Algebraic preconditioners for parallel hybrid solvers

Maison de la Simulation, Saclay Nov. 5, 2013

L. Giraud
Outline

Introduction

Quick overview of a sparse direct solver

Hybrid Linear Solvers

Numerical experiments on large 3D problems

Next step toward exascale

Acknowledgments
Forewords

HiePACS objectives: Contribute to the design of effective tools for frontier simulations arising from challenging research and industrial multi-scale applications towards exascale computing.
Sparse linear solvers

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

**Usual trades off**

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

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

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

Usual trades off

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

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

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

Usual trades off

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

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

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

**Usual trades off**

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

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

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

Usual trades off

Direct

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

Iterative

- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
- Possible high weak scalability
Sparse linear solvers

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

Usual trades off

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

**Iterative**
- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
- Possible high weak scalability
Quick overview of a sparse direct solver
Parallel sparse direct solver - PaStiX Main Features

Numerical features

- $LL^T$, $LDL^T$, $LU$ factorization with supernodal implementation
- Static pivoting + Refinement: CG/GMRES
- 1D/2D block distribution + Full BLAS3
- Simple/Double precision + Float/Complex operations

Implementation features

- MPI/Threads implementation (SMP/Cluster/Multicore/NUMA)
- Dynamic scheduling inside SMP nodes (static mapping)
- Support external ordering library (PT-Scotch/METIS)

Additional information

- Multiple RHS (direct factorization)
- Incomplete factorization with ILU(k) preconditionner
- Schur complement computation
- Out-of Core implementation (in SMP mode only)
Outline

Hybrid Linear Solvers
Hybrid Linear Solvers

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
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

\[
\mathcal{A} = \begin{pmatrix} A_{1,1} & A_{1,\delta} \\ A_{\delta,1} & A_{\delta,\delta} & A_{\delta,2} \\ A_{\delta,2} & A_{2,2} \end{pmatrix} \implies \mathcal{M}^{\delta}_{AS} = \begin{pmatrix} A_{1,1} & A_{1,\delta}^{-1} \\ A_{\delta,1} & A_{\delta,\delta}^{-1} & A_{\delta,2}^{-1} \\ A_{\delta,2} & A_{2,2}^{-1} \end{pmatrix}
\]

Classical Additive Schwarz preconditioners $N$ subdomains case

\[
\mathcal{M}^{\delta}_{AS} = \sum_{i=1}^{N} (R_i^\delta)^T (A_i^\delta)^{-1} R_i^\delta
\]
Non-overlapping Domain Decomposition

Goal: solve linear system $Ax = b$

Apply partially Gaussian elimination

Solve the reduced system $Sx_\Gamma = f$

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

\[
\begin{pmatrix}
A_{1,1} & 0 & A_{1,\Gamma} \\
0 & A_{2,2} & A_{2,\Gamma} \\
0 & 0 & S
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_\Gamma
\end{pmatrix} =
\begin{pmatrix}
b_1 \\
b_2 \\
b_\Gamma - \sum_{i=1}^{2} A_{\Gamma,i} A_{i,i}^{-1} b_i
\end{pmatrix}
\]

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

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

In an assembled form:

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

Algebraic Additive Schwarz preconditioner

[ L.Carvalho, L.G., G.Meunant - 01]

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

\[ S = \begin{pmatrix}
  \ddots \\
  S_{kk} & S_{k\ell} \\
  S_{\ell k} & S_{\ell \ell} & S_{\ell m} \\
  S_{m\ell} & S_{mm} & S_{mn} \\
  S_{nm} & S_{nn}
\end{pmatrix} \Rightarrow \mathcal{M} = \begin{pmatrix}
  \ddots \\
  S_{kk} & S_{k\ell} & -1 \\
  S_{\ell k} & S_{\ell \ell} & S_{\ell m} \\
  S_{m\ell} & S_{mm} & S_{mn} \\
  S_{nm} & S_{nn}
\end{pmatrix}
\]

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

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

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

Similarity with Neumann-Neumann preconditioner

[ J.F Bourgat, R. Glowinski, P. Le Tallec and M. Vidrascu - 89] [Y.H. de Roek, P. Le Tallec and M. Vidrascu - 91]
Parallel preconditioning features

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

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

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

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

Assembled local Schur complement

\[ S_{mm} = \sum_{j \in \text{adj}(m)} S_{mm}^{(j)} \]
Parallel implementation

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

$$A^{(i)} \equiv \begin{pmatrix} A_{I_i I_i} & A_{I_i \Gamma_i} \\ A_{\Gamma_i I_i} & A_{\Gamma_i \Gamma_i} \end{pmatrix}$$

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

$$S^{(i)} = A_{\Gamma \Gamma}^{(i)} - A_{\Gamma_i I_i} A_{I_i I_i}^{-1} A_{I_i \Gamma_i}$$

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

- Compute simultaneously the solution for the interior unknowns

$$A_{I_i I_i} x_{I_i} = b_{I_i} - A_{I_i \Gamma_i} x_{\Gamma_i}$$
Algebraic Additive Schwarz preconditioner

