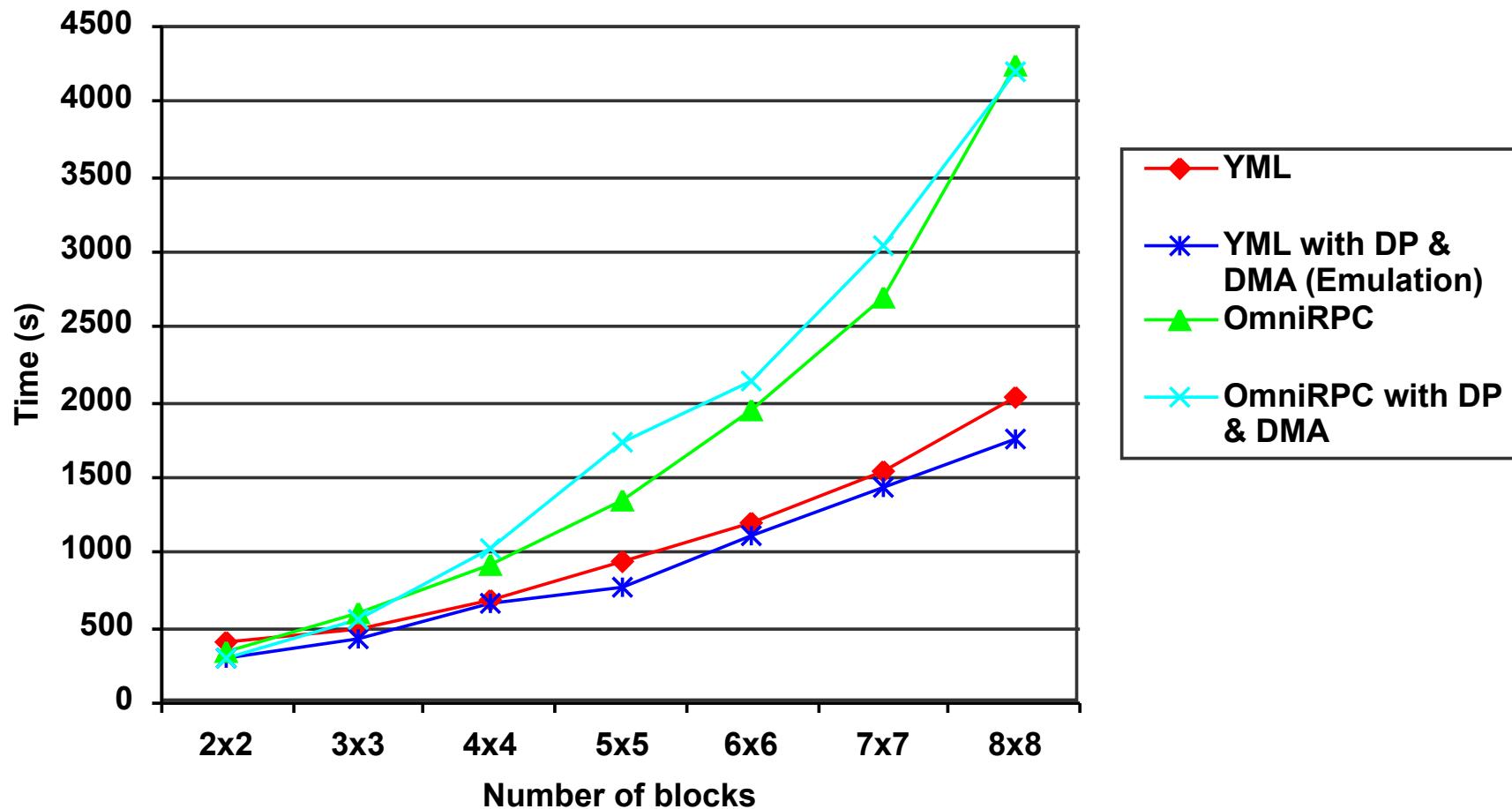


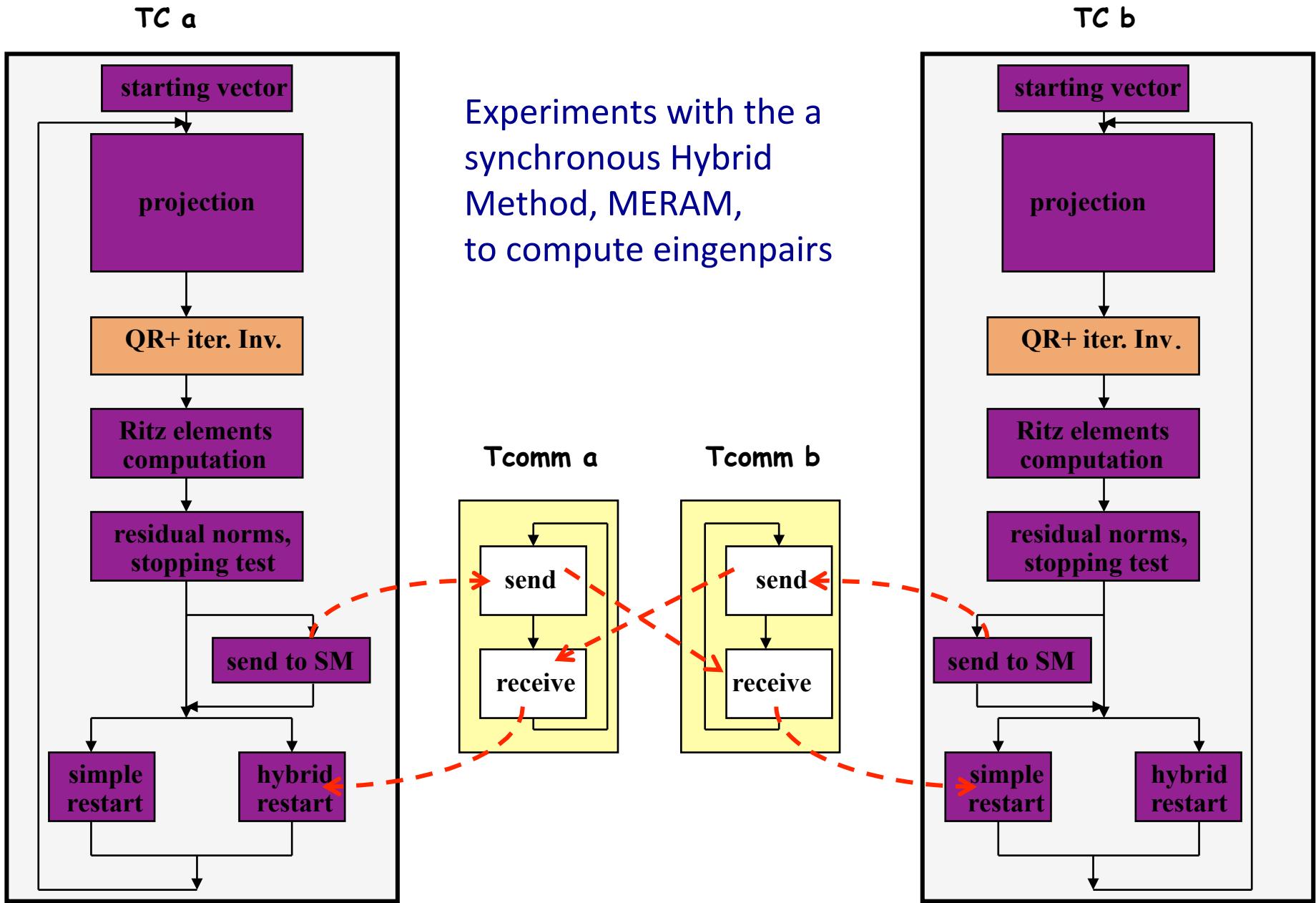
Exascale Workshop, Portland



# BGJ DP<sup>1</sup> & DMA<sup>2</sup> on 200 processors distributed over 4 clusters (Cluster of Clusters)

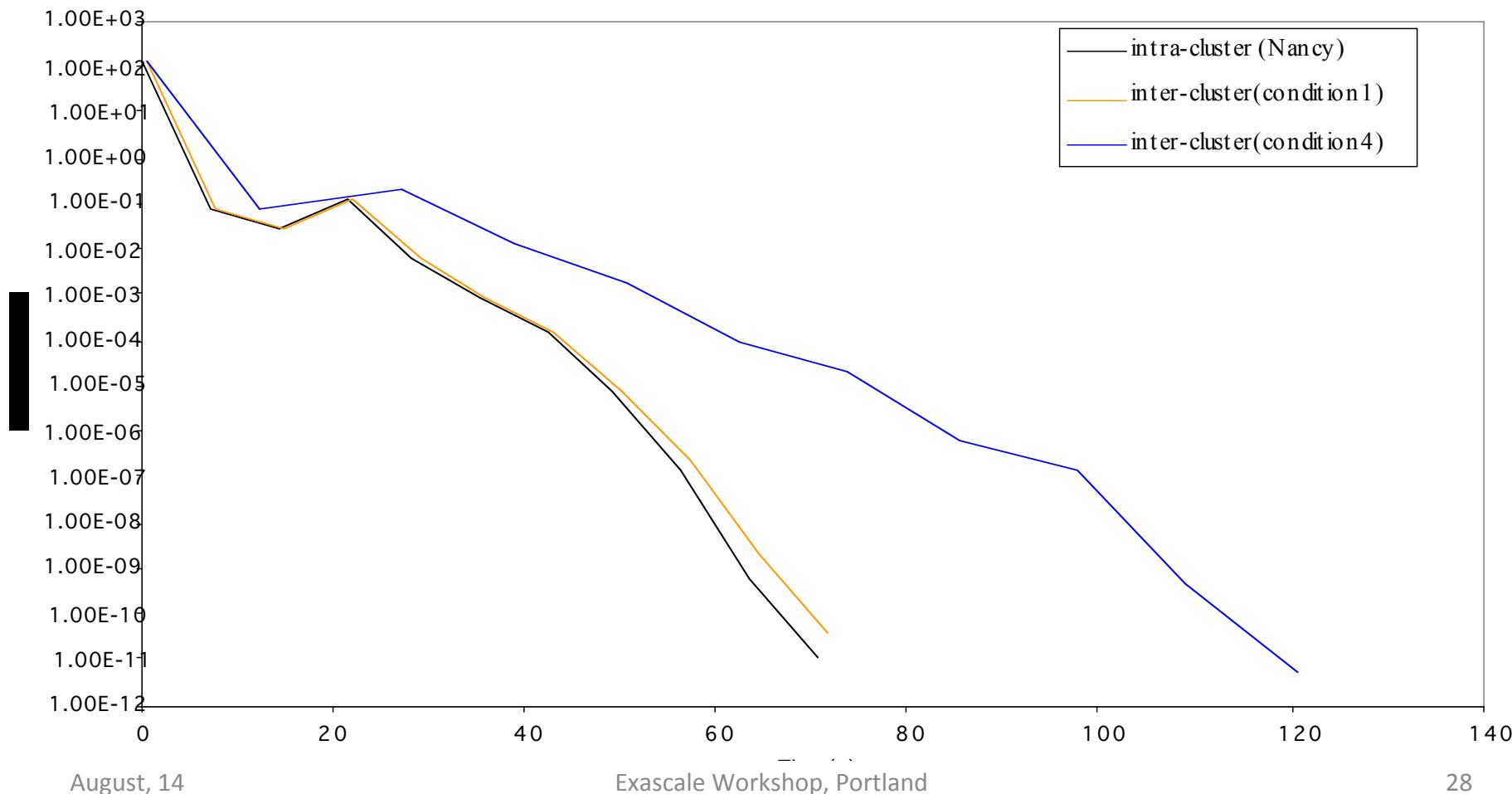
Time of execution on a Cluster of Clusters, block size = 1500





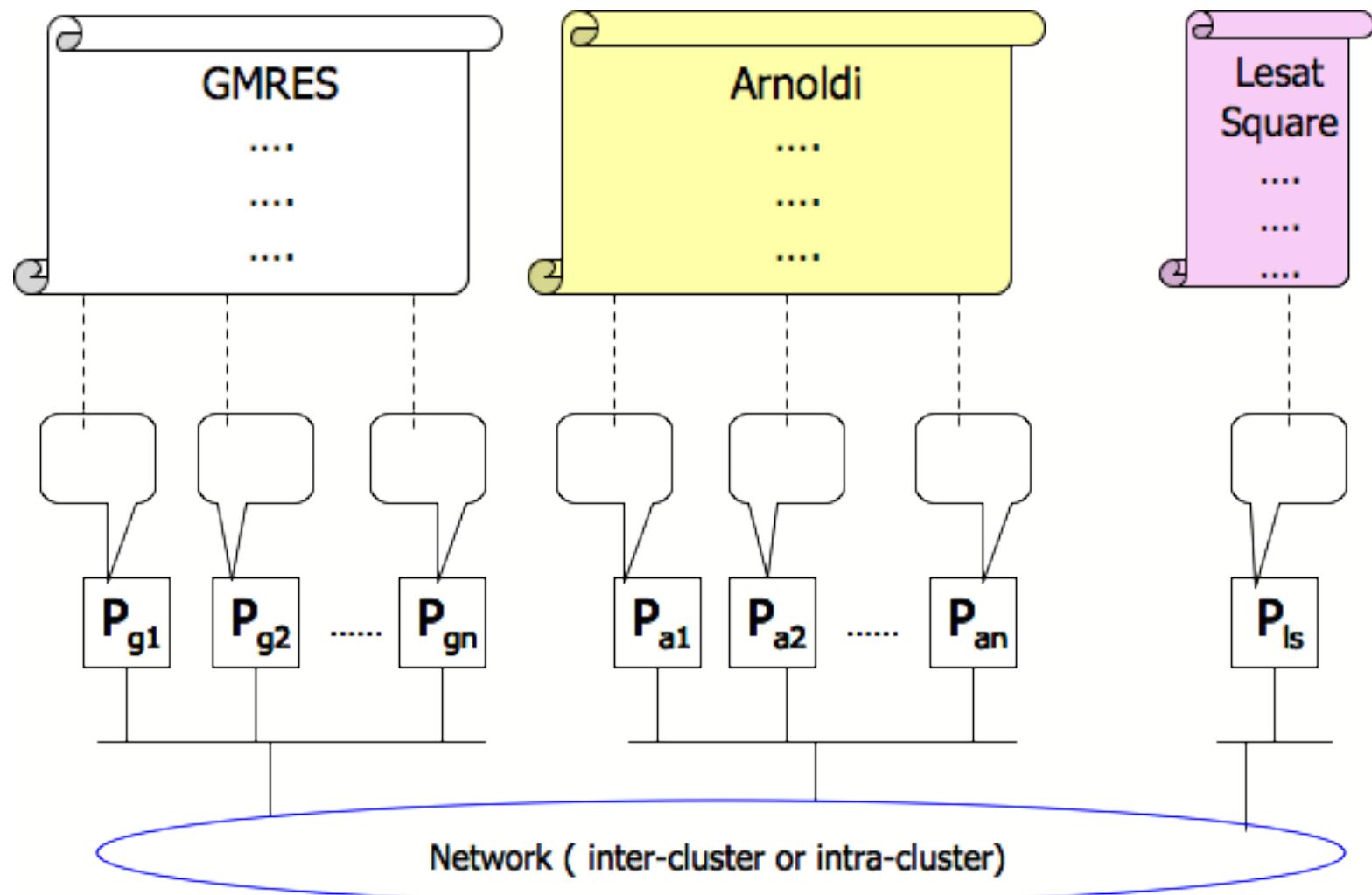
# Comparison between different configurations on Grid5000

## Implementation intra-cluster and inter-cluster (17000)



# Asynchronous Iterative Restarted Methods

Collaboration with Guy Bergère and Ye Zhang



# Outline

- Introduction
- Multilevel programming paradigms
- The YML environment and experiments
- **YML/XMP/StarPU and the Japanese-French FP3C project**
- Reusable library, experiment with PETSc, SLEPc and YML
- Conclusion



University of Tsukuba  
筑波大学

informatics mathematics  
*inria*

cea  
énergie atomique • énergies alternatives

東京工業大学  
Tokyo Institute of Technology



FP3C Group



東京大学  
THE UNIVERSITY OF TOKYO



## Framework and Programming for Post-Petascale Computing (FP3C)

An 4 years ANR-JST suported project

Japanese PI: Mitsuhsia Sato

French PI: Serge Petiton

YML/XMP integration :

