On a hierarchical parallel algebraic domain decomposition linear solver

Emmanuel AGULLO
Luc GIRAUD
Abdou GUERMOCHE
Yan-Fei JING
Stojce NAKOV
Jean ROMAN

Aquitaine-Euskadi Workshop on Applied Mathematics
Bilbao, June 6-7, 2012
Motivations

Goal: solving $Ax = b$, where $A$ is large and sparse

Usual trades off

**Direct**
- Robust/prescribed accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems
- Limited weak scalability

**Iterative**
- Problem dependent efficiency / monitored accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
- Possible high weak scalability

Full direct
- Direct on nearby matrix
- Block diagonal prec.
- Partial factorization
- Schur complement
- Block incomplete factorization
- Incomplete factorization
- Stationary iteration

Full iterative
Goal: solving $Ax = b$, where $A$ is large and sparse

**Usual trades off**

**Direct**
- Robust/prescribed accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems
- Limited weak scalability

**Iterative**
- Problem dependent efficiency / monitored accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
- Possible high weak scalability
Motivations

Goal: solving $Ax = b$, where $A$ is large and sparse

Usual trades off

<table>
<thead>
<tr>
<th>Direct</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robust/prescribed accurate for general problems</td>
<td>Problem dependent efficiency / monitored accuracy</td>
</tr>
<tr>
<td>BLAS-3 based implementations</td>
<td>Sparse computational kernels</td>
</tr>
<tr>
<td>Memory/CPU prohibitive for large 3D problems</td>
<td>Less memory requirements and possibly faster</td>
</tr>
<tr>
<td>Limited weak scalability</td>
<td>Possible high weak scalability</td>
</tr>
</tbody>
</table>
Motivations

Goal: solving $Ax = b$, where $A$ is large and sparse

Usual trades off

**Direct**
- Robust/prescribed accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems
- Limited weak scalability

**Iterative**
- Problem dependent efficiency / monitored accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
- Possible high weak scalability
Motivations

Goal: solving $Ax = b$, where $A$ is large and sparse

Usual trades off

**Direct**
- Robust/prescribed accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems
- Limited weak scalability

**Iterative**
- Problem dependent efficiency / monitored accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
- Possible high weak scalability
Motivations

Goal: solving $Ax = b$, where $A$ is large and sparse

Usual trades off

<table>
<thead>
<tr>
<th>Direct</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robust/prescribed accurate for general problems</td>
<td>Problem dependent efficiency / monitored accuracy</td>
</tr>
<tr>
<td>BLAS-3 based implementations</td>
<td>Sparse computational kernels</td>
</tr>
<tr>
<td>Memory/CPU prohibitive for large 3D problems</td>
<td>Less memory requirements and possibly faster</td>
</tr>
<tr>
<td>Limited weak scalability</td>
<td>Possible high weak scalability</td>
</tr>
</tbody>
</table>
Develop robust scalable parallel hybrid direct/iterative linear solvers

- Exploit the efficiency and robustness of the sparse direct solvers
- Develop robust parallel preconditioners for iterative solvers
- Take advantage of the natural scalable parallel implementation of iterative solvers

Domain Decomposition (DD)

- Natural approach for PDE’s
- Extend to general sparse matrices
- Partition the problem into subdomains, subgraphs
- Use a direct solver on the subdomains
- Robust preconditioned iterative solver
Overlapping Domain Decomposition

Classical Additive Schwarz preconditioners [Schwarz, 1870]

- **Goal:** solve linear system $Ax = b$
- Use iterative method
- Apply the preconditioner at each step
- The convergence rate deteriorates as the number of subdomains increases

$$A = \begin{pmatrix} A_{1,1} & A_{1,\delta} \\ A_{\delta,1} & A_{\delta,\delta} & A_{\delta,2} \\ A_{2,\delta} & A_{2,2} \end{pmatrix} \Rightarrow M_{\delta AS} = \begin{pmatrix} A_{1,1} & A_{1,\delta} & -1 \\ A_{\delta,1} & A_{\delta,\delta} & A_{\delta,2} \\ A_{2,\delta} & A_{2,2} \end{pmatrix}^{-1}$$

Classical Additive Schwarz preconditioners, $N$ subdomains case

$$M_{\delta AS} = \sum_{i=1}^{N} \left( R_{i}^{\delta} \right)^{T} \left( A_{i}^{\delta} \right)^{-1} R_{i}^{\delta}$$
**Goal:** solve linear system $Ax = b$

