CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

MAISON DE LA SIMULATION
PARIS-SACLAY, FRANCE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

SPPEXA Workshop Japan 2017

Towards an Intelligent linear algebra for extreme computing

Serge G. Petiton

CNRS/Maison de la Simulation and CRIStAL Laboratory,
University Lille 1, Sciences et Technologies

AKIHABARA Convention Hall

April the 6th, 217

Outline

- **Introduction**
- Krylov subspace auto-tuning algorithm
- GMRES Cache Aware Auto-tuning strategies
- Orthogonalization auto-tuning algorithm
- ERAM restart strategy auto-tuning
- Unite and Conquer MERAM
- Sparse format auto-tuning
- Intelligent Krylov method for extreme computing
- Conclusion

Researches on programming paradigms, languages and tools for extreme computing have to considered (new) methods and applications which would be adapted for future machines

- Minimize the global computing time,
- Accelerate the convergence,
- Minimize the number of communications (optimized Ax, asynchronous comp, communication compiler and mapper,….)
- Minimize the number of longer size scalar products (global reductions),…
- Minimize memory space, cache optimization….
- Select the best sparse matrix compressed format,
- Mixed arithmetic
- Resilience – fault tolerance
- Minimize energy consumption…

Nevertheless we have not access to exascale machines yet and some behaviors will appear only in a few months/years.

The goal of this talk is to illustrate that we would need "smart" auto-tuning algorithms for intelligent linear algebra methods to create the next generation of High Performance Numerical software

Then, we have to propose adapted programming paradigms, language and algorithmic for those new/future methods.

Outline

- Introduction
- **Krylov subspace auto-tuning algorithm**
- GMRES Cache Aware Auto-tuning strategies
- Orthogonalization auto-tuning algorithm
- ERAM restart strategy auto-tuning
- Asynchonous Hybrid ERAM methods (MERAM)
- Experimental results
- Intelligent Krylov method for extreme computing
- Conclusion

GMRES example : about memory space, dot products and sparse matrix-vector multiplication

1. *Start*: Choose x_0 and compute $r_0 = f - Ax_0$ and $v_1 = r_0/\|r_0\|$.
2. *Iterate*: For $j = 1, 2, \cdots, m$ do:
   $h_{i,j} = (Av_j, v_i), i = 1, 2, \cdots, j,$
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i,$
   $h_{j+1,j} = \|\hat{v}_{j+1}\|$, and
   $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$.
3. *Form the approximate solution*:
   $x_m = x_0 + V_m y_m$, where y_m minimizes $\|\beta e_1 - \bar{H}_m y\|$, $y \in R^m$.
4. *Restart*:
   Compute $r_m = f - Ax_m$; if satisfied then stop
   else compute $x_0 := x_m$, $v_1 := r_m/\|r_m\|$ and go to 2.

A : very sparse matrix of size n x n

Dense matrix of size n x m

Memory space :
sparse matrix : nnz elements
Krylov basis vectors : **n** m
Hessenberg matrix : m m

Scalar products, at j fixed:
Sparse Matrix-vector product : n of size C
Orthogonalization : j of size n

m, the subspace size, may be auto-tuned at runtime to minimize the space memory occupation and the number of scalar product, with better or approximately same convergence behaviors.

GMRES example : about memory space, dot products and sparse matrix vector multiplication

1 matrix vector multiplication

m, subspace size

J scalar product

1. *Start*: Choose x_0 and compute $r_0 = f - Ax_0$ and $v_1 = r_0/\|r_0\|$.
2. *Iterate*: For $j = 1, 2, \cdots, m$ do:
   $h_{i,j} = (Av_j, v_i), i = 1, 2, \cdots, j,$
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i,$
   $h_{j+1,j} = \|\hat{v}_{j+1}\|$, and
   $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$.
3. *Form the approximate solution*:
   $x_m = x_0 + V_m y_m$, where y_m minimizes $\|\beta e_1 - \bar{H}_m y\|$, $y \in R^m$.
4. *Restart*:
   Compute $r_m = f - Ax_m$; if satisfied then stop
   else compute $x_0 := x_m$, $v_1 := r_m/\|r_m\|$ and go to 2.

Subspace computation : $O(m^3)$

Memory space :
sparse matrix : nnz elements
Krylov basis vectors : n **m**
Hessenberg matrix : **m m**

Scalar products, at j fixed:
Sparse Matrix-vector product : n of size C
Orthogonalization : j of size n

m, the subspace size, may be auto-tuned at runtime to minimize the space memory occupation and the number of scalar product, optimize the convergence behaviors, considering the "less parallel" computation on the subspace.

GMRES : about memory space and dot products

1. *Start*: Choose x_0 and compute $r_0 = f - Ax_0$ and $v_1 = r_0/\|r_0\|$.
2. *Iterate*: For $j = 1, 2, \cdots, m$ do:
   $h_{i,j} = (Av_j, v_i), i = 1, 2, \cdots, j,$
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i,$
   $h_{j+1,j} = \|\hat{v}_{j+1}\|$, and
   $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}.$
3. *Form the approximate solution*:
   $x_m = x_0 + V_m y_m$, where y_m minimizes $\|\beta e_1 - \bar{H}_m y\|$, $y \in R^m$.
4. *Restart*:
   Compute $r_m = f - Ax_m$; if satisfied then stop
   else compute $x_0 := x_m$, $v_1 := r_m/\|r_m\|$ and go to 2.

Incomplete orthogonalization (Y. Saad): i.e. i= from max(1,j-q) to j
q>0. Then, J-q+1 bands on the Hesseberg matrix.

Memory space :

- sparse matrix : nnz (i.e. < C n) elements
- Krylov basis vectors : n m
- Hessenberg matrix : m m

Scalar products, at j fixed:

- Sparse Matrix-vector product : n of size C
- Orthogonalization : m of size n

m, the subspace size, may be auto-tuned at runtime to minimize the space memory occupation and the number of scalar product, with better or approximately same convergence behaviors. The number of vectors othogonalized with the new one may be auto-tuned at runtime. The subspace size may be large!

Previous works, subspace auto-tuning algorithms

- Subspace size : different auto-tuning at runtime
  - Subspace size increase, until a fixed limit [Katagiri00][Sosonkina96],..
  - Subspace size decrease, until a fixed limit [Baker09],…..
  - Restart Trigger [Zhang04], restart when stagnation is detected.
- Orthogonalization : no auto-tuning at runtime
  - Prior to execution : [Jia94]

Remark, *in general*:

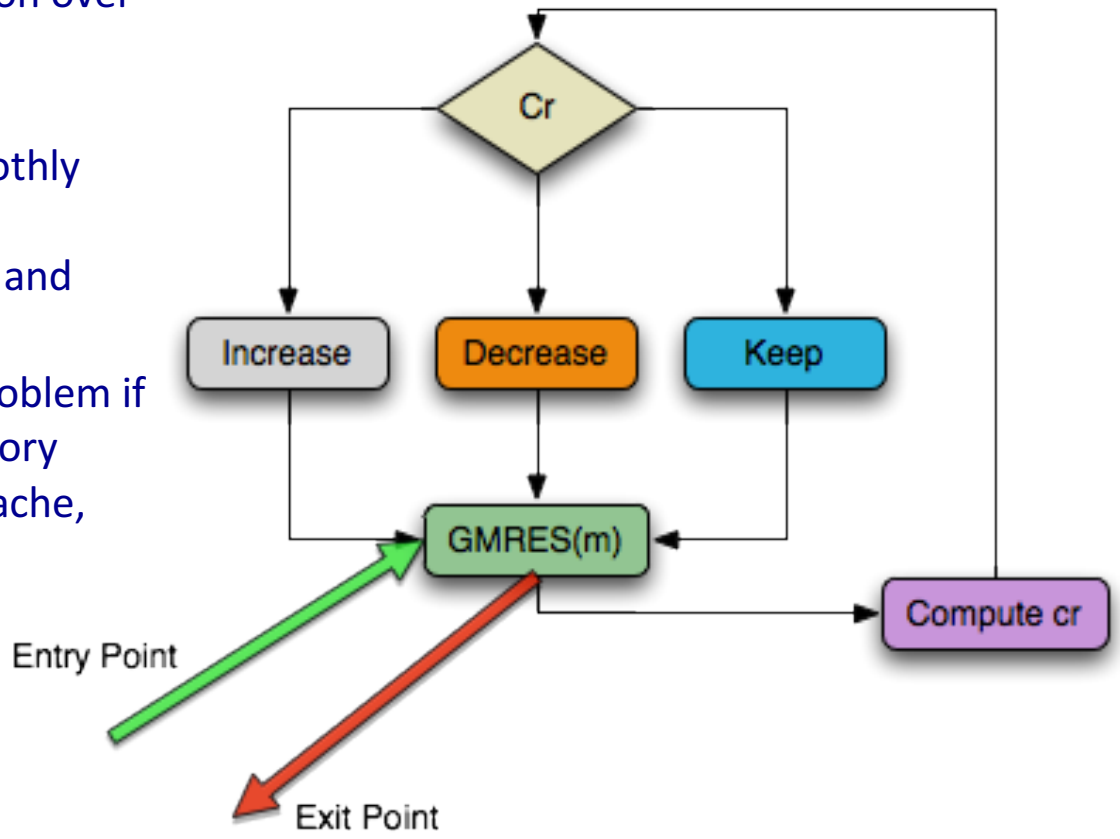
- Greater subspace size -> better convergence/long restart, less iterations
- Smaller subspace size -> slow convergence, stagnation, short restart, more iteration
- Choice of m is mandatory.

For difficult problems we have to use a large subspace size to reach convergence : the numerical stability and the quality of the orthogonalization are crucial.

Auto-Tuning Algorithms

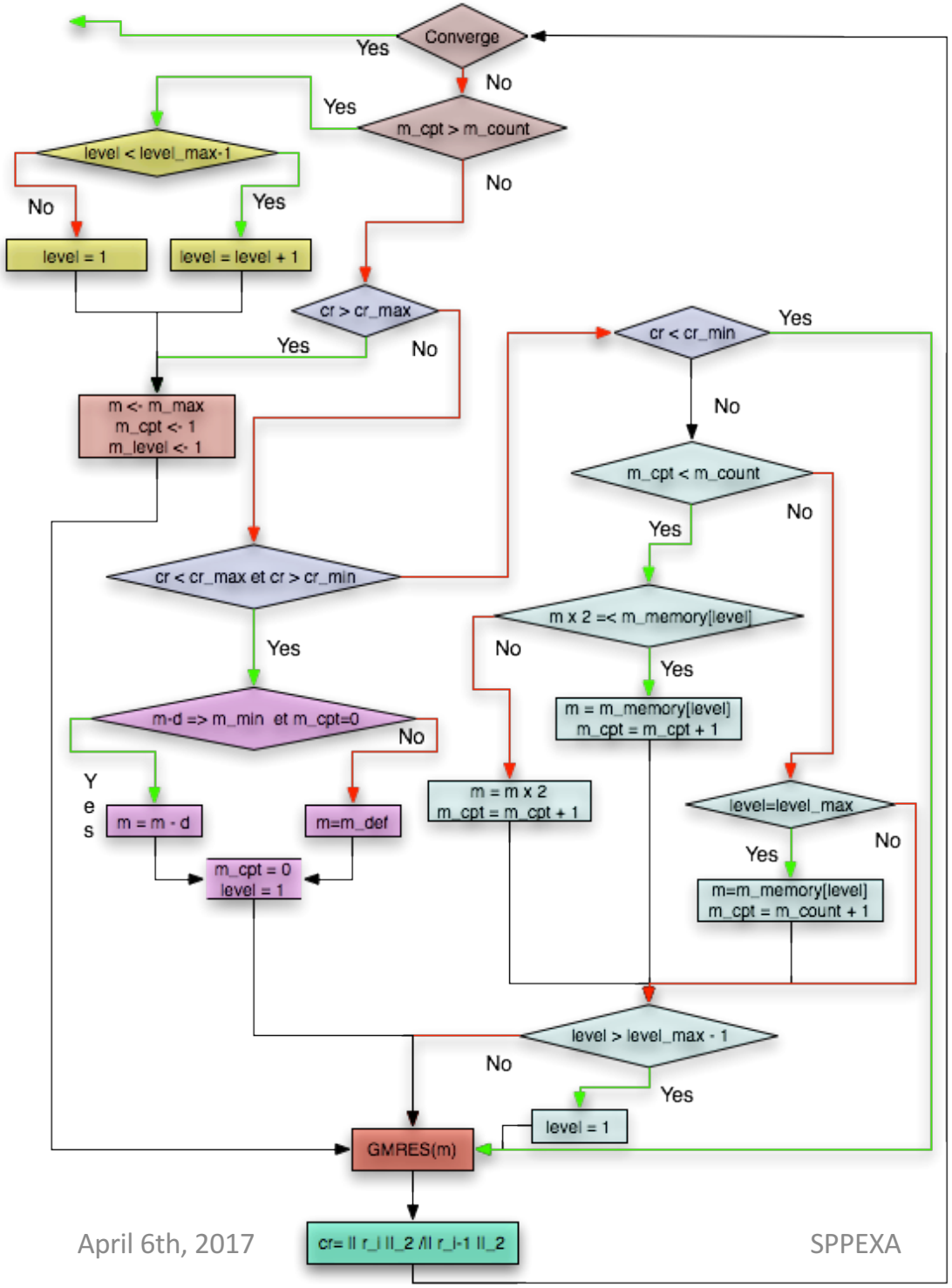
with Pierre-Yves Aquilenti (TOTAL/U. Lille 1, now at TOTAL/Houston)

- Subspace size
  - Evaluate convergence progression over some iterations.
  - Decrease if convergence are monotonous or if they are smoothly slowing (approximately same convergence but minimize time and space)- **Cr medium**
  - Increase if convergence stall (problem if we increase too much the memory space), Track memory levels : Cache, RAM, Nodes. **Cr low**
  - Do nothing if **Cr high**



Cr = norm2 (r_i) / norm2(r_{i_1})