Mitsuhsia Sato

Serge Petiton

Maxime Hugues

Miwako Tsuji

[jfli.nii.ac.jp](http://jfli.nii.ac.jp)

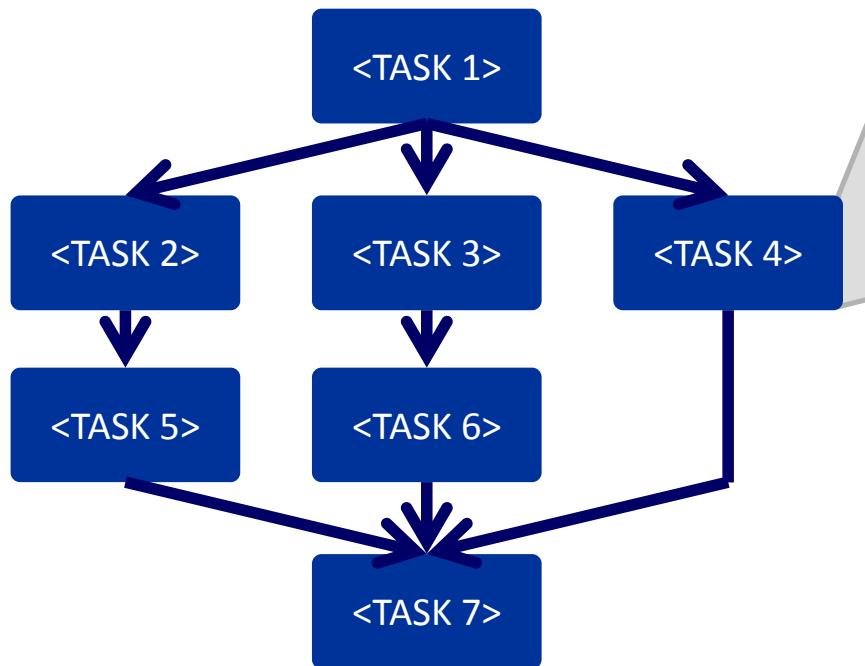
Projects, and FP3C

# YML/XMP/StarPU

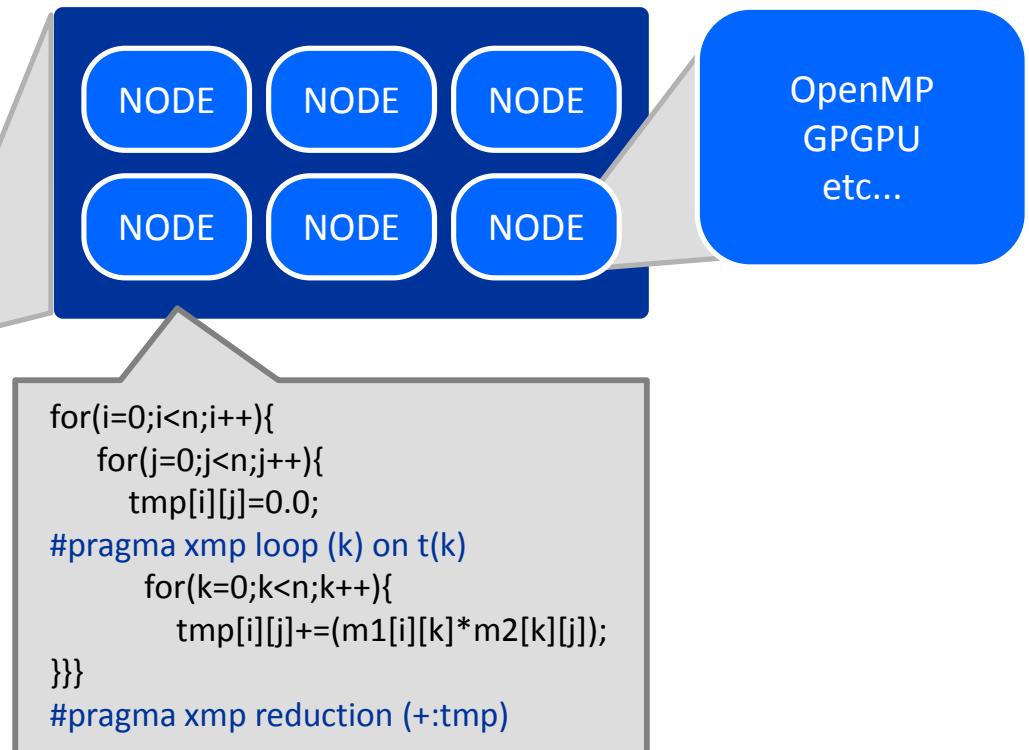
- Multi-level programming paradigm proposal:
  - Top level for inter-gang/nodes communications
  - Intermediate level with gang of nodes
  - Low level at lot-of-cores processors
- A Multi-Level Parallel Programming Framework Implementation:
  - Top: **YML** a graph description language and framework
  - Intermediate: **XMP** a directive based programming
  - Low: Thread programming paradigm (**StarPU**)
- First: Integrate XMP programming in YML (Done)
- Second : Integrate XMP and StarPU
- Third : Integrate YMP/XMP/StarPU

# Multi-Level Parallelism Integration: YML-XMP

N dimension graphs available



YML provides a workflow programming environment and high level graph description language called YvetteML



Each task is a parallel program over several nodes.  
XMP language can be used to describe parallel program easily!

# XcalableMP (XMP)

- Directive-based language extension for scalable and performance-aware parallel programming
- It will provide a base parallel programming model and a compiler infrastructure to extend the base languages by directives.
- Source (C+XMP) to source (C+MPI) compiler
- Data mapping & Work mapping using template

```
#pragma xmp nodes p(4)
#pragma xmp template t(0:7)
#pragma xmp distribute t(block) onto p
int a[8];
#pragma xmp align a[i] with t(i)

int main(){
#pragma xmp loop on t(i)
    for(i=0;i<8;i++)
        a[i] = i;
```



# XcalableMP: Code Example

```
int array[YMAX][XMAX];
```

```
#pragma xmp nodes p(4)
#pragma xmp template t(YMAX)
#pragma xmp distribute t(block) on p
#pragma xmp align array[i][*] with t(i)
```

data distribution

```
main(){
    int i, j, res;
    res = 0;
```

add to the serial code : incremental parallelization

```
#pragma xmp loop on t(i) reduction(+:res)
for(i = 0; i < 10; i++)
    for(j = 0; j < 10; j++){
        array[i][j] = func(i, j);
        res += array[i][j];
    }
}
```

work sharing and data synchronization

*June 2011 : Future extensions, targeting accelerators*

# YML/XMP Integration

- Create a backend to run MPI programs
  - Reminder: XMP generates MPI code
  - Extend OmniRPC middleware to support MPI
  - Process creation with MPI\_Spawn
    - Portable
    - Batch scheduler needs one executable
- XMP component generator
  - Support XMP compiler
  - Create the suitable binary for OmniRPC extended
- YML needs to support distributed arrays
  - Benefit to be high level oriented
  - Introduction of new attribute in the implementation component
  - Data distribution generated by the XMP component generator
- Data exchange between components through shared file system (first stage)
  - Initial design of YML is not convenient to be executed by batch scheduler of supercomputers
  - Use MPI-IO that gives efficient accesses

