Design of Data Management for Multi SPMD Workflow Programming Model

T. Dufaud, M. Tsuji, M. Sato

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

RIKEN

XMP Workshop, Tsukuba, 2018-11-1

1 Motivation

2 Data exchange issues and model

3 Implementation through YML user interface

4 Development of Block Gauss-Jordan Algorithm

Motivation

Multi SPMD for large scale computing

Why Multi SPMD

- Flat MPI reaches a limit
- Simulation code may consists in several parallel programs
- Each SPMD is an independant program $\rightarrow$ component approach,reusability

Programming environment

- Dominant execution model approach: MPI + X
- Coupling between models and apps involves I/O
- **Proposed approach for 2-level programming model**
  - Workflow programing with task dependencies
  - Single Program Multiple Data at the fine layer
  - Separation of data and computation
- **Programing environment YML-XMP**
  - XMP is a PGAS language`http://www.xcalablemp.org`
  - YML: graph of task, each task is described by a component, `http://www.yml.prism.uvsq.fr`

Multi SPMD model

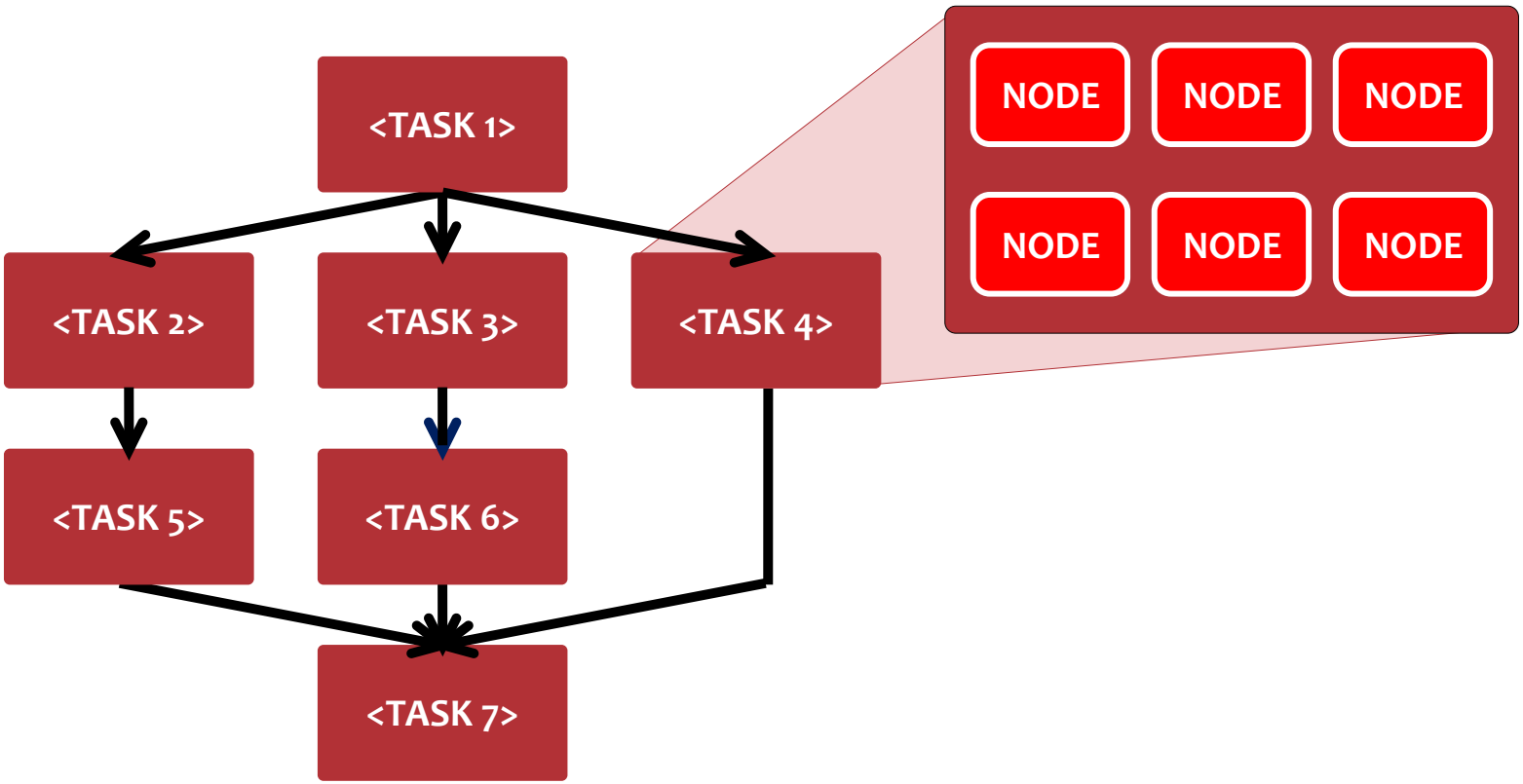


Figure: Multi SPMD - Graph of task (YML) task following SPMD model on distributed nodes (XMP)

