First contact with YML: Training with a Virtual Machine

T.DUFAUD

LI-PaRAD/Maison de la Simulation

YML VM - 2017/04/19

Plan

1. Communication
2. YML for users
   - Developing an application
   - Multilevel programming
   - Methodology
   - Creation of components
   - Graph
3. The Virtual Machine
   - Settings
   - Tutorial
   - Training with YML-XMP
4. Appendix I - YML Language
5. Appendix II - Gauss-Jordan Method

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Communication

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Acces to YML

- Official website : http://yml.prism.uvsq.fr

Collaboration

- **A code in France**
  - yml-2.2.1.tar.gz
  - version from yml-1.0.0 to yml-1.0.7 (2011) tar.gz
  - **git repository** for yml-1.0.0 to yml-1.0.7 http://sourceforge.net/p/yml/code/ci/master/tree/
  - yml-2.3.0 on gitlab.maisondelasimulation

A virtual machine to discover YML

- First version for **Workshop (JDev 2015, France, July 2015)** for users on yml.prism.fr
- June 2016: new version of VM for developers (testing during summer internships)
- **April 2017: The VM includes the last YML-XMP** development with the backend omniRPC-MPI and the use of XMP in YML
- Upgrade of the system and english version.

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

YML for users

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Developing an application with YML

- Create tasks defining **components** (using XML language)
- Write an application with a **graph of tasks with Yvette language**
- Execute the application in distributed environment

Advantages

- Components approach
  - modularity
  - reusability
- Workflow programming:
  - facilitate the expression of parallelism for user
  - close to computational methods (Algorithm)
  - high level language (Yvette)
  - deduce dataflow

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Developing an application with YML

1. Create tasks defining **components** (using XML language)
2. Write an application with a **graph of tasks with Yvette language**
3. Execute the application in distributed environment

1. Task definition

A task is a service define by:

- **interface**: «abstract» component
  - Input / Output data
- **realization**: «implementation» component
  - C/C++/XMP-C/XMP-FORTRAN
  - libraries etc.

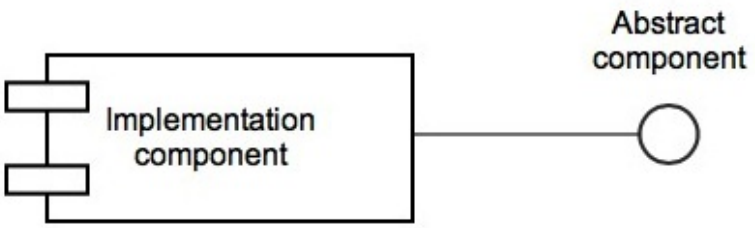


Figure : A service

Component approach

- modularity
- reusability

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Developing an application with YML

1. Create tasks defining **components** (using XML language)
2. Write an application with a **graph of tasks with Yvette language**
3. Execute the application in distributed environment

1. Task definition

A task is a service define by:

- **interface**: «abstract» component
  - Input / Output data
- **realization**: «implementation» component
  - C/C++/XMP-C/XMP-FORTRAN
  - libraries etc.

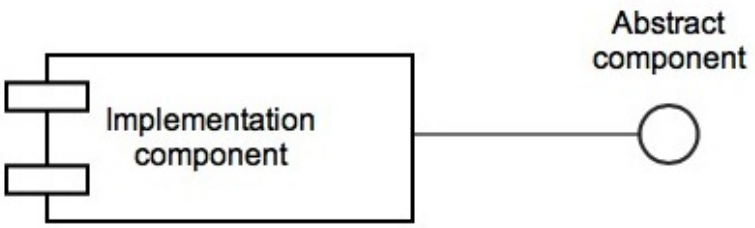


Figure : A service

Component approach

- modularity
- reusability

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

2. Workflow programming

- **facilitate the expression of parallelism for user**
- close to computational methods (Algorithm)
- high level language (Yvette) -> workflow
- deduce dataflow
- $\Rightarrow$ **enable optimization combining two aspects: workflow and dataflow.**