# YML-XMP

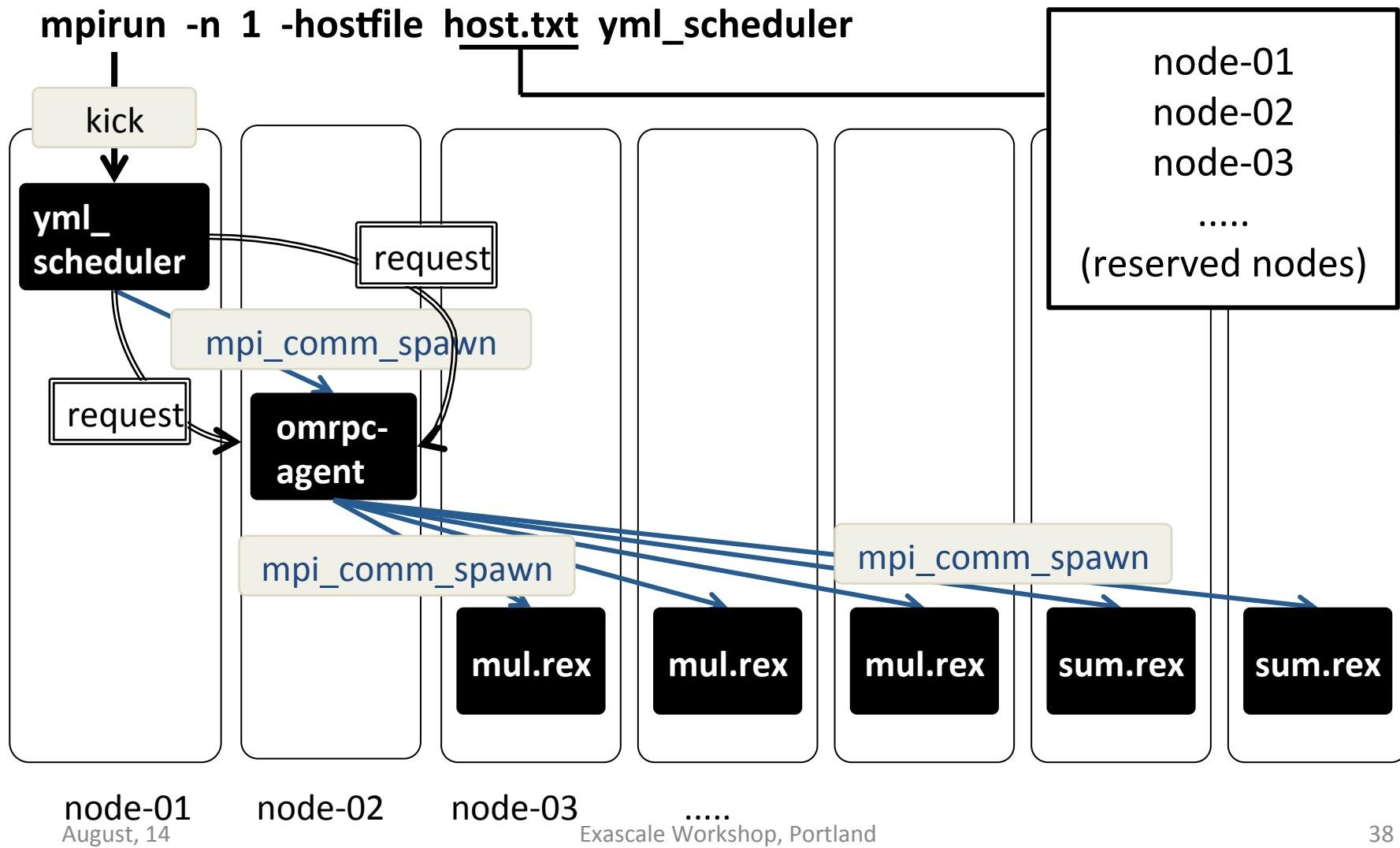
Number of core/nodes and data distribution have to be specified to the scheduler for each XMP task.

Two back-ends will be developed, one based on MPI, the other on omniRPC (which will be deployed on the « K » computer.)

First, abstract component will be adapted **to allow the end-user to give some expertise.**

```
<?xml version="1.0" ?>
<component type="abstract" name="prodMat" description="Matrix Matrix Product" >
<here have to be included end-user expertises, for example to allow the scheduler to
know the number of processor associate to this task, and data distributions>
<params>
    <param name="matrixBkk" type="Matrix" mode="in" />
    <param name="matrixAki" type="Matrix" mode="inout" />
    <param name="blocksize" type="integer" mode="in" />
</params>
</component>
```

## Processes management: *OmniRPC Extension*



# Implementation Component Extension

- Topology and number of processors are declared to be used at compile and run-time.
- Data distribution and mapping are declared
- Automatic generation for distributed language (XMP, CAF, ...)
- Used at run-time to distribute data over processes

```
<?xml version="1.0"?>
<component type="impl" name="Ex" abstract="Ex" description= "Example">
    <impl lang="XMP" nodes="CPU:(5,5)" libs=" " >
        <distribute>
            <param template=" block,block " name="A(100,100) " align="[i][j]:(j,i) " />
            <param template=" block " name="Y(100);X(100)" align="[i]:(i,*)" />
        </distribute>
        <header />
        <source>
            <![CDATA[
                /* Computation Code */
            ]]>
        </source>
        <footer />
    </impl>
</component>
```

Additional node declaration is allowed.  
If node is not declared for a template,  
default node declaration, (4) in this case,  
is used.

```
<?xml version="1.0"?>
<component type="impl" name="test" abstract="test">
<impl lang="XMP" nodes="CPU:(4)" >

<templates>
<template format="block,block" name="t" size="10,10" nodes="p(2,2)"/>
<template format="block" name="u" size="20" />
</templates>

<distribute>
<param name="X(10,10)" align="[i][j]:(i,j)" template="t"/>
<param name="Y(10,10)" align="[i][j]:(i,j)" template="t"/>
<param name="Z(20)" align="[i]:(i)" template="u"/>
</distribute>
```

# XMP & C source code

```
#pragma xmp nodes _XMP_default_nodes(4)

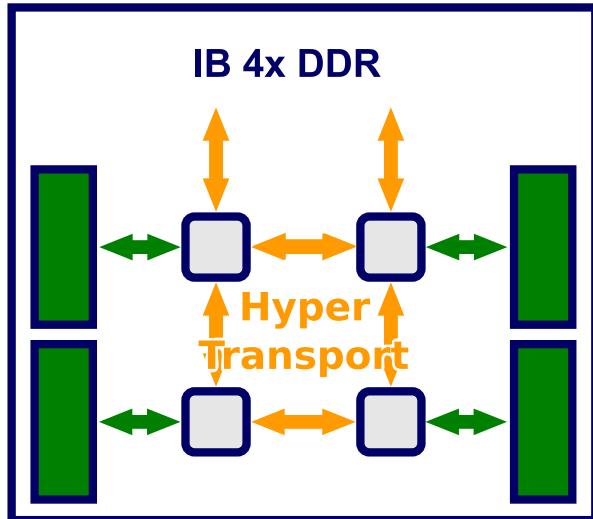
#pragma xmp nodes p(2,2)
#pragma xmp template t(0:9,0:9)
#pragma xmp distribute t(block,block) onto p

#pragma xmp template u(0:19)
#pragma xmp distribute u(block) onto _XMP_default_nodes

XMP_CMatrix X[10][10];
XMP_CMatrix Y[10][10];
XMP_CVector Z[20];

#pragma xmp align X[i][j] with t(i,j)
#pragma xmp align Y[i][j] with t(i,j)
#pragma xmp align Z[i] with u(i)
```

# Experiments : T2K-Tsukuba



Node:

Opteron Barcelona B8000 CPU  
2.3GHz x 4FLOP/c x 4core x 4socket  
= 147.2 GFLOPS/node  
8x4=32GB memory/node

16cores in a node.

