Krylov Subspace and Incomplete Orthogonalization
Auto-tuning Algorithms for GMRES

Serge G. Petiton (LIFL), Christophe Calvin (CEA), Jérome Dubois (CEA),
Laurine Decobert (LIFL), and Rqiya Abouchane (LIFL)
+ Pierre-Yves Aquilanti (LIFL), and France Boillod (LIFL)

MS6 : Towards Smart-Tuning : state-of-art of Auto-Tuning
Technologies and Future Direction
Outline

• Introduction
• GMRES and auto-tuning algorithms
• Incomplete orthogonalization auto-tuning algorithms
• Experimental results as a contribution to illustrate the state-of-art of Auto-Tuning technologies
• Towards smart-Tuning
• Conclusion
Outline

• Introduction
• GMRES and auto-tuning algorithms
• Incomplete orthogonalization auto-tuning algorithms
• Experimental results as a contribution to illustrate the state-of-art of Auto-Tuning technologies
• Towards smart-Tuning
• Conclusion
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:
   \[
   h_{i,j} = (Av_j, v_i), \quad i = 1, 2, \cdots, j,
   \hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i,
   h_{j+1,j} = \|\hat{v}_{j+1}\|, \text{ and }
   v_{j+1} = \hat{v}_{j+1} / h_{j+1,j}.
   \]
3. Form the approximate solution:
   \[
   x_m = x_0 + V_m y_m, \quad \text{where } y_m \text{ minimizes } \|\beta e_1 - \tilde{H}_m y\|, \quad y \in \mathbb{R}^m.
   \]
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.

Memory space:
- Sparse matrix: \text{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, with a minimized computing time.
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 $j = 1, 2, \ldots, m$ do:  
   - $h_{i,j} = (Av_j, v_i)$, $i = 1, 2, \ldots, j$, Incomplete orthogonalization: $\sum i = j - q_j$; $q > 0$.  
   - $\hat{v}_{j+1} = Av_j - \sum_{i=1}^j h_{i,j}v_i$, Then, $q+1$ bands on the Hesseberg matrix.  
   - $h_{j+1,j} = \|\hat{v}_{j+1}\|$, and  
   - $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$.  
3. **Form the approximate solution:**  
   - $x_m = x_0 + V_my_m$, where $y_m$ minimizes $\|\beta e_1 - \tilde{H}_my\|$, $y \in R^m$.  
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.

Memory space:  
- sparse matrix: nnz (i.e. $< Cn$) elements  
- Krylov basis vectors: $n \times m$  
- Hessenberg matrix: $m \times m$

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 orthogonalized with the new one may be auto-tuned at runtime.
Auto-tuned Krylov methods, some correlated goals

- Minimize the global computing time,
- Accelerate the convergence,
- Minimize the number of communications,
- Minimize the number of longer size scalar products,
- Minimize memory space,
- Select the best sparse matrix compressed format,
- Mixed arithmetic.

Criteria which are some of the requirements for Petascale and future Exascale computing.

*The goal of this talk is to illustrate the difficulties to analyze auto-tuning algorithm efficiency and to conclude that we would need “smart” auto-tuning algorithms to create the next generation of High Performance Numerical software.*

*Experiences on cluster of GPU confirm the difficulties to conclude with the today auto-tuning algorithms, to many correlated criteria have to be analyzed and it is quite impossible to have some stable “conclusions”*
Outline

• Introduction
• **GMRES and auto-tuning algorithms**
  • Incomplete orthogonalization auto-tuning algorithms
  • Experimental results as a contribution to illustrate the state-of-art of Auto-Tuning technologies
• Towards smart-Tuning
• Conclusion
Previous works

• 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.
Subspace size tuning principle

We both increase and decrease the subspace size: based on the adaptive subspace size

\[ Cr = \text{norm2} (r_i) / \text{norm2}(r_{i-1}) \]

1 - Keep previous subspace size (cr low): satisfying

2 - Decrease subspace size (cr medium): reduce cycle time, we will have approximately the same convergence rate but with less operations and communications per restart.

3 - Increase restart (cr high): we want to accelerate the convergence
   • Track memory levels: Cache, RAM, Nodes
Auto-Tuning Algorithms

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

\[
Cr = \frac{\text{norm}_2(r_i)}{\text{norm}_2(r_{i-1})}
\]

Easy to implement using libraries
Auto-Tuning Algorithms

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

\[
Cr = \frac{\text{norm2}(r_i)}{\text{norm2}(r_{i-1})}, \text{ Is it really a good criteria?}\\
\]

Easy to implement using libraries

Do we have to change at each restart?
**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 classical increase before intending a more important one
- **m_memory**: array containing subspace size values for important increase
- **Cr-min**: if <, then “Cr low”
- **Cr-max**: if > then “Cr high”
GMRES Autotuning
Lehmer Matrix, size 2568*2568
GMRES(subspace size)

