Algorithmes répartis sur grilles et cluster de clusters

7 décembre 2007

Serge G. Petiton

Goals and hypothesis

Questions :

Is Parallel Matrix Computing adaptable to *large scale* P2P Platforms?

Or, is it possible to define a new programming paradigm for scientific matrix-based computing on large scale P2P plateforms? Other that only the task-farming programming one.

What algorithms for classical linear algebra problems?
Is the class of well - adapted methods large enough?
What language and framework for the **end-users**?
What evaluation and economic model?

Large scale P2P computing, hypothesis

- Each peer can :
  - compute
  - Send and receive data
  - *Participate to the system coordination*
- Several thousand of peers targeted :
  - Between 4Gflops and 250Mflops per peer
  - Different Memory size, between 1Gbytes and 128Mbytes
  - Different software on each peer (BLAS and LAPACK available)
- Fault tolerant system (including MPI communications)
  - Indeterminism
  - MPICH - V as a Fault-Tolerant MPI ; with multicasts

Platforms

Two configurations for XtremWeb:

- A LAN based with **128** PCs, largely non - dedicated (Polytech - Lille, LIFL, INRIA).

- A WAN based with **128PCs+133PCs** distributed on two geographic sites (Polytech - Lille/LIFL and Paris XI university / INRIA), also non -dedicated.

One XtremWeb - CH testbed of **128** PCs is also deployed.

Platforms

Polytech'Lille/LIFL/INRIA Platform (128 PCs)

- 28 PIII, 450MHz,
- 81 Intel Celerons 600Mhz to 2.4 GHz,
- 15 AMD Duran, 750MHz
- 4 PIV, 2.4 GHz.

Memory size from 128MBytes to 512MBytes

Paris 11 Univ./LRI/INRIA (133 PCs)

- 101 AMD Athlon and K7, 7 bi proc., 600MHz to 2.8GHz,
- 10 PIV, 2GHz,
- 12 PIII, 500 MHz,
- 9 PII, 400 MHz.

Memory Size from 128Mbytes to 1Gbytes

Ethernets with heterogeneous speeds from 10Mbits/s to 100Mbits/s

No-dedicated to our experimentations : students and seti@home are also using these computers

Algorithms, Evaluations and/or Experimentations

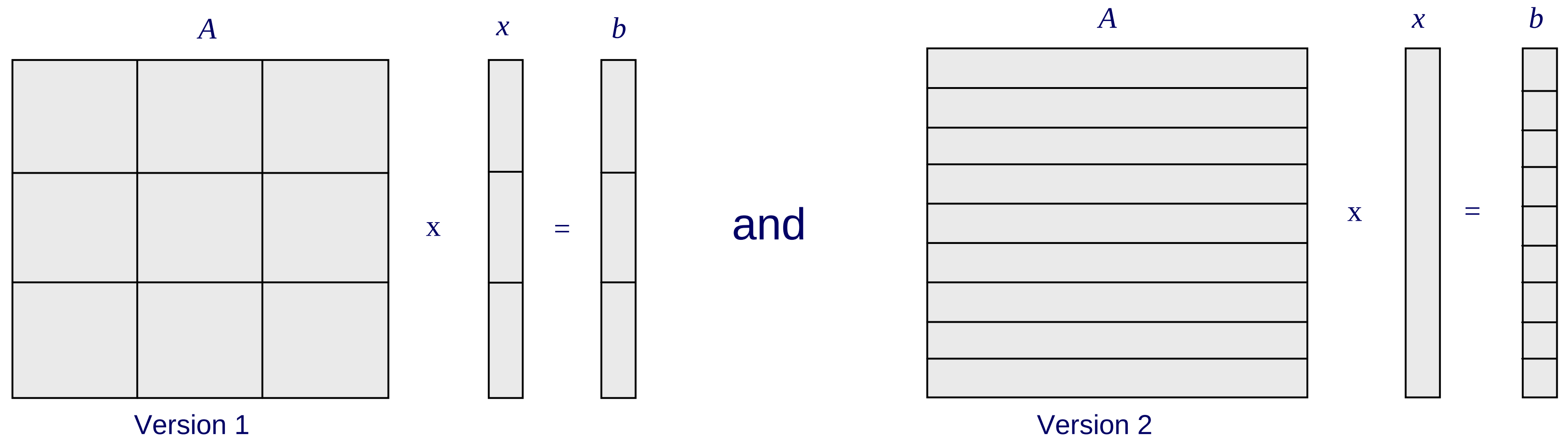
- BLAS2 computation (for iterative methods)
- Eigenvalues end eigenvectors

Algorithms, Evaluations and/or Experimentations

- **BLAS2 computation (for iterative methods)**
- Eigenvalues end eigenvectors
- Gauss Jordan par blocs

Distributed Block Dense Matrix Vector Multiplication with XtremWeb

- Two traditional ways to distributed matrices in block matrix algebra generate two different algorithms, as follow :The matrix vector product is a simple but fundamental operation in matrix computing. It forms the core of iterative methods, for example.
- It is often one of the primary operations evaluated to benchmark supercomputers.

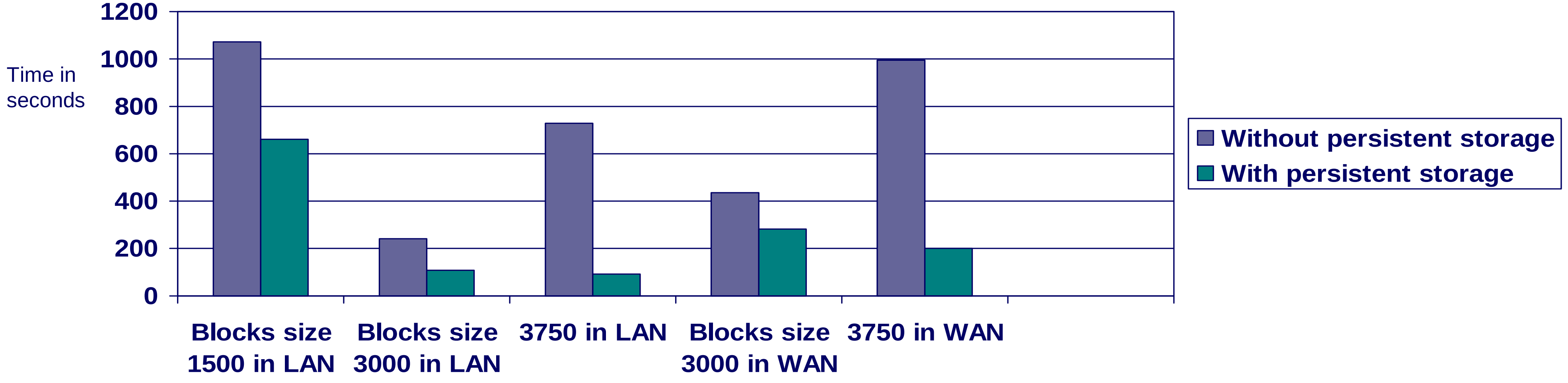


The matrix *A* remains unchanged, this enables us to pre-deploy data and use “persistent data storage” (for iterative method for example).

Results 1

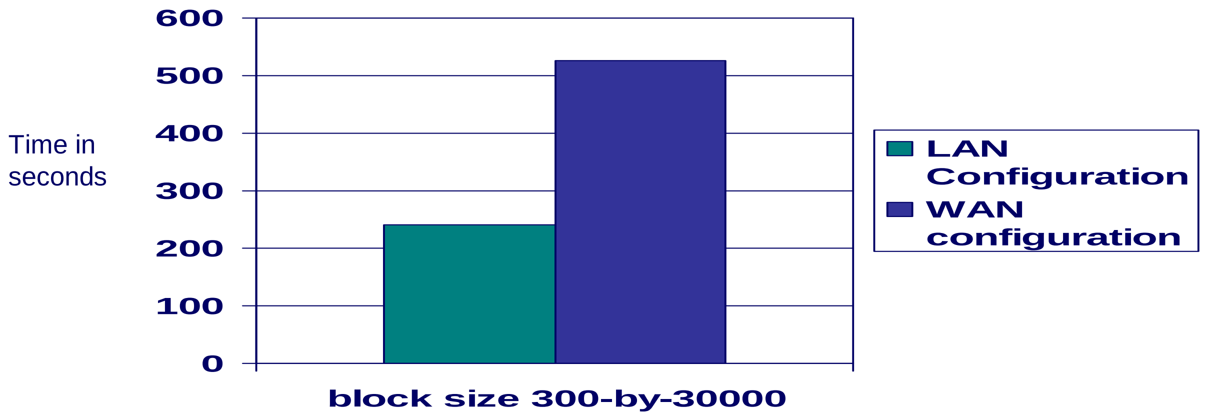
- The computing times of different runs vary greatly . We present here the average computing times (in seconds) for each execution,

- Version 1 : with or without persistent storage in LAN and WAN configurations.
  - For blocks size of 1500, we introduce 420 tasks into the XtremWeb network at each execution (more tasks than available workers).
  - 110 tasks for blocks size of 3000.
  - 72 tasks for blocks size of 3750.