I/O based data exchange

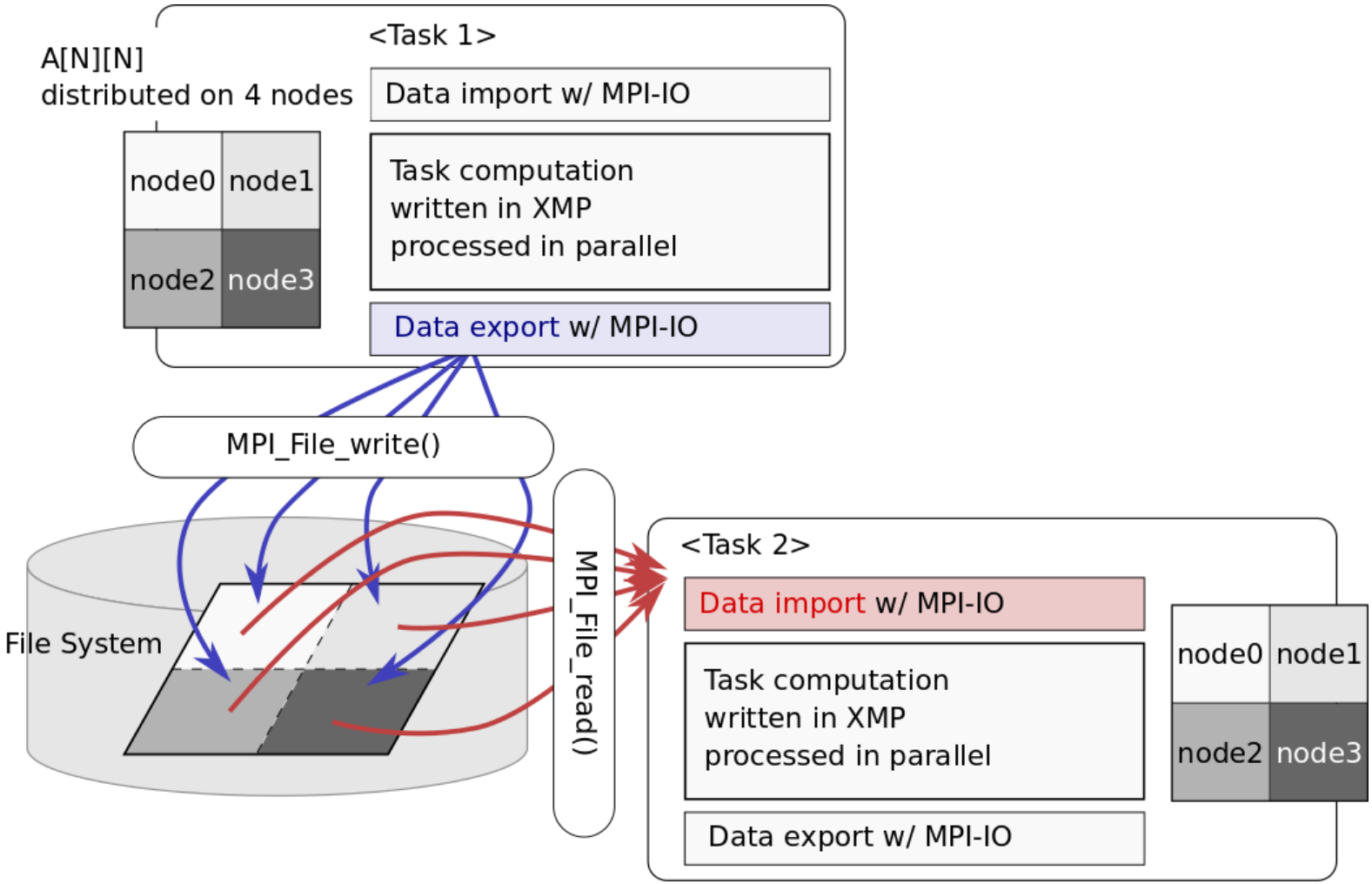


Figure: Import export based on I/O in YML-XMP

Data exchange issues and model

Data exchange for Multi SPMD

Context

- Parallel programming model on each level ?
- Separation of data and computation ?
- Data placement and dependencies ?
- Data persistency ?

Choices

- **Graph of tasks** on the higher level
- **SPMD** at the fine level
- Component approach at the higher level (large parallel tasks)
- Intensive data access and Persistency of certain data for fault tolerance

System quality

Assumption

- Data repository $\rightarrow$ store a global data, separate from computation
- High level $\rightarrow$ Global data (To be Imported/Exported)
- Low level $\rightarrow$ Group of local data
- Import/Export are synchronous

Quality

- High performance in case of intense computation
- Fault tolerance, persistency of important data

Key point of the system

Main design

- **Detection of data distribution** (Given by execution model)
- **Performance** of communication according to **data distribution**...
- ... and **architecture and middleware**
- adapt strategy at runtime(considering size of data and distribution)
- earn benefit of both dataflow and workflow knowledge (Drawing our inspiration from **M. Hugues *et. al.* ASIODS 2011**)
  - consider a **task scheduler and a data scheduler**
  - **interaction** between schedulers for optimization
  - $\Rightarrow$ **anticipate data migration, remapping of parallel data**

Remarks

- Component approach + ability to get information from interfaces
- $\Rightarrow$ enable integration of automatic decision tools