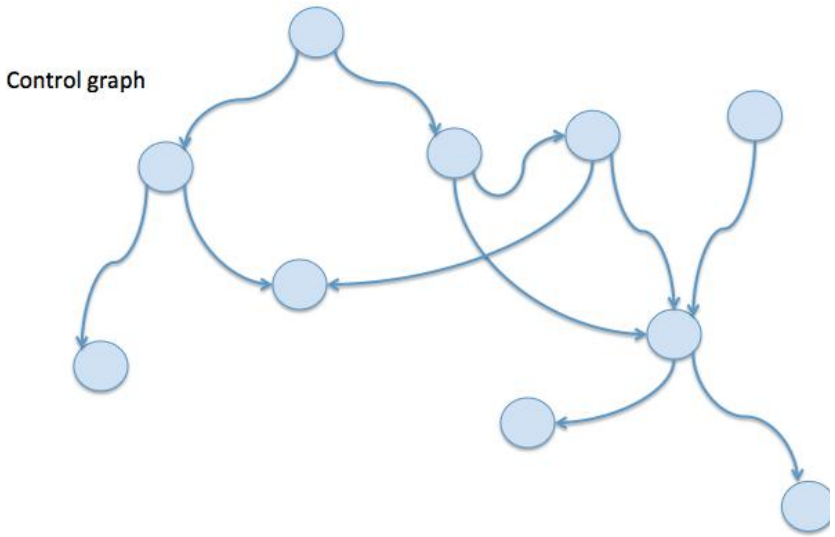


Figure : Graph of tasks

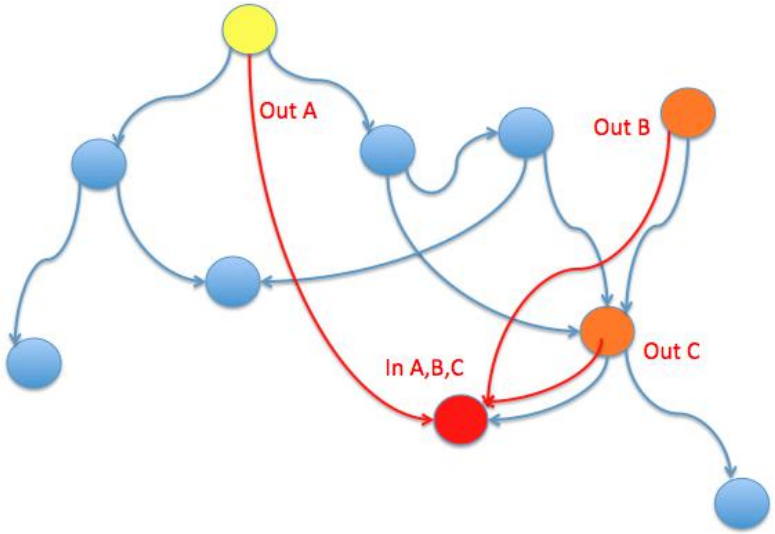


Figure : Deducing dataflow

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

3. Compilation and Execution

- Components are registered in the development catalog (yml_component)
- Components implementation are compiled in execution catalog (yml_component)
- The application (graph) is compiled and give information to a scheduler (yml_compiler)
- The yml_scheduler command enable to run the application with a given middleware

See YML Architecture (Next slide)

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

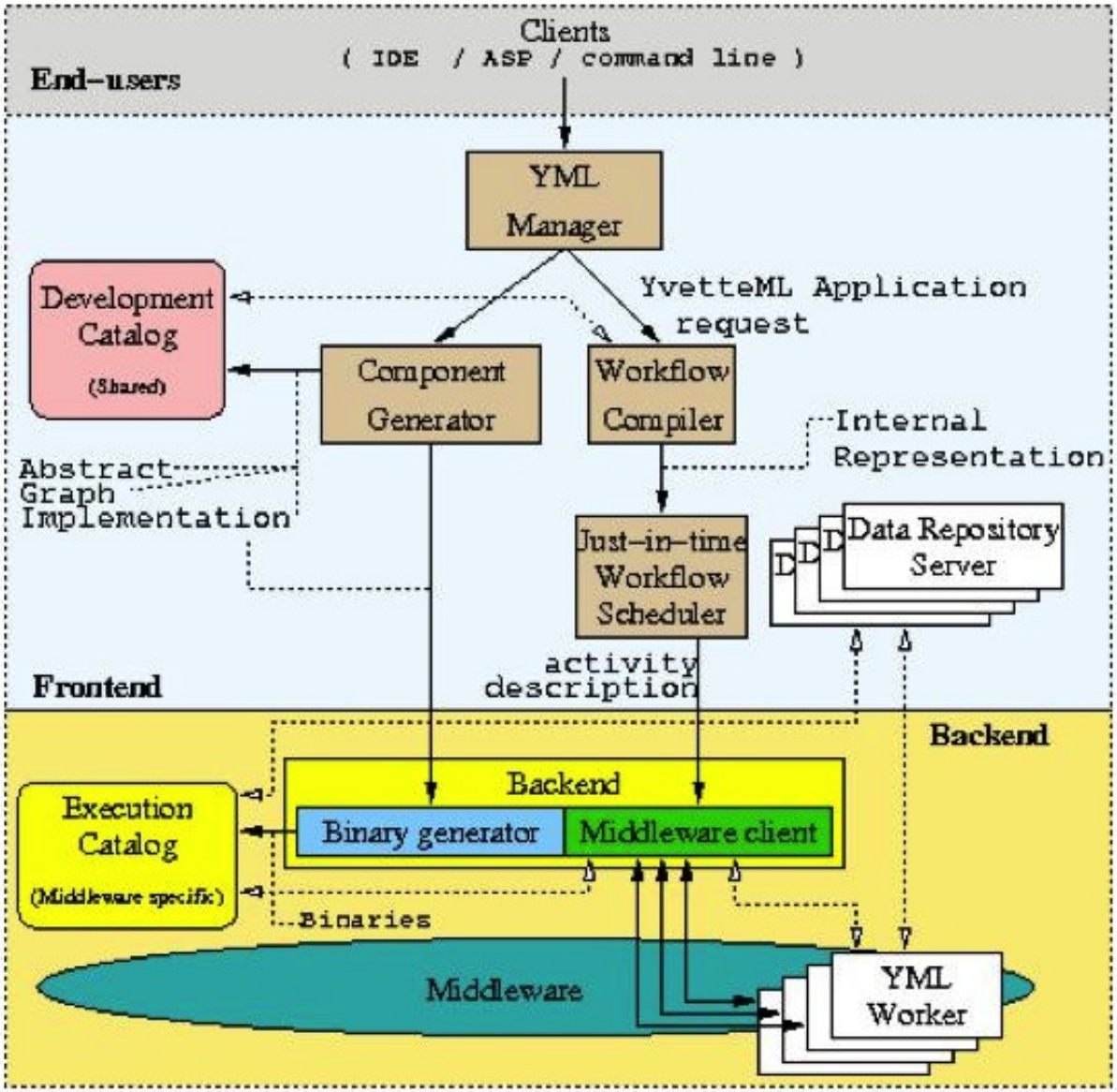


Figure : YML Architecture

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Note on backend interface

- Backend interface provides information from catalogs
- Middleware uses this information
- **Separation of Data definition, computation and communication**

yml_scheduler

1. get information on application (graph, binaries)
2. Scheduling loop
   1. Scheduler schedule pending task (with BackendManager)
   2. BackendManager execute task (with Backend)
   3. Backend execute Implementation

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Multilevel programming paradigm

- **High level**: communication inter nodes/group of nodes
- **Low level**: group of nodes / cores

Multilevel programming

- **High level: YML** - coarse grain parallelism - asynchronous communication
- **Low level: XMP** programming with pragma, PGAS language

Backend and compiler on last VM (2017-04)

- omnicompiler
- A component can be a subgraph (component «graph» (non functional))