Easy to implement using libraries



Parameters

d : number of steps between successive decreases,

m_min: minimum subspace size value,

m_max : maximum subspace size value,

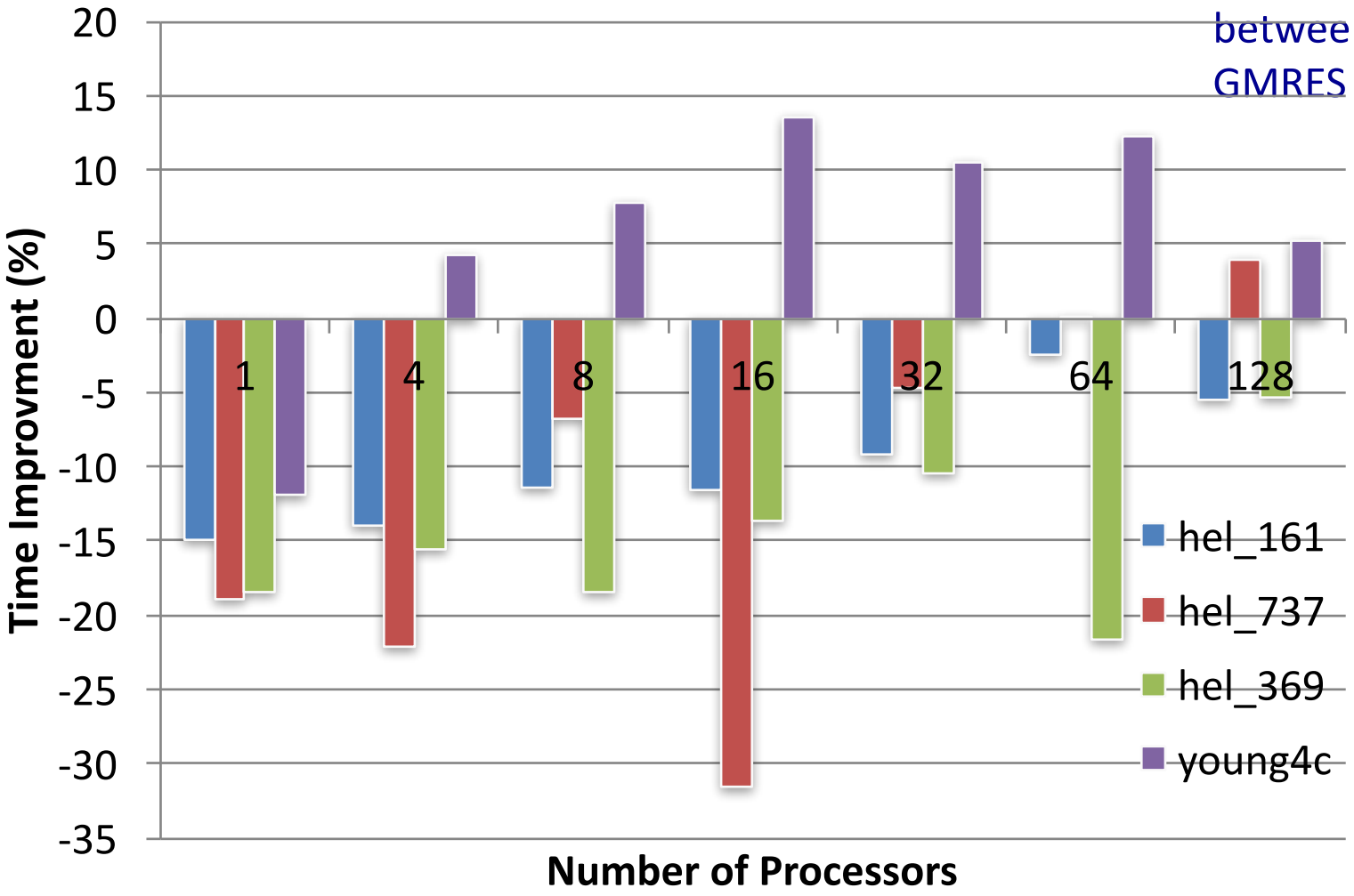
m_counts : number of successive soft increase before intending “special” Increases,

m_memory[ ] : array containing subspace size values for hardware increase

Just a few lines of code to add at the beginning and at the end of the main loops, without any other modifications

- Our algorithm compared to no auto-tuning

Simple Strategy, decrease or keep the subspace size between successives GMRES restarts



Less is better

Outline

- Introduction
- Krylov subspace auto-tuning algorithm
- **GMRES Cache Aware Auto-tuning strategies**
- Orthogonalization auto-tuning algorithm
- ERAM restart strategy auto-tuning
- Unite and Conquer MERAM
- Sparse format auto-tuning
- Intelligent Krylov method for extreme computing
- Conclusion

Auto-Tune Restarted GMRES With Caches

with Pierre-Yves Aquilenti (TOTAL)

- We define levels of GMRES restart parameter auto-tuning increase depending on levels of memory

Cache L1	Cache L2	Cache L3	RAM	SWAP
4	10	20	200	1000

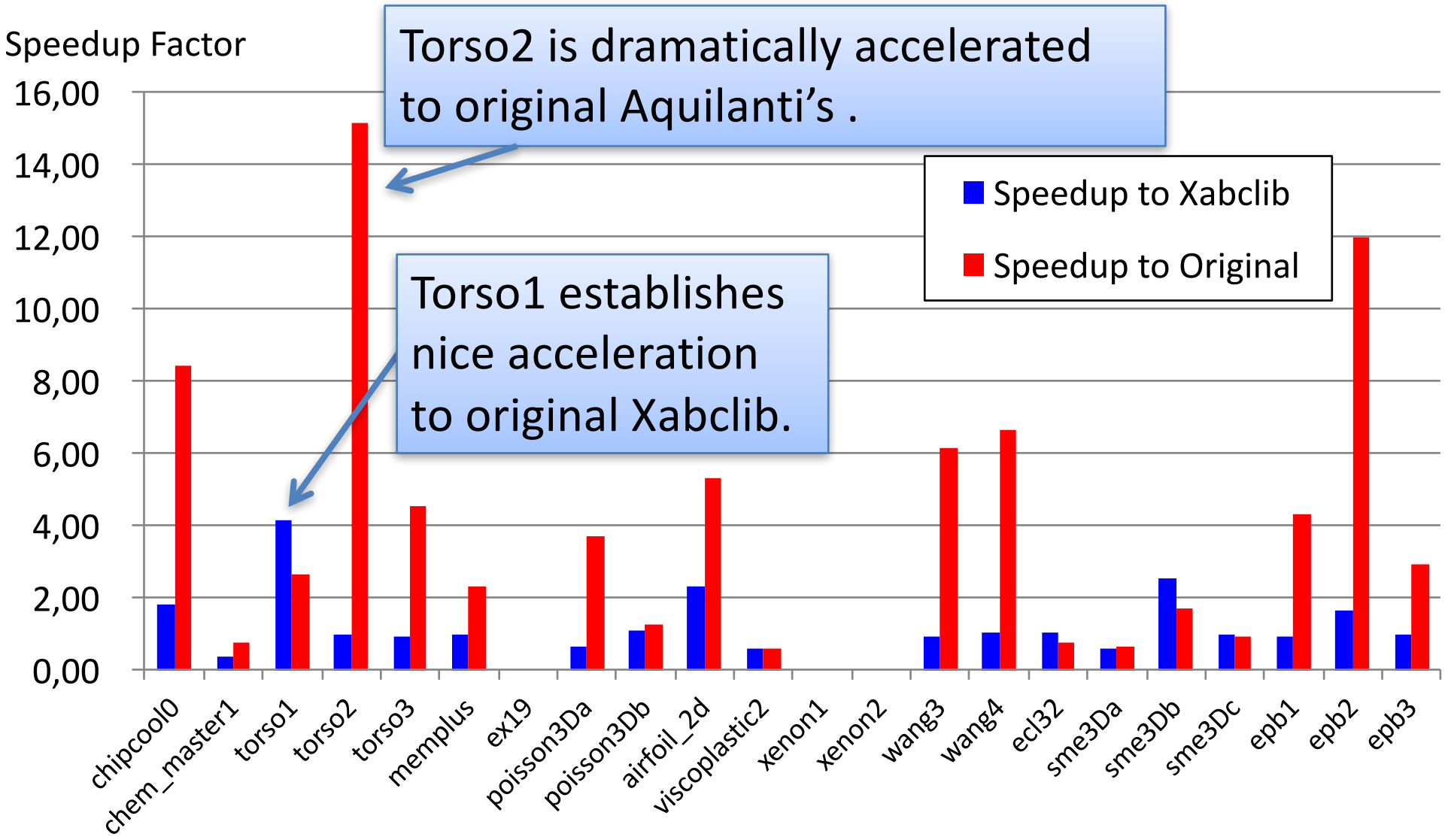
$$(nnz + 3n + m(m+1) + n(m+1)) \times SizeOfScalar \geq MemoryBytesLevel$$

$$\frac{MemoryBytesLevel}{SizeOfScalar} - nnz - 4n = m^2 + m(n+1)$$

nnz = number of non zeros of the matrix
m = subspace size / restart parameter
n = matrix size

With Pierre-Yves Aquilenti (TOTAL) and Takahiro Katagari (U. Tokyo)

16 Threads on the T2K Open Supercomputer (1 node), Xabclib_GMRES V1.00



Outline

- Introduction
- Krylov subspace auto-tuning algorithm
- GMRES Cache Aware Auto-tuning strategies
- **Orthogonalization auto-tuning algorithm**
- ERAM restart strategy auto-tuning
- Unite and Conquer MERAM
- Experimental results
- Intelligent Krylov method for extreme computing
- Conclusion

Incomplete orthogonalization Auto-Tuning

Complete orthogonalisation : we orthogonalise with all the previous computed vectors of the basis, i.e. at step k, we ortogonalise with k vectors, which generates k scalar product at step k.

Incomplete orthogonalisation : we orthogonalize with only min(k,q) previous computed vectors of the basis, i.e. at step k, we ortogonalise only with min(k,q) vectors, q < m. DQGMRES : [Saad '94], DQGMRES : [Wu '97] IGMRES : [Brown '86][Jia '07]

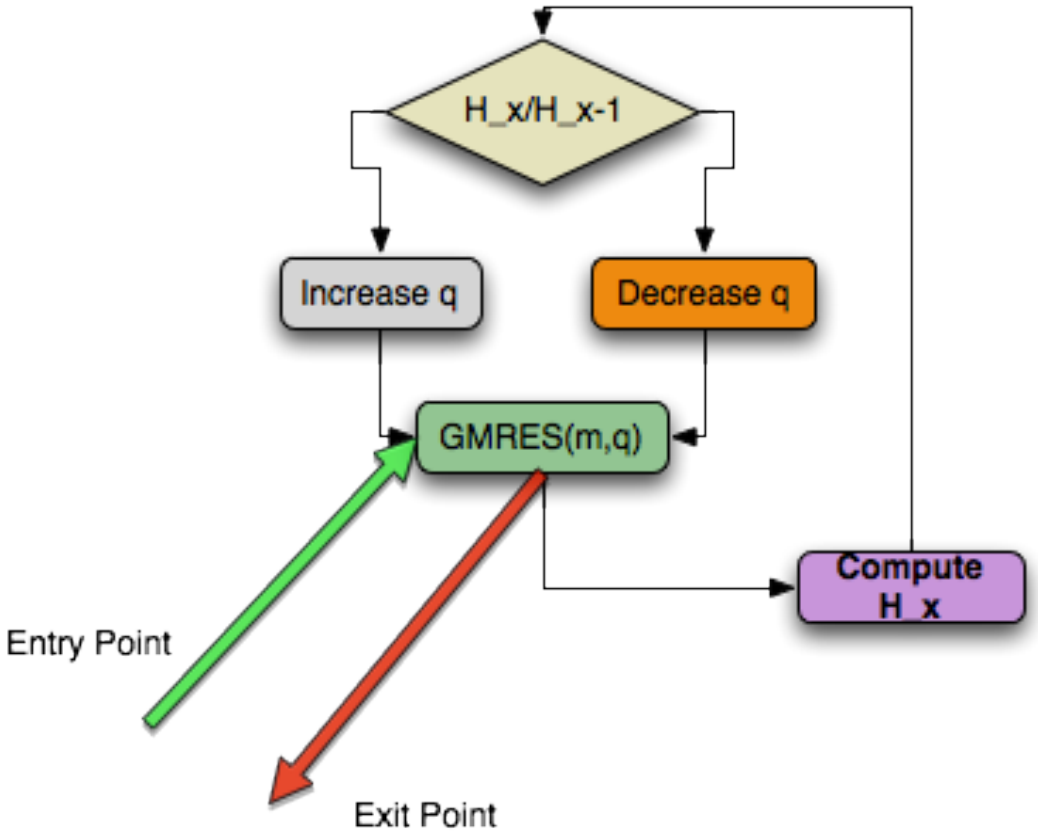
Then, we have only q scalar product at step k (for k > or equal to q).

Complete ortogonalization : k scalar product for k fixed
Incomplete orthogonalization : q scalar product for k fixed, q < k

We may then save k-q scalar products, for q < k, and, then, several synchronized communications .

Even, if the number of iterations may be a little larger, we minimize a lot of long globa communications generated by scalar products.

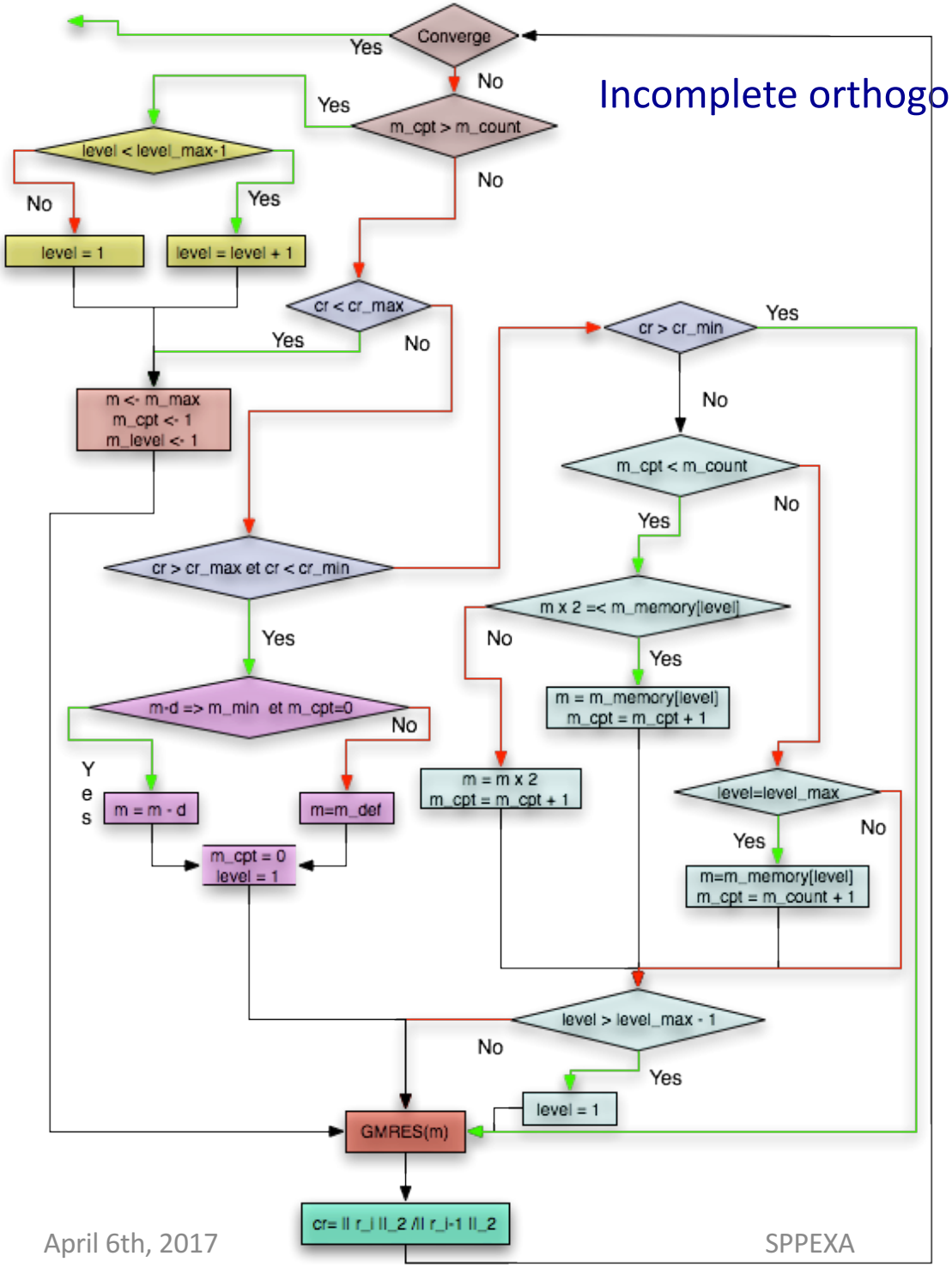
Incomplete orthogonalization algorithm at runtime



- Evaluate iteration costs in time vs. Convergence
- Decrease number of orthogonalized vectors q if ratio convergence/(time iteration) decrease

A complex heuristic-based algorithm : With respect to the variation of the residual between restarts, we change the number q of vectors concerned by the orthogonalization

Still, a lot of researches to achieve to optimize this algorithm.