Target system

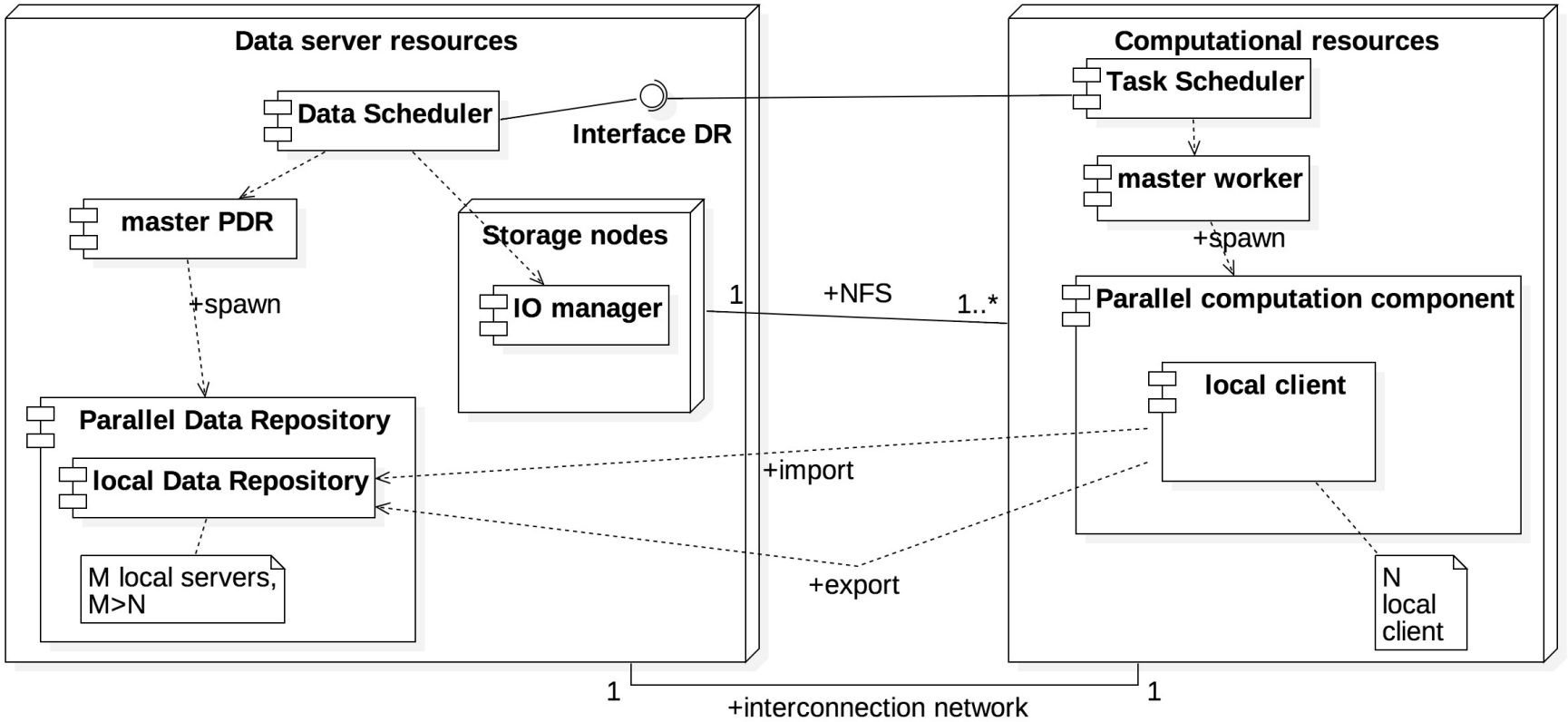


Figure: Target system

Implementation through YML user interface

First step implementation enable by YML front end

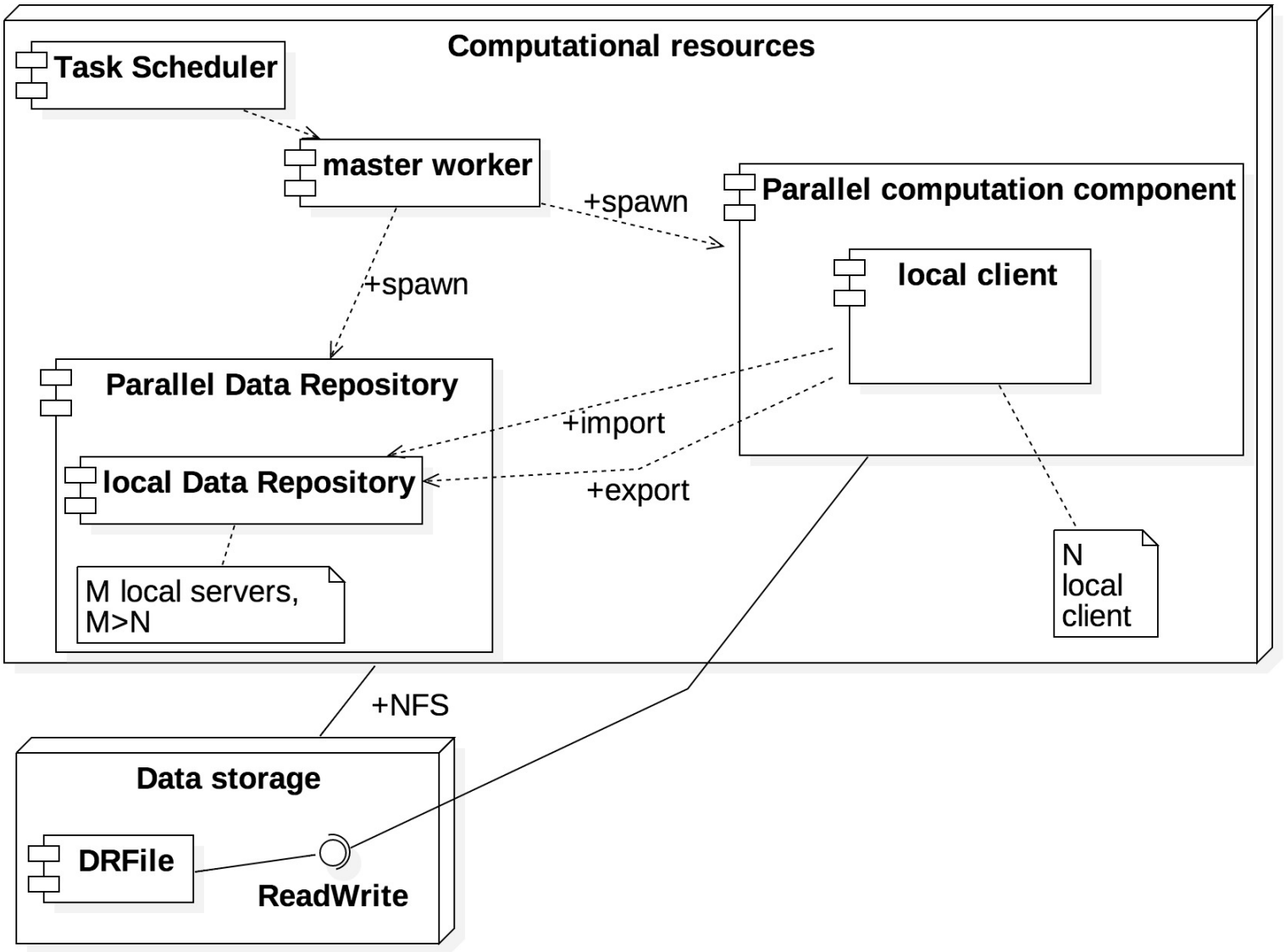


Figure: Integrate new data management through YML front end

I/O based data exchange

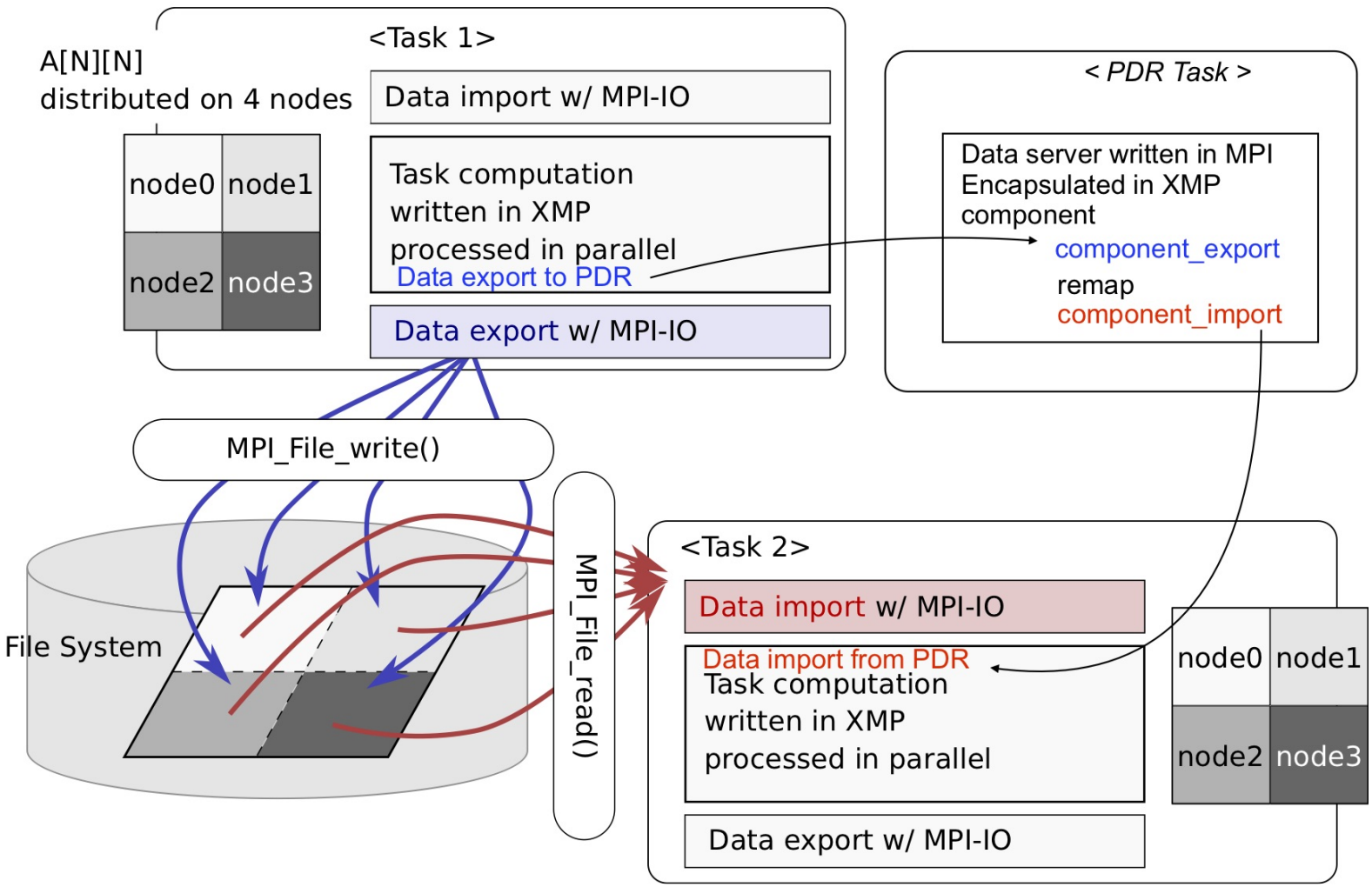


Figure: Import export with 2 strategies in YML-XMP

Integration of new PDR

```
par
  # ALLOCATE DATA IN PDR
  par(i:=0;blockcount-1)(j:=0;
       blockcount-1)
  do
    ij:=i*n+j; tidA[ij]:=0; tidB[ij]:
         =1;
    par
      compute allocateInPdr(ij,tidA[ij
           ],count);
    //
      compute allocateInPdr(ij,tidB[ij
           ],count);
    endpar
    notify(begin[ij]);
  enddo
//
  # RUN PARALLEL DATA REPOSITORY
  par(i:=0;blockcount-1)(j:=0;
       blockcount-1)
  do
    ij:=i*n+j; tidA[ij]:=0; tidB[ij]:
         =1;
    par
      compute pdr(ij,tidA[ij]);
    //
      compute pdr(ij,tidB[ij]);
    endpar
  enddo
//
  # ALGORITHM
  notify(end);
//
  # STOP PARALLEL DATA REPOSITORY
  wait(end);
  par(i:=0;blockcount-1)(j:=0;