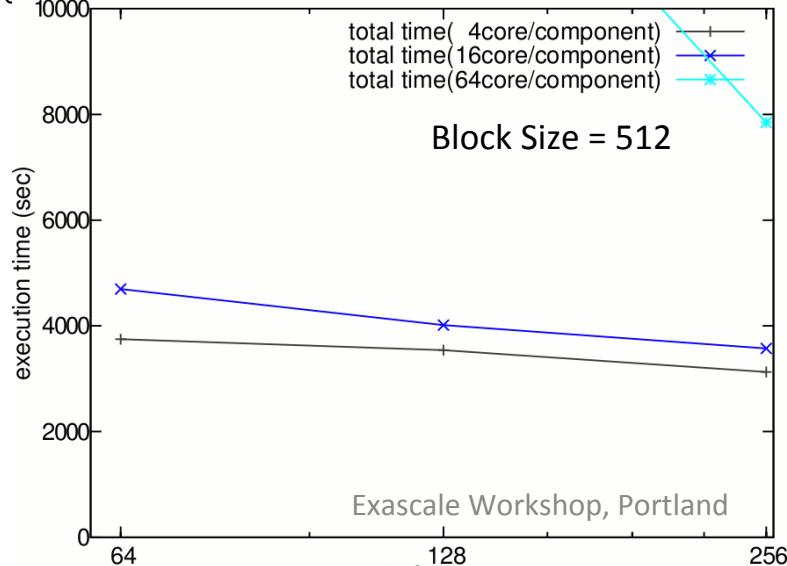
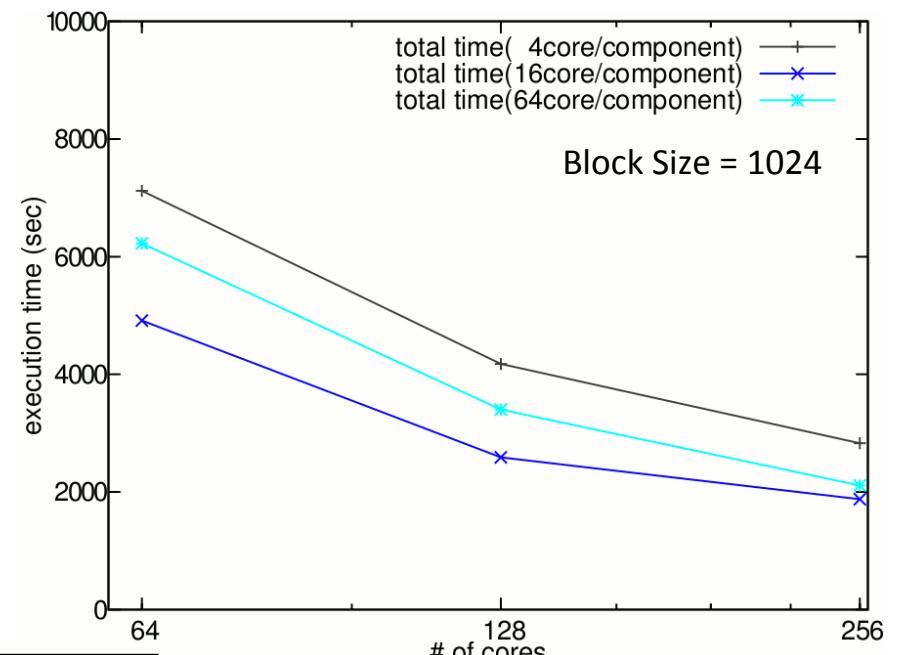
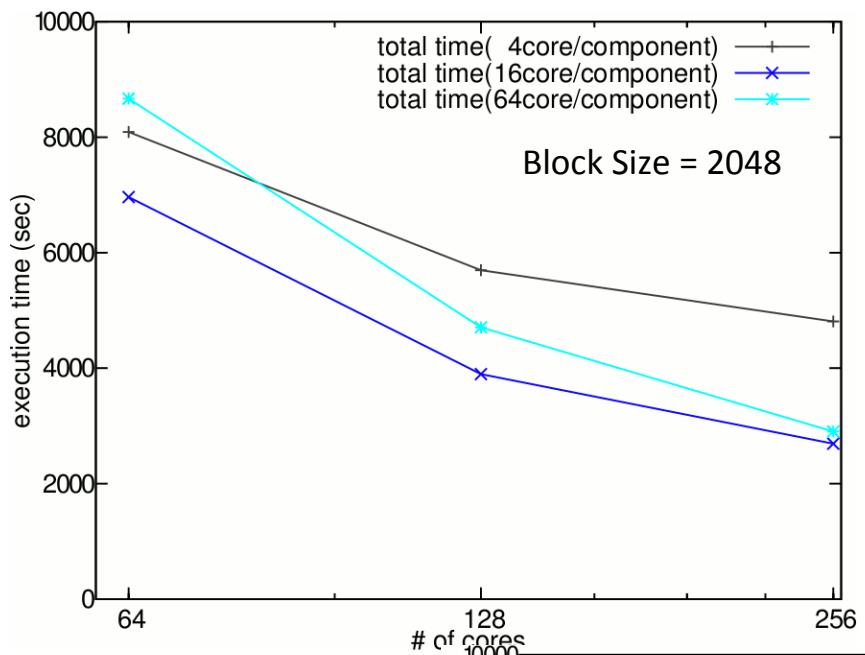
Flat MPI  
August, 14

AMD Opteron  
8350 Barcelona  
Quad-core  
2.3GHz  
8GB  
DDR2-667 memory



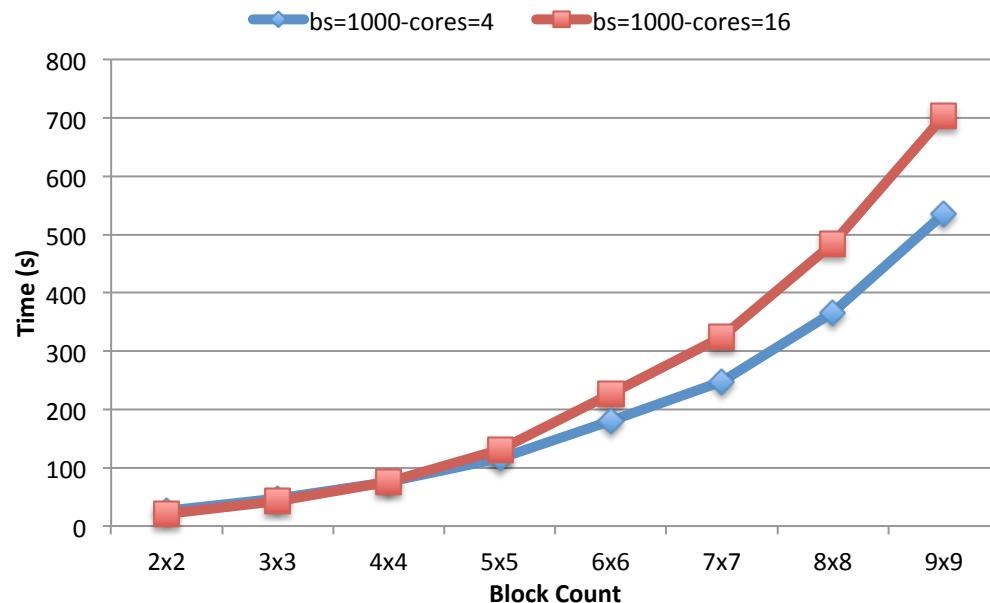
System:  
648 nodes  
95.3 TFLOPS/system  
20.8 TB memory/system  
800 TB Raid-6 Luster cluster file system  
Fat-tree  
full-bisection interconnection  
Quad-rail of InfiniBand

# Block Gauss-Jordan on T2K (Tsukuba)

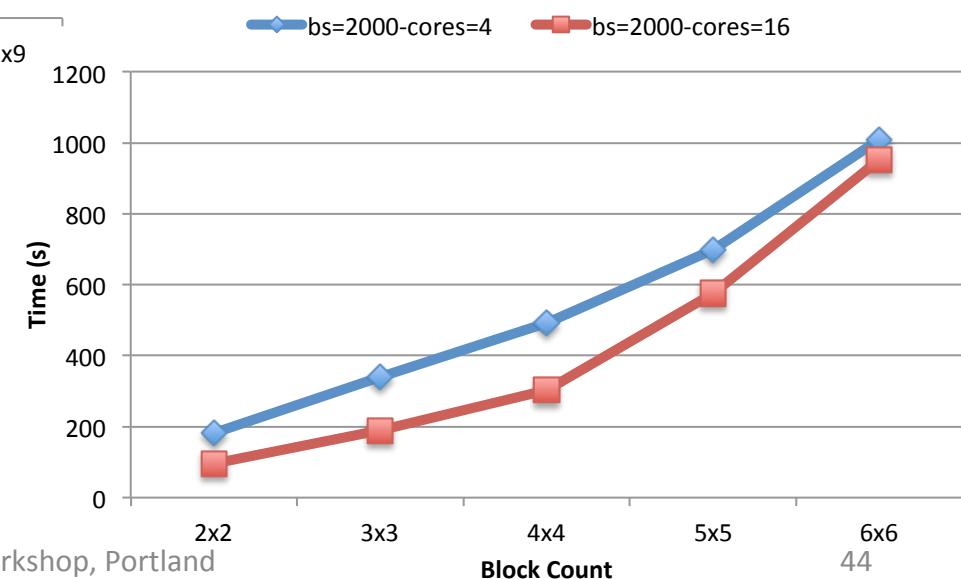


- Block Gauss-Jordan
- Matrix size 16384x16384
- Various block size
- Various nb cores per component

# Block Gauss-Jordan on Grid'5000



- More cores for a component does not mean more performance
- Amount of data must be not too small