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

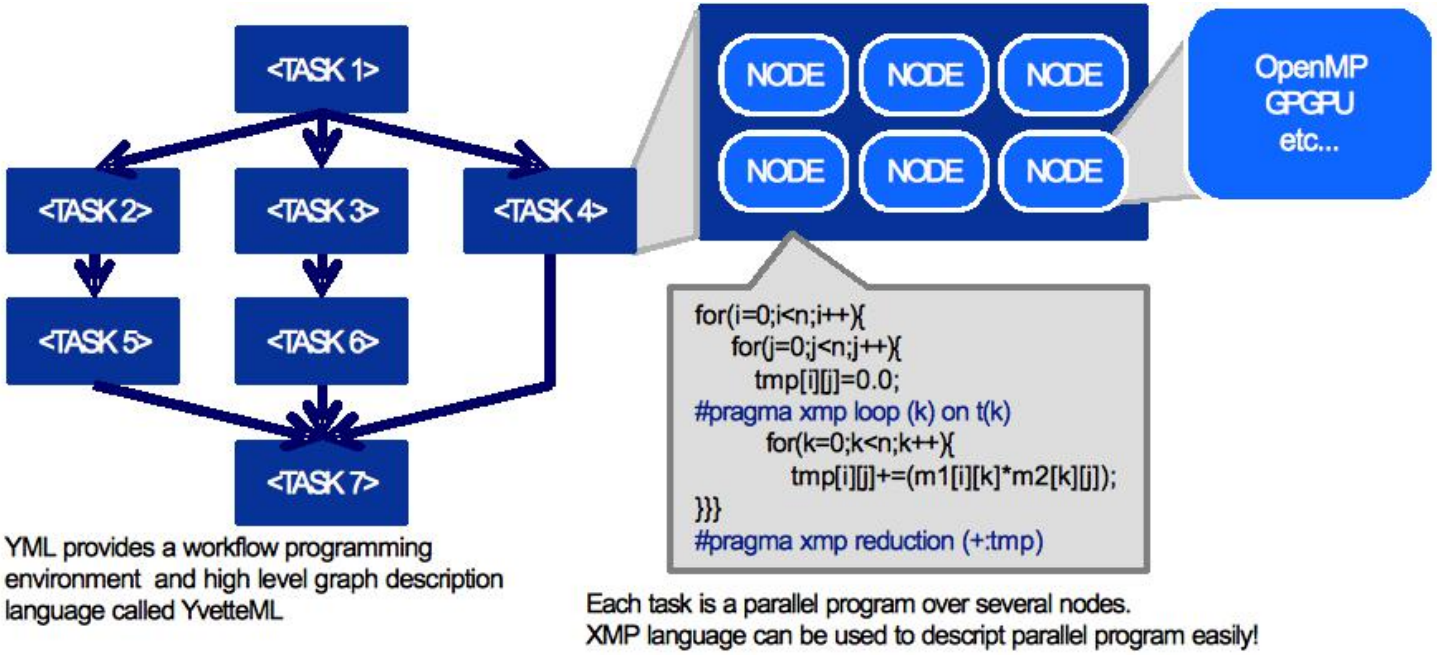


Figure : Multilevel programming

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

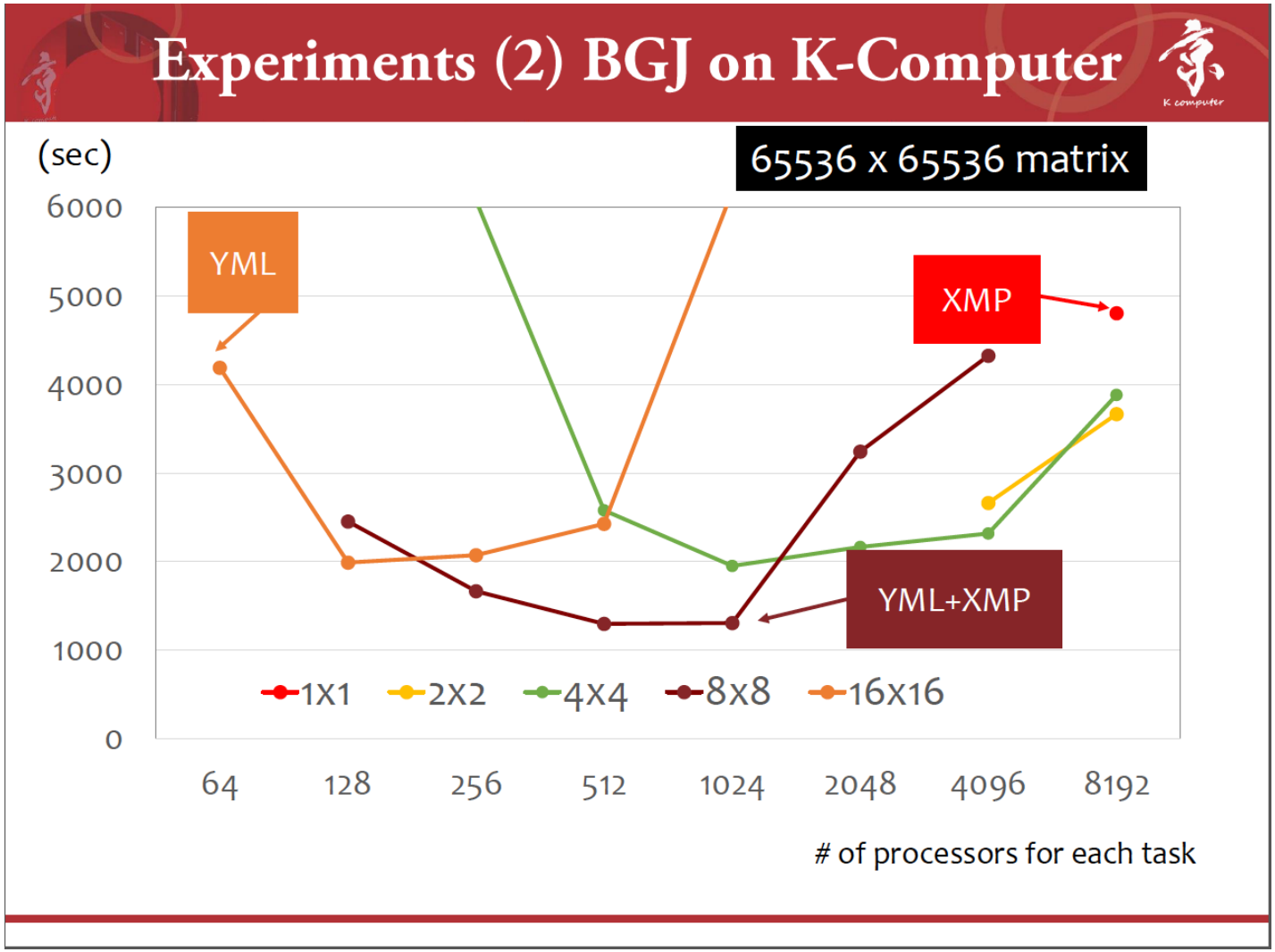


Figure : A^{-1} computation with Gauss-Jordan Block (YML-XMP on K computer (Japan)

Results given by Miwako Tsuji.