Results 2

- Version 2 : only one step.
  - At each execution, we introduce 100 tasks into the XtremWeb network



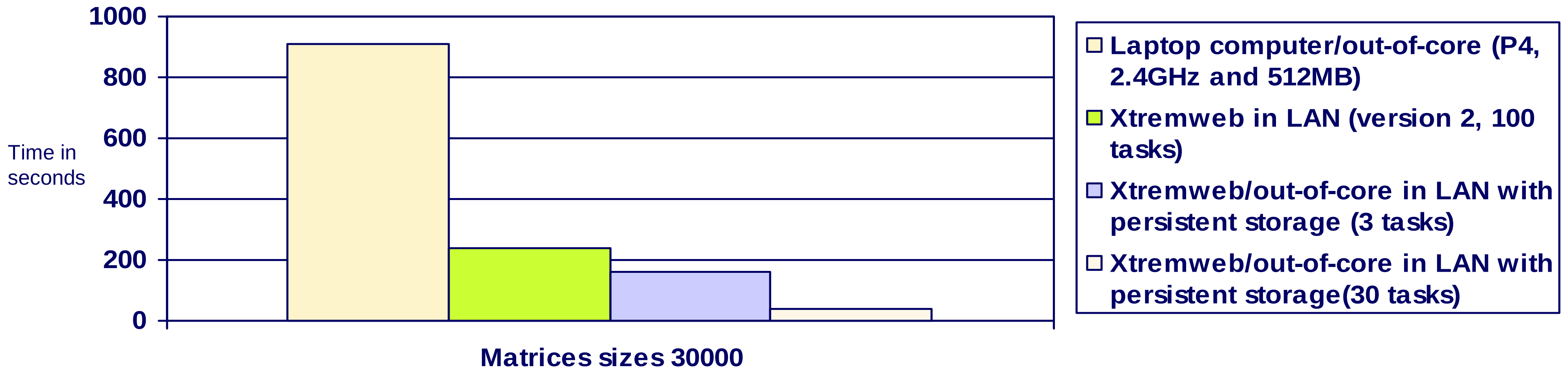
Lack of an efficient scheduling.

P2P Out-of-core programming 1

- The out-of-core technique solves large problems on standard computers. We will use it for increasing the size of problems on each peer.
  – Classical out-of-core matrix vector product.

- The average computing times (in seconds) :

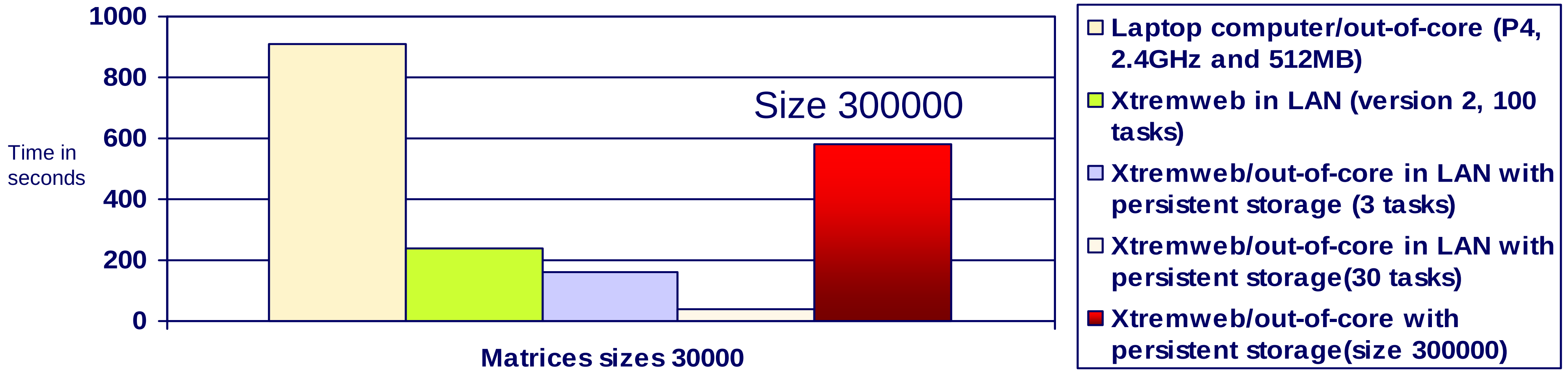
  - For out-of-core version (size 30000) we introduce between 3 and 30 tasks in the XtremWeb network.



P2P with out-of-core programming

- **The out-of-core technique solves large problems on standard computers. We will use it for increasing the size of problems on peers.**
  - **The matrix vector product is scheduled out-of-core in very simple and efficient way**
- **The average computing times (in seconds) :**

- For out-of-core version (size 30000) we introduce between 3 and 30 tasks in the XtremWeb network.



P2P with out-of-core programming

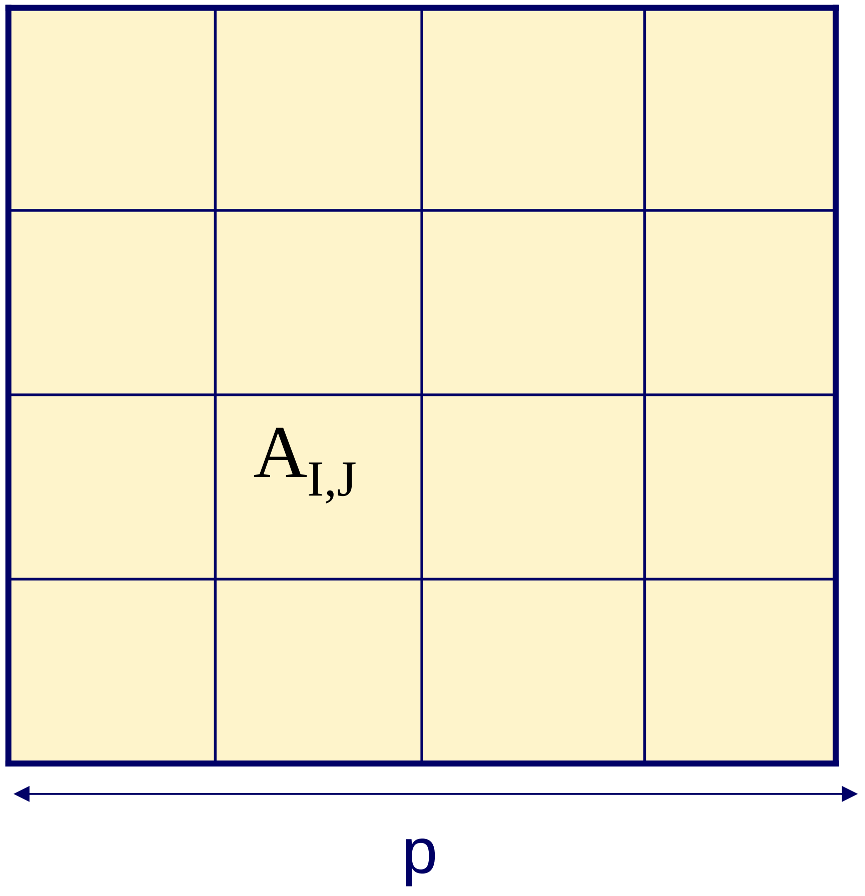
- For matrix size of 30000, we obtain a speed-up up to 20 compared to the 2.4 GHz laptop execution ; using the out-of-core product. The vector and all matrix blocks are pre-deployed.