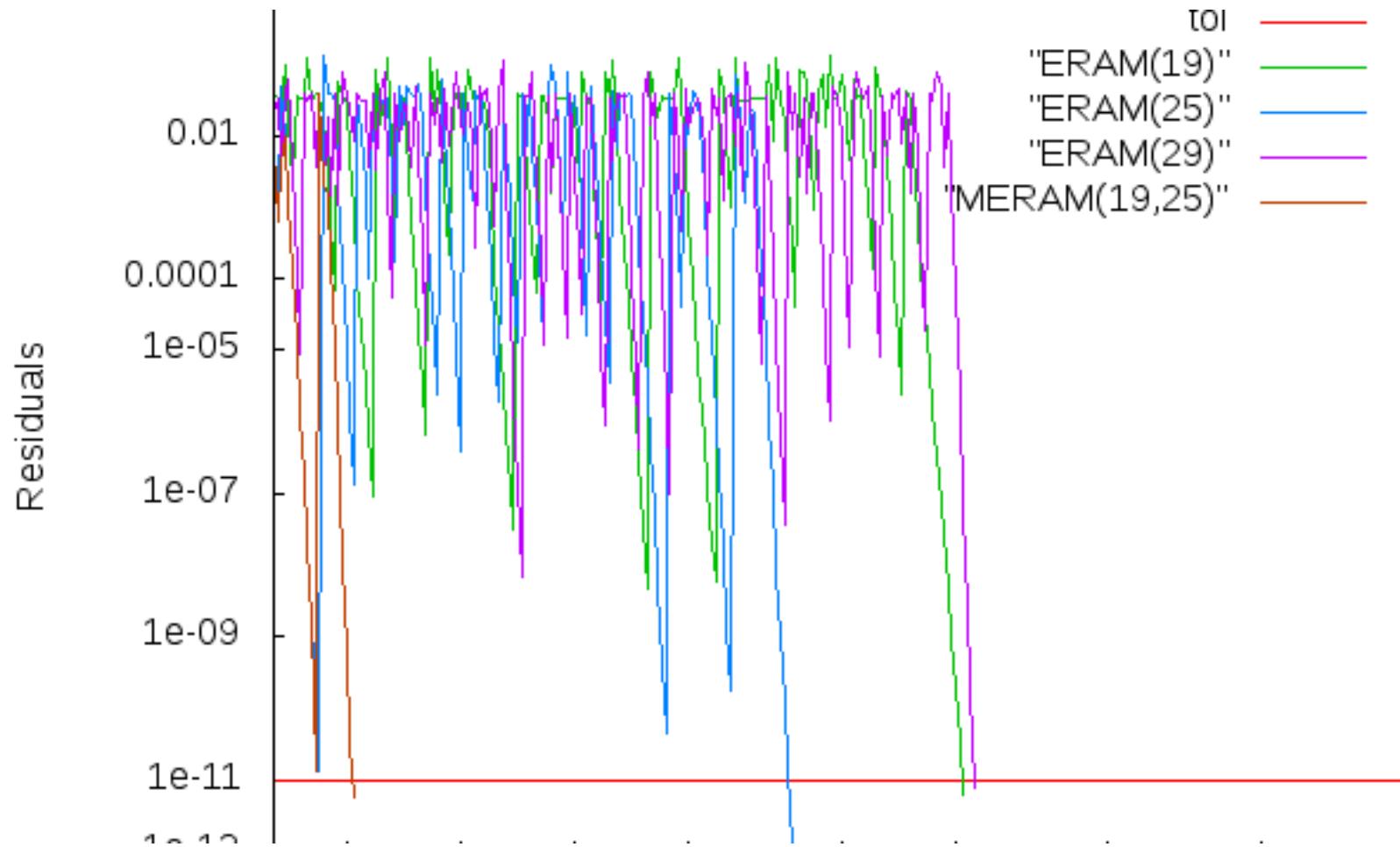


- More parallelism at high level does not mean performance increase
- So what ?
  - Exchange through file system: bottleneck

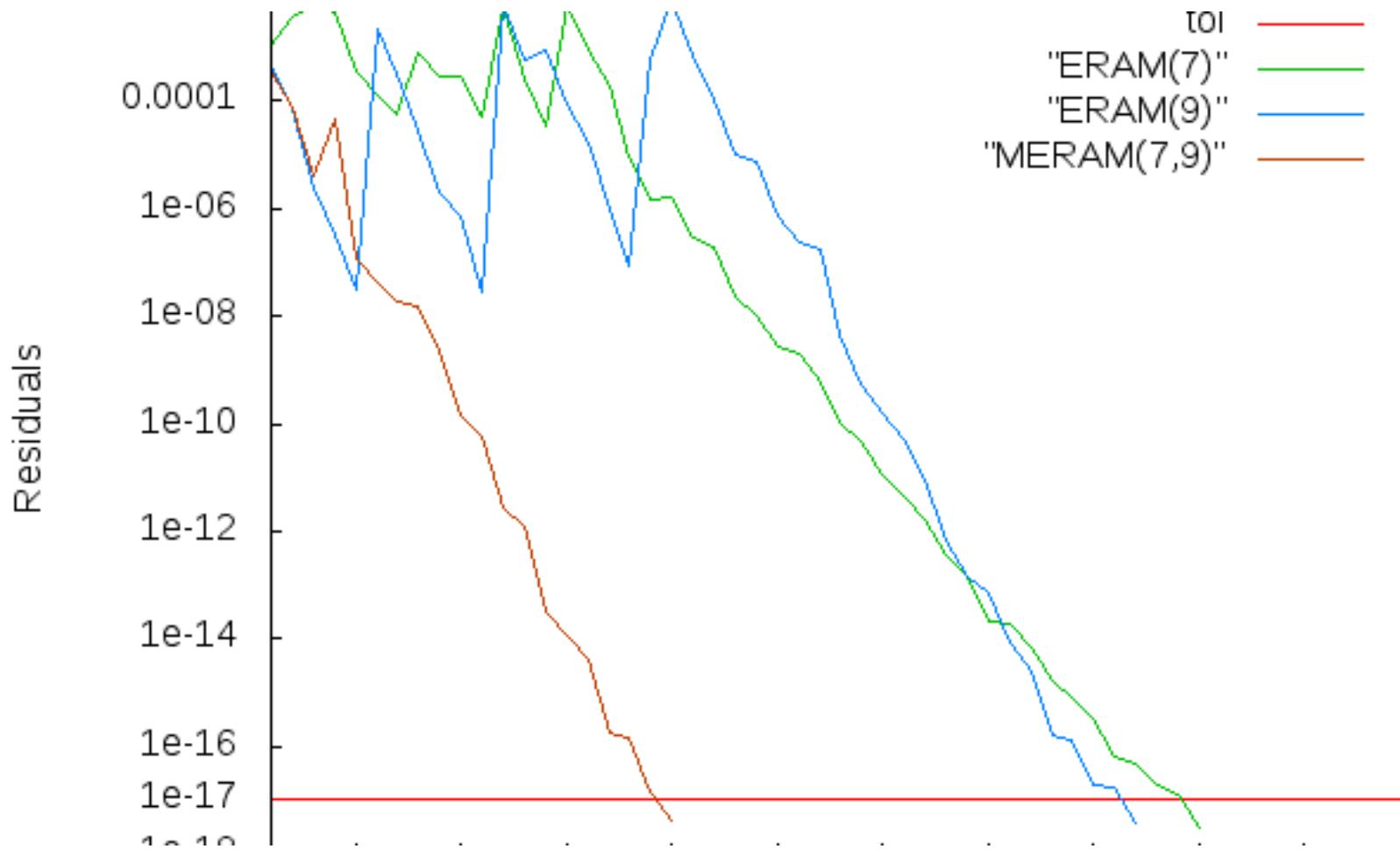
# MERAM vs ERAM

## YML/XMP on Hopper

Nahid Emad, Makarem Dandouna, Leroy Drummond



# MERAM vs ERAM YML/XMP on Hopper



# Outline

- Introduction
- Multilevel programming paradigms
- The YML environment and experiments
- YML/XMP/starPu and the Japanese-French FP3C project
- **Reusable library, experiment with PETSc, SLEPc and YML**
- Conclusion