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Methodology

- Create abstract components (Interfaces)
- Create implementation components (Interfaces's realization)
- Write graph of control with Yvette
- Register components and compile application.
- Execute application.

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Abstract component

- XML code: encapsulation **component**
- Type: *abstract*
- Parameters encapsulation: **params**
  - *name*
  - *type*: real, integer, ... or defined by user
  - *mode*: in, out, inout

Syntaxe

```
<?xml version="1.0"?>
<component type="abstract" name="test"
description="This␣component␣sums␣two␣double␣and␣stores␣the␣
    result␣in␣res">
    <params>
       <param name="res" type="real" mode="out" />
       <param name="a1"  type="real" mode="in" />
       <param name="a2"  type="real" mode="in" />
     </params>
</component>
```

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Concrete component

- XML code: encapsulation **component**
- Type: *impl*
- Declaration of its interface: *abstract*
- Code encapsulation: **impl** and specification of language *lang* and libraries *libs*
  - **header**: contains header
  - **source**: contains source code

Syntax

```
<?xml version="1.0" ?>
<component name="test_impl" abstract="test" type="impl">
    <impl lang="CXX" libs="">
        <header></header>
        <source>
            res = a1 + a2;
        </source>
        <footer></footer>
    </impl>
</component>
```

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Defining an application

- XML code: encapsulation **application**
- I/O in **params**
- Graph written in Yvette **graph**

Yvette Language

- **Parallel Section:** par section1 // ... // section N endpar
- **Sequential loop:** seq (i:=begin;end) do ... enddo
- **Parallel loop:** par (i:=begin;end) do ... enddo
- **Conditional structure:** if (condition) then ... else ... endif
- **Synchronization:** wait(event) / notify(event)
- **Component call:** compute NameOfComponent(args,...,...)

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Syntaxe

```
  <?xml version="1.0"?>
    <application name="test2">
    <description>
A starter example of an YML application with parameters
    </description>
    <params>
        <param name="v1" type="real" mode="in" />
        <param name="v2" type="real" mode="in" />
        <param name="result" type="real" mode="out" />
    </params>
    <graph>
        compute test(result, v1, v2);
        compute test(result, result, result);
    </graph>
 </application>
```

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Working Directories

- ***var/*** in YML installation directory
- ***DefaultDevelopmentCatalog*** contains .xml files from analysis of created components.
- ***DefaultExecutionCatalog*** contains **include** and **dependancies**, *libs* needed for compilation of components, and **index** files, path to binaries of concrete components (in *binaries*).
- ***binaries*** contains *source codes* extracted from components and their *binaries*.

Data types

- Data are managed by a component called *Data Repository*
- A *worker* communicates with *Data Repository*
- communication by *packing*
- **definition of import/export in header (*.type.hh)** to put in *DefaultExecutionCatalog*

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Libraries

- name of the dependency in component **impl** with *libs=""*
- **in *DefaultExecutionCatalog* create a file <libs-name>.txt** which contains:
  - CFLAGS
  - binaries and libs to link

Calling codes

- It is possible to call a precompiled code
- Enable the portability of codes in YML
- **deploy existing application on distributed system writing algorithms with Yvette**

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Register and compile components

- **component file**: <name_file>.query
- Register abstract component: yml_component [option] <name_file>.query
  - .xml file in **DefaultDevelopmentCatalog**
  - Info on standard ouput
- Compile and register concrete component: yml_component [option] <name_file_impl>.query
  - source code and binaries in **DefaultExecutionCatalog**
  - Info on standard ouput

Compilation et execution

- yml_compiler [option] <name_application>.query
  - create binary : <name_application>.query.yapp
  - Info on standard ouput
- yml_scheduler <name_application>.query.yapp
  - execute application with desired backend **configure yml.xcf**
  - create file runXXX which contains execution info (data, log, description...)

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

The Virtual Machine

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Information

- Virtual box
- Ubuntu system
- YML installed

Documents

- This presentation - **from Workshop (JDev 2015, France, July 2015)**
- PhD thesis and articles in the document directory of the VM.
- Specific user's documentation for YML-XMP by M. Tsuji (Riken AICS)
- Maxime Hugues's code (Gauss-Jordan) in Tutorial.

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Working Environment

- Linux distribution **Ubuntu 64 bits (14.04)** for VirtualBox
- **yml-2.3.0** and dependancies installed in /home/ymlusr/local
- **Process backend (POSIX)**
  - enable to use YML-C on every system
  - used for debugg
  - to use for the tutorial.
- **Mpi backend (OmniRPC-MPI)**
  - enable to use YML-XMP
  - to use for examples in yml-2.3/samples/fp2c*
- Command line
- Help and documentation in files **README and .txt**
- A directory dedicated to **tutorials for YML-C**: cd $TUTO
- A directory dedicated to the **solution** of tutorials for YML-C: cd $TUTOSOL
- Samples for YML-XMP in YMLEnvironment/yml-omrpc-xmp/yml-2.3/samples
- Shell bash
- Edit: vi, emacs, gedit

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Tutorials

- in $TUTO
- **3 exercises** to discover how to use YML
- **Interface for C language**
- Sequential code for each component
- ⇒ coarse grain parallelism
- Documentation in README and .txt (English)
- Solution in directory $TUTOSOL