- We increase the size of matrix with 300000 (100 times the initial matrix, >700 GB). We introduce 30 tasks in the LAN XtremWeb network (each task with block size of 1000-by-300000)

  - ➢ the average computing time is 575 seconds!

Conclusion

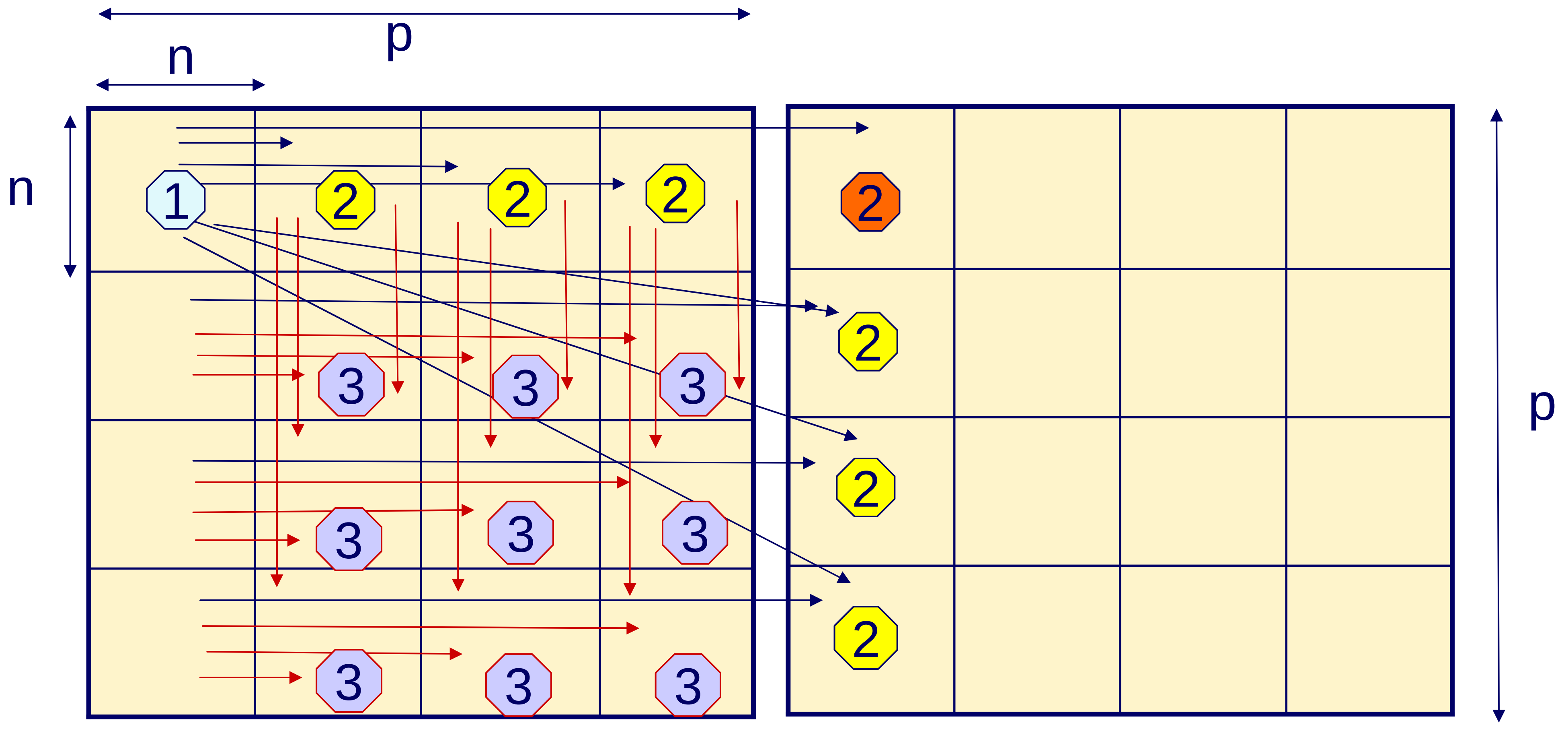
- The central management of communications is a strong bottleneck
- We need smart scheduler and efficient persistent data P2P storage
- Out-of-core computing on each peer as a programming paradigm.
- Possible to get a numerical result which was not computable on a unique computer.
- New end-users



$$\mathbf{B} = \begin{array}{cccc} n & & & \\ \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \end{array}$$