# Reusable library, experiment with PETSc, SLEPc and YML

Nahid Emad, University of Versailles

Leroy Drummond, LBNL

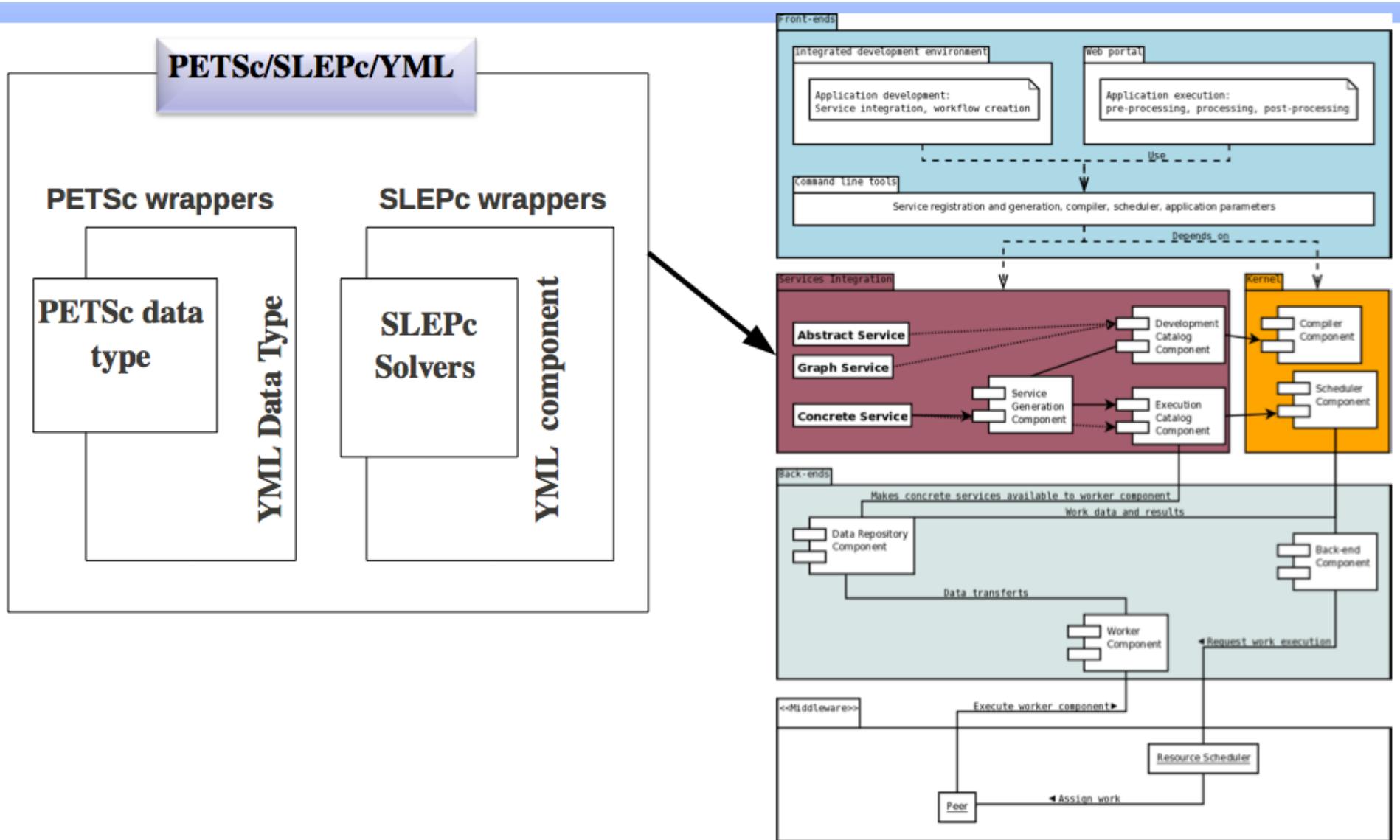
Makaram Dandouna, University of Versailles

Poster this evening:

*Sustainability of Numerical Libraries for Extreme Scale Computing*

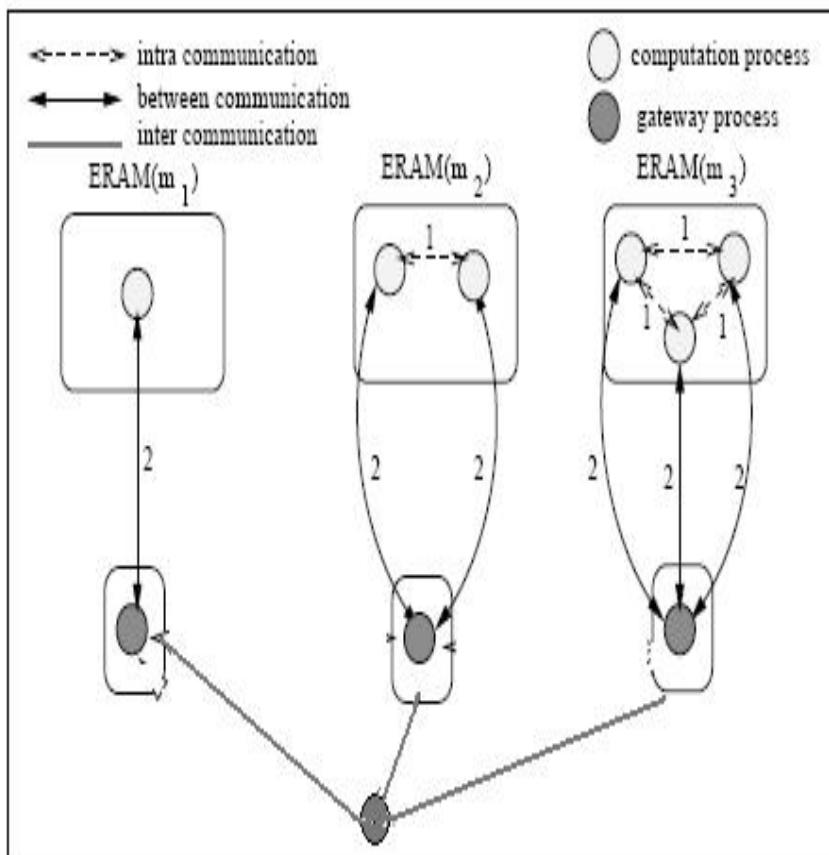
Nahid Emad (University of Versailles, France), Makarem Dandouna (University of Versailles, France), and Leroy Drummond (LBNL)

# Proposed reusable design with PETSc and SLEPc



# Experiments, MERAM

## MERAM/SLEPc/MPI



## MERAM/SLEPc/YML

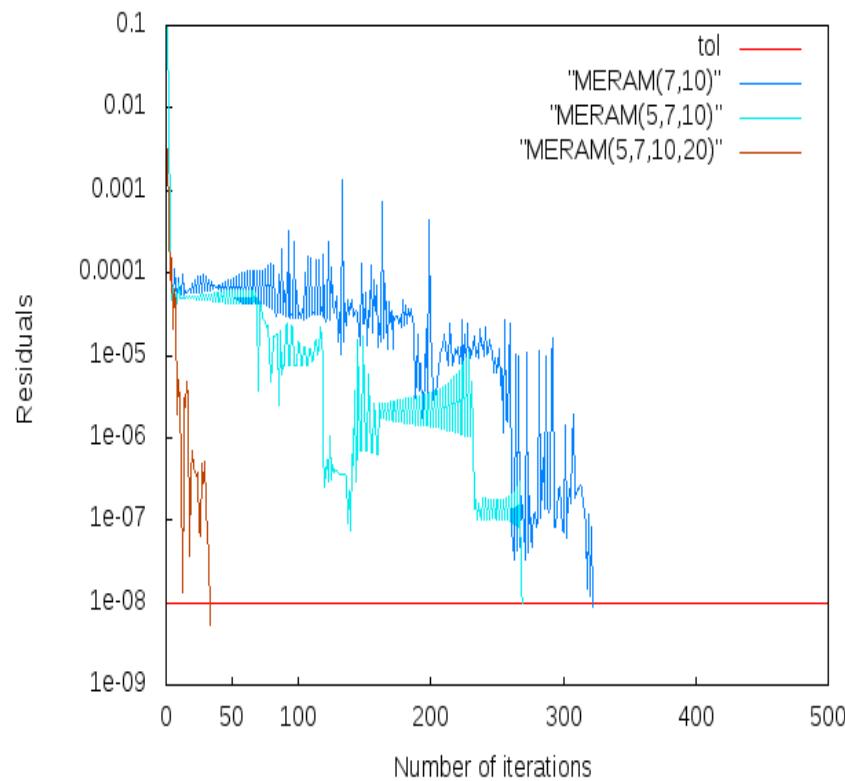
```
#express the coarse grain //ism
par(i:=1;nberam) do
    seq(iter:=1;maxiter) do
        par(j:=1 ; nbProcess[i]) do
            Compute SolverProjection(...)
            notify(result[j])
        ...
        wait(result[j])
    enddo
    compute Solve(...);
    compute reduce(...);

enddo
enddo
```