YML configuration

- file .xcf in $PREFIX/etc/yml
- in yml.xcf:
  - for module set the value **ProcessBackend**
  - for init set the value **process**

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Tutorial 1

- cd $TUTO/tutorial1-sum
- Instruction tutorial1-sum.txt
- Perform **the sum of 2 integers** (several times))

Goals

- Write an abstract component
- Write a concrete component
- Register components
- Compile and execute a simple application
- Manipulate I/O with the yml_parameter command

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Tutorial 2

- cd $TUTO/tutorial2-dsort
- Instruction tutorial2-dsort.txt
- **Sort a distributed array of integers**

Goals

- Understand the conception of the application
- Define data types for YML
- Learn the Yvette language to express the graph
- Apply the skills learnt in tutorial 1

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Tutorial 3

- cd $TUTO/tutorial3-Gauss-Jordan
- Instruction tutorial3-Gauss-Jordan.txt
- **Inverse Matrices with Gauss-Jordan block method**

Goals

- Understand the conception of the application
- Integrate several implementations for a same component
- Link libraries
- Express parallelism with 2D graphs or 3D graphs for the same algorithm
- Apply the skills learnt in tutorials 1 and 2

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Examples

- in YMLEnvironment/yml_omrpc_xmp/yml-2.3.0/samples
- Sum of matrices in fp2c_tutorial1
- Block Gauss-Jordan in fp2c_bgj
- Parallel code for each component using XMP

YML configuration

- file .xcf in $PREFIX/etc/yml
- in yml.xcf:
  - for module set the value **MpiBackend**
  - for init set the value **mpi**

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Appendix I - YML Language

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

- **Component call**: This is the most important construction. It corresponds to the execution of a component on the remote peers. The component call is composed of a scheduling policy that can be use in the scheduler (or not) the name of an abstract component and a list of parameters. Parameters can be a data or a constant value. In the next example there is two component calls both take three parameters Each parameter represents a data.

```
compute test(res, 1, 2);
compute MatrixVectorProduct(A[1], x, y[1]);
```

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

- **Parallel section**: In YvetteML like most programming language the control flow is sequential until you explicitly mark section as parallel. YvetteML supports two kind of parallel sections. The generic form is illustrated below. This example calls three times the test component in parallel. Parallel section are separated by a // separator. Multiple statement can compose a parallel section. Endpar keyword is an implicit synchronization.

```
par
  compute test(res[1], 3, 4);
//
  compute test(res[2], 4, 3);
//
  compute test(res[3], 4, 5);
endpar
```

The second form of parallel section as the same semantics but it make use of iterator to control the number of parallel branch executed in parallel. The example below execute 81 component calls organized in a grid of 10 by 10. The diagonal cells of the grid are ignored.

```
par (i:= 1; 10) (j:= 1;10) {i neq j}
do
  compute test(Res[i][j], i, j);
enddo
```

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

- **Sequential loop**: You can also express sequential loop controled by an iterator using a form similar to the parallel iterated loop describe previously. The example does the same as before but sequentially.

```
seq (i:= 1; 10) (j := 1; 10) {i neq j}
do
  compute test(Res[i][j], i, j);
enddo
```

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

- **Conditional**: YML allows the definition of conditional branches based on a boolean condition. The condition can use the operator available in the language as well as any user defined function. The example bellow does the same as the sequential loop presented previously using a conditional construction to prevent the generation of the diagonal elements of the cell. It is possible to define alternate branch using else keyword like in the second example.

```
seq (i:= 1; 10) (j := 1; 10)
do
  if i neq j then
    compute test(Res[i][j], i, j);
  endif
enddo
```

```
seq (i:= 1; 10) (j := 1; 10)
do
  if i neq j then
    compute test(Res[i][j], i, j);
  else
    compute test(Res[i][j], j, i);
  endif
enddo
```

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

- **Events manipulation**: The events can be explicitly manage using two constructions called notify and wait. The first construction set a list of events. The second one is used to wait until a boolean expression of events is true. The parallel loop describe in the following shows the way to create sequential loop in YML private releases.

```
notify(loop[0]);
par (i := 1; 10)
do
  wait (loop[i - 1]);
  compute test(Res[i], i, i);
  notify(loop[i]);
enddo
```

The second example illustrates the use of explicit synchronization between parallel branches. component A and B are executed in parallel while C is waiting for A and B to terminate D is executed independently.

```
par
   compute A(a);
   notify(eva);
//
   compute B(b);
   notify(evb);
//
   wait(eva and evb);
   compute C(c, a, b);
//
   compute D(d);
endpar
```

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

Appendix II - Gauss-Jordan Method

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

GAUSS-JORDAN method

The Gauss-Jordan method for the solution of a linear system $Au = b$ consists in the following steps:

1. Successive elimination: construct a non-singular matrix M such that $B = MA$ be `diagonal`.
2. Compute $y = Mb$.
3. Solve: $MAx = Mb$, with MA Diagonal

The first step is the diagonalization of a non-sigular matrix A: Let $A = A_1$, $A \in \mathbb{R}^{n \times n}$. We look for M such that MA be `diagonal`.
At the end of the $(k-1)$-th step we have a matrix A_k:

$$A_k = E_{k-1} P_{k-1} \dots E_2 P_2 E_1 P_1 A_1 = \begin{pmatrix} a^k_{1,1} & & & a^k_{1,k} & \dots & a^k_{1,n} \\ & a^k_{2,2} & & a^k_{2,k} & \dots & a^k_{2,n} \\ & & \ddots & \vdots & & \vdots \\ & & & a^k_{k,k} & \dots & a^k_{k,n} \\ & & & \vdots & & \vdots \\ & & & a^k_{n,k} & \dots & a^k_{n,n} \end{pmatrix}$$

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

We choose one of the $a^k_{i,k}$, $k \leq i \leq n$, non-zero element and take it as a `pivot`. This corresponds to the multiplication of A_k by P_k. $P_k A_k = (\alpha^k_{i,j})$ is such that $\alpha^k_{k,k} \neq 0$. Then E_k is as follow:

$$E_k = \begin{pmatrix} 1 & & & \frac{-a^k_{1,k}}{\alpha^k_{k,k}} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & \frac{-a^k_{k-1,k}}{\alpha^k_{k,k}} & & & \\ & & & 1 & & & \\ & & & \frac{-a^k_{k+1,k}}{\alpha^k_{k,k}} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & \frac{-a^k_{n,k}}{\alpha^k_{k,k}} & & & 1 \end{pmatrix}$$

After the $(n-1)$-th step,

$$A_n = MA, \text{ with } M = E_{k-1} P_{k-1} \ldots E_2 P_2 E_1 P_1$$

be diagonal.