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

To invert a matrix
$2N^3$ operations
Challenge : $N = 10^6$

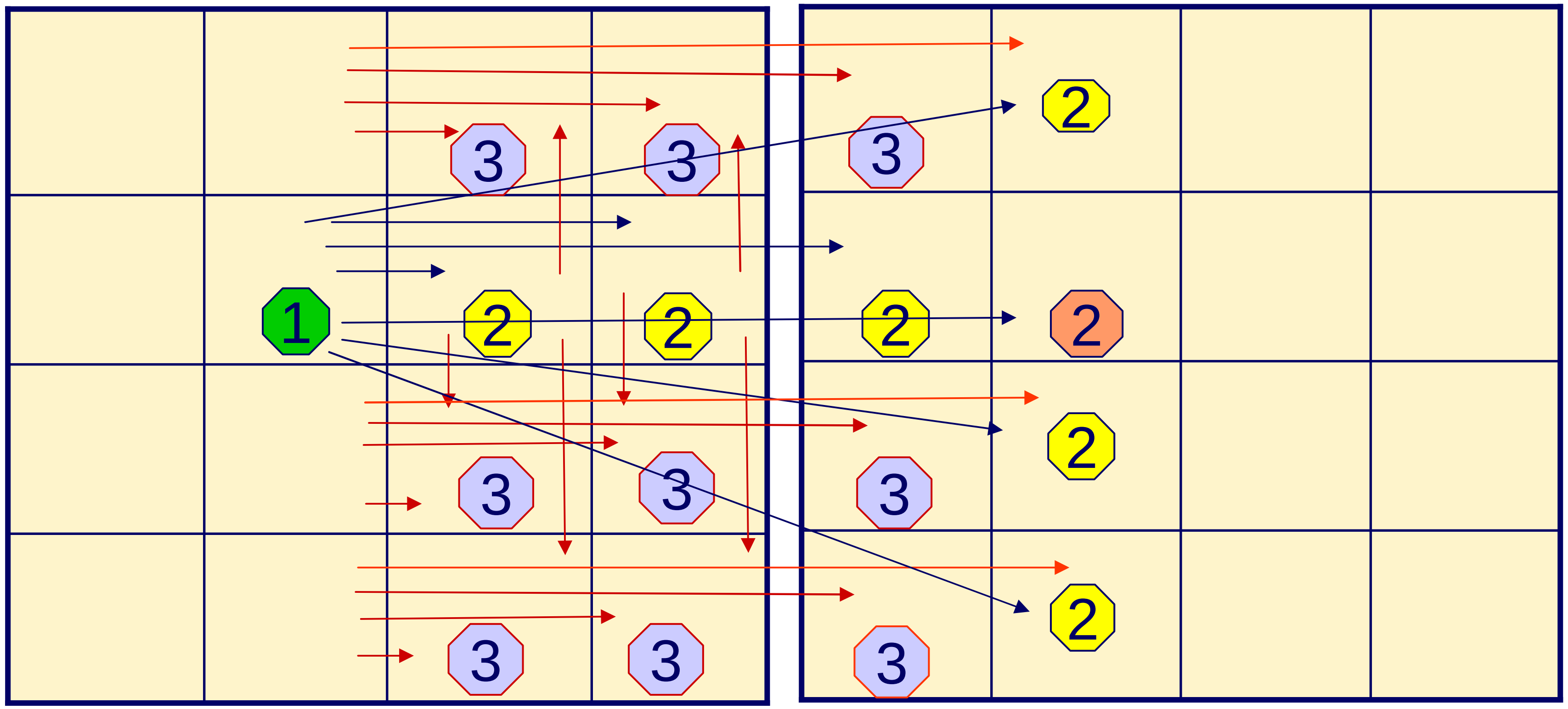


1 Element Gauss-Jordan, LAPACK, $cx = 2n^3 + O(n^2)$

2 $A = +/- A B$; BLAS3, $cx = 2 n^3 – n^2$, 2 $A = B$

3 $A = A – B C$; BLAS3, $cx = 2n^3$

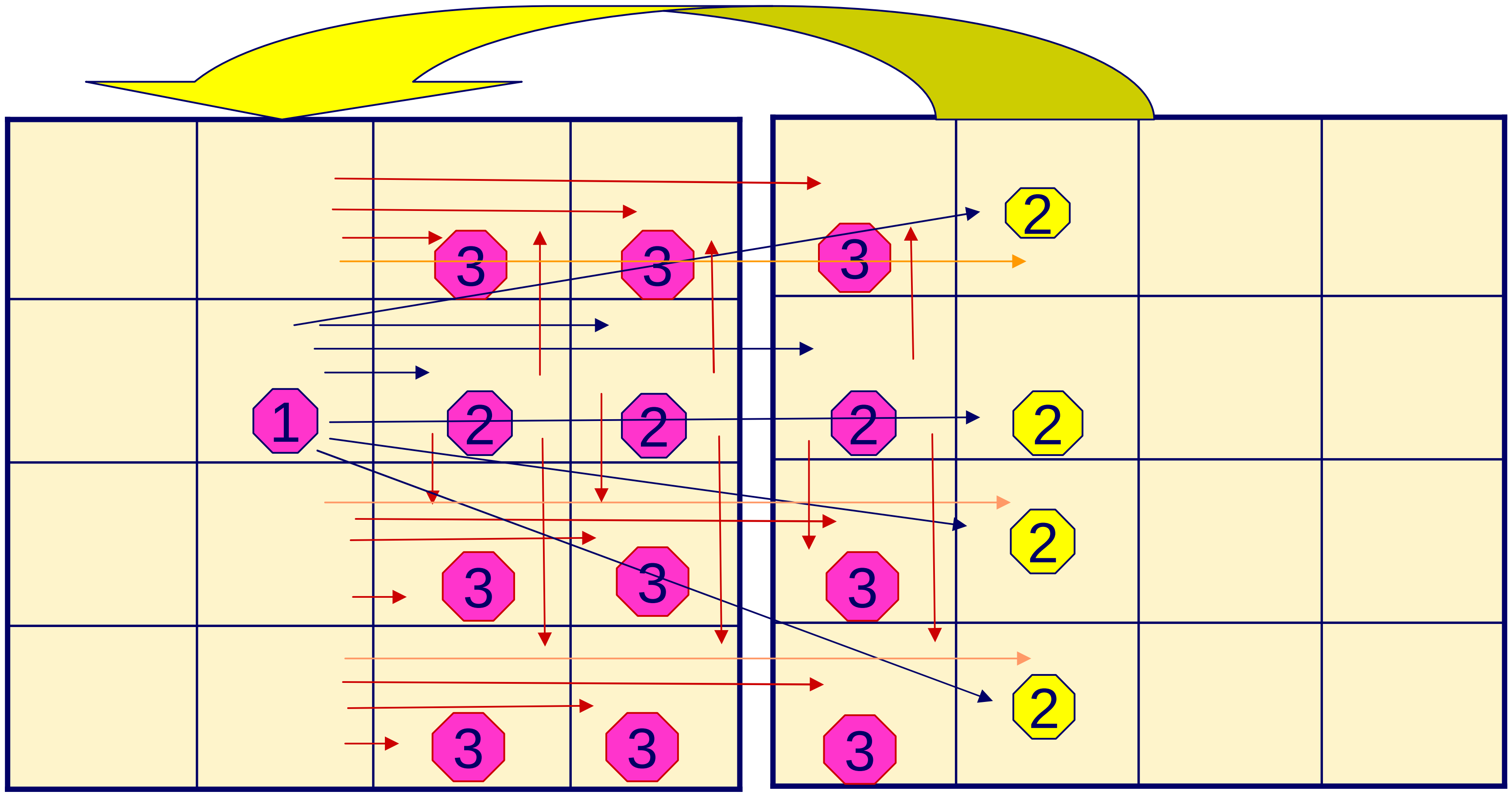
→ n^2 64 bit floating point numbers



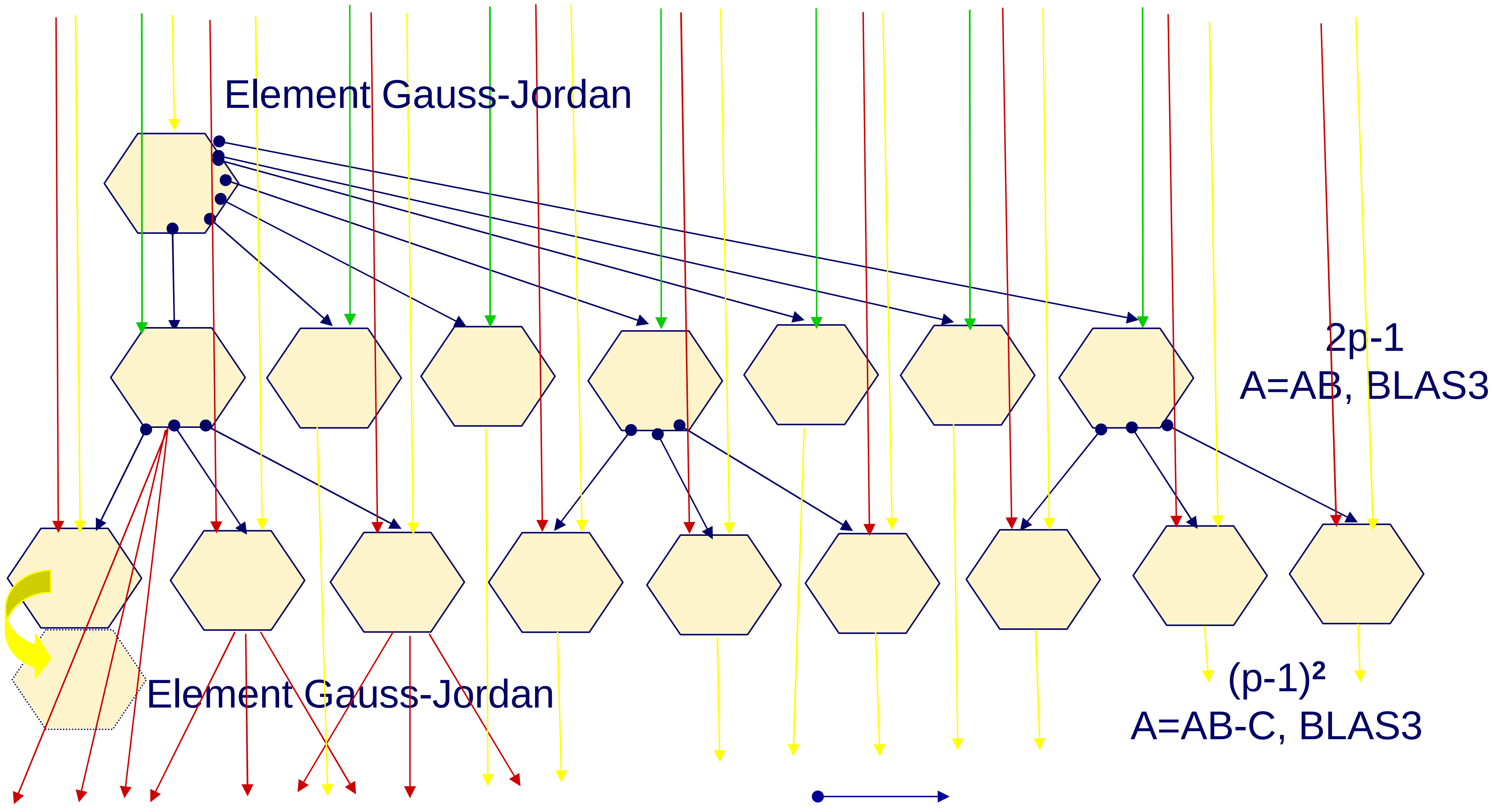