Incomplete orthogonalization Auto-tuning Algorithm

q_{min}= minimum number of vector to orthogonalize

q_{max}= maximum number of vector to orthogonalize, typically = m the gmres subspace size

T_x = time of the x^{th} restart

N_x = norm of the residual variation, equal to the norm of the duration of the x^{th} restart minus the duration of the $x\text{-}1^{th}$ restart

H_x = relative variation
$= N_x / T_x$

Heuristic = ratio of the relative variation between restart x and x-1, equal H_x / H_{x-1}

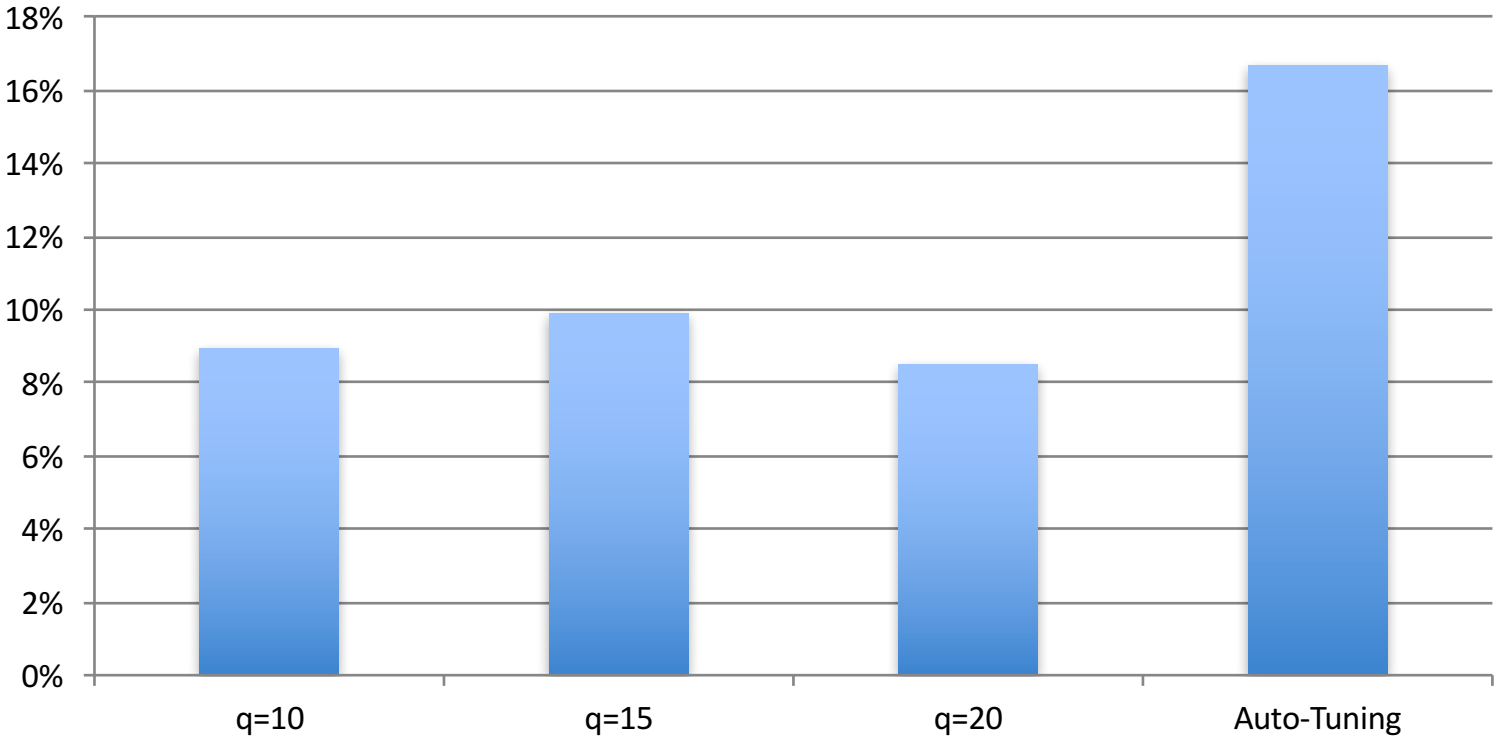
RESULTS Incomplete orthogonalization auto-tuning

Hardware : 2.26Ghz, Core2Duo, 4GB ram, PETSc 3.0

Matrix young4c from matrix market
Solution at 10^{e}-6
GMRES subspace size m=30
Truncation at 10, 15, 20
Serial processing

6 experiments for each case,
took the best time, SEQUENTIAL

Percentage of improvement over full orthogonalisation
in term of computation time. Iteration number does not vary much



Higher is better

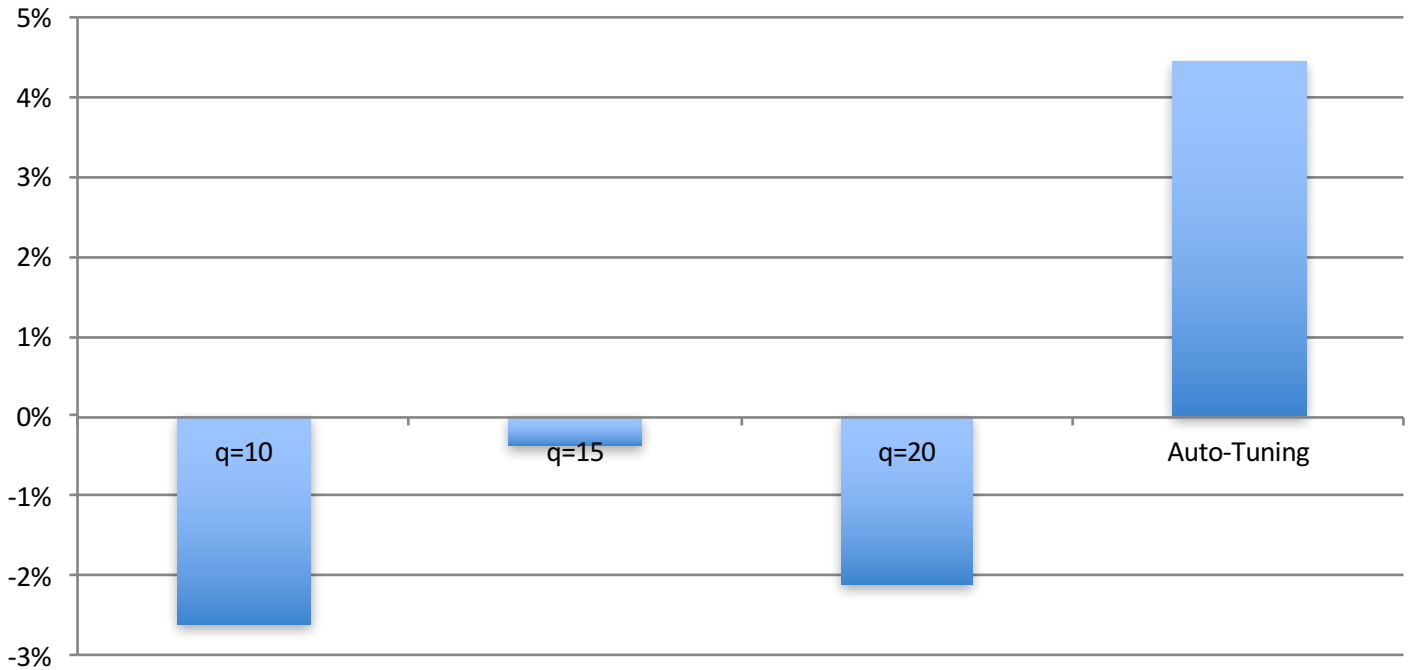
RESULTS Incomplete orthogonalization auto-tuning

Hardware : 2.26Ghz, Core2Duo, 4GB ram, PETSc 3.0

Matrix hel369 from matrix market
Solution at 10^{e}-3
GMRES subspace size m=30
Truncation at 10, 15, 20
Serial processing

6 experiments for each case,
took the best time

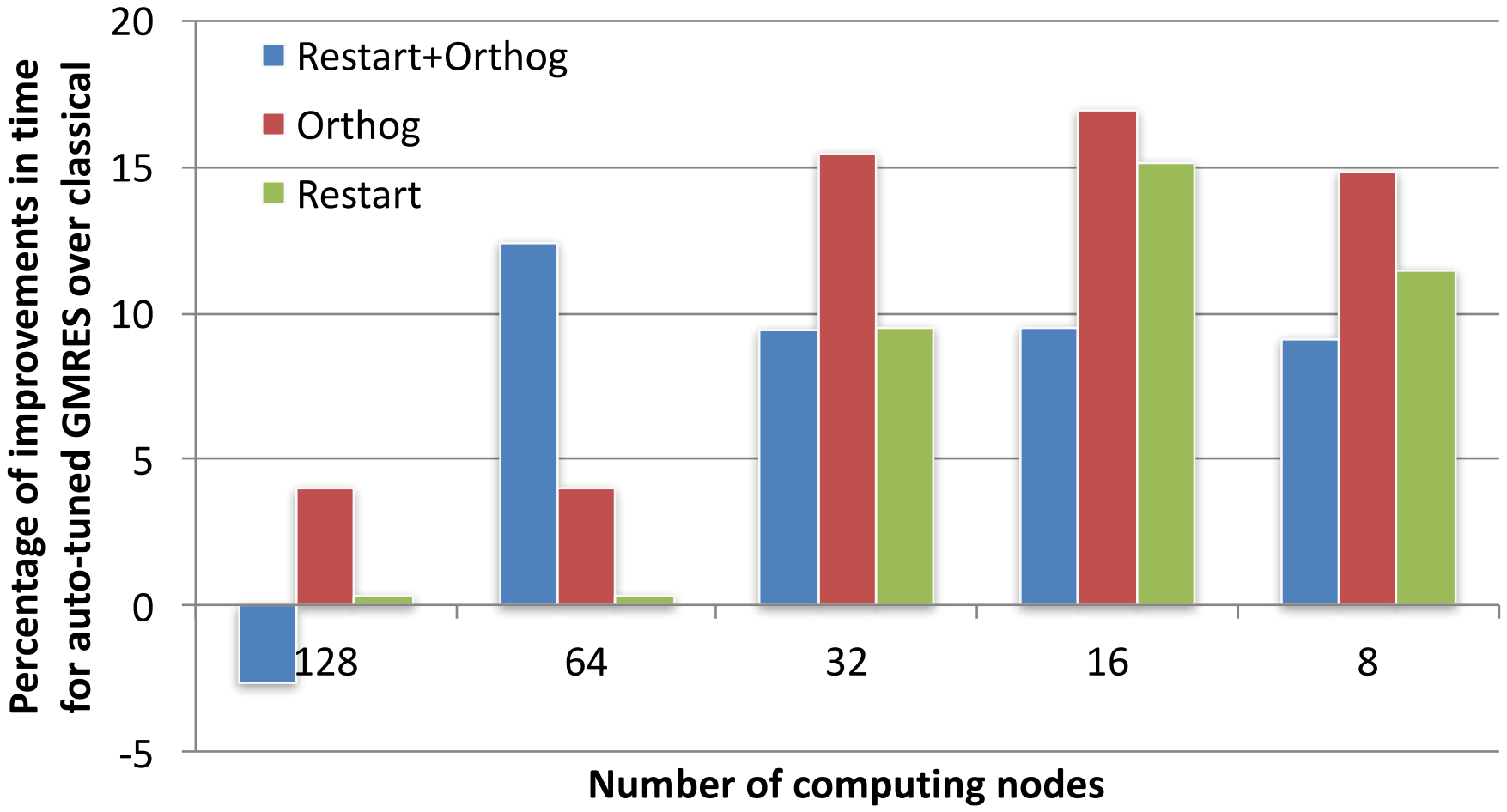
Percentage of improvement over full orthogonalisation
in term of computation time.