`Remarque` The Gauss-Jordan method is used to compute the invers of a matrix. Let $A^{-1} = C = [c_1, c_2, \ldots, c_n]$ where c_j is the j-th column of matrix C. Computing the inverse of A is equivalent to the solution of the following n linear systems:

$$(MA)c_j = Me_j \text{ with } 1 \leq j \leq n$$

where e_j is the j-th vector of the identity matrix ($e_j(i) = 0$ if $i \neq j$ and $e_j(j) = 1$).

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

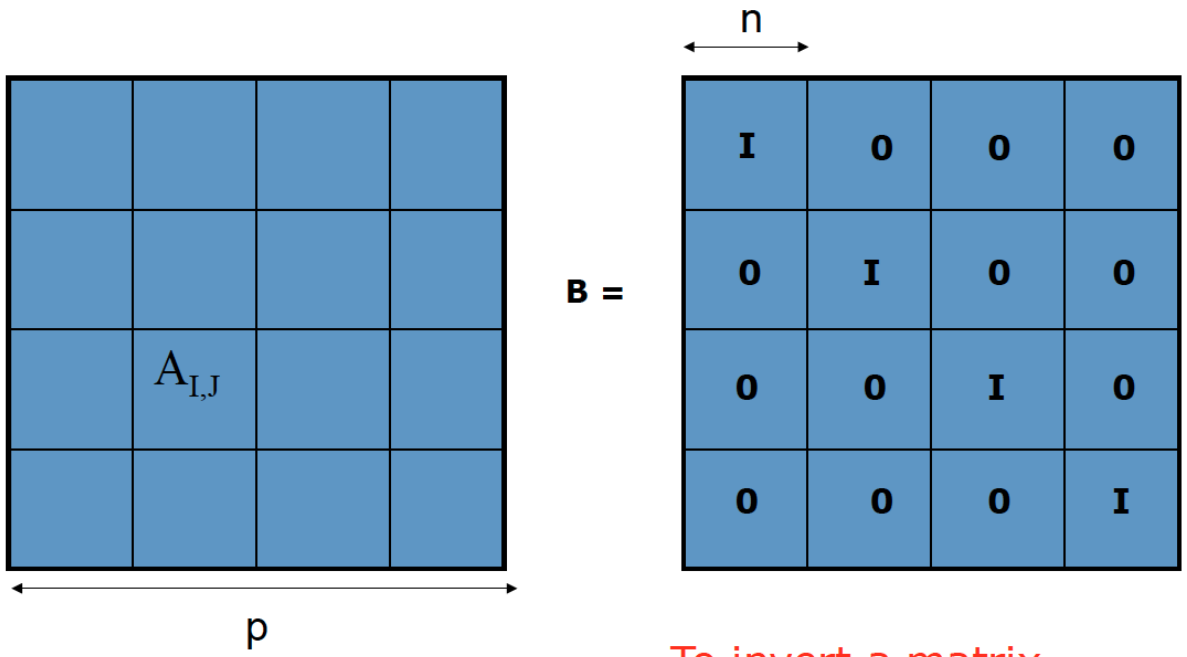
Algorithm 1 The block-based Gauss-Jordan matrix inversion

Input: A (partitioned into $p \times p$ blocks)
Output: $B = A^{-1}$

```
For k = 0 to p − 1
  B_kk = A^{-1}_kk
  For i = k + 1 to p − 1 (1)
    A_ki = B_kk × A_ki
  End For
  For i = 0 to p − 1 (2)
    If (i ≠ k)
      B_ik = −A_ik × B_kk
    End If
    If (i < k)
      B_ki = B_kk × B_ki
    End If
  End For
  For i = 0 to p − 1 (3)
    If (i ≠ k)
      For j = k + 1 to p − 1
        A_ij = A_ij − A_ik × A_kj
      End For
      For j = 0 to k − 1
        B_ij = B_ij − A_ik × B_kj
      End For
    End If
  End For
End For
```

Figure : Block Gauss-Jordan Algorithm, M. Hugues *et.al.*

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method



A B = B A = I
Block Gauss-Jordan
Matrix size = N = p n

To invert a matrix
$2N^3$ operations

Figure : Block Diagonalization and complexity (by S. Petiton)