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

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

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

\[
p\text{ILU}(A(i)) \equiv p\text{ILU}(A) = \begin{pmatrix} A_{ii} & A_{i\Gamma_i} \\ A_{\Gamma_i i} & A_{\Gamma_i \Gamma_i(i)} \end{pmatrix} \equiv \begin{pmatrix} \tilde{L}_i & 0 \\ A_{\Gamma_i} \tilde{U}_i^{-1} & I \end{pmatrix} \begin{pmatrix} \tilde{U}_i & \tilde{L}_i^{-1} A_{i\Gamma_i} \\ 0 & \tilde{S}_{(i)} \end{pmatrix}
\]

- 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 ressort 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]
Outline

Numerical experiments on large 3D problems
Numerical experiments on large 3D problems

**Academic model problems**

- Problem patterns

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

\[
\begin{align*}
-\epsilon \text{div}(K.\nabla u) + v.\nabla u &= f \quad \text{in} \quad \Omega, \\
    u &= 0 \quad \text{on} \quad \partial\Omega.
\end{align*}
\]

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

3D heterogeneous diffusion problem with 43 Mdof mapped on 1000 processors

- For $\xi \ll$ the convergence is marginally affected while the memory saving is significant 15%
- For $\xi \gg$ 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.
Numerical experiments on large 3D problems

Numerical behaviour of mixed preconditioners

Convergence history of PCG

Time history of PCG

- 3D heterogeneous diffusion problem with 43 MdoF 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
Numerical experiments on large 3D problems

Scaled scalability on massively parallel platforms

Numerical scalability

Parallel performance

- The solved problem size varies from 2.7 up to 74 M dof
- Control the grow 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
Numerical alternative: numerical scalability in 3D

Domain based coarse space: \( M = M_{AS} + R_O^T A_O^{-1} R_0 \) where \( A_0 = R_0 S R_0^T \)

- “As many” dof in the coarse space as sub-domains

[Carvalho, Giraud, Le Tallec, 01]

- Partition of unity: \( R_0^T \) simplest constant interpolation

**2D Heterogenous diffusion**

**3D Heterogenous diffusion**
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.
Numerical experiments on large 3D problems

Indefinite systems in structural mechanics  S.Pralet, SAMTECH

Fuselage of 6.5 M dof

- Linear elasticity
- Composed of its skin, stringers and frames
- Midlinn shell elements are used
- Each node has 6 unknowns
- One extremity is fixed
- On the other extremity a rigid body element is added
- A force perpendicular to the axis is applied
Numerical experiments on large 3D problems

Numerical behaviour of sparse preconditioners

Convergence history

Time history

- Fuselage problem of 6.5 Mdof dof mapped on 16 processors
- The sparse preconditioner setup is 4 times faster than the dense one (19.5 v.s. 89 seconds)
- In term of global computing time, the sparse algorithm is about twice faster
- The attainable accuracy of the hybrid solver is comparable to the one computed with the direct solver
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, keeping 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
## Numerical improvement benefits

<table>
<thead>
<tr>
<th># total processors</th>
<th>Algo</th>
<th># subdomains</th>
<th># processors/subdomain</th>
<th># iter</th>
<th>iterative loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 processors</td>
<td>1-level parallel</td>
<td>16</td>
<td>1</td>
<td>147</td>
<td>77.9</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>8</td>
<td>2</td>
<td>98</td>
<td>51.4</td>
</tr>
<tr>
<td>32 processors</td>
<td>1-level parallel</td>
<td>32</td>
<td>1</td>
<td>176</td>
<td>58.1</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>16</td>
<td>2</td>
<td>147</td>
<td>44.8</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>8</td>
<td>4</td>
<td>98</td>
<td>32.5</td>
</tr>
<tr>
<td>64 processors</td>
<td>1-level parallel</td>
<td>64</td>
<td>1</td>
<td>226</td>
<td>54.2</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>32</td>
<td>2</td>
<td>176</td>
<td>40.1</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>16</td>
<td>4</td>
<td>147</td>
<td>31.3</td>
</tr>
<tr>
<td></td>
<td>2-level parallel</td>
<td>8</td>
<td>8</td>
<td>98</td>
<td>27.4</td>
</tr>
</tbody>
</table>

- Reduce the number of subdomains $\implies$ reduce the number of iterations
- Though the subdomain size increases, the time of the iterative loop decreases as:
  - The number of iterations decreases
  - Each subdomain is handled in parallel
  - All the iterative kernels are efficiently computed in parallel
- The speedup factors of the iterative loop vary from 1.3 to 1.8
- Very attractive especially when the convergence rate depends on the # of subdomains
### Parallel performance benefits