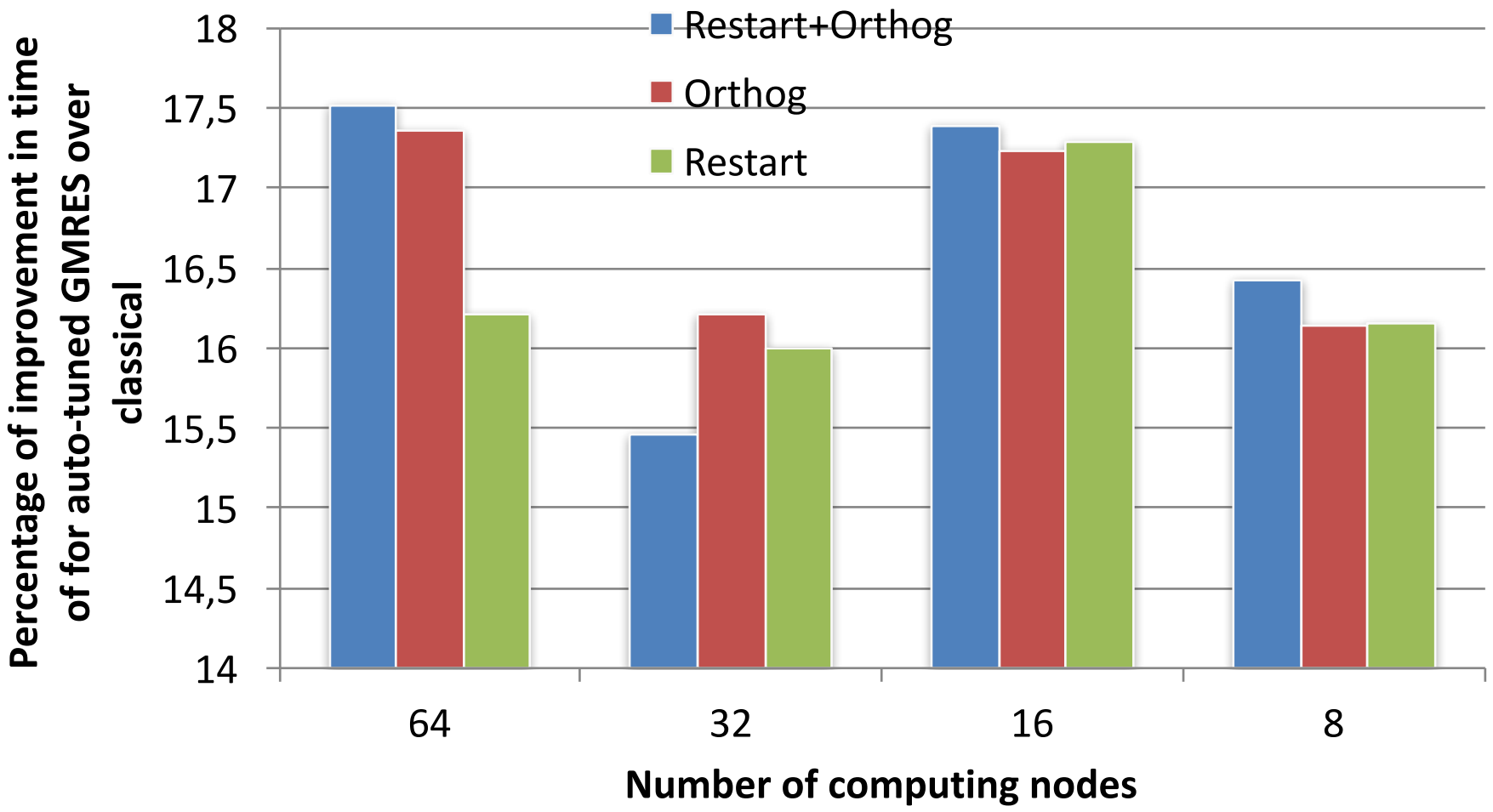
Higher is better

Results : Industrial Case

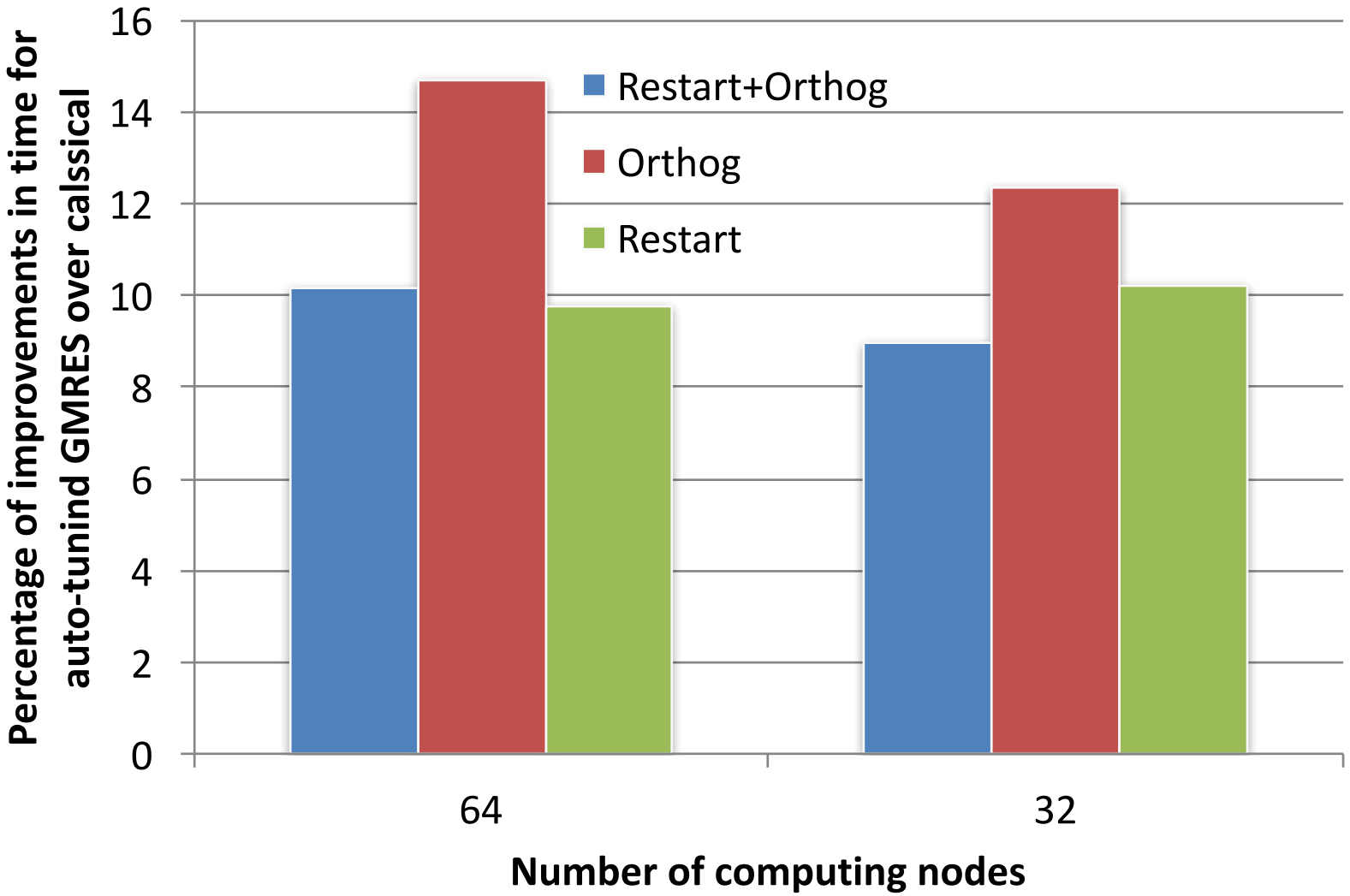
number of unknown = (119 x 119 x 115), 3Hz, m=10



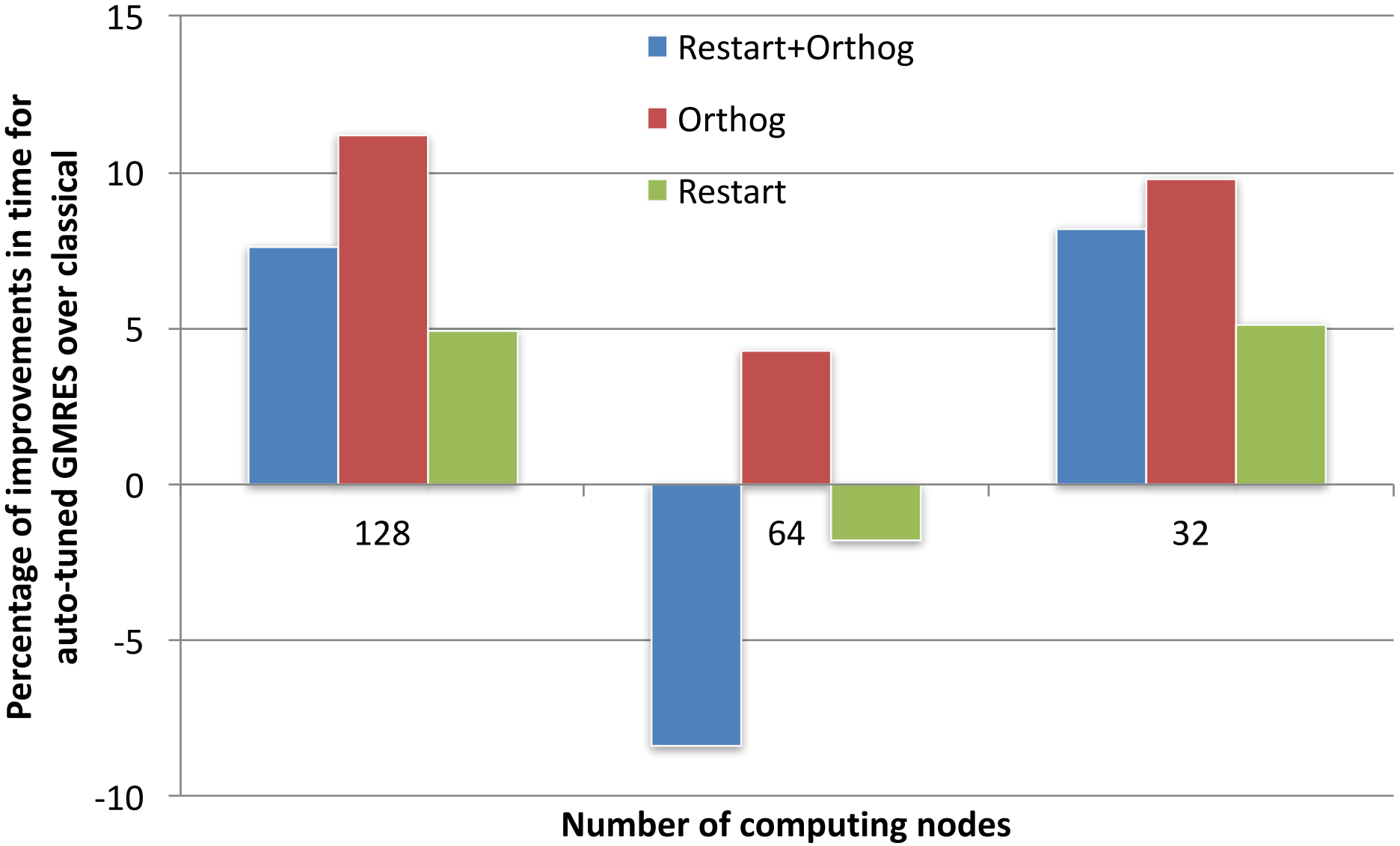
number of unknown = (119 x 119 x 115), 3 Hz, m=30



number of unknown = (183 x 183 x 191), 5 Hz, m=10



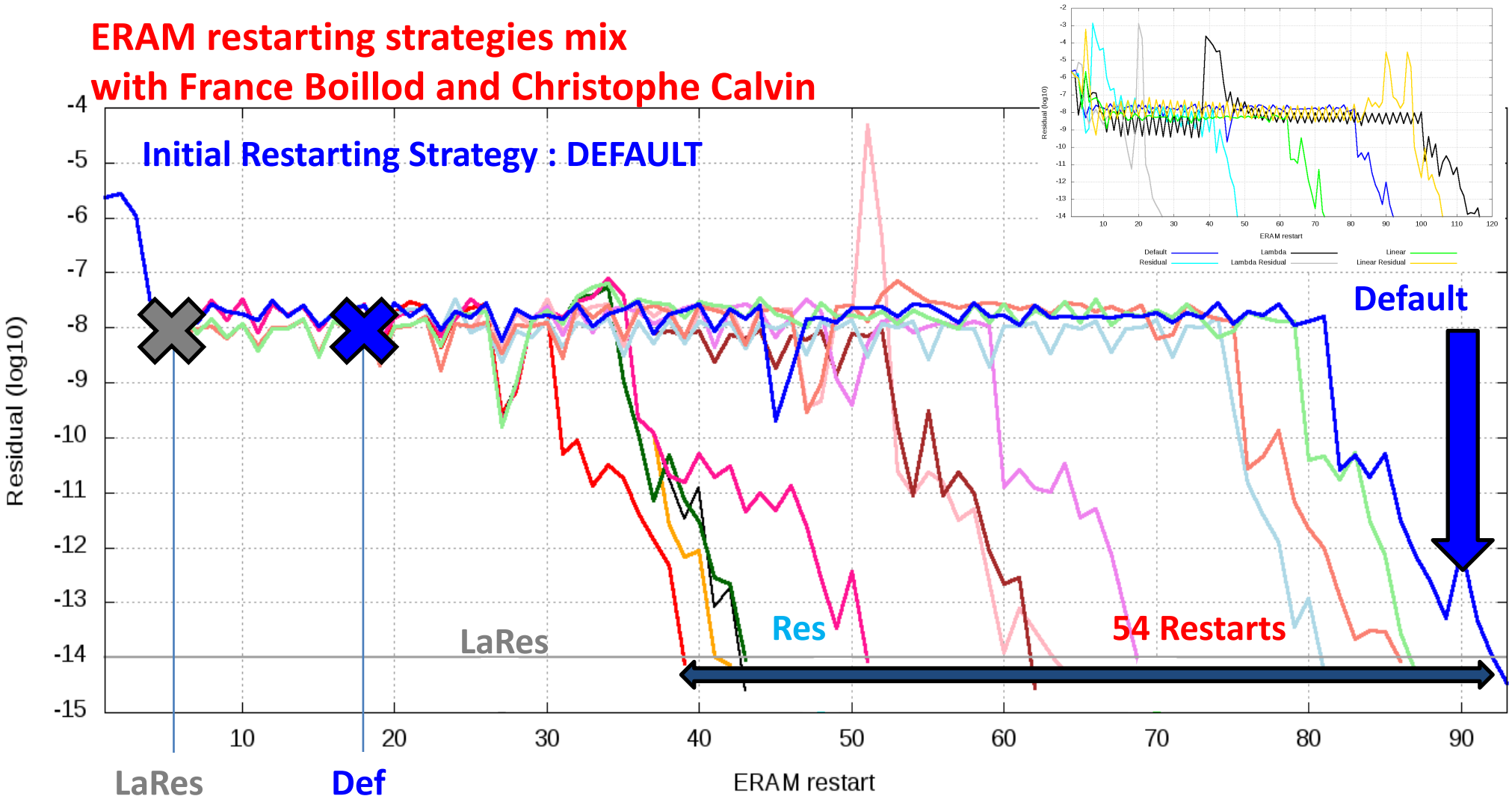
number of unknown = (335 x 327 x 383), 5Hz, m=30



Outline

- Introduction
- Krylov subspace auto-tuning algorithm
- GMRES Cache Aware Auto-tuning strategies
- Orthogonalization auto-tuning algorithm
- **ERAM restart strategy auto-tuning**
- Unite and Conquer MERAM
- Sparse format auto-tuning
- Intelligent Krylov method for extreme computing
- Conclusion

ERAM restarting strategies mix
with France Boillod and Christophe Calvin



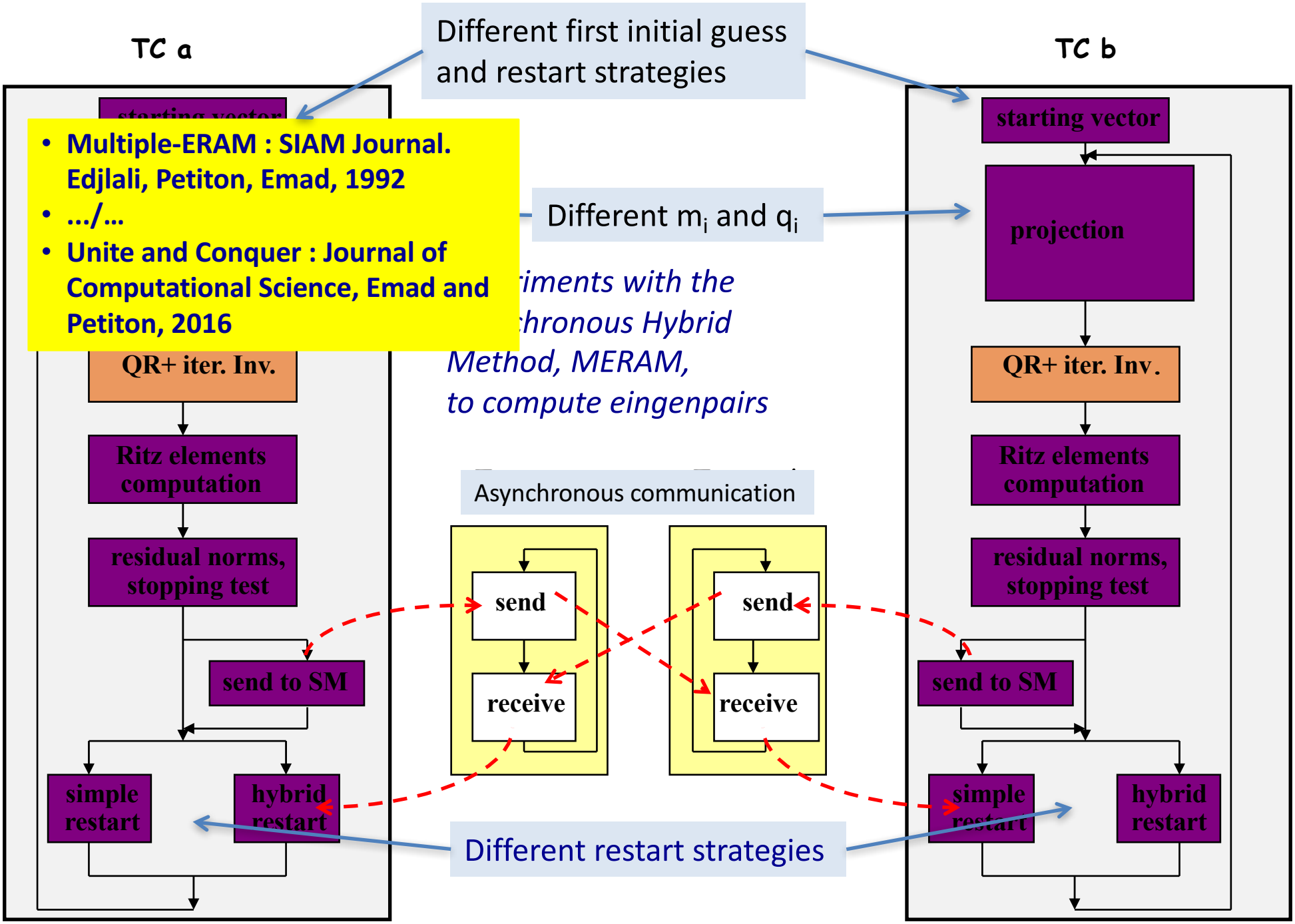
4 eigenpairs,
m=15, CGSr
Bayer04 Matrix
- N=20545
- nnz=85537
- 1 Intel i5-2430M