Apply partially block Gaussian elimination

Solve the reduced system $Sx_\Gamma = f$

Then solve $A_i x_i = b_i - A_{i,\Gamma}x_\Gamma$

---

Solve $Ax = b \implies$ solve the reduced system $Sx_\Gamma = f \implies$ then solve $A_i x_i = b_i - A_{i,\Gamma}x_\Gamma$

where $S = A_{\Gamma,\Gamma} - \sum_{i=1}^{2} A_{\Gamma,i}A_{i,\Gamma}^{-1}A_{i,\Gamma}$,

and $f = b_\Gamma - \sum_{i=1}^{2} A_{\Gamma,i}A_{i,\Gamma}^{-1}b_i$. 

---

**Non-overlapping Domain Decomposition**

**Schur complement reduced system**
Nonoverlapping Domain Decomposition

**Schur complement reduced system**

\[ \Gamma = k \cup \ell \cup m \cup n \]

\[ \Omega = \Omega_t \cup \Omega_{t+1} \cup \Omega_{t+2} \]

\[ S_{\ell\ell} = S_{\ell\ell}^{(i)} + S_{\ell\ell}^{(i+1)} \implies S_{\ell\ell} = \sum_{i \in \text{adj}} S_{\ell\ell}^{(i)} \]

**Distributed Schur complement**

\[ \begin{pmatrix} S_{kk} & S_{k\ell} \\ S_{\ell k} & S_{\ell\ell}^{(i)} \end{pmatrix} \]

\[ \begin{pmatrix} S_{\ell\ell}^{(i+1)} & S_{\ell m} \\ S_{ml} & S_{mm} \end{pmatrix} \]

\[ \begin{pmatrix} S_{mm} & S_{mn} \\ S_{nm} & S_{nn} \end{pmatrix} \]
Non-overlapping Domain Decomposition

Algebraic Additive Schwarz preconditioner [L.Carvalho, L.G., G.Meaurant - 01]

\[ S = \sum_{i=1}^{N} R_{\Gamma_i}^T S^{(i)} R_{\Gamma_i} \]

\[ S = \begin{pmatrix} \cdots & S_{kk} & S_{k\ell} & S_{\ell m} & S_{\ell\ell} & \cdots \end{pmatrix} \quad \Rightarrow \quad M = \begin{pmatrix} \cdots & S_{kk} & S_{k\ell} & -1 \cr S_{\ell k} & S_{\ell\ell} & -1 \cr S_{\ell m} & S_{mm} & -1 \cr S_{\ell\ell} & S_{nn} & S_{nn} \end{pmatrix} \]

\[ M = \sum_{i=1}^{N} R_{\Gamma_i}^T (\bar{S}^{(i)})^{-1} R_{\Gamma_i} \]

where \( \bar{S}^{(i)} \) is obtained from \( S^{(i)} \)

\[ S^{(i)} = \begin{pmatrix} S_{kk}^{(i)} & S_{k\ell}^{(i)} \\ S_{\ell k}^{(i)} & S_{\ell\ell}^{(i)} \end{pmatrix} \quad \Rightarrow \quad \bar{S}^{(i)} = \begin{pmatrix} S_{kk} & S_{k\ell} \\ S_{\ell k} & S_{\ell\ell} \end{pmatrix} \]

Similarity with Neumann-Neumann preconditioner [J.F. Bourgat, R. Glowinski, P. Le Tallec and M. Vidrascu - 89] [Y.H. de Roeck, P. Le Tallec and M. Vidrascu - 91]

\[ \sum_{\iota \in \text{adj}} S_{\ell\ell}^{(i)} \]
Parallel preconditioning features

\[ S(i) = A_{\Gamma_i \Gamma_i} - A_{\Gamma_i \ell_i} A_{\ell_i \Gamma_i}^{-1} A_{\ell_i \Gamma_i} \]

\[ M_{AS} = \sum_{i=1}^{\#\text{domains}} R_i^T (\bar{S}(i))^{-1} R_i \]

\[ \bar{S}(i) = \begin{pmatrix} S_{mm} & S_{mg} & S_{mk} & S_{ml} \\ S_{gm} & S_{gg} & S_{gk} & S_{gl} \\ S_{km} & S_{kg} & S_{kk} & S_{kl} \\ S_{lm} & S_{lg} & S_{lk} & S_{ll} \end{pmatrix} \]