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

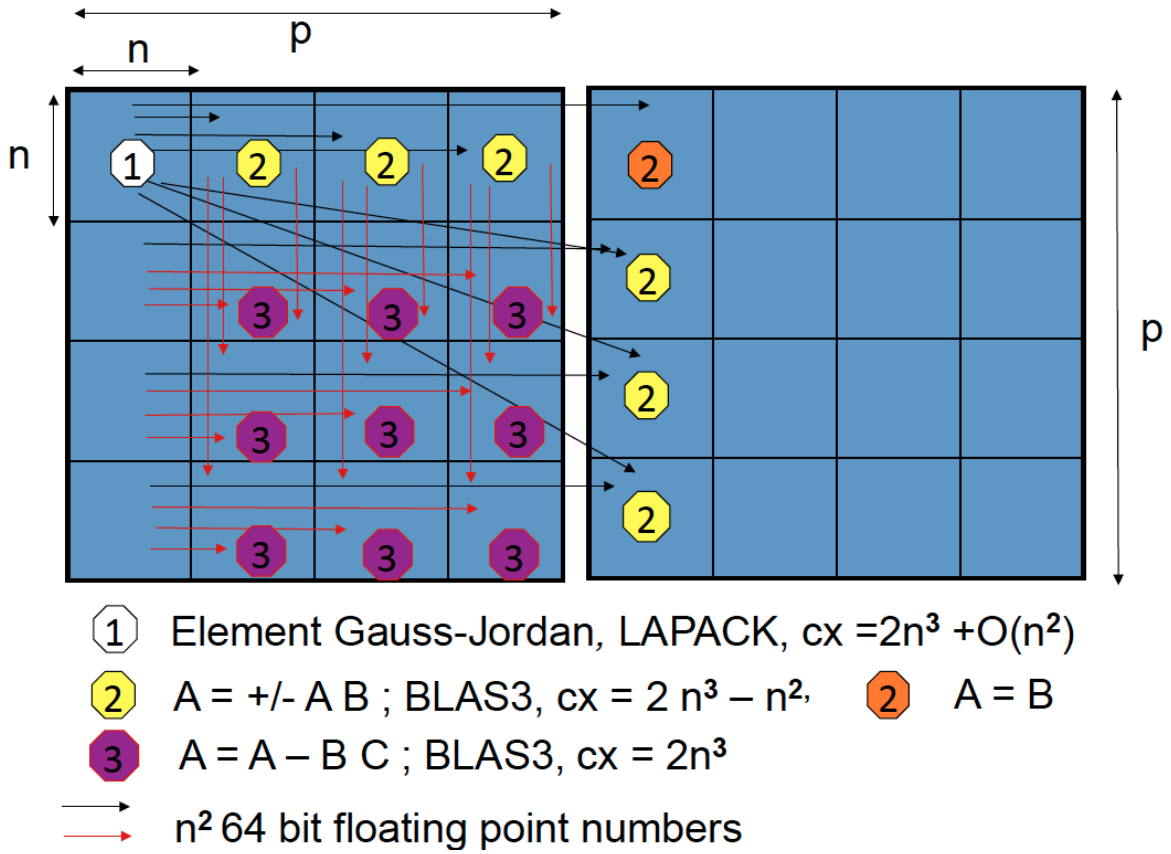
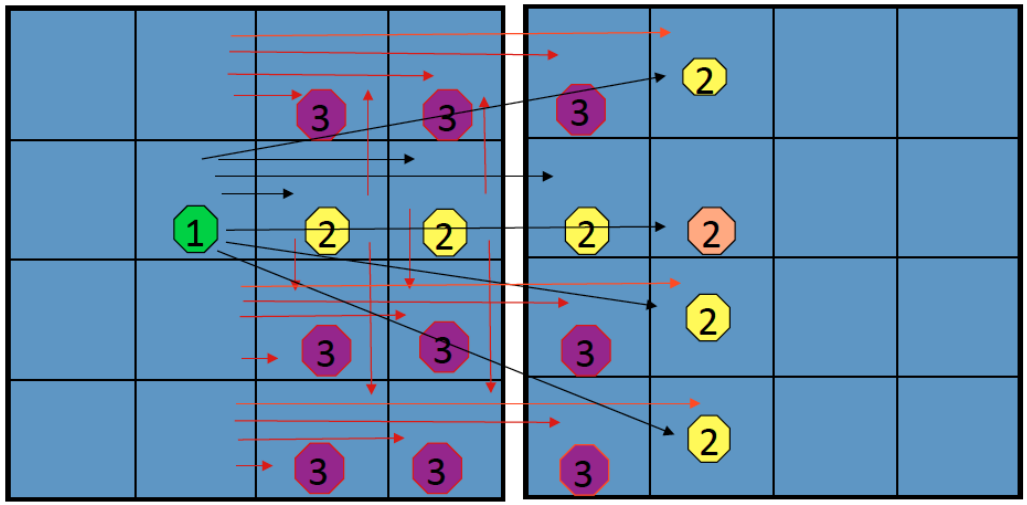


Figure : Task execution according to data distribution: Step 1 (by S. Petiton)

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method



Each computing task : 1 up to 3 blocks
maximum n < (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

August, 14 Exascale Workshop, Portland

Figure : Task execution according to data distribution: Step 2 (by S. Petiton)