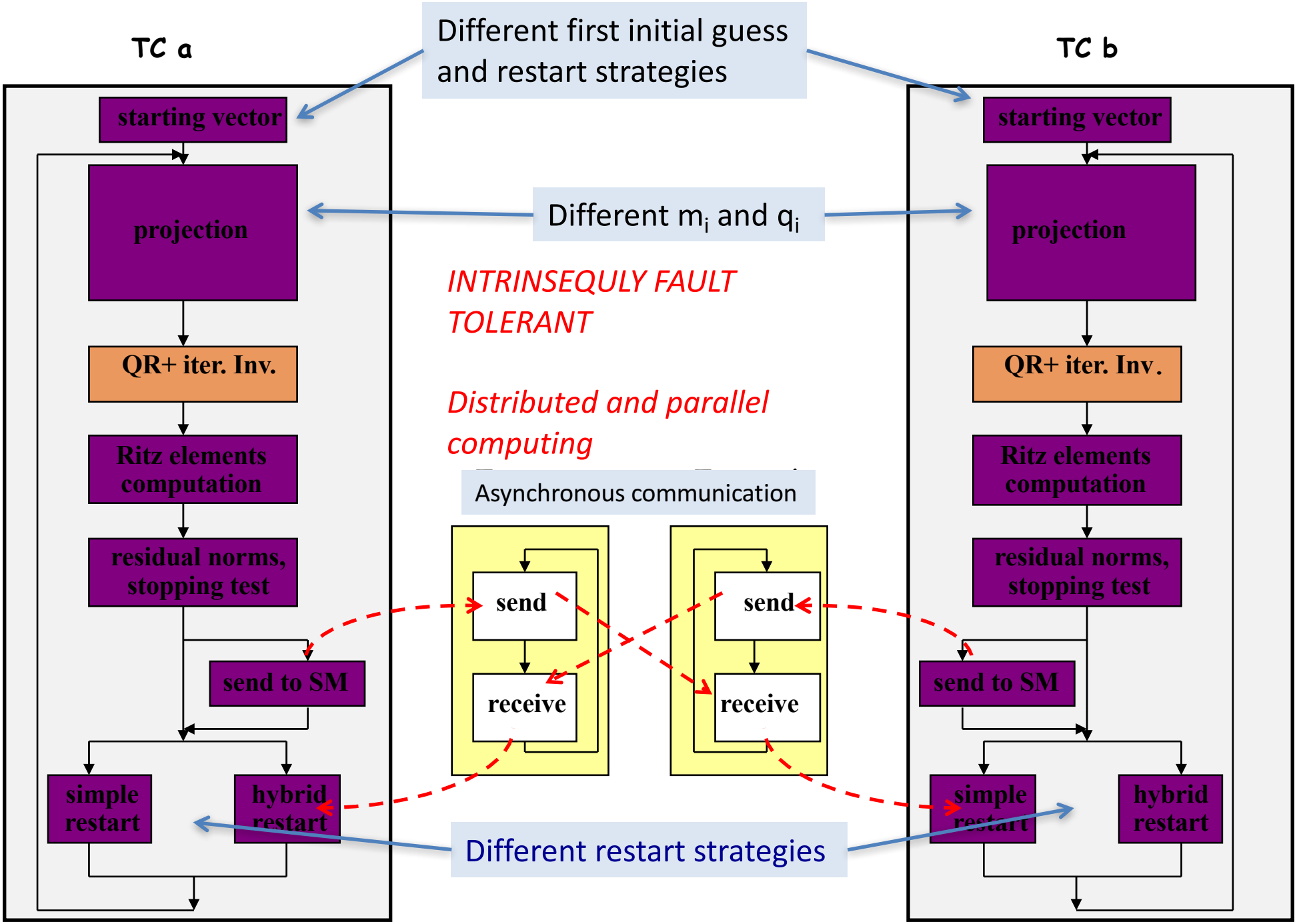
LaRes(5), Def(18)

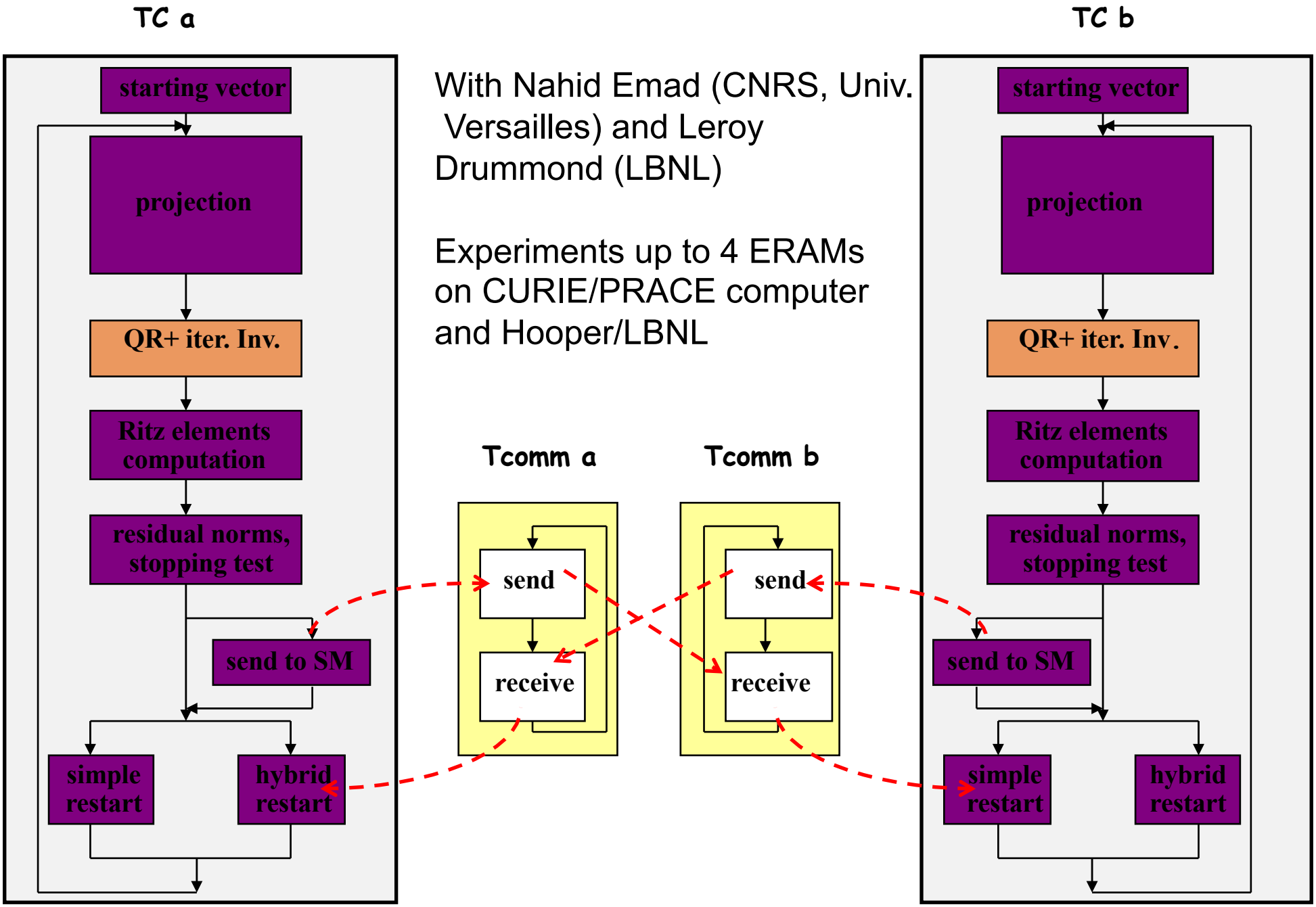
Li(38)
Res(26)
Res(14), LaRes(24)
LaRes(4), Def(30)
LaRes(5), Def(21)
DEFAULT

Outline

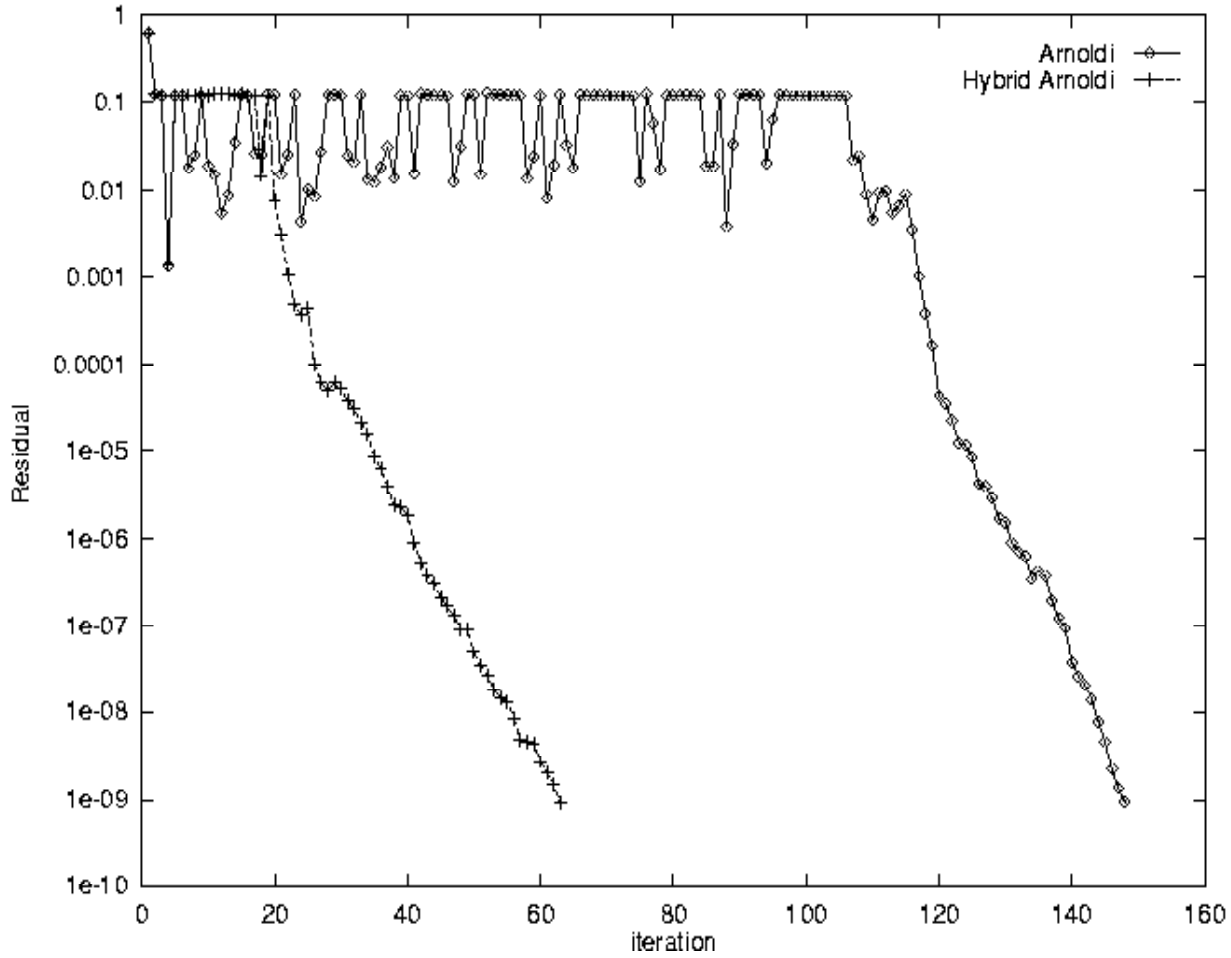
- Introduction
- Krylov subspace auto-tuning algorithm
- GMRES Cache Aware Auto-tuning strategies
- Orthogonalization auto-tuning algorithm
- ERAM restart strategy auto-tuning
- **Unite and Conquer MERAM**
- Sparse format auto-tuning
- Intelligent Krylov method for extreme computing
- Conclusion


TC a
TC b
Different first initial guess and restart strategies
Multiple-ERAM : SIAM Journal. Edjlali, Petiton, Emad, 1992
.../...
Unite and Conquer : Journal of Computational Science, Emad and Petiton, 2016
Different m_i and q_i
iments with the
hronous Hybrid
Method, MERAM,
to compute eingenpairs
starting vector
projection
QR+ iter. Inv.
Ritz elements computation
residual norms, stopping test
send to SM
simple restart
hybrid restart
Asynchronous communication
send
receive
send
receive
starting vector
projection
QR+ iter. Inv.
Ritz elements computation
residual norms, stopping test
send to SM
simple restart
hybrid restart
Different restart strategies

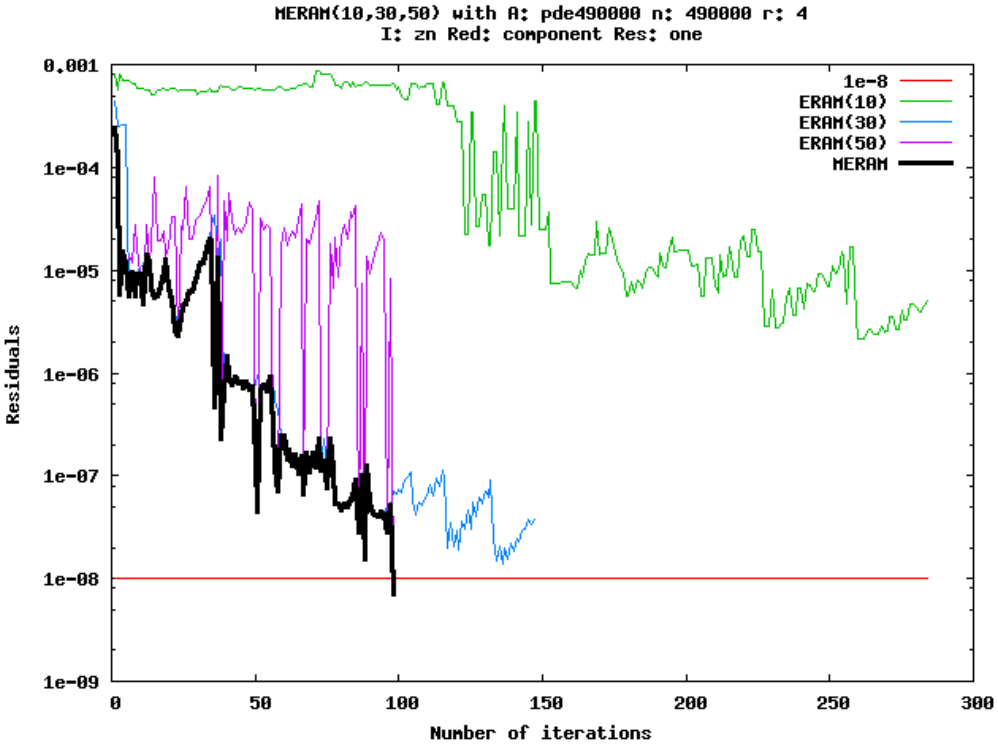

TC a
Different first initial guess and restart strategies
TC b
starting vector
projection
QR+ iter. Inv.
Ritz elements computation
residual norms, stopping test
send to SM
simple restart
hybrid restart
Different m_i and q_i
INTRINSEQULY FAULT TOLERANT
Distributed and parallel computing
Asynchronous communication
send
receive
send
receive
starting vector
projection
QR+ iter. Inv.
Ritz elements computation
residual norms, stopping test
send to SM
simple restart
hybrid restart
Different restart strategies