<table>
<thead>
<tr>
<th># subdomains or SMP node</th>
<th>Algo</th>
<th>proc/subdom or “working”</th>
<th>Precond setup time</th>
<th># iter</th>
<th>iterative loop time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1-level</td>
<td>1</td>
<td>208.0</td>
<td>98</td>
<td>94.1</td>
<td>525.1</td>
</tr>
<tr>
<td></td>
<td>2-level</td>
<td>2</td>
<td>124.6</td>
<td></td>
<td>51.5</td>
<td>399.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>70.8</td>
<td></td>
<td>32.5</td>
<td>326.4</td>
</tr>
<tr>
<td>16</td>
<td>1-level</td>
<td>1</td>
<td>89.0</td>
<td>147</td>
<td>77.9</td>
<td>217.2</td>
</tr>
<tr>
<td></td>
<td>2-level</td>
<td>2</td>
<td>52.7</td>
<td></td>
<td>44.8</td>
<td>147.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>30.4</td>
<td></td>
<td>31.3</td>
<td>112.0</td>
</tr>
<tr>
<td>32</td>
<td>1-level</td>
<td>1</td>
<td>30.0</td>
<td>176</td>
<td>58.1</td>
<td>124.1</td>
</tr>
<tr>
<td></td>
<td>2-level</td>
<td>2</td>
<td>20.4</td>
<td></td>
<td>40.8</td>
<td>97.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>13.0</td>
<td></td>
<td>22.7</td>
<td>71.7</td>
</tr>
</tbody>
</table>

- When running large simulations that need all the memory available on the nodes
- The *1-level parallel* algo “wastes” ressource performance (it lose 48 “idle” processors on 16 SMP)
- The *2-level parallel* algo exploits the computing facilities of the remaining “idle” processors
- The *2-level parallel* algo runs closer to the peak of per-node performance
- The preconditioner setup time benefits vary from 1.5 to 3
- The speedup factors of the iterative loop vary from 1.8 to 2.7
Outline

Next step toward exascale
Exascale platforms

- Dramatic increase in the number of resources
  - Increase of the number of cores
  - Probable loss of cache coherency between processors
  - Drastic reduction of the size of memory per-core
- Highly hierarchical platforms
  - A lot of efforts have to be done to manage intra-node parallelism
- Democratization of accelerators
  - GPUs or other type of accelerators will be “somehow” integrated in processors as specialized cores
  - Heterogeneity management
- How about storage, hardware failures, ...?
Multiple layer approach

Governing ideas: Enable advanced numerical algorithms to be executed on a scalable unified runtime system for exploiting the full potential of future exascale machines.

Basics:

- Graph of tasks
- Out-of-order scheduling
- Fine granularity
Algorithms

Governing ideas: Design high-level algorithms

Main challenges:

- Increase concurrency
- Control granularity of tasks
- Trade off numerical accuracy and stability with performance

Fundings and collaborations:

- National: Total, ANR-SFGPU
- International: AT-FastLA (Berkeley, Stanford), AT-MORSE (UTK, KAUST, UC Denver)
**Kernels**

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>RUNTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>KERNELS</td>
<td></td>
</tr>
</tbody>
</table>

**Governing ideas:** Use optimized low-level kernels

**Main challenges:**
- Possibly use existing kernels
- Otherwise design new kernels for complex hardware
- Automatic generation

**Projects and collaborations:**
- INRIA Bordeaux: MANAO
- International: AT-FastLA (Berkeley, Stanford), AT-MORSE (UTK, KAUST, UC Denver)
Next step toward exascale

A first example in dense linear algebra

Parallel Hierarchical Linear Solvers  
Nov. 5 - 35/40
A first example in dense linear algebra

+ 200 GFlop/s but 12 cores = 150 GFlop/s
Heterogeneity

<table>
<thead>
<tr>
<th>Kernel</th>
<th>CPU</th>
<th>GPU</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>sgeqrt</td>
<td>9 Gflops</td>
<td>60 Gflops</td>
<td>≈6</td>
</tr>
<tr>
<td>stsqrt</td>
<td>12 Gflops</td>
<td>67 Gflops</td>
<td>≈6</td>
</tr>
<tr>
<td>sormqr</td>
<td>8.5 Gflops</td>
<td>227 Gflops</td>
<td>≈27</td>
</tr>
<tr>
<td>stsmqr</td>
<td>10 Gflops</td>
<td>285 Gflops</td>
<td>≈27</td>
</tr>
</tbody>
</table>

- Task distribution observed on StarPU:
  - sgeqrt: 20% of tasks on GPUs
  - stsmqr: 92.5% of tasks on GPUs

- Taking advantage of heterogeneity!
  - Only do what you are good for
  - Don’t do what you are not good for
PaStiX: multicore results
PaStiX: results with GPUs over StarPU

Figure: Audi

Figure: MHD
Acknowledgments
Acknowledgments

Credit to co-workers

- Numerical methodologies:

- Applications:
  H. Benhadjali (SEISCOPE), D. Goudin (CEA-CESTA), S. Operto (Géosciences Azur), S. Pralet (SAMTECH), J. Virieux (LGIT).
Merci .... QUESTIONS ?