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

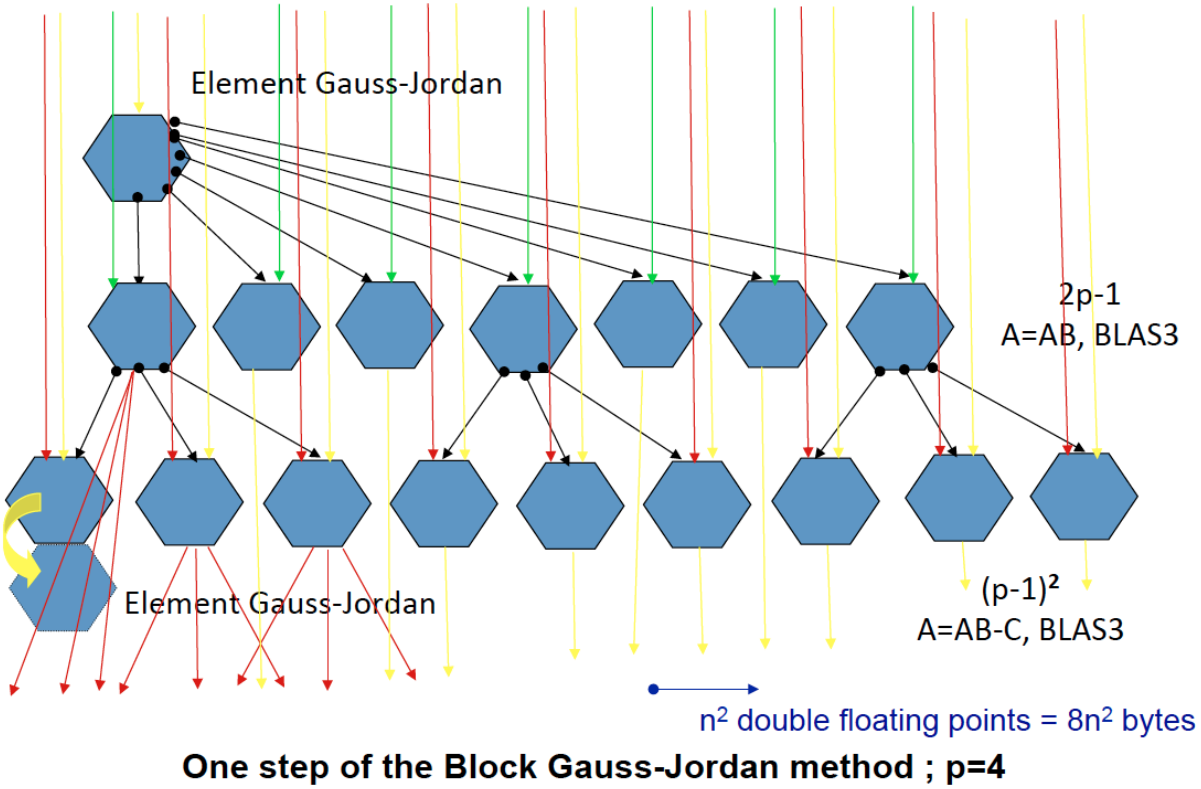
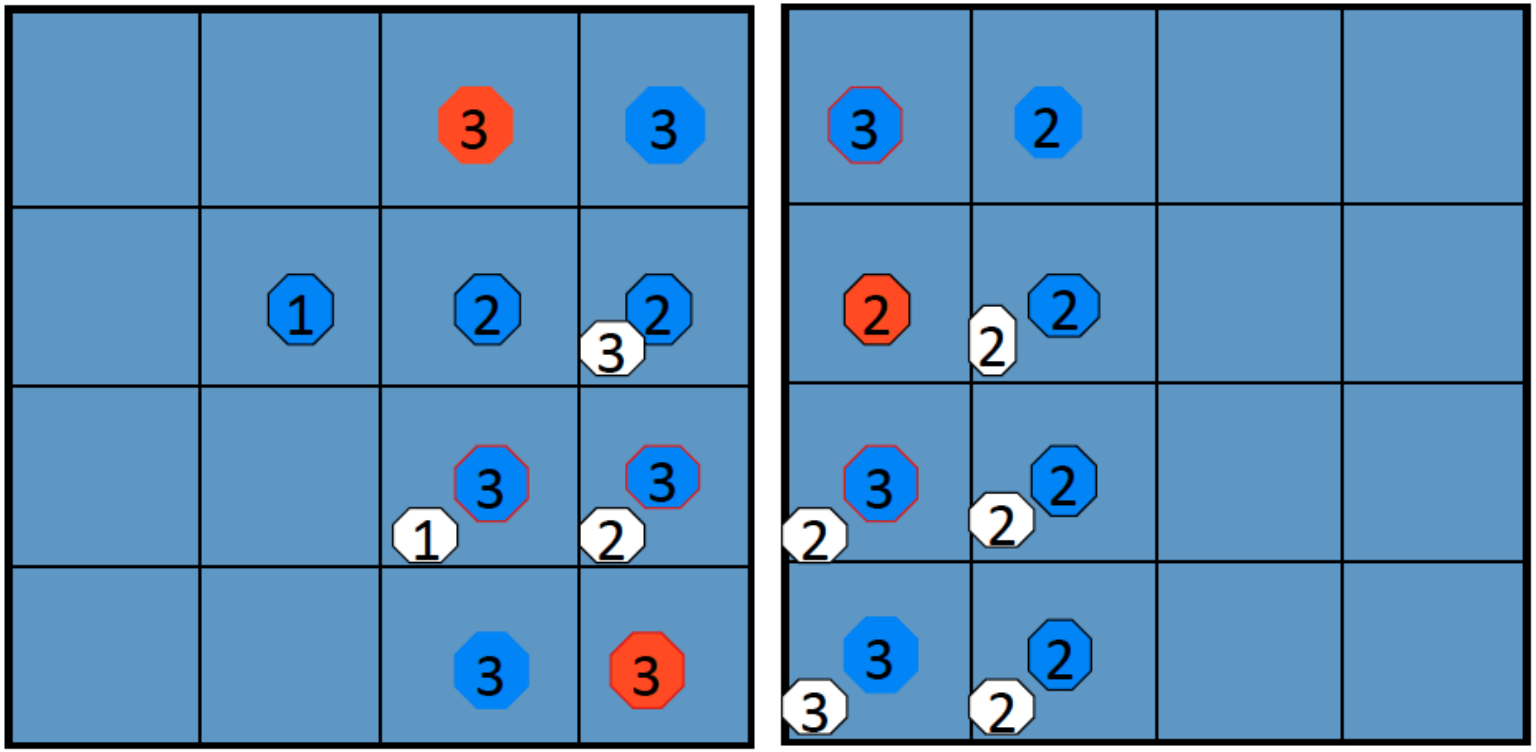


Figure : Workflow of one step (S. Petiton)

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method



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).

Figure : Parallel tasks (S. Petiton)

References (Selection)

Communication
YML for users
Developing an application
Multilevel programming
Methodology
Creation of components
Graph
The Virtual Machine
Settings
Tutorial
Training with YML-XMP
Appendix I - YML Language
Appendix II - Gauss-Jordan Method

- **YML-XMP**
  - S. Petiton, M. Sato, N. Emad, C. Calvin, M. Tsuji and M. Dandouna, Multi level programming Paradigm for Extreme Computing, Published online: 06 June 2014
- **YML**
  - N. Emad, O. Delannoy, and M. Dandouna. Numerical library reuse in parallel and distributed platforms. In the proceedings of 9th International Meeting on High Performance Computing for Computational Science, VecPar'10, Lawrence Berkeley National Labratory, California, USA, June, 22-25 2010
  - L. Choy, O. Delannoy, N. Emad and S. Petiton - Federation and abstraction of heterogeneous global computing platforms with the YML framework, in The Third International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC-2009), March 2009, Japan
  - O. Delannoy, YML: A scientific Workflow for High Performance Computing, Ph.D. Thesis, Septembre 2008, Versailles
- **Algorithms**
  - Shang, Ling; Petiton, Serge; Hugues, Maxime; ,A new parallel paradigm for block-based Gauss-Jordan algorithm,"Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on",193-200,2009,IEEE

Enjoy the discovery!

YML VM

April 19, 2017