Each computing task : 1 up to 3 blocks

maximum n < (memory size of one pair) / 3

Up to $(p-1)^2$ peers

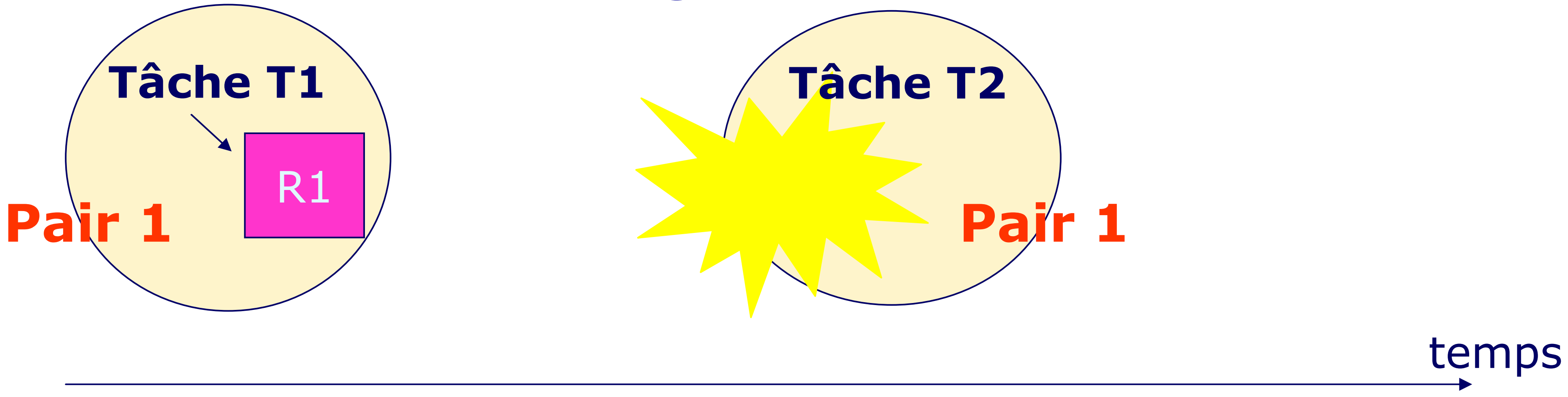


- •Computation of « new » blocks on peer which minimize communications
- •« update » of block at step k, on peer who updated the block a step k-1
- • data send to dedicate peer ASAP



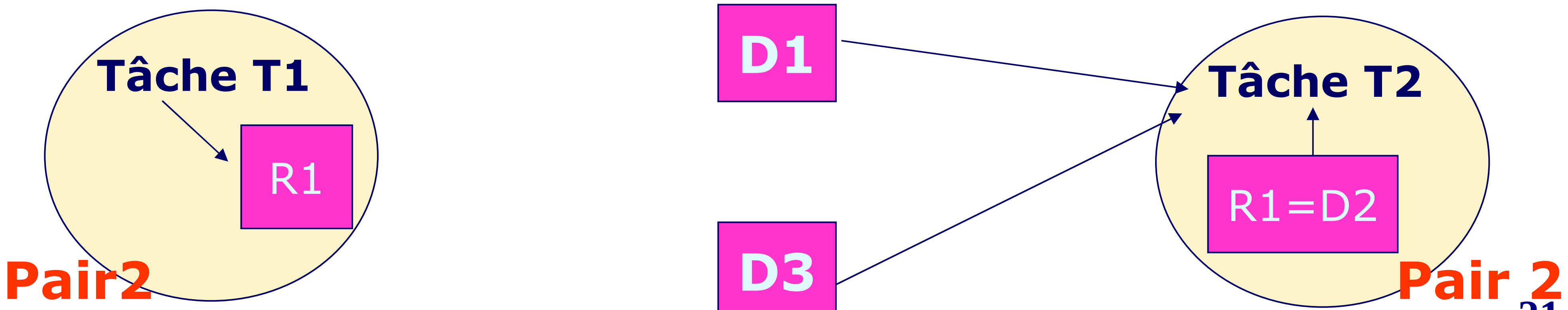
One step of the Block Gauss-Jordan method ; p=4