       blockcount-1)
  do
    ij:=i*n+j;
    par
      compute stopPdr(ij,idA[ij]);
    //
      compute stopPdr(ij,idB[ij]);
    endpar
  enddo
endpar
```

Performance of our model implementation

Platform

- 92 nodes IBM cluster (iDataPlex dx360 M4 servers)
- 2 CPU Sandy Bridge E5-2670 (2.60GHz)
  - 8 cores per CPU / 16 cores per node
  - 32 GB RAM per node

Key points of evaluation

- (I) Import export by **one XMP task increasing load**
- (II) **Weak scaling** in case of one XMP task
- (III) **Multiple accesses** (N XMP tasks one PDR)

Performance of our model implementation I

Performance for Import/Export a real matrix

- $nt \times nt$ XMP matrix on 16 XMP nodes (**Fix number of MPI process**)
- D0: Current YML-XMP implementation using MPI-IO
- D3: No I/O, use MPI_Comm_connect (etc.) and MPI_Send / MPI_Recv
- 10 import/export

Local size	10×10	100×100	500×500	1000×1000
Design 0	0.10 s	0.18 s	0.87 s	2.98 s
Design 3	1.09 s	0.94 s	1.08 s	1.26 s

Table: 1 client of 16 XMP nodes : Time to perform 10 import/export, **Design 3 bounded by connection time**

