C2S@Exa (1st year meeting)
December, Sophia Antipolis

Parallel Hierarchical Hybrid Algebraic Linear Solvers : current and future

HiePACS Inria Project
Joint Inria-CERFACS lab
INRIA Bordeaux Sud-Ouest

Motivations

$$Ax = b$$

The “spectrum” of linear algebra solvers

Direct

- Robust/accurate for general problems
- BLAS-3 based implementations

- Memory/CPU prohibitive for large 3D problems
- Limited parallel scalability

Iterative

- Problem dependent efficiency/controlled accuracy
- Only mat-vect required, fine grain computation

- Less memory computation, possible trade-off with CPU
- Attractive “build-in” parallel features

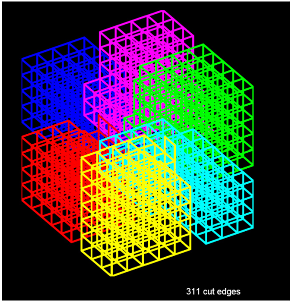
Goal: design Hybrid Linear Solvers

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



Two Hybrid Linear Solvers

Algebraic non-overlapping domain decomposition

- Partitioning of the adjacency graph of the sparse matrix
- Perform a partial Gaussian elimination (sparse direct solution on the internal variables)
- Solve the Schur complement system using a preconditioned Krylov subspace method
- Backsolve for the internal variables

Two parallel implementations

- HIPS (Hierarchical Iterative Parallel Solver)
  Incomplete LU factorization of the Schur complement based on a hierarchical interface decomposition ordering
- MaPHyS (Massively Parallel Hybrid Solver)
  Algebraic additive Schwarz preconditioner for the Schur complement

MaPHyS: Algebraic Additive Schwarz preconditioner [L.Carvalho, L.G., G.Meurant- NLAA - 01]

$$\mathcal{S} = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \mathcal{S}^{(i)} \mathcal{R}_{\Gamma_i}$$

$$\mathcal{S} = \begin{pmatrix} \ddots & & & & \\ & \mathcal{S}_{kk} & \mathcal{S}_{k\ell} & & \\ & \mathcal{S}_{\ell k} & \mathcal{S}_{\ell\ell} & \mathcal{S}_{\ell m} & \\ & & \mathcal{S}_{m\ell} & \mathcal{S}_{mm} & \mathcal{S}_{mn} \\ & & & \mathcal{S}_{nm} & \mathcal{S}_{nn} \end{pmatrix} \Longrightarrow \mathcal{M} = \begin{pmatrix} \ddots & & & & \\ & \begin{bmatrix} \mathcal{S}_{kk} & \mathcal{S}_{k\ell} \\ \mathcal{S}_{\ell k} & \mathcal{S}_{\ell\ell} \end{bmatrix}^{-1} & & & \\ & & \begin{bmatrix} \mathcal{S}_{\ell\ell} & \mathcal{S}_{\ell m} \\ \mathcal{S}_{m\ell} & \mathcal{S}_{mm} \end{bmatrix}^{-1} & & \\ & & & \begin{bmatrix} \mathcal{S}_{mm} & \mathcal{S}_{mn} \\ \mathcal{S}_{nm} & \mathcal{S}_{nn} \end{bmatrix} & \end{pmatrix}$$

$$\mathcal{M} = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T (\bar{\mathcal{S}}^{(i)})^{-1} \mathcal{R}_{\Gamma_i}$$

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

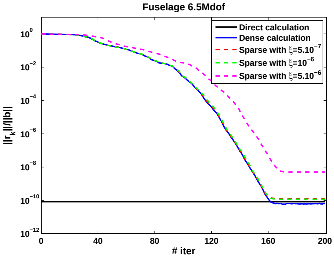
$$\underbrace{\mathcal{S}^{(i)} = \begin{pmatrix} \mathcal{S}_{kk}^{(\iota)} & \mathcal{S}_{k\ell} \\ \mathcal{S}_{\ell k} & \mathcal{S}_{\ell\ell}^{(\iota)} \end{pmatrix}}_{\text{local Schur}} \Longrightarrow \underbrace{\bar{\mathcal{S}}^{(i)} = \begin{pmatrix} \mathcal{S}_{kk} & \mathcal{S}_{k\ell} \\ \mathcal{S}_{\ell k} & \mathcal{S}_{\ell\ell} \end{pmatrix}}_{\text{local assembled Schur}}$$

$$\searrow \quad \sum_{\iota \in adj} \mathcal{S}_{\ell\ell}^{(\iota)} \quad \nearrow$$