Clouage et volatilité

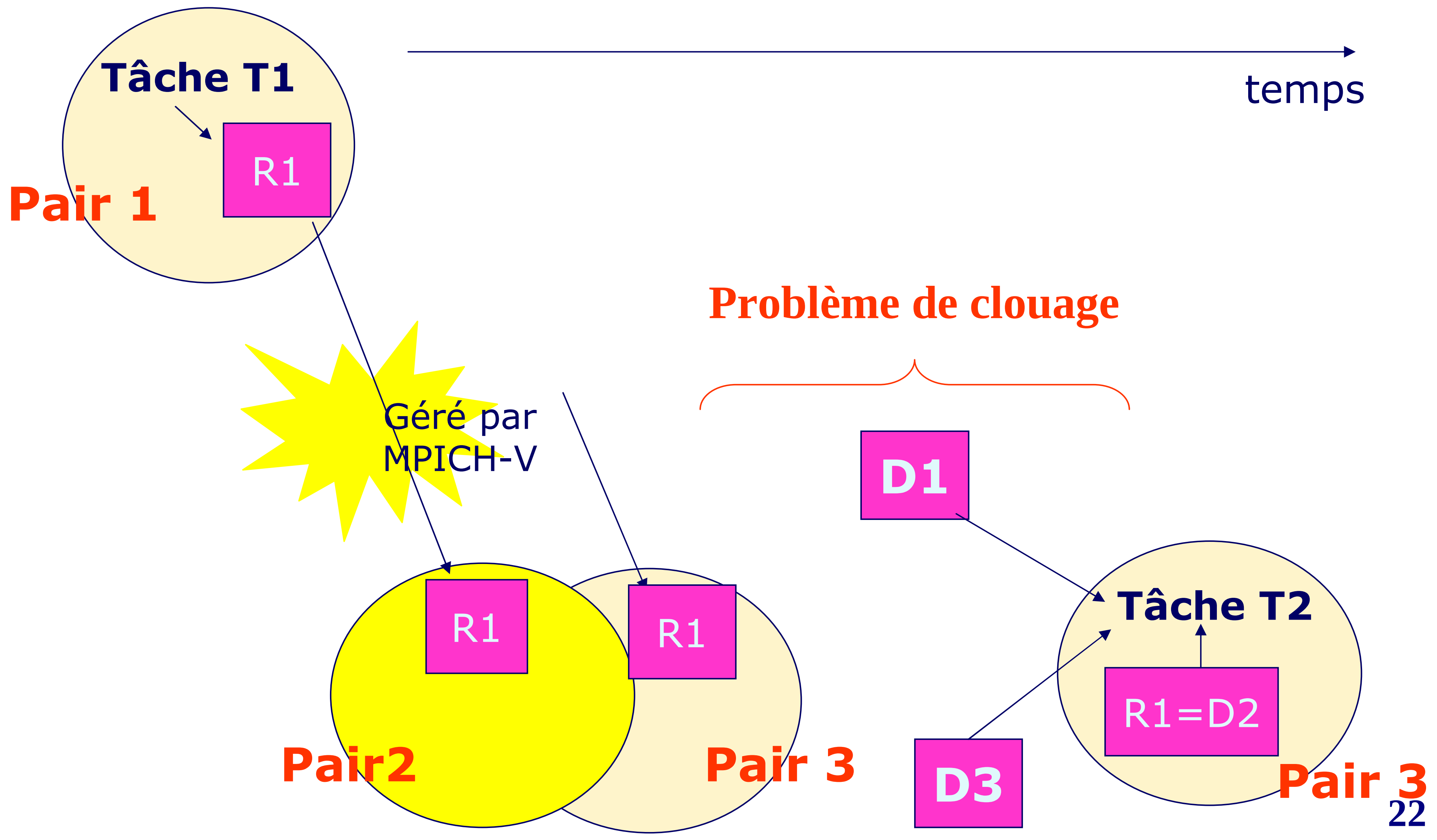


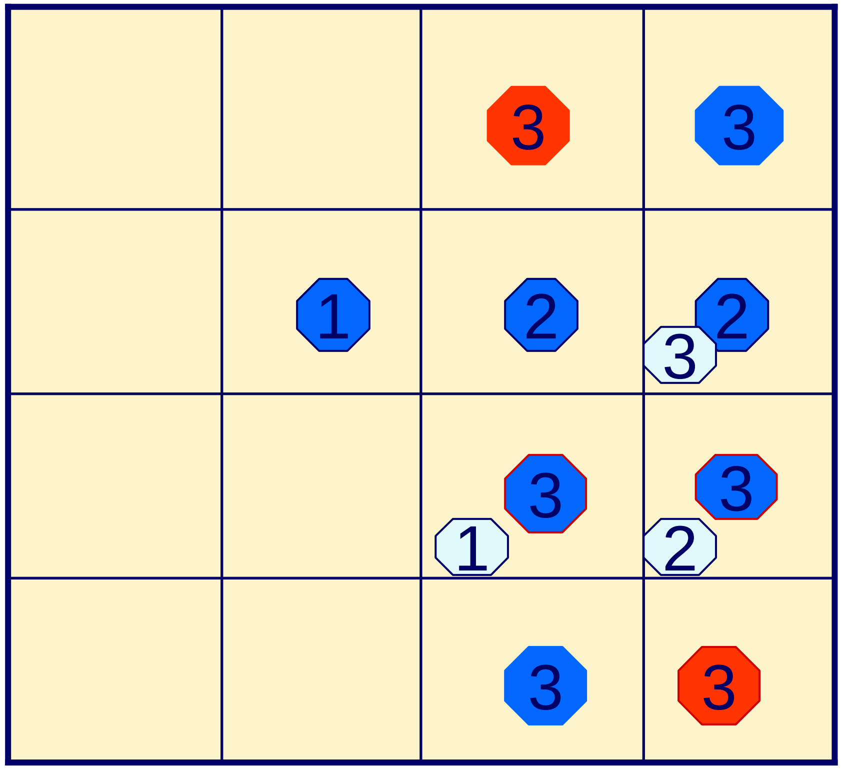
Si le Pair 1 se retire de l'ensemble des pairs?

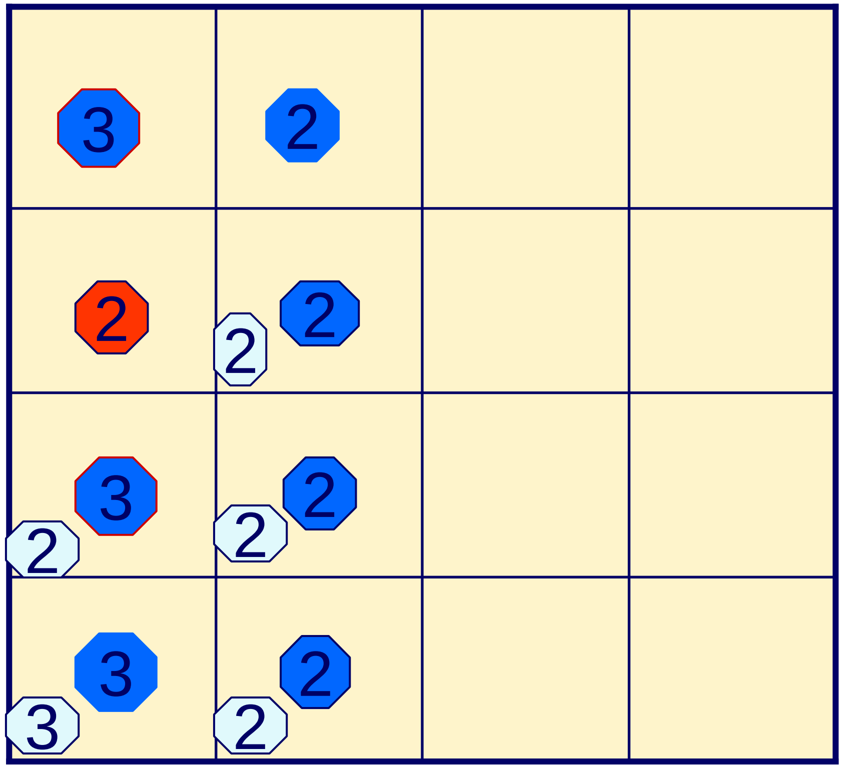
Nécessité de dupliquer la tâche pour garantir le clouage



Anticipation et volatilité







Nevertheless, peers are not stables. Then, we can have in parallel computing from several steps of the method.

We have to use an inter and intra steps dependency graph.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<component type="Graph">
  <name>GaussJordan</name>
  <description>
This component invert a bloc matrix using Gauss-Jordan
       algorithm.
  </description>
  <parameters>
    <parameter name="Minput"
       type="BLOC_MATRIX_REAL" access="in"/>
    <parameter name="Moutput"
       type="BLOC_MATRIX_REAL" access="ou"/>
    <parameter name="bloc_count"  type="INTEGER"
       access="in"/>
    <parameter name="size"       type="INTEGER"
       access="in"/>
  </parameters>

   <graph>
   /* Declarations */
   const twop := 2 * bloc_count,
         pp1  := bloc_count+ 1,
         pm1  := bloc_count - 1,
         m    := size/ bloc_count ;
   domain a[1..bloc_count, 1..twop] of
   MATRIX_MATRIX_DOUBLE;
   bind a[1..bloc_count,1..bloc_count] to Minput;
   bind a[1..bloc_count,pp1..twop] to Moutput;
   event call[1..bloc_count, 1..bloc_count, 1..twop];
```

```
/* Calcul */
compute invmatcp(a[1,1], a[1,pp1], m);