Performance of our model implementation II

Weak scaling

- $nt \times nt$ XMP matrix
- D0: Current YML-XMP implementation using MPI-IO
- D3: No I/O, use MPI_Comm_connect (etc.) and MPI_Send / MPI_Recv
- **Increase XMP nodes**
- 100 import/export

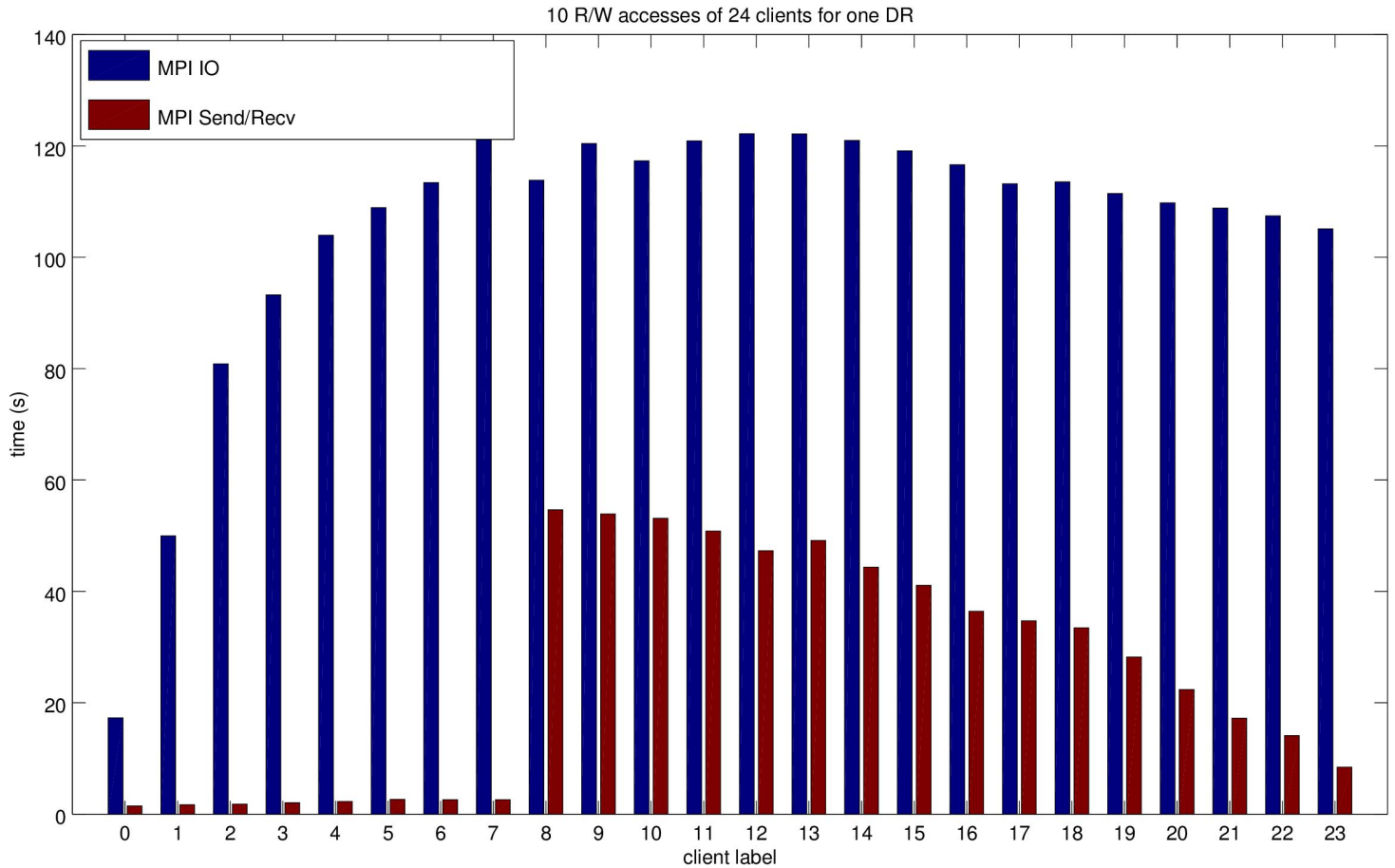
XMP grid size	4×4	5×5	6×6	7×7
Design 0	44.68 s	69.61 s	263.61 s	268.45 s
Design 3	17.8 s	21 s	20 s	22.3 s

Table: **Weak scaling**, increasing the grid size, with local array of size 1000×1000 : Time to perform 100 import/export

Performance of our model implementation III

Multiple accesses

- $nt \times nt$ XMP matrix on 16 XMP nodes
- **One Data Repository Multiple Task R/W**



Performance of our model implementation III (2)

Multiple accesses

- $nt \times nt$ XMP matrix on 16 XMP nodes
- **One Data Repository Multiple Task R/W**