Assembled local Schur complement

\[ S_{mm} = \sum_{j \in \text{adj}(m)} S_{mm}^{(j)} \leftarrow \text{dense matrix} \]
Parallel algorithm

- Each subdomain $\mathcal{A}^{(i)}$ is handled by one processor

$$
\mathcal{A}^{(i)} \equiv \begin{pmatrix}
\mathcal{A}_{\mathcal{I}_i\mathcal{I}_i} & \mathcal{A}_{\mathcal{I}_i\mathcal{Gamma}_i} \\
\mathcal{A}_{\mathcal{Gamma}_i\mathcal{I}_i} & \mathcal{A}_{\mathcal{Gamma}_i\mathcal{Gamma}_i}
\end{pmatrix}
$$

- Concurrent partial factorizations are performed on each processor to form the so-called “local Schur complement”

$$
S^{(i)} = \mathcal{A}_{\mathcal{Gamma}_i\mathcal{Gamma}_i} - \mathcal{A}_{\mathcal{Gamma}_i\mathcal{I}_i} \mathcal{A}_{\mathcal{I}_i\mathcal{I}_i}^{-1} \mathcal{A}_{\mathcal{I}_i\mathcal{Gamma}_i}
$$

- The reduced system $S\mathbf{x}_{\Gamma} = f$ is solved using a distributed Krylov solver
  - One matrix vector product per iteration each processor computes $S^{(i)}(\mathbf{x}^{(i)}_\Gamma)^k = (\mathbf{y}^{(i)}_\Gamma)^k$
  - One local preconditioner apply $(\mathcal{M}^{(i)})(\mathbf{z}^{(i)}_\Gamma)^k = (\mathbf{r}^{(i)}_\Gamma)^k$
  - Local neighbor-neighbor communication per iteration
  - Global reduction (dot products)

- Compute simultaneously the solution for the interior unknowns

$$
\mathcal{A}_{\mathcal{I}_i\mathcal{I}_i} \mathbf{x}_{\mathcal{I}_i} = \mathbf{b}_{\mathcal{I}_i} - \mathcal{A}_{\mathcal{I}_i\mathcal{Gamma}_i} \mathbf{x}_{\Gamma}
$$
## Main characteristics in 2D
- The ratio interface/interior is small
- Does not require large amount of memory to store the preconditioner
- Computation/application of the preconditioner are fast
- They consist in a call to LAPACK/BLAS-2 kernels

## Main characteristics in 3D
- The ratio interface/interior is large
- The storage of the preconditioner might not be affordable
- The construction of the preconditioner can be computationally expensive
- Need cheaper Algebraic Additive Schwarz form of the preconditioner
What tricks exist to construct cheaper preconditioners

### Sparsification strategy through dropping

\[
\hat{S}_{k\ell} = \begin{cases} 
\bar{s}_{k\ell} & \text{if } \bar{s}_{k\ell} \geq \xi(|\bar{s}_{kk}| + |\bar{s}_{\ell\ell}|) \\
0 & \text{else}
\end{cases}
\]

### Approximation through ILU - [INRIA PhyLeas - A. Haidar, L.G., Y.Saad - 10]

\[
pILU(A^{(i)}) \equiv pILU \begin{pmatrix} A_{ii} & A_{i\Gamma_i} \\ A_{\Gamma_i i} & A_{\Gamma_i \Gamma_i} \end{pmatrix} \equiv \begin{pmatrix} \bar{L}_i & 0 \\ A_{\Gamma_i} \bar{U}_i^{-1} & I \end{pmatrix} \begin{pmatrix} \tilde{U}_i & \tilde{I}_i^{-1} A_{i\Gamma} \\ 0 & \tilde{S}^{(i)} \end{pmatrix}
\]

### Mixed arithmetic strategy

- Compute and store the preconditioner in 32-bit precision arithmetic Is accurate enough?
- Limitation when the conditioning exceeds the accuracy of the 32-bit computations Fix it!
- Idea: Exploit 32-bit operation whenever possible and resort to 64-bit at critical stages
- Remarks: the backward stability result of GMRES indicates that it is hopeless to expect convergence at a backward error level smaller than the 32-bit accuracy [C.Paige, M.Rozložník, Z.Strakoš - 06]
- Idea: To overcome this limitation we use FGMRES [Y.Saad - 93]
Academic model problems