par

  par (j := 2, bloc_count) do
    compute diad3(a[1,j], a[1,1], m);
    signal(cal[1,1,j], cal[1,1,j]);
  end par do
//
    par (i := 2, bloc_count) (j := 2, bloc_count) do
    wait(cal[1, 1, j]);
    if (i eq 2 and j eq 2) then
      compute triad3inv(a[2,2], a[2,2 + bloc_count], a[2,1],
        a[1,2], m);
      signal(cal[2, 2, 2], cal[2, 2, 2 + bloc_count]);
    else
      compute triad3(a[i,j], a[i,1], a[1,j], m);
      signal(cal[1, i, j]);
    endif
  end par do
 //
    par (i := 2, bloc_count) do
    compute mdiad3(a[i,1+bloc_count], a[1,1], m);
    signal(cal[1, i, pp1]);
  end par do
//
……
```

Méthodes hybrides asynchrones

Multiple Explicit Restarted Arnoldi Method (MERAM)

La méthode d’Arnoldi comporte 3 phases par itération :

- projection dans un sous-espace de taille $m << n$
- calcul dans ce sous espace (faible complexité)
- redémarrage : nouvelle itération

Projeter dans plusieurs sous-espaces différents de tailles distinctes

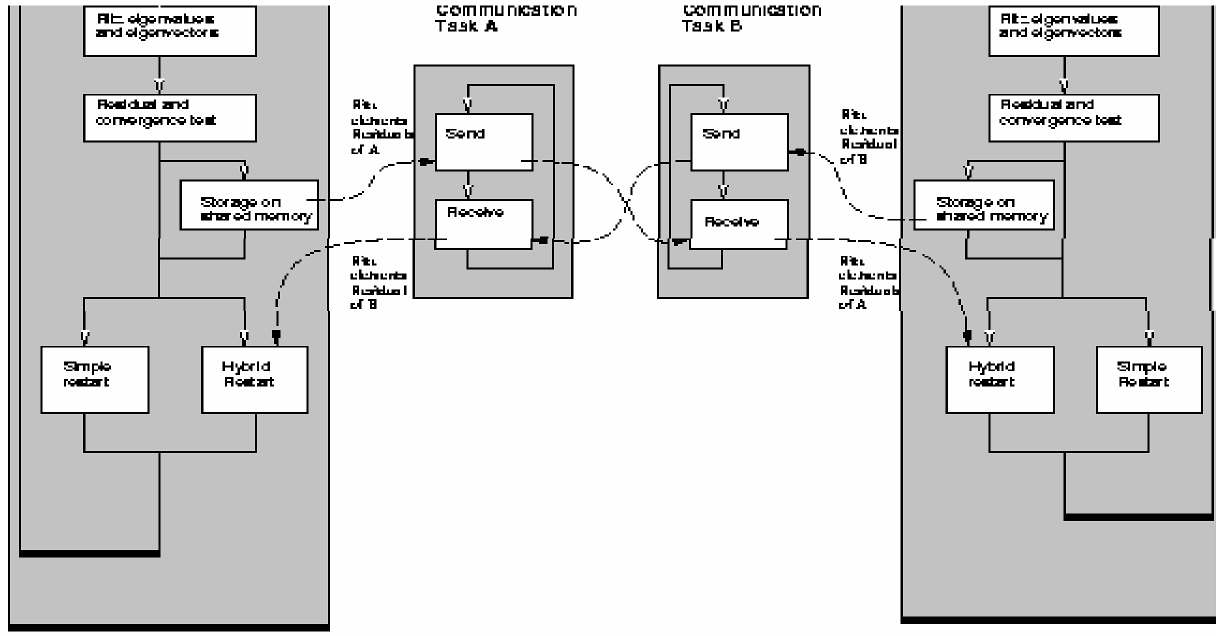
Les informations supplémentaires permettent d’accélérer la convergence vers les valeurs propres recherchées

MERAM : 1993, SEH, ETCA, avec N. Emad et G. Edjlali

- Complètement asynchrone et à gros grains
- Plusieurs stratégies de redémarrage sélectionnées
- Communications en parallèle des calculs
- Lisse les convergences comportant souvent des paliers

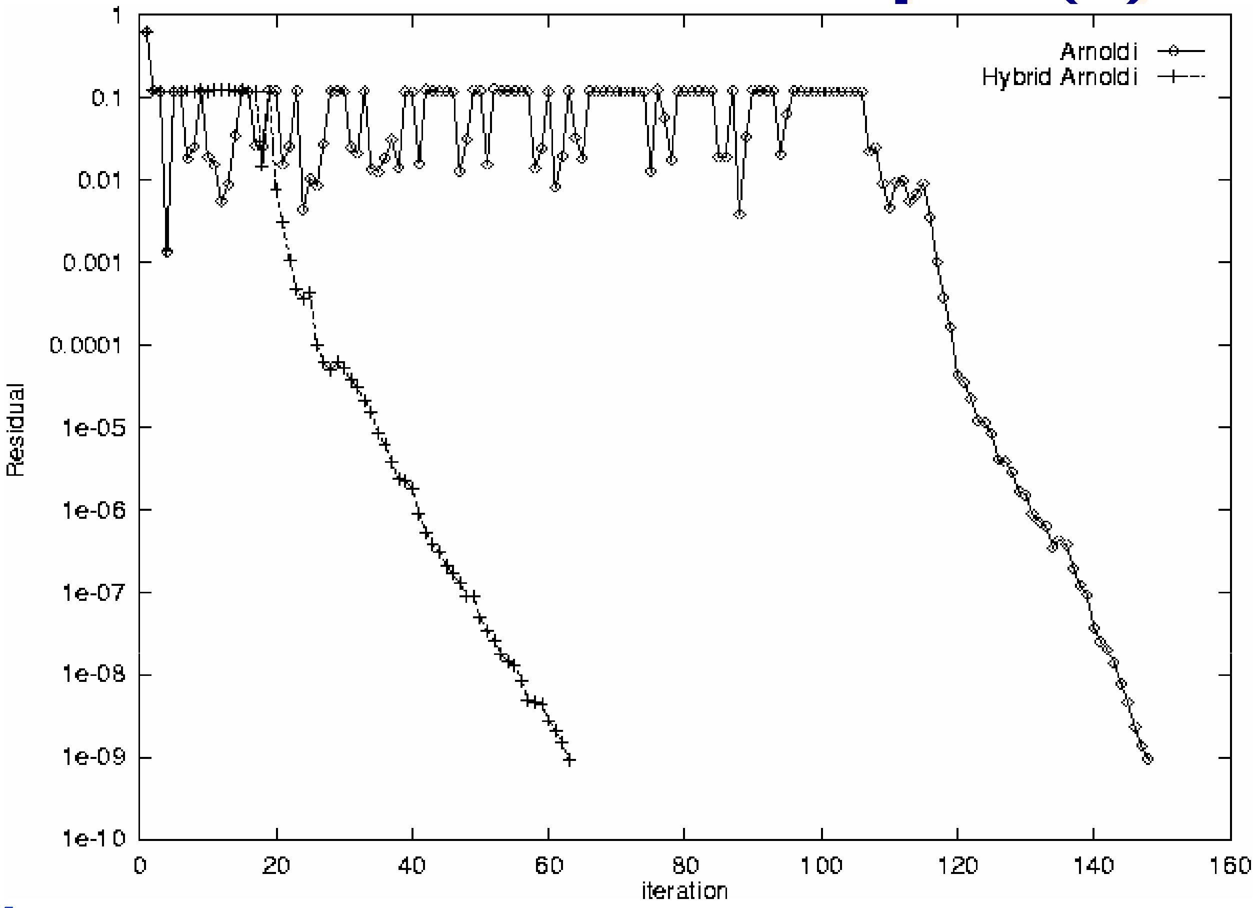
Expérimentations avec deux machines parallèles