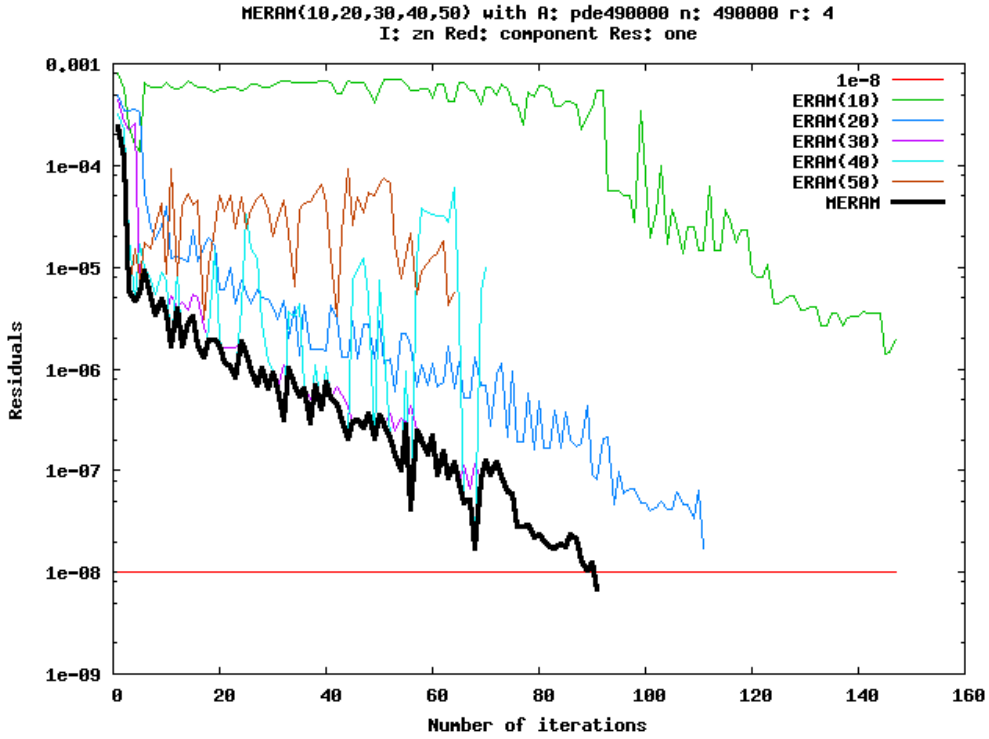

TC a
TC b
starting vector
projection
QR+ iter. Inv.
Ritz elements computation
residual norms, stopping test
send to SM
simple restart
hybrid restart
Tcomm a
Tcomm b
send
receive
With Nahid Emad (CNRS, Univ. Versailles) and Leroy Drummond (LBNL)
Experiments up to 4 ERAMs on CURIE/PRACE computer and Hooper/LBNL

MERAM, since 1993



Experiments on Grid'5000 (Nahid Emad et al)

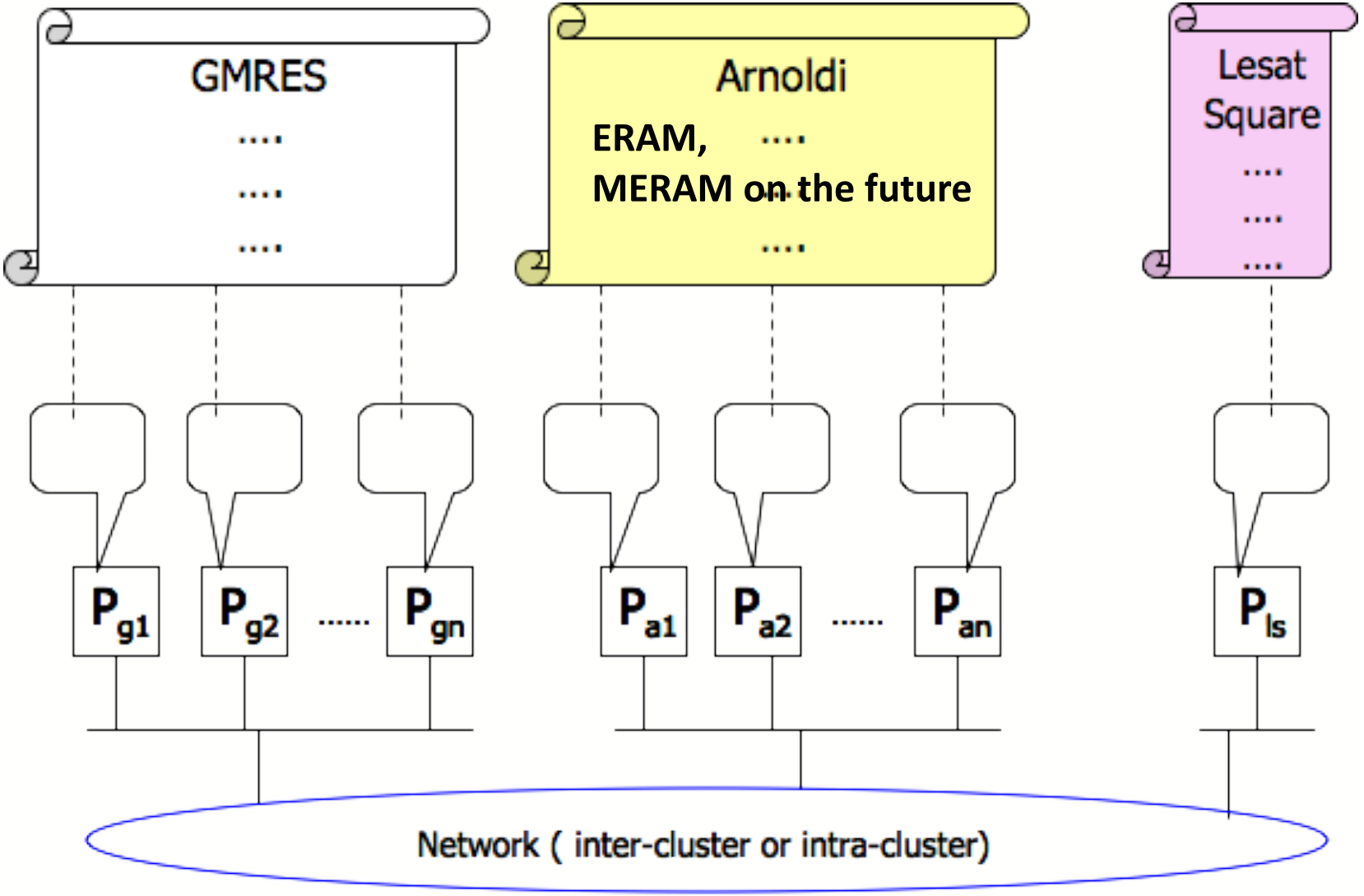




Convergence of MERAM for matrix 490000 with different number of co-methods

Asynchronous Iterative Restarted Methods

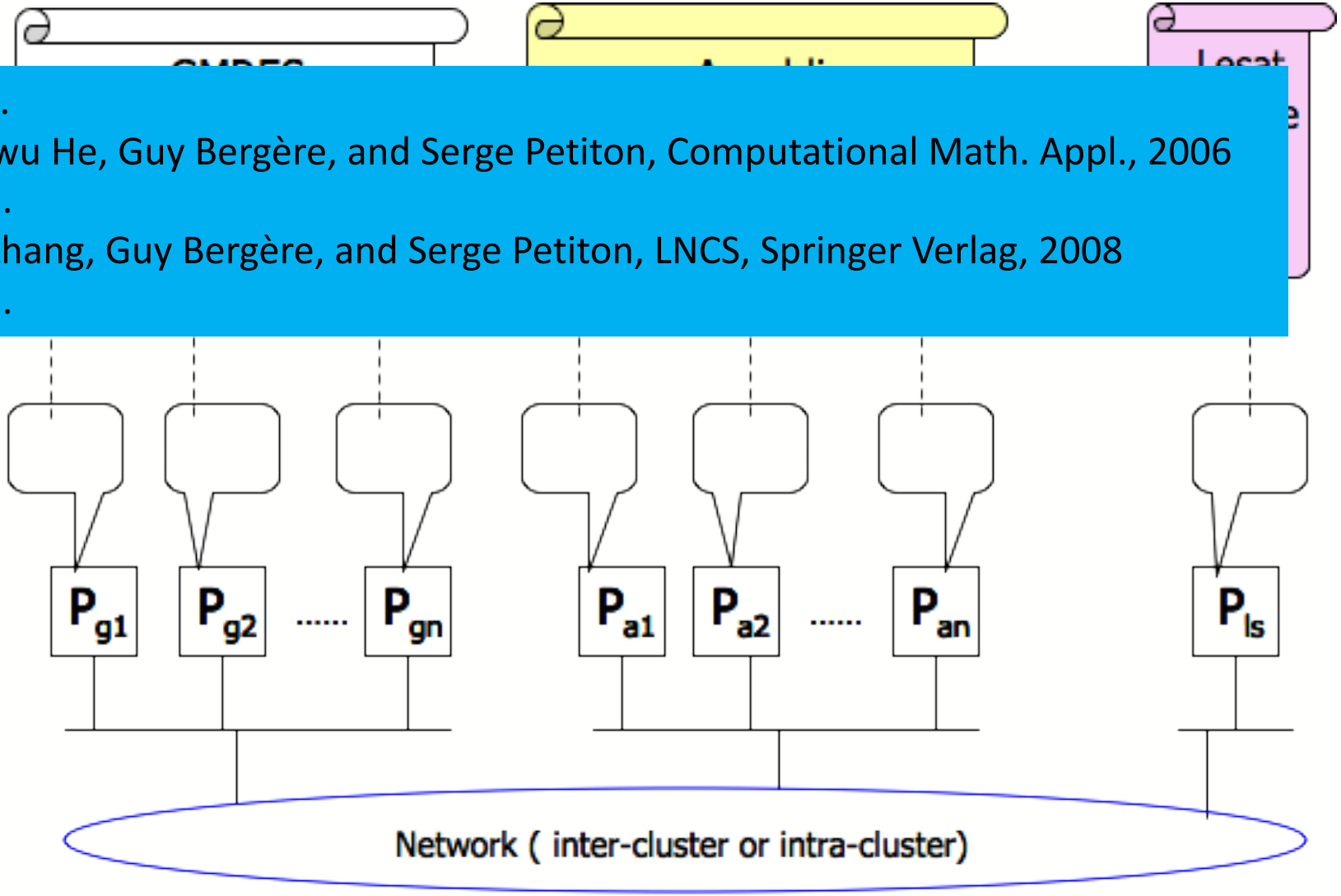
Collaboration with He Haiwu and Guy Bergère (U. Lille 1, CNRS)
and Ye Zhang (Hohai Univ. Nanjing) , Salim Nahi (Maison de la simulation),
and Pierre-Yves Aquilenti (TOTAL)



Asynchronous Iterative Restarted Methods

Collaboration with He Haiwu and Guy Bergère (U. Lille 1, CNRS)
and Ye Zhang (Hohai Univ. Nanjing) , Salim Nahi (Maison de la simulation),
and Pierre-Yves Aquilenti (TOTAL)

- .../...
- Haiwu He, Guy Bergère, and Serge Petiton, Computational Math. Appl., 2006
- .../...
- Ye Zhang, Guy Bergère, and Serge Petiton, LNCS, Springer Verlag, 2008
- .../...



Asynchronous Iterative Restarted Methods

Collaboration with He Haiwu and Guy Bergère (U. Lille 1, CNRS)
and Ye Zhang (Hohai Univ. Nanjing) , Salim Nahi (Maison de la simulation),
and Pierre-Yves Aquilenti (TOTAL)

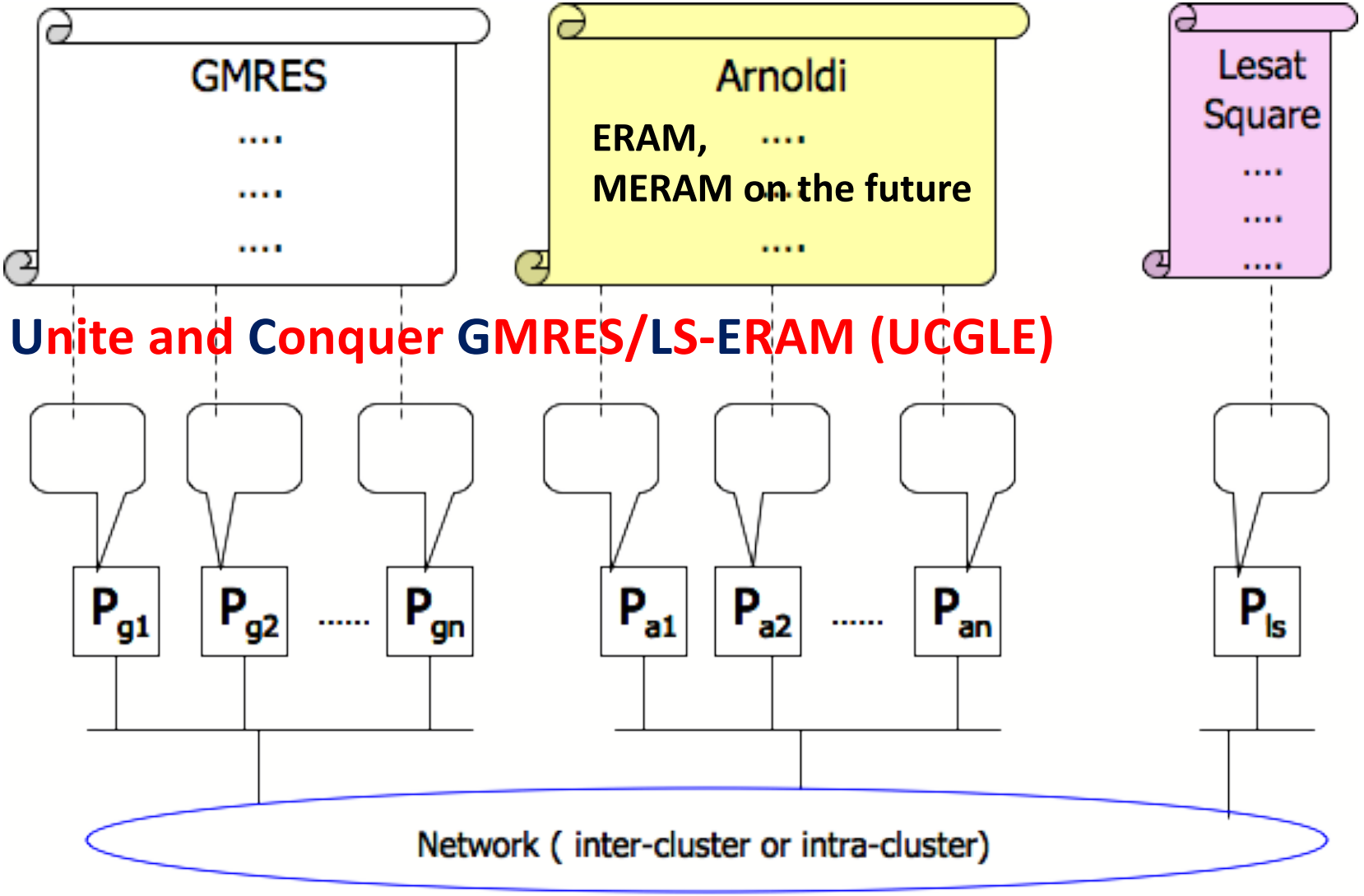
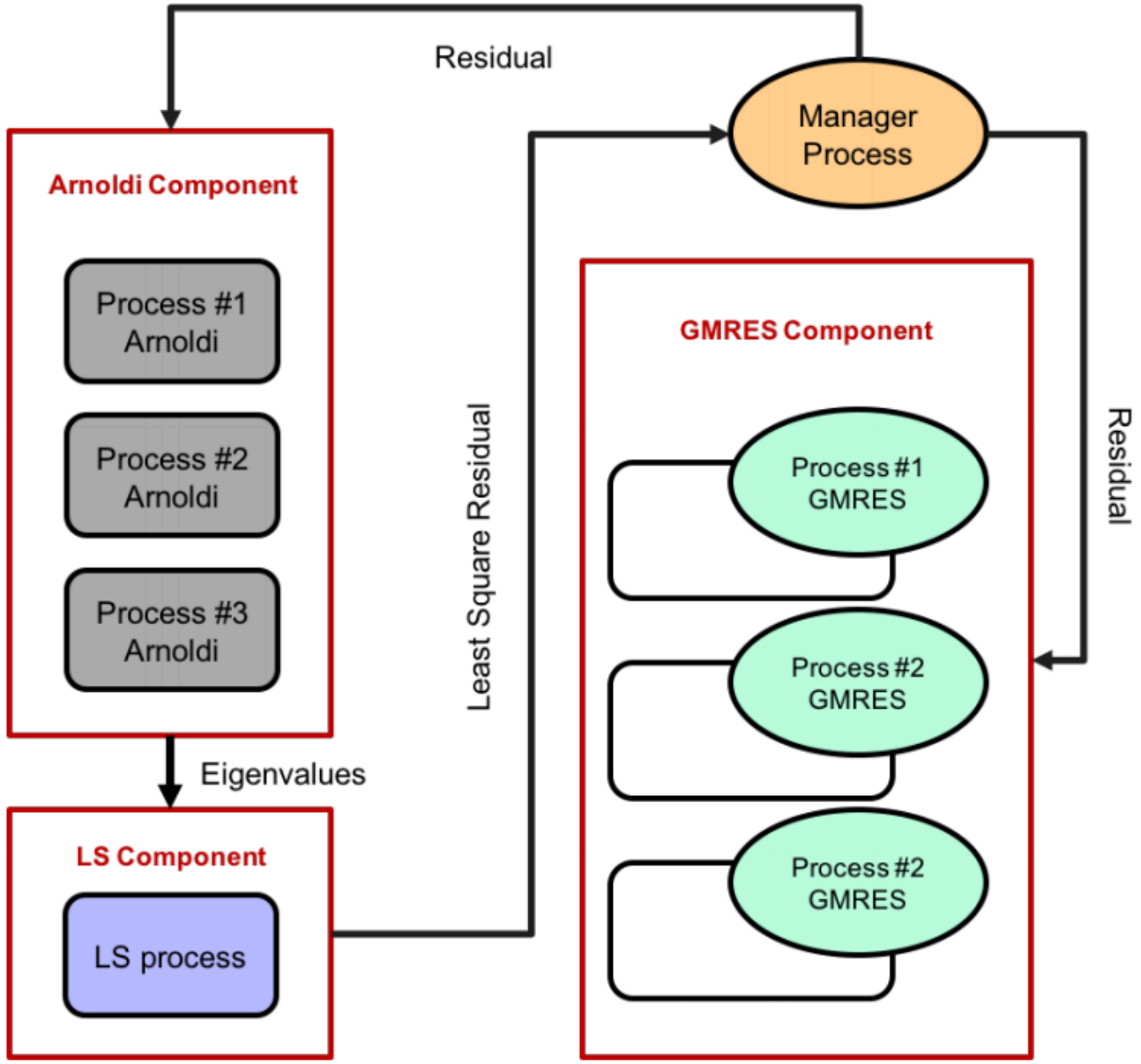