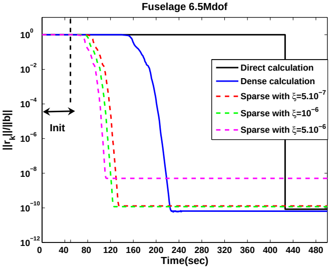
Similarity with Neumann-Neumann preconditioner [J.F Bourgat, R. Glowinski, P. Le Tallec and M. Vidrascu - 89] [Y.H. de Roek, P. Le Tallec and M. Vidrascu - 91]

MaPHyS: main future features

Convergence history



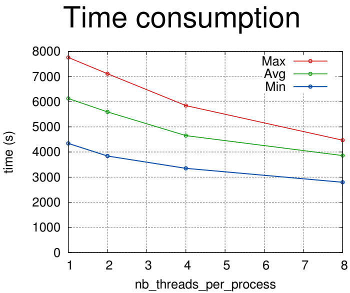
Time history

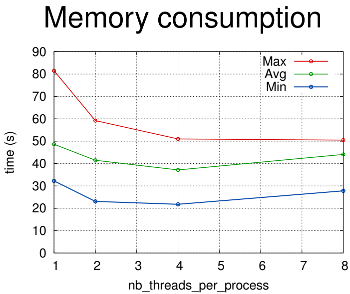


Ongoing/future software development

- Improved software flexibility (rely on MUMPS, PastiX, Scotch, ...)
- Hybrid MPI-Thread implementation (PhD thesis - DIP Inria/Total) on top of PaStiX
- Implementation on top of runtime systems - cf Emmanuel's talk

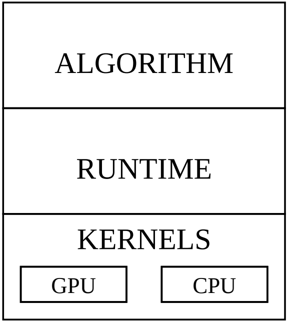
MaPHyS: main MPI-Threads features





Audi test case: 32 cores - 4 to 32 domains/MPI processes

Multiple layer approach

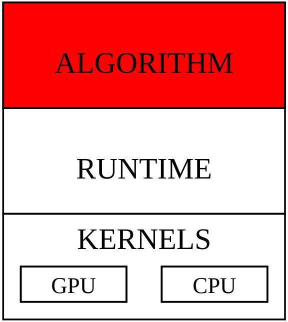


Governing ideas: Enable advanced numerical algorithms to be executed on a scalable unified runtime system for exploiting the full potential of future exascale machines.

Basics:

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

Algorithms



Governing ideas: Design high-level algorithms

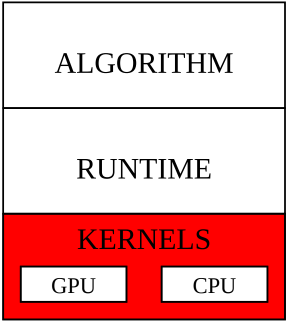
Main challenges:

- 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



Governing ideas: Use optimized low-level kernels

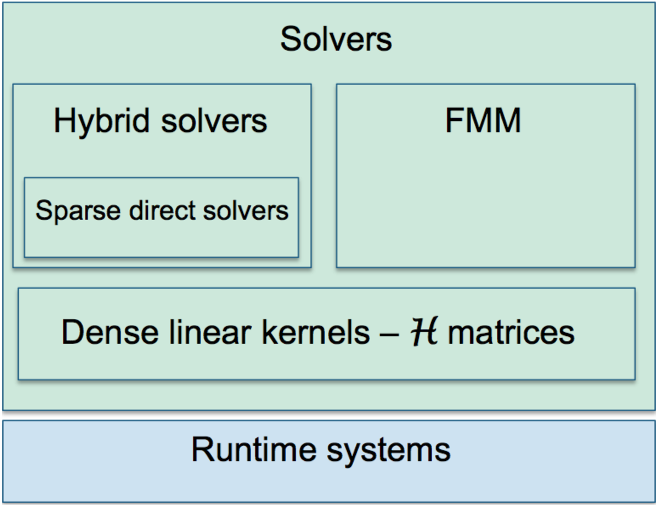
Main challenges:

- Possibly use existing kernels
- Otherwise design new kernels for complex hardware
- Automatic generation

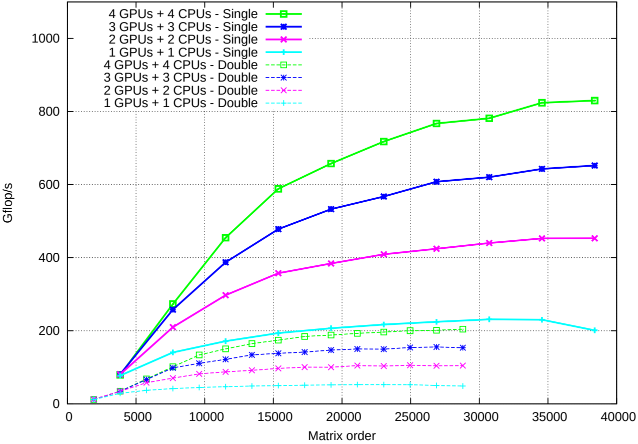
Projects and collaborations:

- INRIA Bordeaux: MANAO
- International: AT-FastLA (Berkeley, Stanford), AT-MORSE (UTK, KAUST, UC Denver)

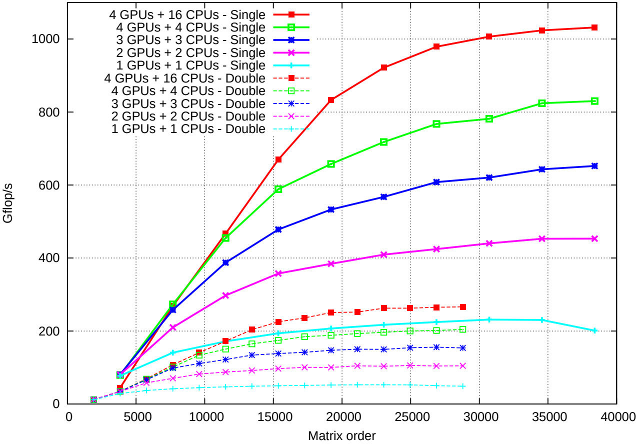
Targeted solver stack



A first example in dense linear algebra

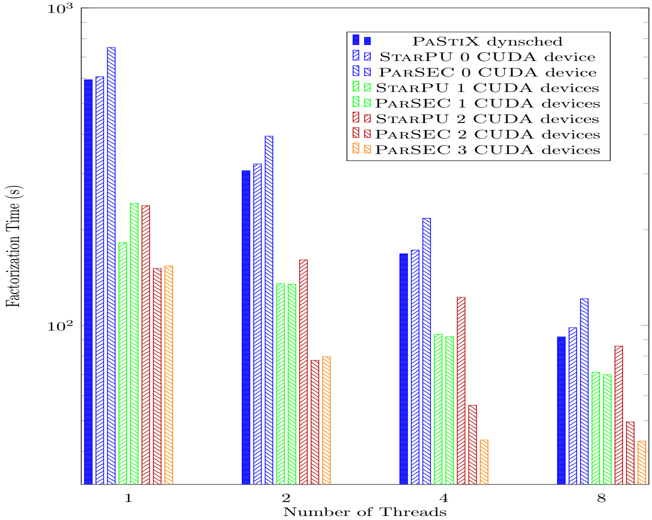


A first example in dense linear algebra



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

A second example in sparse linear algebra



LL^T A factorization on Audi test case

What’s next

Ongoing/future numerical development

- Improved numerical robustness - hierarchical toward global preconditioning (ADT Maphys@Exa via C2S@Exa)
- Coarse space mechanisms through augmentation and/or deflation (FP7 Exa2CT via C2S@Exa)
- New Krylov subspace methods (block variants, hiden/avoiding communicaitons, ...) - shared with sparse directs
- H-matrix arithmetic (Stanford/Berkeley collaboration via AT FASTLA)