Problem patterns

Circular flow velocity Problem

Diffusion equation ($\epsilon = 1$ and $v = 0$) and convection-diffusion equation

$$\left\{ \begin{array}{ll}
-\epsilon \text{div}(K \cdot \nabla u) + v \cdot \nabla u &= f \quad \text{in} \quad \Omega, \\
u &= 0 \quad \text{on} \quad \partial \Omega.
\end{array} \right.$$  

- Heterogeneous problems
- Anisotropic-heterogeneous problems
- Convection dominated term
Numerical behaviour of sparse preconditioners

Convergence history of PCG

Time history of PCG

- **3D** heterogeneous diffusion problem with 43 Mdof mapped on 1000 processors
- For ($\xi \ll \ll $) the convergence is marginally affected while the memory saving is significant 15%
- For ($\xi \gg \gg $) a lot of resources are saved but the convergence becomes very poor 1%
- Even though they require more iterations, the sparsified variants converge faster as the time per iteration is smaller and the setup of the preconditioner is cheaper.
3D heterogeneous diffusion problem with 43 M dof mapped on 1000 processors

64-bit and mixed computation both attained an accuracy at the level of 64-bit machine precision

The number of iterations slightly increases

The mixed approach is the fastest, down to an accuracy that is problem dependent
Scaled scalability on massively parallel platforms

**Numerical scalability**

- 3D heterogeneous diffusion problem
- Dense 64-bit calculation
- Dense mixed calculation
- Sparse with $\xi=10^{-4}$

<table>
<thead>
<tr>
<th># proc</th>
<th># iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>216 343 512 729 1000 1331 1728</td>
</tr>
<tr>
<td>Time (sec)</td>
<td></td>
</tr>
<tr>
<td>0 20 40 60 80 100 120 140 160 180</td>
<td># proc</td>
</tr>
</tbody>
</table>

- The solved problem size varies from 2.7 up to 74 MdoF
- Control the growth in the # of iterations by introducing a coarse space correction
- The computing time increases slightly when increasing # sub-domains
- Although the preconditioners do not scale perfectly, the parallel time scalability is acceptable
- The trend is similar for all variants of the preconditioners using CG Krylov solver

**Parallel performance**

- 3D heterogeneous diffusion problem
- Dense 64-bit calculation
- Dense mixed calculation
- Sparse with $\xi=10^{-4}$

<table>
<thead>
<tr>
<th># proc</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>216 343 512 729 1000 1331 1728</td>
</tr>
<tr>
<td>0 20 40 60 80 100 120 140 160 180</td>
<td></td>
</tr>
</tbody>
</table>

L. GIRAUD  A Hierarchical Parallel Hybrid Solver
Experiments on large 3D real life applications

Application areas

- Structural mechanics: real SPD and symmetric indefinite linear systems.
- Electromagnetism: complex symmetric non-Hermitian.
- Seismic: complex symmetric non-Hermitian.
3D SEG/EAGE Overthrust model

- Solve the Helmholtz equation
- $20 \times 20 \times 4.65 \text{ km}^3$
- 4 grid points per minimum wavelength
- PML (Perfectly-Matched Layer) [J.P. Berenger - 94]
- 5.6 M dof at 7 Hz

Main characteristics

- Frequency-domain full-waveform tomography [F. Sourbier, S. Operto, J. Virieux - 08]
- The inversion of a few frequencies is enough to build velocity models
- Multisource frequency-domain wave modeling requires the solution of multiple RHS
- Traditional method of choice for solving these systems relies on sparse direct solvers
- To overcome this limitation, the goal is to develop efficient hybrid methods for large 3D
### Performance on 3D unsymmetric complex systems

#### 3D SEG/EAGE Overthrust model

<table>
<thead>
<tr>
<th># of subdomains</th>
<th>Memory size</th>
<th>Memory interface</th>
<th>Memory All (GB)</th>
<th>Initial-</th>
<th>Precond-</th>
<th># of iterations</th>
<th>Time per RHS</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>54 × 54 × 31</td>
<td>11056</td>
<td>191.6</td>
<td>614</td>
<td>497.1</td>
<td>81</td>
<td>67.8</td>
<td>1178.9</td>
</tr>
<tr>
<td>72</td>
<td>45 × 45 × 31</td>
<td>8833</td>
<td>179.3</td>
<td>334</td>
<td>273.5</td>
<td>103</td>
<td>73.9</td>
<td>681.4</td>
</tr>
<tr>
<td>96</td>
<td>45 × 33 × 31</td>
<td>7405</td>
<td>167.8</td>
<td>184</td>
<td>153.8</td>
<td>119</td>
<td>61.1</td>
<td>398.9</td>
</tr>
<tr>
<td>98</td>
<td>38 × 38 × 31</td>
<td>7216</td>
<td>169.7</td>
<td>189</td>
<td>141.5</td>
<td>148</td>
<td>66.7</td>
<td>397.2</td>
</tr>
<tr>
<td>192</td>
<td>33 × 33 × 21</td>
<td>5578</td>
<td>147.4</td>
<td>90</td>
<td>78.2</td>
<td>235</td>
<td>85.8</td>
<td>254.0</td>
</tr>
</tbody>
</table>