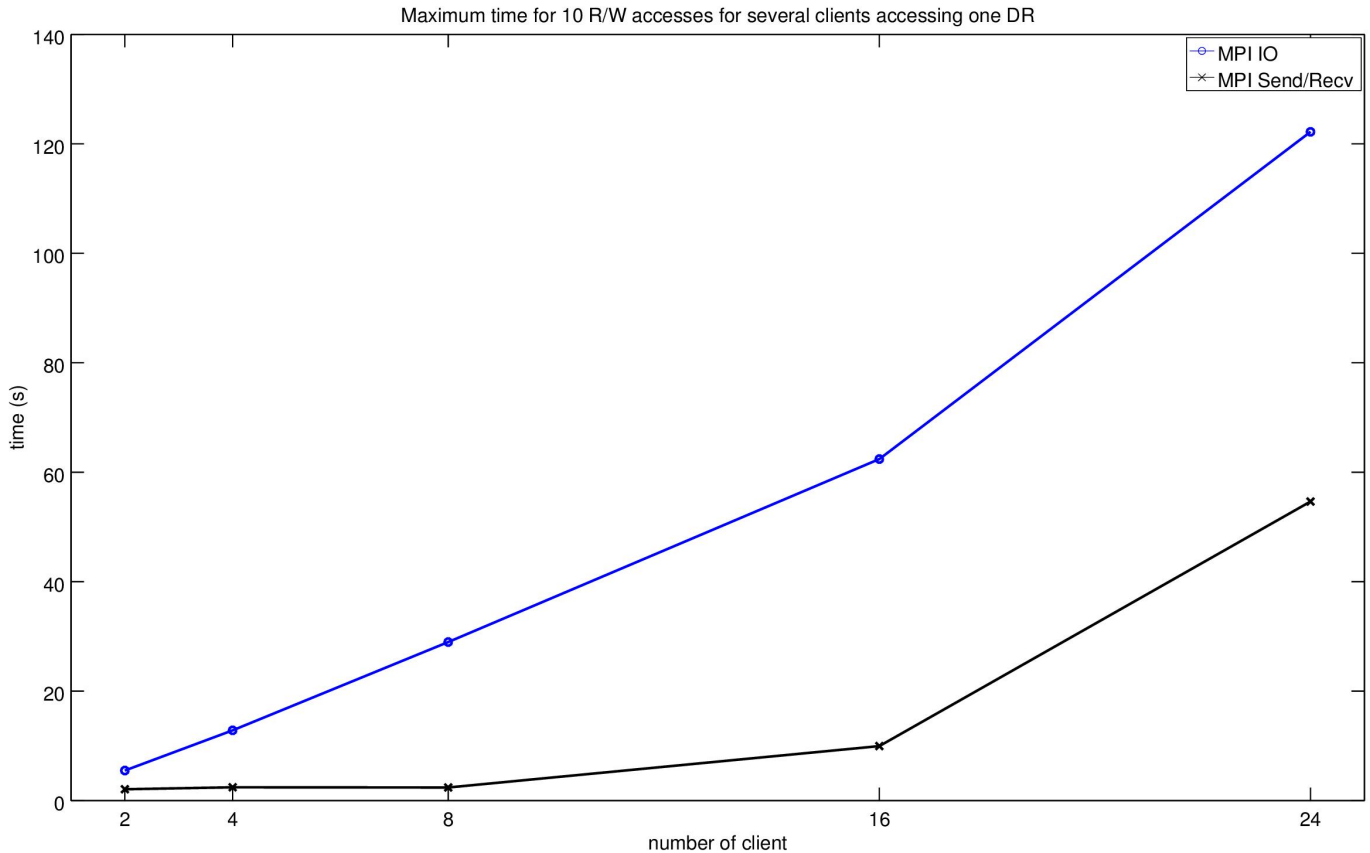


Figure: Maximum time in case **multiple accesses to one data repository.**

Development of Block Gauss-Jordan Algorithm

The Block Gauss Jordan Algorithm

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 $p-1$
  1 $B_{kk} = A^{-1}{}_{kk}$
  For $i = k+1$ to $p-1$
    2 $A_{ki} = B_{kk} \times A_{ki}$
  End For
  For $i = 0$ to $p-1$
    If $(i \neq k)$
      2 $B_{ik} = -A_{ik} \times B_{kk}$
    End If
    If $(i < k)$
      2 $B_{ki} = B_{kk} \times B_{ki}$
    End If
  End For
  For $i = 0$ to $p-1$
    If $(i \neq k)$
      For $j = k+1$ to $p-1$
        3 $A_{ij} = A_{ij} - A_{ik} \times A_{kj}$
      End For
      For $j = 0$ to $k-1$
        3 $B_{ij} = B_{ij} - A_{ik} \times B_{kj}$
      End For
    End If
  End For
End For

```
par(k:=0;blockcount-1)
do
  par
    #inversion
    if (k neq 0) then
      wait(prodDiffA[k][k][k-1]);
    endif
1   compute XMP_inversion(A[k][k],B[k][k]);
    notify(bInversed[k][k]);
//
    if (k neq blockcount-1) then
      par (i:=k+1; blockcount-1)
      do
        wait(bInversed[k][k]);
2       compute XMP_prodMat(B[k][k],A[k][i]);
        notify(prodA[k][i]);
      enddo
    endif
//
    wait(bInversed[k][k]);
    par(i:=0;blockcount-1)
    do
      if(i neq k) then
        compute XMP_mProdMat(A[i][k],B[k][k],B[i][k]);
2       notify(mProdB[k][i][k]);
      endif
      if(k gt i) then
2       compute XMP_prodMat(B[k][k],B[k][i]);
        notify(prodB[k][i]);
      endif
    enddo
//
    par(i:= 0;blockcount-1)
    do
      if (i neq k) then
        if (k neq blockcount - 1) then
          par (j:=k + 1;blockcount - 1)
          do
            wait(prodA[k][j]);
            compute XMP_prodDiff(A[i][k],A[k][j],A[i][j]);
3           notify(prodDiffA[i][j][k]);
          enddo
        endif
        if (k neq 0) then
          par(j:=0;k - 1)
          do
            wait(prodB[k][j]);
3           compute XMP_prodDiff(A[i][k],B[k][j],B[i][j]);

          enddo
        endif
      endif
    enddo
endpar
enddo
```

Figure: "As you write" implementation: 1. **Pivot**, 2. **Update row and column**, 3. **Update A and B**
Algorithm from **M. Hugues, S. Petiton,** A Matrix Inversion Method with YML/OmniRPC on a Large Scale Platform, **VECPAR 2008**