## Scalability of MERAM vs number of co-methods for af23560 on Grid5000

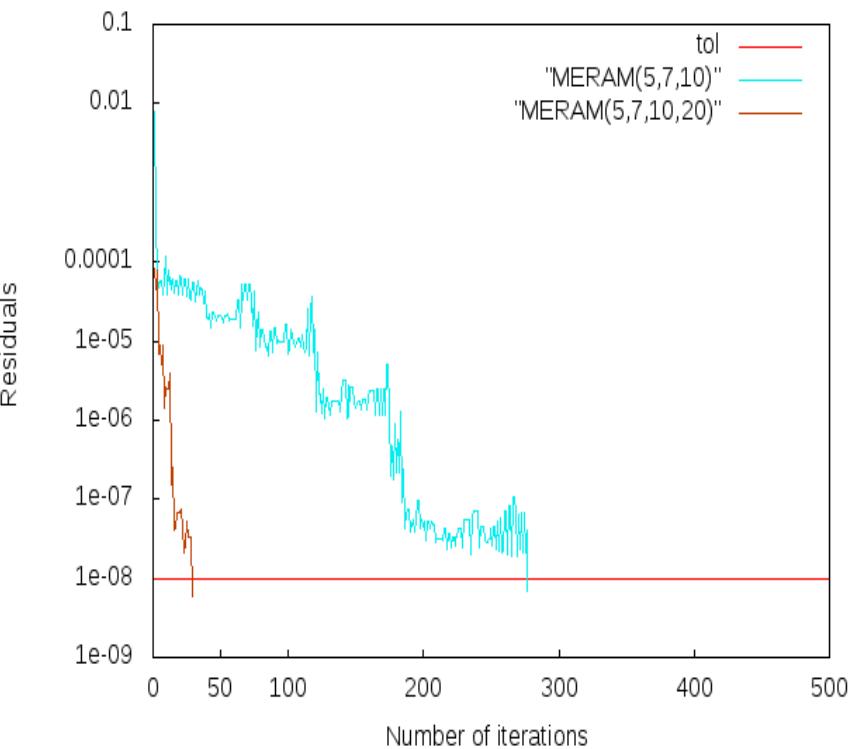
### PETSc\MPI

Scalability of the solution: number of co-methods on A:af23560 n:23560 r:2 tol:



### PETSc\YML

scalability of the solution: number of comethods on A:af23560 n:23560 r:2 tol:

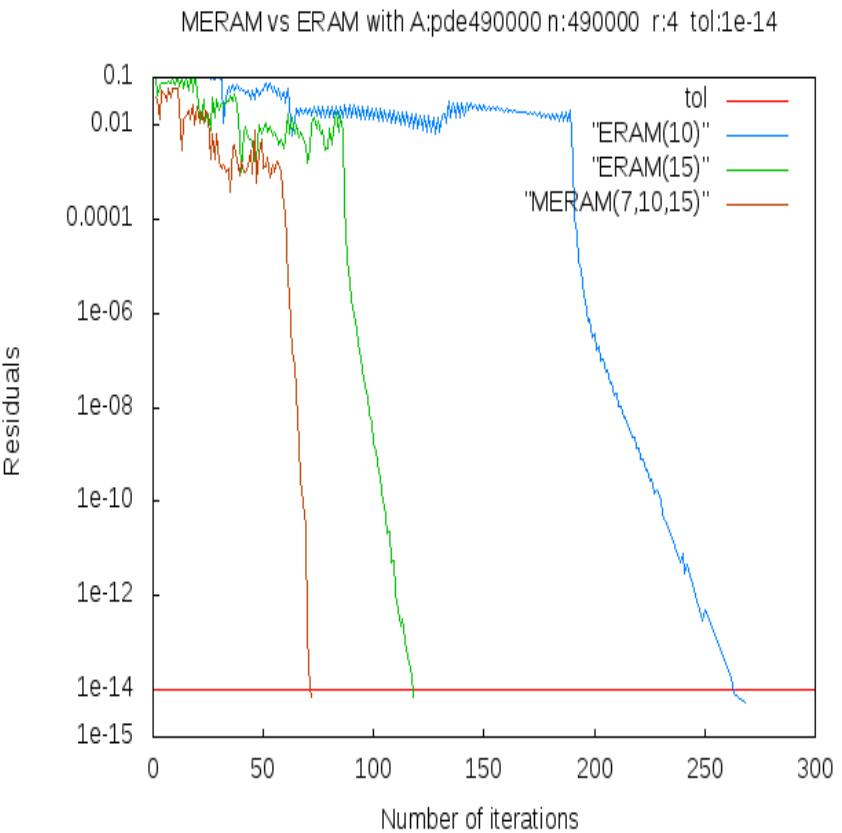


## Scalability the solution : the matrix size with PETSc\YML (matrix pde490000)

PETSc/MPI

Nahid Emad's poster

PETSc/YML

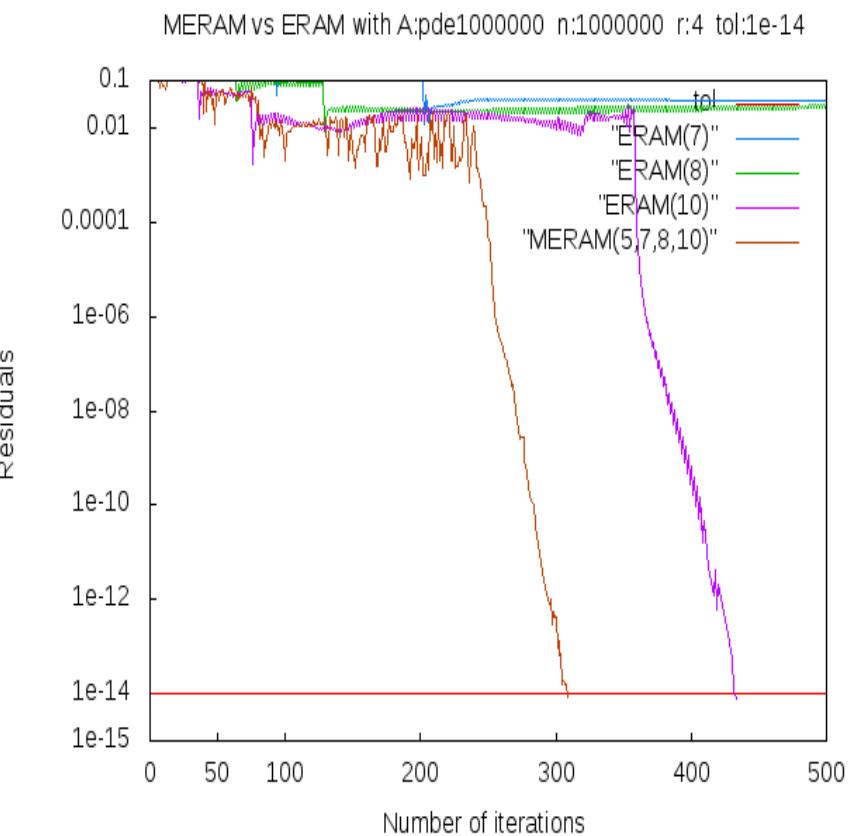


## Scalability the solution : the matrix size with PETSc\YML (matrix pde1000000)



Nahid Emad's poster

### PETSc/YML



# Outline

- Introduction
- Multilevel programming paradigms
- The YML environment and experiments
- YML/XMP/starPu and the Japanese-French FP3C project
- Reusable library, experiment with PETSc, SLEPc and YML
- Conclusion

# Conclusion

- Multilevel programming paradigms would be a potential solution for Exascale computing
- Still a lot of work to do
- Experiment in PRACE and K are scheduled
- Discussions