- Problem of 5.6 Mdof at 7 Hz
- Increasing the number of subdomains reduces the memory requirement
- Increasing the number of subdomains decreases the computational cost
- Increasing the number of subdomains increases the number of iterations
Exploiting $2$-levels of parallelism - motivations

"The numerical improvement"
- Classical parallel implementations ($1$-level) of DD assign one subdomain per processor.
- Parallelizing means increasing the number of subdomains.
- Increasing the number of subdomains often leads to increasing the number of iterations.
- To avoid this, one can instead of increasing the number of subdomains, keep it small while handling each subdomain by more than one processor introducing $2$-levels of parallelism.

"The parallel performance improvement"
- Large $3D$ systems often require a huge amount of data storage.
- On SMP node: classical $1$-level parallel can only use a subset of the available processors.
- Thus some processors are "wasted", as they are "idle" during the computation.
- The "idle" processors might contribute to the computation and the simulation runs closer to the peak of per-node performance by using $2$-levels of parallelism.
### 2-levels of parallelism on 3D Overthrust SEG/EAGE

#### Frequency equal to 7 Hz

<table>
<thead>
<tr>
<th>Available processors</th>
<th>Algo</th>
<th># subdomains</th>
<th>Processors/subdomain</th>
<th># iter</th>
<th>Iterative loop</th>
<th>Time per RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>≈ 200 processors</td>
<td>1-level parallel</td>
<td>192</td>
<td>1</td>
<td>235</td>
<td>79.0</td>
<td>85.8</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>96</td>
<td>2</td>
<td>119</td>
<td>38.2</td>
<td>45.1</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>50</td>
<td>4</td>
<td>81</td>
<td>28.1</td>
<td>35.5</td>
</tr>
<tr>
<td>≈ 100 processors</td>
<td>1-level parallel</td>
<td>96</td>
<td>1</td>
<td>119</td>
<td>57.0</td>
<td>61.1</td>
</tr>
<tr>
<td></td>
<td>1-level parallel</td>
<td>98</td>
<td>1</td>
<td>148</td>
<td>66.7</td>
<td>66.7</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>50</td>
<td>2</td>
<td>81</td>
<td>39.1</td>
<td>45.1</td>
</tr>
</tbody>
</table>

- Reduce the number of subdomains $\Rightarrow$ reduce the number of iterations
- Though the subdomain size increases, the time of the iterative loop decreases as:
- The speedup factor of one RHS simulation varies from 1.3 to 2.5
- Very attractive approach for multiple right-hand sides simulations
Parallel performance benefits

2-levels of parallelism on 3D Overthrust SEG/EAGE

<table>
<thead>
<tr>
<th># subdomains</th>
<th>Algo</th>
<th>proc/subdom or “working”</th>
<th>Precond setup time</th>
<th># iter</th>
<th>iterative loop time</th>
<th>Time per RHS</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1-level</td>
<td>1</td>
<td>497.1</td>
<td>81</td>
<td>64.4</td>
<td>67.8</td>
<td>1178.9</td>
</tr>
<tr>
<td></td>
<td>2-level</td>
<td>2</td>
<td>262.4</td>
<td></td>
<td>39.1</td>
<td>45.1</td>
<td>854.5</td>
</tr>
<tr>
<td></td>
<td>2-level</td>
<td>4</td>
<td>135.3</td>
<td></td>
<td>28.1</td>
<td>35.5</td>
<td>419.8</td>
</tr>
<tr>
<td>81</td>
<td>1-level</td>
<td>1</td>
<td>256.3</td>
<td>109</td>
<td>73.6</td>
<td>77.4</td>
<td>557.7</td>
</tr>
<tr>
<td></td>
<td>2-level</td>
<td>2</td>
<td>169.2</td>
<td></td>
<td>53.7</td>
<td>57.2</td>
<td>431.4</td>
</tr>
<tr>
<td>96</td>
<td>1-level</td>
<td>1</td>
<td>153.8</td>
<td>119</td>
<td>57.0</td>
<td>61.1</td>
<td>398.9</td>
</tr>
<tr>
<td></td>
<td>2-level</td>
<td>2</td>
<td>81.2</td>
<td></td>
<td>38.2</td>
<td>45.1</td>
<td>299.3</td>
</tr>
</tbody>
</table>