Illustration of BGJ steps

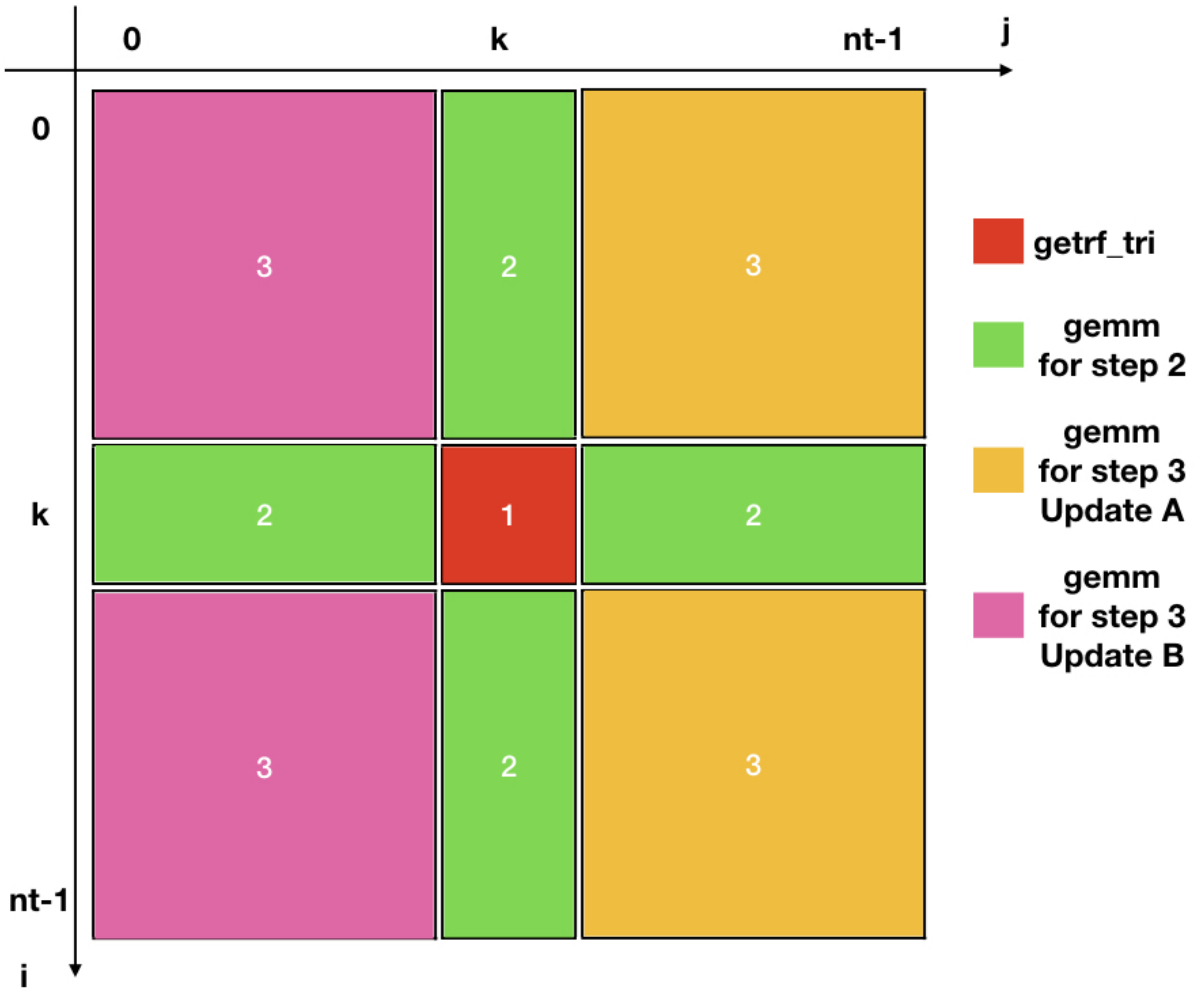


Figure: Illustration of one BGJ step

Large block size

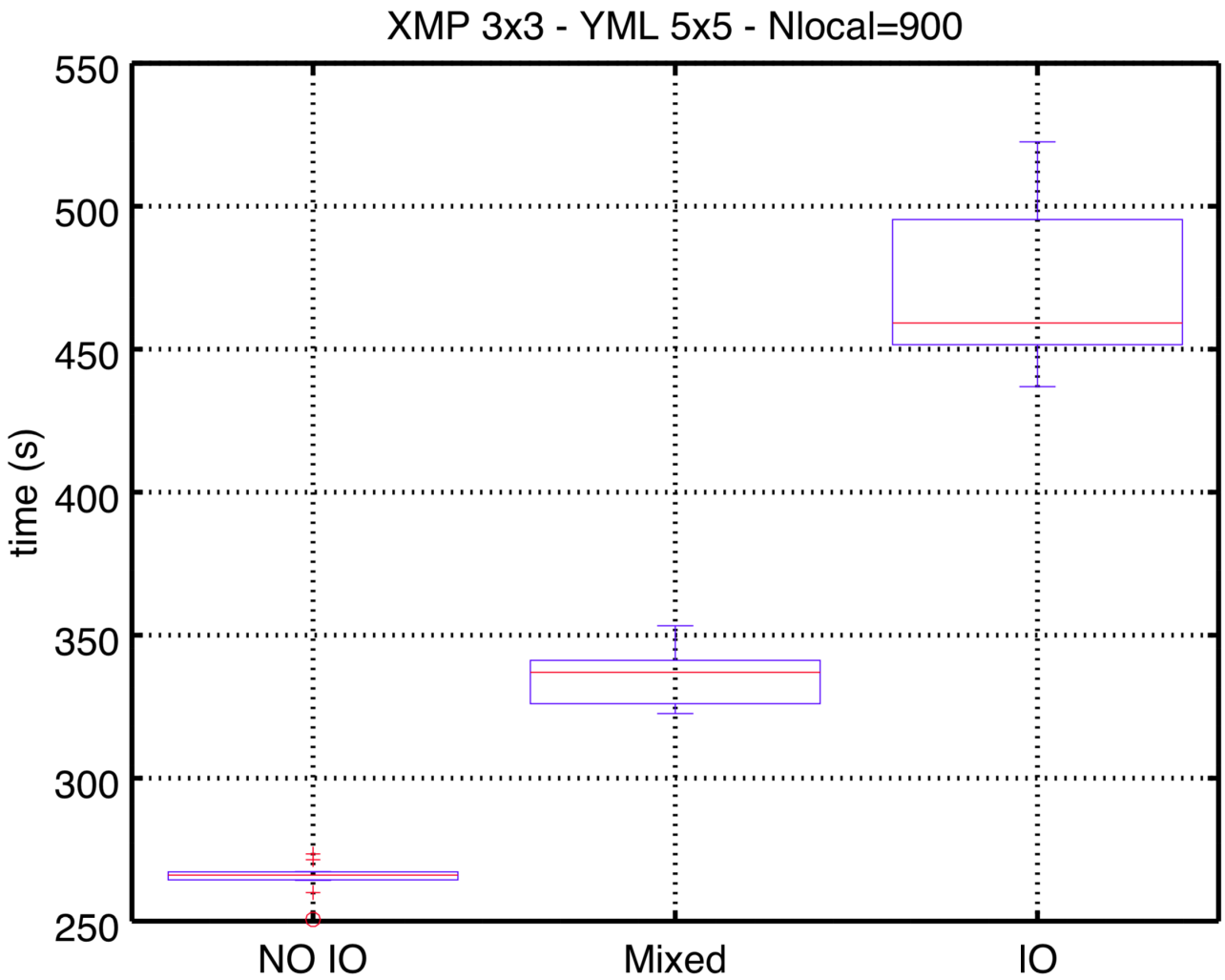


Figure: **NO I/O strategy = faster (1.73x speedup), more stable** / Mixed = compromise **1.36x**, checkpoint, stable

Small block size

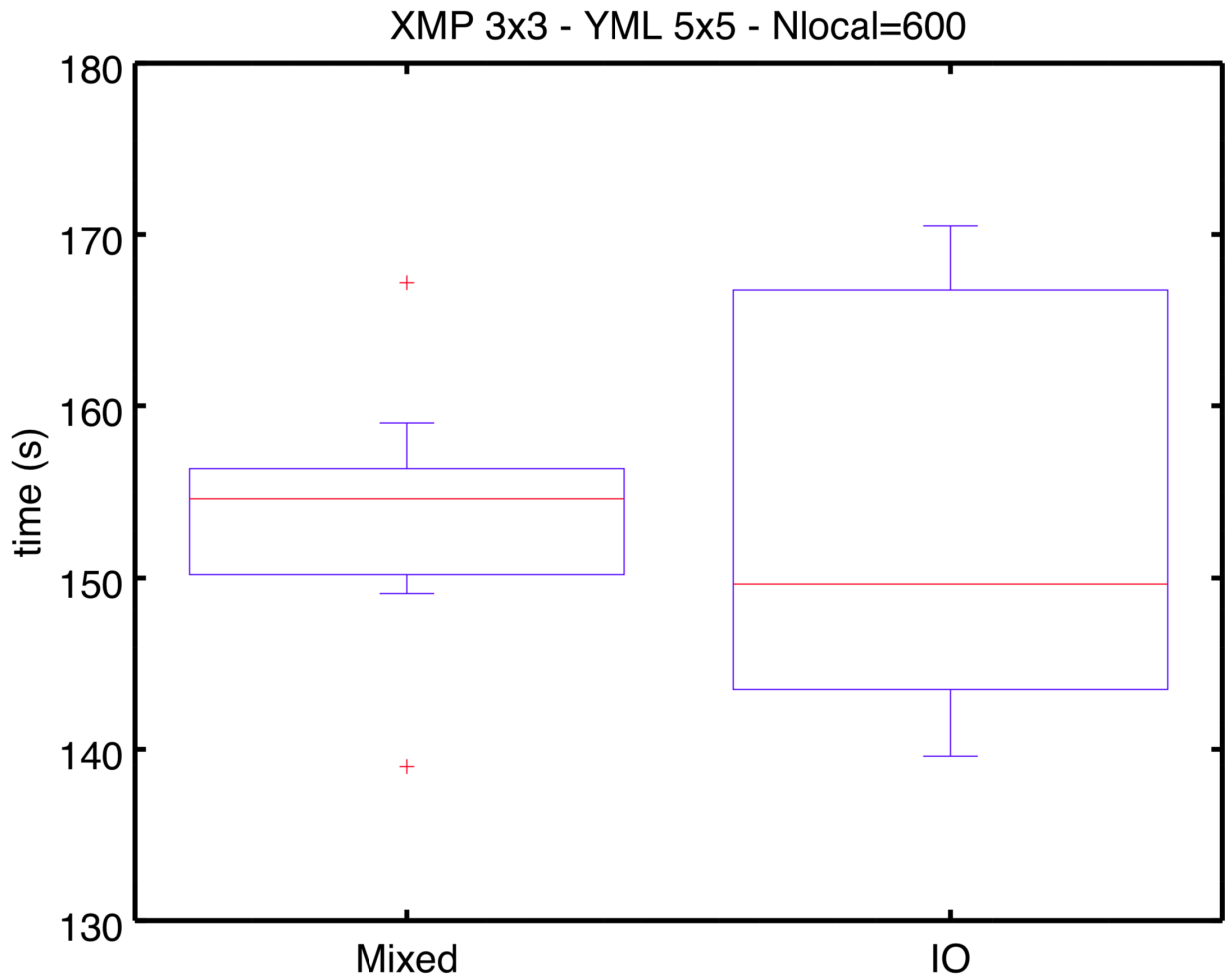


Figure: Mixed strategy **improve stability and limit loss of performance**

Conclusion

Design

- Design of data exchange for Multi SPMD
- **Grap of tasks + PGAS**
- **Design enables to extract and use information**
  - parallel programing model
  - execution model
  - architecture

Toward implementation

- YML-XMP and exchange through a data repository
- Task scheduler
- YML interface can be used
- **Choices: MPI-IO or MPI server (No I/O) or mix**
- **Speedup up to 2x for benchmark application (BGJ)**

References (Selection)

- **Block-Gauss Jordan**
  - M. Hugues, S. Petiton, A Matrix Inversion Method with YML/OmniRPC on a Large Scale Platform, VECPAR 2008
- **YML and 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
  - 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
- **ASIODS**
  - M. R. Hugues, M. Moretti, S. G. Petiton, H. Calandra, ASIODS - An Asynchronous and Smart I/O Delegation System, Procedia Computer Science, Volume 4, 2011, Pages 471-478,
- **Parallel I/O**
  - W. Gropp, *Lecture 32: Introduction to MPI I/O*, http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture32.pdf
  - T. Nakamura, M.Sato, *XMP-IO function and its application to MapReduce on the K computer* Parallel C: Accelerating Computational Science and Engineering (CSE), IOS Press, 2014