Figure: Asynchronous Communication of UCGLE method



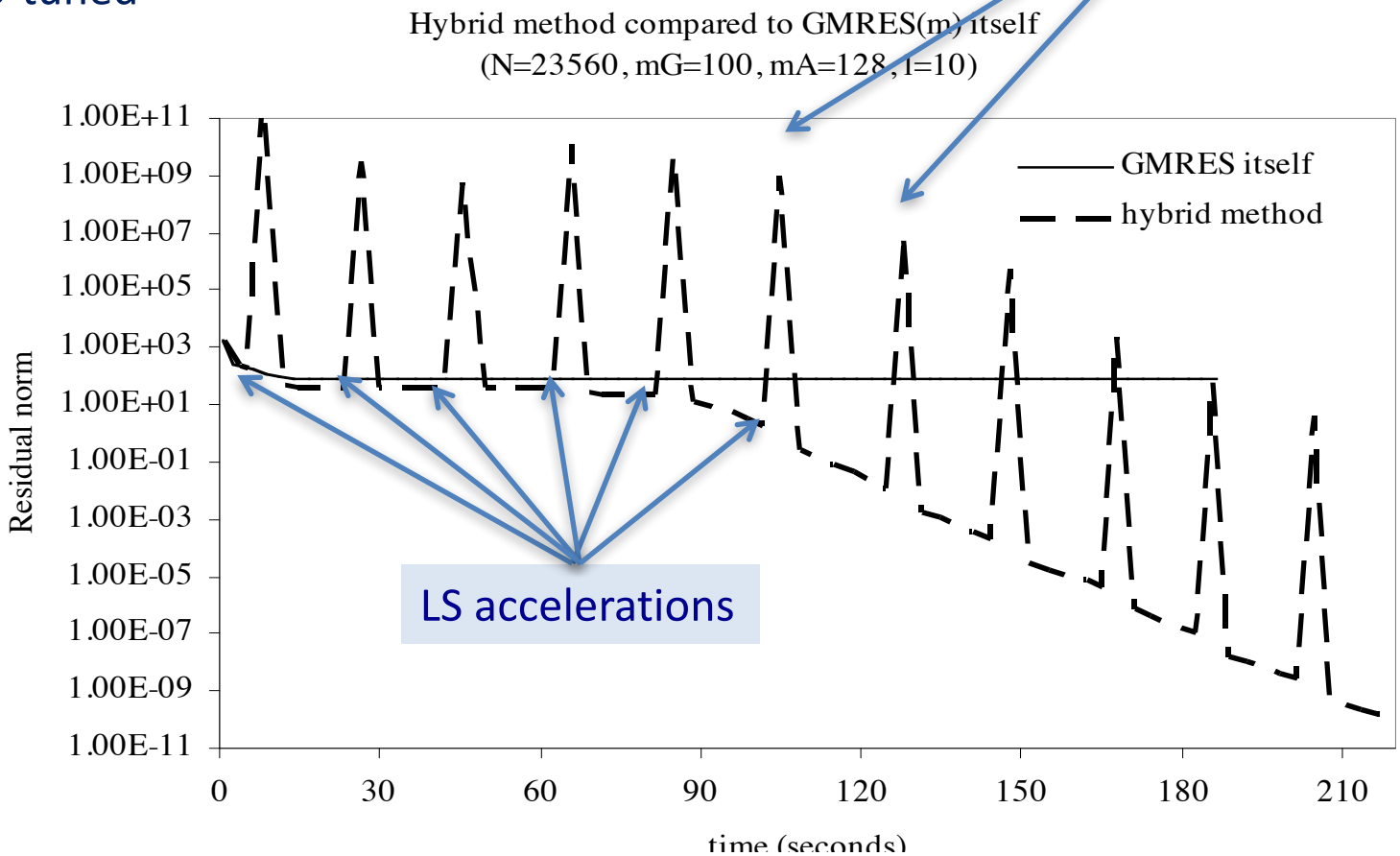
A

Numeric Results and Analysis

advantage over GMRES (difficult convergence case)

Degrees of polynomial,
number of iterations between two LS “accelerations”,….
may be auto-tuned

Well- known behaviours

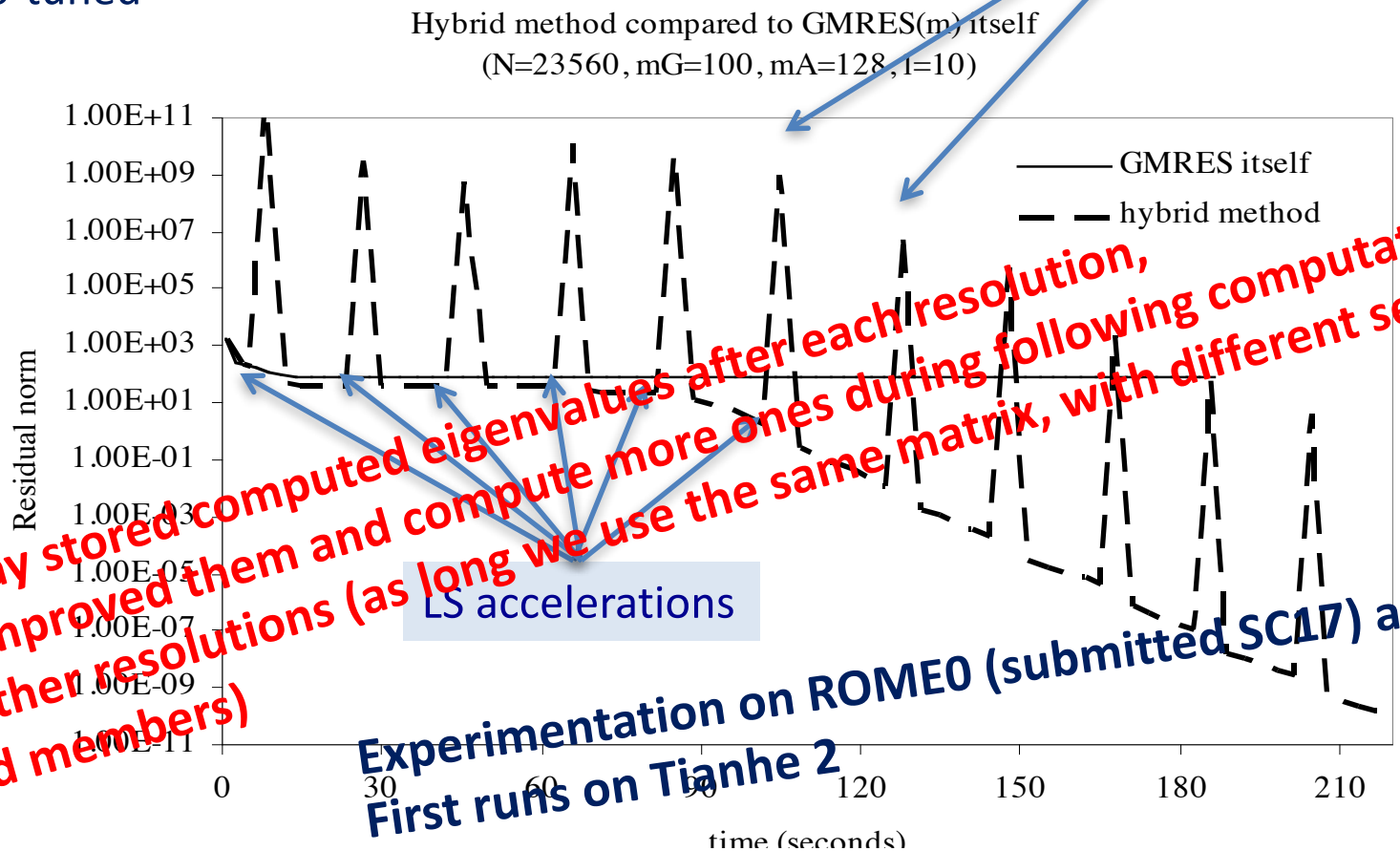


Numeric Results and Analysis

advantage over GMRES (difficult convergence case)

Degrees of polynomial,
number of iterations between two LS “accelerations”,….
may be auto-tuned

Well- known behaviours



LS accelerations

We may stored computed eigenvalues after each resolution, and improved them and compute more ones during following computation for other resolutions (as long we use the same matrix, with different second hand members)

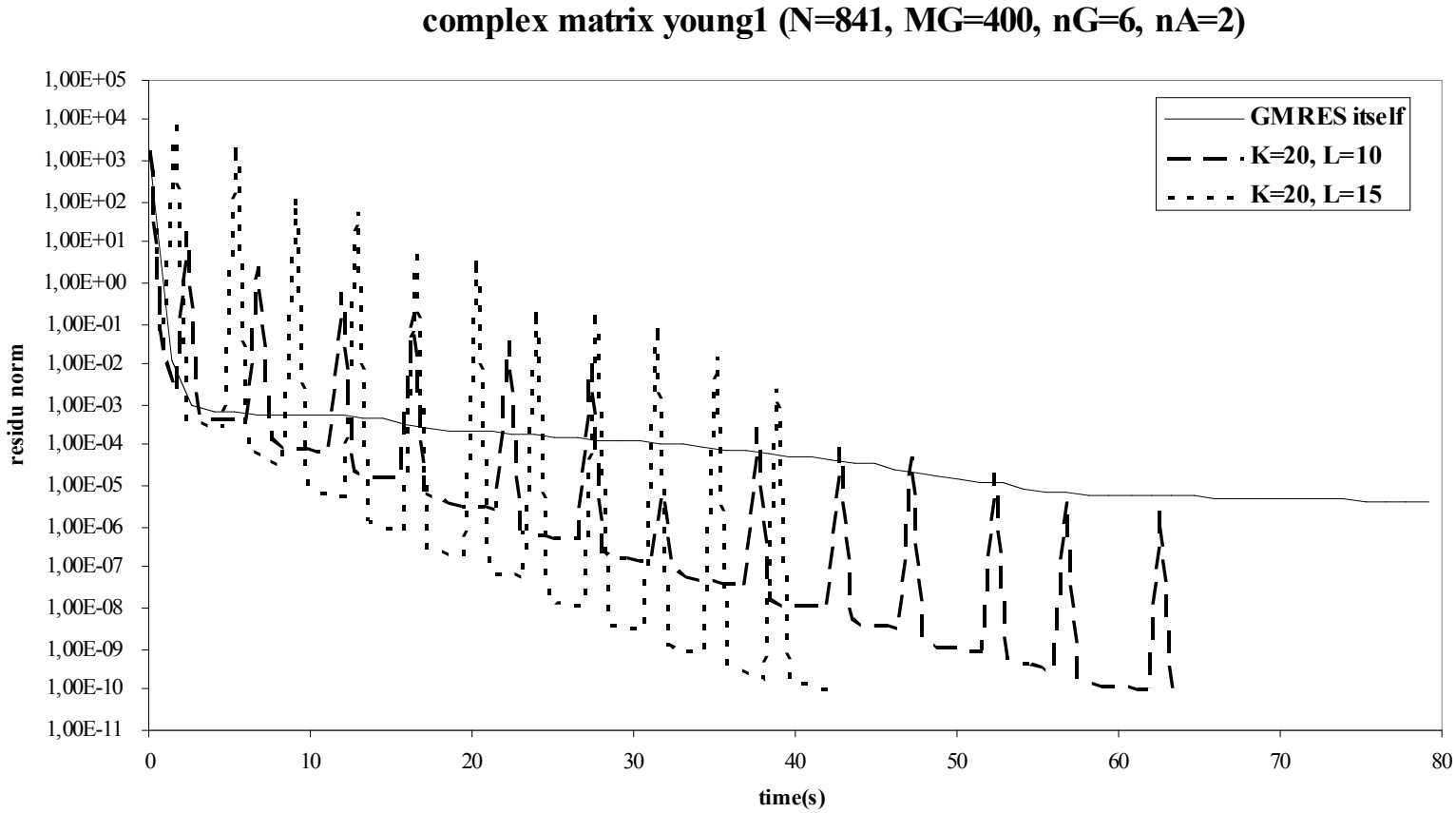
Experimentation on ROME0 (submitted SC17) and First runs on Tianhe 2

Complex-element matrix (compute as a double-size real-element matrix)

More parameters (to tuned!?) :

- Degree k,
- Number of restarts L between two « LS accelerations »