GMRES Autotuning evaluation

Time Execution Comparison (sec)
Lehmer Matrix (size: 2568*2568)
Outline

• Introduction
• GMRES and auto-tuning algorithms
• **Incomplete orthogonalization auto-tuning algorithms**
• Experimental results as a contribution to illustrate the state-of-art of Auto-Tuning technologies
• Towards smart-Tuning
• Conclusion
Incomplete orthogonalization Auto-Tuning

**Complete orthogonalisation**: we orthogonalise with all the previous computed vectors of the basis, i.e. at step $j$, we orthogonalise with $j$ vectors, which generates $j$ scalar product at step $j$. ($j=1,m$)

**Incomplete orthogonalisation**: we orthogonalize with only $\min(j,q)$ previous computed vectors of the basis, i.e. at step $j$, we orthogonalise only with $\min(j,q)$ vectors, $q < m$. DQGMRES : [Saad ’94], DQGMRES : [Wu ’97]

IGMRES : [Brown ’86][Jia ’07]

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

Complete orthogonalization : $j$ scalar product for $j$ fixed ; $1,m$
Incomplete orthogonalization : $q$ scalar product for $j$ fixed, $q < j$

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

Even, if the number of iterations may be a little larger, we minimize a lot of long blocking gobal 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

$q_{\text{min}}$ = minimum number of vector to orthogonalize

$q_{\text{max}}$ = 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 $x-1^{th}$ restart

$H_x$ = relative variation

$= \frac{N_x}{T_x}$

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

• Introduction
• GMRES and auto-tuning algorithms
• Incomplete orthogonalization auto-tuning algorithms
• Experimental results as a contribution to illustrate the state-of-art of Auto-Tuning technologies
• Towards smart-Tuning
• Conclusion
Results: subspace size

- **Serial**: Auto-Tuning VS. No Auto-Tuning

### Diagram

- **GMRES(3,30) Random Strategy**
- **GMRES(3,30) Simple Strategy**
- **GMRES(30) No Strategy**
- **GMRES(3) No Strategy**

**Interest of varying the subspace size at runtime**

- No strategy: m remain fixed for the full resolution
- Random: m is a random value between 3 and 30
- Simple: Decrease m from 30 to 3 cyclically

**Residual Norm vs. Computing Time**

- **Norme résiduelle**: Residual Norm
- **Temps de calcul (s)**: Computing Time

Feb 15, 2012

SIAM PP12
• Our algorithm compared to no auto-tuning

![Graph showing time improvement (%) vs. number of processors.](image)

*Less is better*
RESULTS Incomplete orthogonalization auto-tuning

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

Matrix young4c from matrix market
Solution at $10^{-6}$
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. 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^{-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
Outline

- Introduction
- GMRES and auto-tuning algorithms
- Incomplete orthogonalization auto-tuning algorithms
- Experimental results as a contribution to illustrate the state-of-art of Auto-Tuning technologies
- Towards smart-Tuning
- Conclusion
HYBRID KRYLOV METHODS

Experiments up to 4 ERAMs on CEA Titane computer shown very important accelerations (more that x 4 faster); these Methods are good candidates for Exascale Computing.

We may auto-tune parameters of each co-methods asynchronously.
Asynchronous Iterative Restarted Methods
Collaboration with Guy Bergère and Ye Zhang

The computed Ritz values may be used to analyze the GMRES convergence and decide auto-tuning parameter values.
Performance of SpMV with C-Diagonal in Double Precision on a GPU

Important to auto-tune the compression format first
A(Ax) on Cluster of GPU with a larger matrix

Experiments with C-Diagonal
N=8064000 Q=0.0
Outline

- Introduction
- GMRES and auto-tuning algorithms
- Incomplete orthogonalization auto-tuning algorithms
- Experimental results as a contribution to illustrate the state-of-art of Auto-Tuning technologies
- Towards smart-Tuning
- Conclusion
Each auto-tuning require to manage **several parameters**

It is really very **difficult to analyze results on cluster of GPU** and or a large clusters of multi-cores, to many parameters are concerned, from architecture, software, data structures, auto-tuning algorithm, latencies,.....

The different proposed auto-tuning techniques have interesting behaviours with respect of the matrices and the chose parameters ; but **it is impossible in the today state-of-the-art to really conclude that one is always the best.**

We have to **include more numerical parameter**, such as Ritz elements for example, to be able to analyse the convergence at runtime and take decision about changing parameters,

**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); ex : cluster of eigenvalues,

We have to analyse auto-tuned numerical methods to find **new criteria** to evaluate the quality of the converge and to decide actions

**The auto-tuning strategies have to become smarters and involve perhaps specific computations to allow smart-tuning**