- The preconditioner setup time benefits vary from 1.5 to 3.6
- One RHS simulation time is improved by a factor of 1.3 to 1.7
We want to solve

\[ AX = B, \]

where, \( A \in \mathbb{C}^{N \times N} \), \( B \in \mathbb{C}^{N \times p} \) full rank, and \( X \in \mathbb{C}^{N \times p} \).

For large sparse matrices, it might become prohibitive for sparse direct solvers because of high memory requirement and operation counts.

Suiting candidates are Block Krylov approaches:

- The Krylov subspaces associated with each right-hand side are shared to enlarge the search subspace.
- Matrix-vector products are simultaneously computed.
Background on Krylov subspace methods

\[ Ax = b. \]

- The nested Krylov subspace of dimension \( m \) generated by \( A \) from the initial residual vector \( r_0 = b - Ax_0 \) is of the form

\[ \mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}. \]

- Arnoldi’s procedure is used to build an orthonormal basis of the Krylov subspace.
  - Arnoldi equality:
    \[ AV_m = V_m H_m + [0, \omega_m] \]
  - Block Arnoldi recurrence formula:
    \[ A\gamma_m = \gamma_m \mathcal{H}_m + [0, W_m] \]
  - Block Arnoldi with inexact breakdown [Robbé, Sadkane - 06]
    \[ A\gamma_m = \gamma_m \mathcal{H}_m + [L_{m-1}, W_m] \]
Motivation for this work

- GMRES with deflated restarting–GMRES-DR

- Block variants
Example 1

- IB-BGMRES
- BGMRES-DR
- IB-BGMRES-DR

$\eta_b$ (min, max)

mves
### Table: Comparison results of regular GMRES, GMRES-DR, IB-BGMRES, BGMRES-DR and IB-BGMRES-DR in terms of matrix-vector products.

<table>
<thead>
<tr>
<th>Example</th>
<th>GMRES</th>
<th>GMRES-DR</th>
<th>IB-BGMRES</th>
<th>BGMRES-DR</th>
<th>IB-BGMRES-DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2536</td>
<td>1077</td>
<td>1339</td>
<td>892</td>
<td>587</td>
</tr>
<tr>
<td>2</td>
<td>1069</td>
<td>856</td>
<td>787</td>
<td>667</td>
<td>535</td>
</tr>
<tr>
<td>3</td>
<td>378</td>
<td>378</td>
<td>372</td>
<td>341</td>
<td>334</td>
</tr>
<tr>
<td>4</td>
<td>412</td>
<td>412</td>
<td>444</td>
<td>447</td>
<td>439</td>
</tr>
<tr>
<td>5</td>
<td>845</td>
<td>694</td>
<td>617</td>
<td>474</td>
<td>386</td>
</tr>
<tr>
<td>6</td>
<td>464</td>
<td>464</td>
<td>357</td>
<td>294</td>
<td>248</td>
</tr>
<tr>
<td>7</td>
<td>3154</td>
<td>2003</td>
<td>3291</td>
<td>3090</td>
<td>2104</td>
</tr>
<tr>
<td>8</td>
<td>10643</td>
<td>3110</td>
<td>10000*</td>
<td>4426</td>
<td>2197</td>
</tr>
</tbody>
</table>

Block versus “regular” GMRES
Ongoing work

- \(\beta\)-release of the hierarchical parallel algebraic solver (CeCILL licence)
  https://wiki.bordeaux.inria.fr/maphys/doku.php
  [Parallel hierarchical hybrid linear solvers for emerging computing platforms E. Agullo, L. Giraud, A. Guermouche, J. Roman, CRAS, 2011]

- Design and implementation on heterogeneous manycore plateform
  S. Nakov PhD (funded by Total/DIP), MORSE Associated team (ICL Tennessee, Kaust, UC Denver)

- Exploit H-matrix techniques to reduce computational resource consumption
  FastLA Associated team (LBNL, Stanford)

- Move from Matlab prototype to real implementation for IB-BGMRES-DR
Muchas gracias pour votre attention

Questions ?