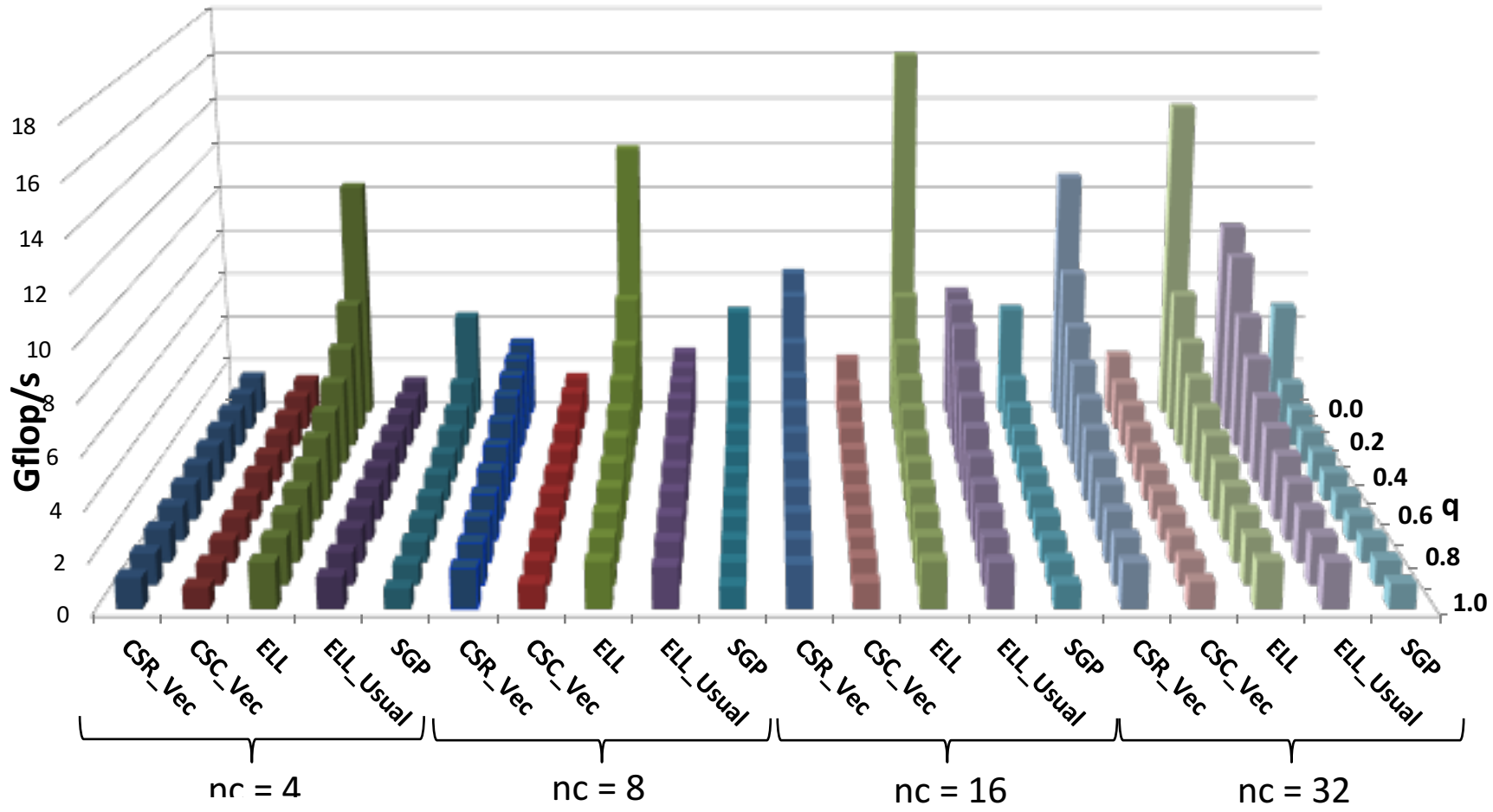
It exists some inclusion properties between the complex matrix and some “converted” in real matrix



Outline

- Introduction
- Krylov subspace auto-tuning algorithm
- GMRES Cache Aware Auto-tuning strategies
- Orthogonalization auto-tuning algorithm
- ERAM restart strategy auto-tuning
- Unite and Conquer MERAM
- **Sparse format auto-tuning**
- Intelligent Krylov method for extreme computing
- Conclusion

Performance of SpMV with C-Diagonal in Double Precision
Auto-tuning of sparse matrix format

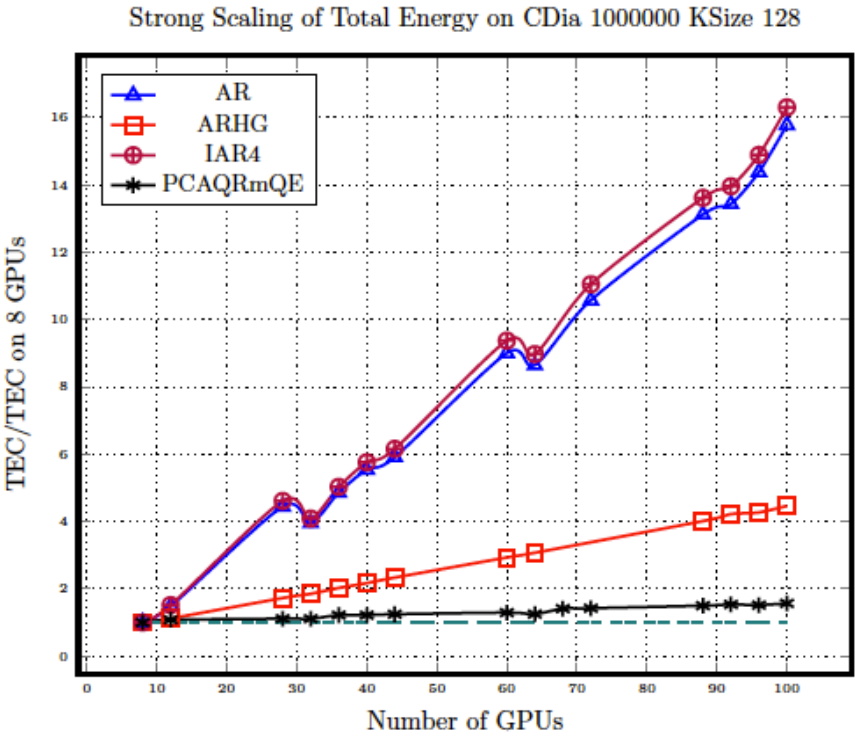


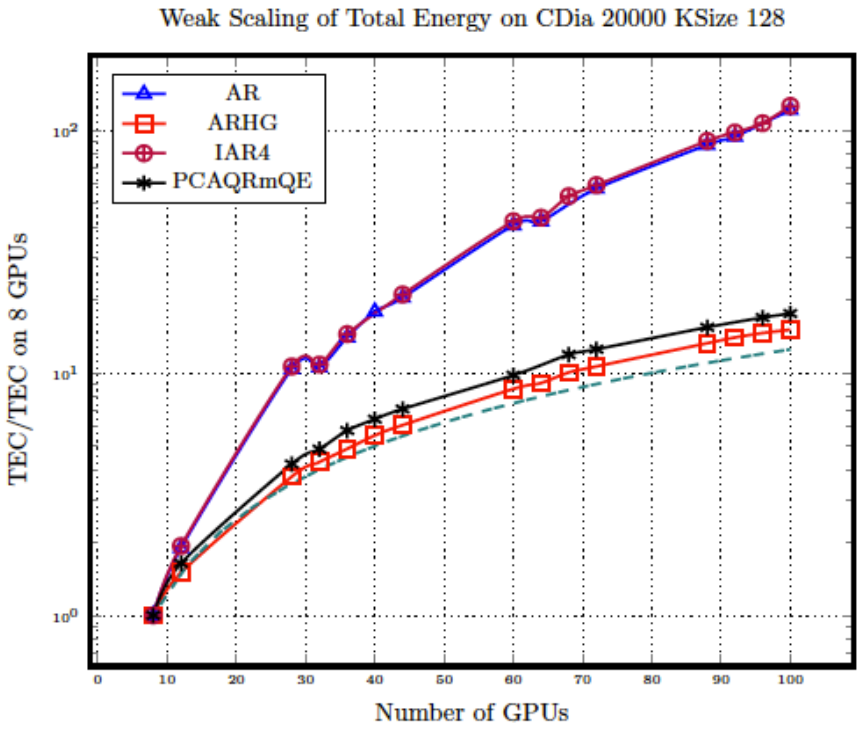
Outline

- Introduction
- Krylov subspace auto-tuning algorithm
- GMRES Cache Aware Auto-tuning strategies
- Orthogonalization auto-tuning algorithm
- ERAM restart strategy auto-tuning
- Unite and Conquer MERAM
- Sparse format auto-tuning
- **Intelligent Krylov method for extreme computing**
- Conclusion

- Each auto-tuning require to manage several parameters
- Multi-auto-tuning algorithms are promising but correlated parameters from different auto-tuning algorithms may be very difficults to optimize at run time
- Hybrid restarted methods will have to asynchronously exchange auto-tuning parameter values to optimize local auto-tuning (ex MERAM(m1,m2,m3,…), GMRES/MERAM-LS)
- Expertise from end-users would be exploited through new high level language and/or framework (yml.prism.uvsq.fr)
- We have to analyse auto-tuned numerical methods to find new crieteria to evaluate the quality of the converge and to decide actions
- *We need to auto-tune the parameters of the different auto-tuning algorithms : smart tuning,……*

Scalability of TEC





- ArnoldiHG and PCAQRmQE have good scalability of TEC
- Communication is important to scalability of energy consumption.

Smart-tuning for extreme computing

- Each parameters of each method may be auto-tuned,
- Each modification of a parameter in one method have to be analyse by the others methods
- We have to analyse the convergence, the efficiency of each iteration, the energy consumed, the accuracy, the stability variation,….

Smart-tuning for extreme computing

- Each parameters of each method may be auto-tuned,
- Each modification of a parameter in one method have to be analyse by the others methods
- We have to analyse the convergence, the efficiency of each iteration, the energy consumed, the accuracy, the stability variation,….

We have to propose high level programming paradigms which allow the end-user and/or the applied mathematician to give some expertise (range of the subspace size, dominant eigenvalue clustering, condition number, ….).

YML is such a programming langage (we have a virtual machine with tutorial and documentation)

We have to use components and high level software strategies to propose tools and language for future numerical analysis methods, who will have to take decisison at runtime : to smart-tune but also tp chose methods and preconditionning, for example

Some elements on YML

- YML[1] 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 OmnRPC for example).

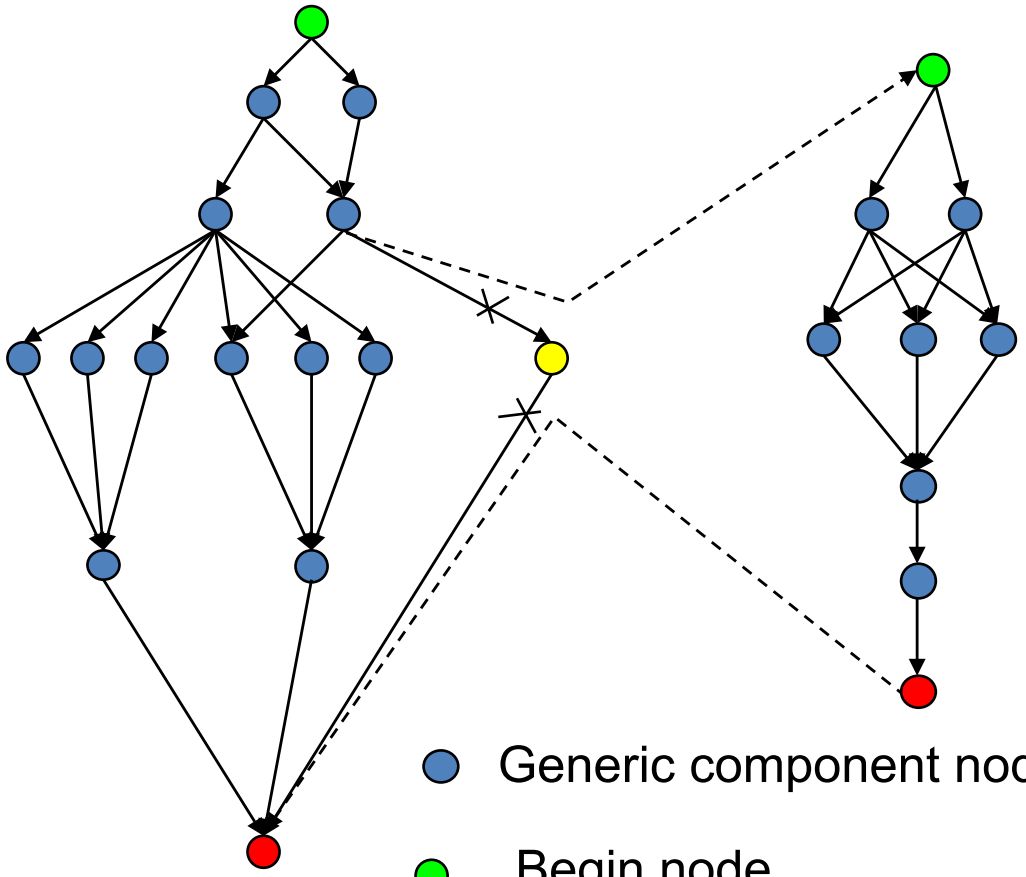
- **Independent from systems and middlewares**
  - The end users can reused their code using another middleware
  - Actually the main system is OmniRPC[3]

- 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*). **LL(1) grammar, easy to parse.**

- Deployed in France, Belgium, Ireland, Japan (T2K, K), China, Tunisia, USA (LBNL, TOTAL-Houston).

Graph (n dimensions) of components/tasksYML




```
par
  compute tache1(..);
  notify(e1);
//
  compute tache2(..); migrate matrix(..);
  notify(e2);
//
  wait(e1 and e2);
  Par (i :=1;n) do
    par
       compute tache3(..);
    notify(e3(i));
    //
    if(I < n)then
       wait(e3(i+1);
       compute tache4(..);
       notify(e4);
     endif;
    //
    compute tache5(..); control robot(..);
    notify(e5); visualize mesh(…) ;
    end par
  end do par
//
  wait(e3(2:n) and e4 and e5);
  compute tache6(..);
  compute tache7(..);
end par
```

End users would be able to add some expertise here

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>
```

<libraries = ScaLAPACk < NAG < PETSC >

Future :

<param   name= "conv" type= " graph_param_float" mode= "inout" />

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>
```

End users may add some expertise here (we'll see 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>
```

A implementation component may be developed using a parallel language and a PGAS one (such as XMP,XMP-StarPU or XcalableACC, XcalableMP)

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
```

```
#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
      enddo
   </graph>
</application>
```

Abstract Component/smart-tuning

```
<?xml version="1.0" ?>
<component type="abstract"    name= »GMRES_Tuned"  description=“restarted
    GMRES method" >
    <Smart_tunings>
    <param name=“m” type=“subspace_size” />
    <param name=“q”, type=“orthogonalization_parameter />
    </Smart_tunings>
    <params>
     <param    name="matrixA"  type="Matrix" mode="in" />
     <param    name="matrixV"    type="Matrix" mode="out" />
     <param    name="size"     type="integer" mode="in" />
    </params>
</component>
```

Future :

```
<param    name= "conv"  type= " graph_param_float" mode= "inout" />
```

Implementation Component and smart-tuning

- Range of parameters to tuned
- Expertise from end-users
- Learning

LL(1) languages may allow end-users to give expertise

```
<?xml version="1.0"?>
<component type="impl" name=   " GMRES_Tuned " abstract=   " GMRES_Tuned"
        description= "Example">
        <range m = {15,100} />
        <Algo_tuning = is method_1 if size larger than 1000, is method_2 otherwise />
        <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>
```

Outline

- Introduction
- Krylov subspace auto-tuning algorithm
- GMRES Cache Aware Auto-tuning strategies
- Orthogonalization auto-tuning algorithm
- ERAM restart strategy auto-tuning
- Unite and Conquer MERAM
- Sparse format auto-tuning
- Intelligent Krylov method for extreme computing
- **Conclusion**

Conclusion

- Linear algebra for very large problems is an important challenge, required to rank data for “big data” applications, to compute a lot of numerical simulations, to decompose a graph into a graph of graphs,……
- The data are often unstructured (large graphs, meshes, sparse matrices,…)
- Smart tunning and hybrid methods are an important propositions to adapted these methods to announced extreme computing machine ; leading to intelligent linear algebra
- High level programming paradigms are required
- We propose both a framework based on multi-level programming and programming paradigm, and new methods and auto-tuning algorithms
- International collaboration are important to be able to evaluate and improve those approaches.
- SPPEXA and MYX allow such basic researches we have to promote