et plusieurs stations

MERAM : 1993, SEH, ETCA,


Ritz eigenvalues and eigenvectors
Residual and convergence test
Storage on shared memory
Simple restart
Hybrid Restart
Communication Task A
Send
Receive
Ritz elements Residuals of A
Ritz elements Residual of B
Communication Task B
Send
Receive
Ritz elements Residual of B
Ritz elements Residuals of A
Ritz eigenvalues and eigenvectors
Residual and convergence test
Storage on shared memory
Hybrid restart
Simple Restart

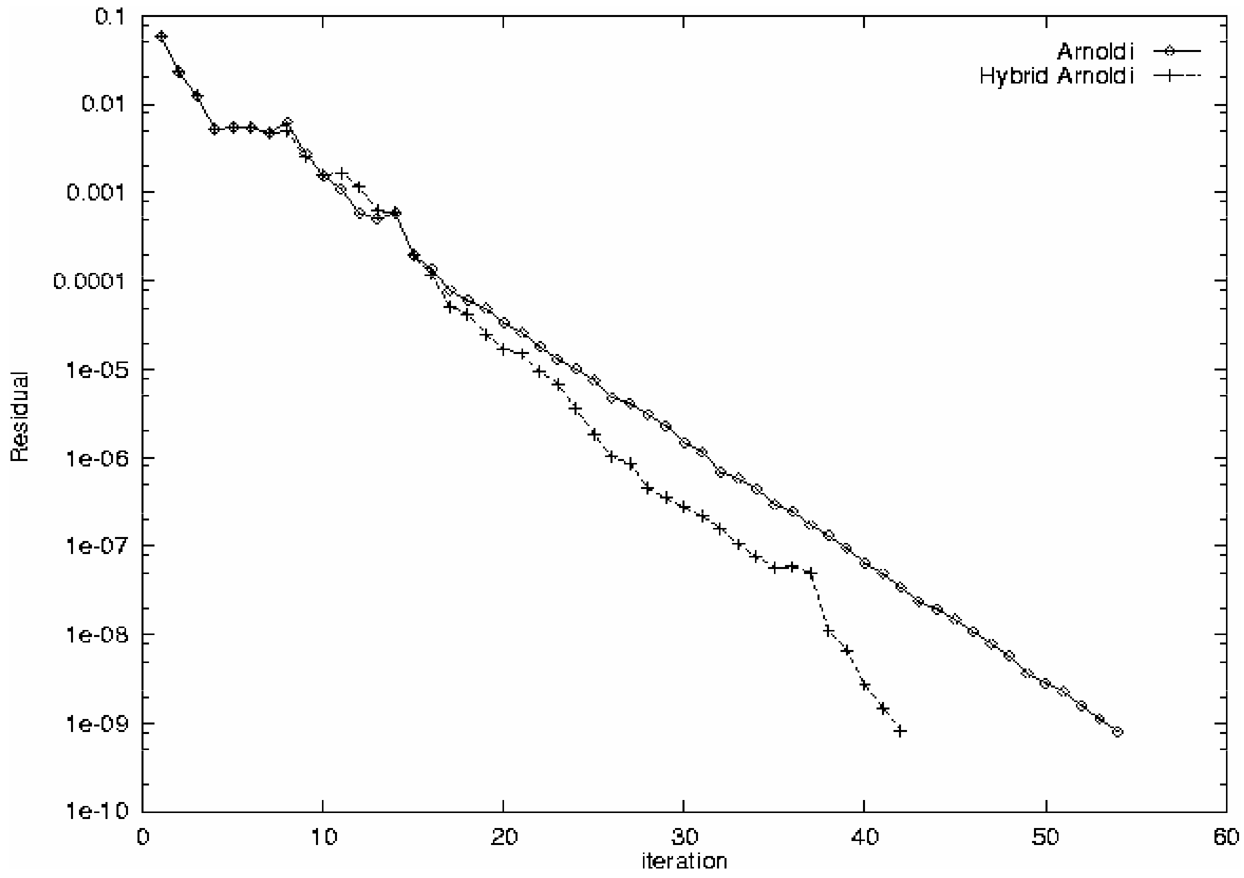
• tâches asynchrones
• communications en parallèle des calculs
• possibilité d’autres calculs déportés

MERAM : 1993, SEH, ETCA, résultats numériques (1)



Expérimentations avec une CM200, une CM5 et des SUN

MERAM : 1993, SEH, ETCA, résultats numériques (2)



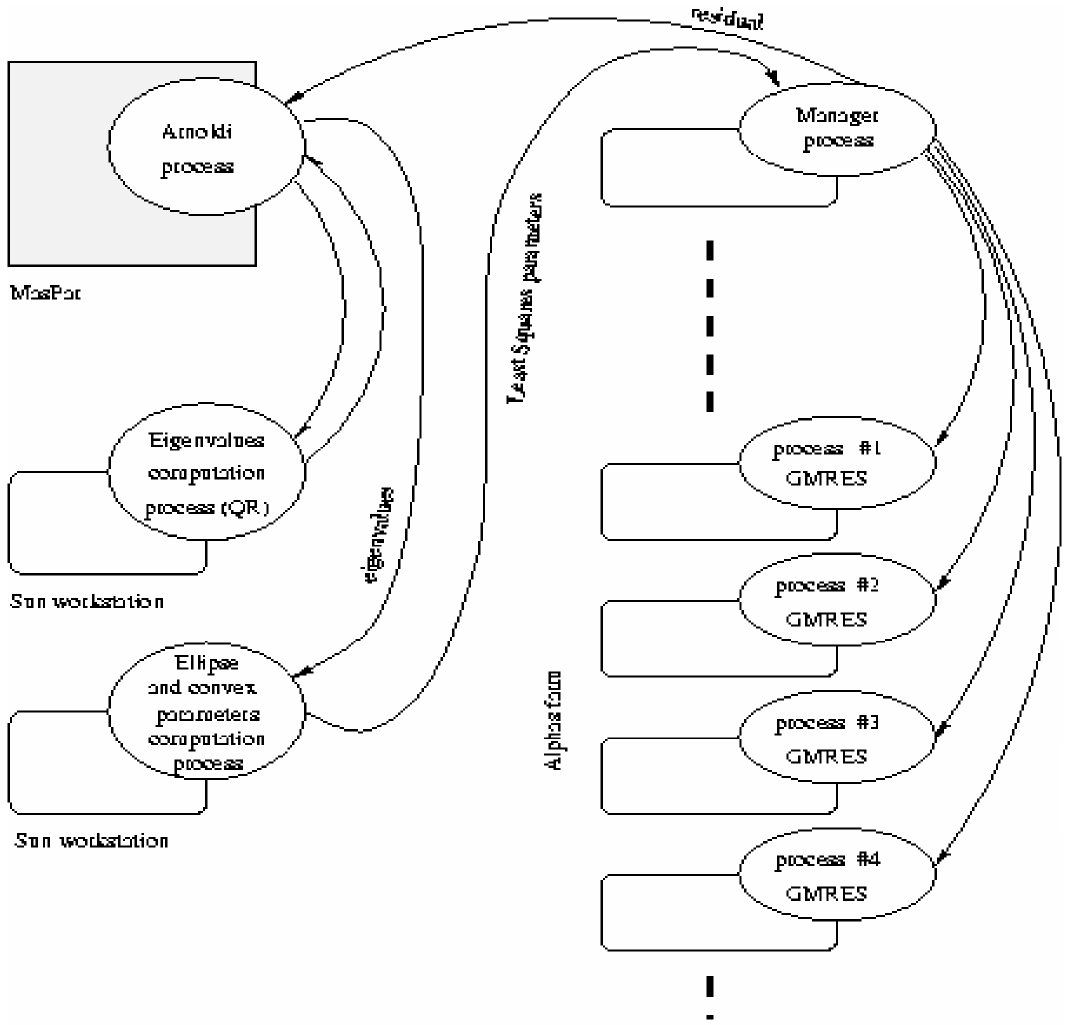
Expérimentations avec une CM200, une CM5 et des SUN

GMRES/LS-Arnoldi : 1995, LIFL, USTL, avec Guy Bergère et Azedine Essai

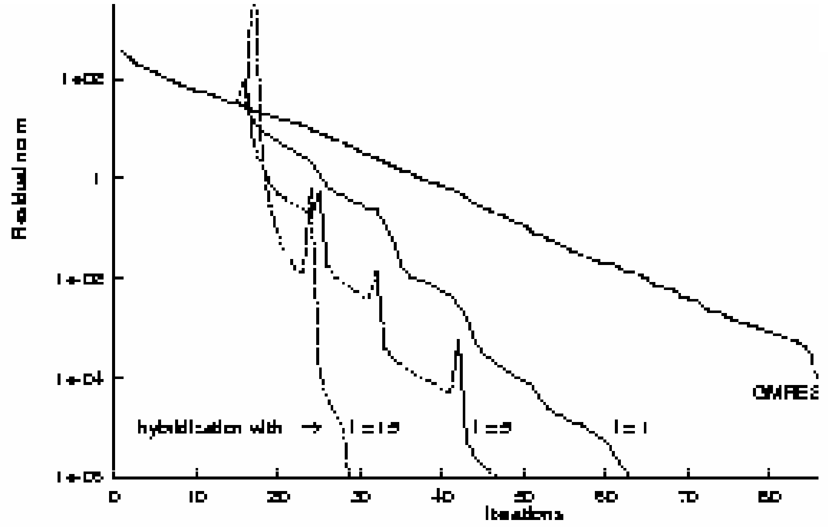
- Calcul des valeurs propres dominantes pour *optimiser* la convergence de GMRES
- Calcul à l’aide de LS des paramètres *optimaux*
- degré du polynôme, fréquence de l’hybridation, ...

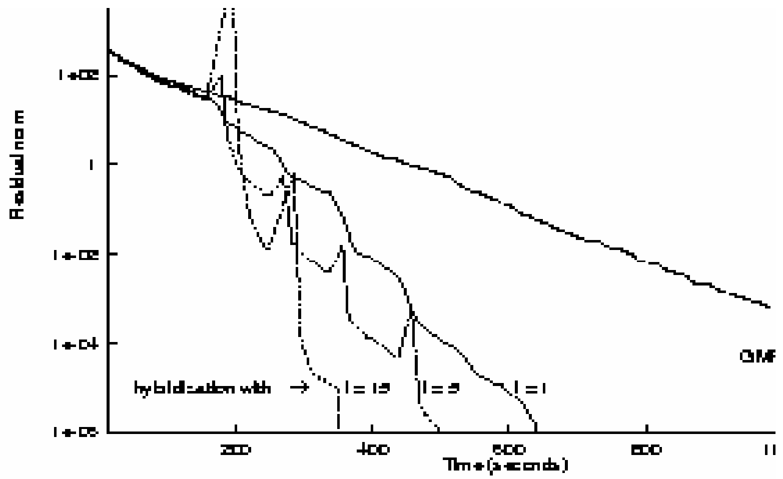
Expérimentations avec deux machines parallèles
et plusieurs stations

GMRES/LS-Arnoldi, LIFL, USTL



GMRES/LS-Arnoldi: 1996, LIFL, USTL
résultats numérique





Expérimentations avec 2 machines parallèles et des SUN