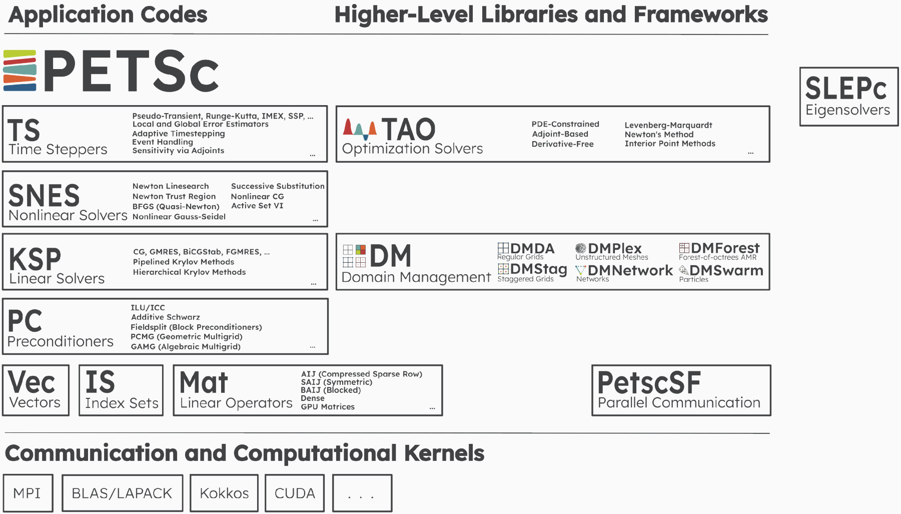
MUMPS in PETSc and HPDDM

Pierre Jolivet — Sorbonne Université, CNRS, LIP6

MUMPS User Days, June 23, 2023

https://joliv.et/MUD_2023.pdf



Application Codes

Higher-Level Libraries and Frameworks

PETSc

SLEPc
Eigensolvers

TS
Time Steppers
- Pseudo-Transient, Runge-Kutta, IMEX, SSP, ...
- Local and Global Error Estimators
- Adaptive Timestepping
- Event Handling
- Sensitivity via Adjoints
- ...

TAO
Optimization Solvers
- PDE-Constrained
- Adjoint-Based
- Derivative-Free
- Levenberg-Marquardt
- Newton's Method
- Interior Point Methods
- ...

SNES
Nonlinear Solvers
- Newton Linesearch
- Newton Trust Region
- BFGS (Quasi-Newton)
- Nonlinear Gauss-Seidel
- Successive Substitution
- Nonlinear CG
- Active Set VI
- ...

KSP
Linear Solvers
- CG, GMRES, BiCGStab, FGMRES, ...
- Pipelined Krylov Methods
- Hierarchical Krylov Methods
- ...

DM
Domain Management
- DMDA: Regular Grids
- DMPlex: Unstructured Meshes
- DMForest: Forest-of-octrees AMR
- DMStag: Staggered Grids
- DMNetwork: Networks
- DMSwarm: Particles

PC
Preconditioners
- ILU/ICC
- Additive Schwarz
- Fieldsplit (Block Preconditioners)
- PCMG (Geometric Multigrid)
- GAMG (Algebraic Multigrid)
- ...

Vec
Vectors

IS
Index Sets

Mat
Linear Operators
- AIJ (Compressed Sparse Row)
- SAIJ (Symmetric)
- BAIJ (Blocked)
- Dense
- GPU Matrices
- ...

PetscSF
Parallel Communication

Communication and Computational Kernels

MPI | BLAS/LAPACK | Kokkos | CUDA | . . .

Matrices in PETSc

- about 100 different types

Matrices in PETSc

- about 100 different types
- not all may be (inexactly) “factored”
  - `MatShell` (matrix-free)
  - `MatKAIJ` (Kronecker product)

Matrices in PETSc

- about 100 different types
- not all may be (inexactly) “factored”
  - `MatShell` (matrix-free)
  - `MatKAIJ` (Kronecker product)
- but many can
  - `MatAIJ` (compressed sparse row)
  - `MatBAIJ` (block CSR)
  - `MatSBAIJ` (symmetric BCSR)
  - `MatNest` (nested submatrices)

Matrices in PETSc

- about 100 different types
- not all may be (inexactly) “factored”
    - `MatShell` (matrix-free)
    - `MatKAIJ` (Kronecker product)
- but many can
    - `MatAIJ` (compressed sparse row)
    - `MatBAIJ` (block CSR)
    - `MatSBAIJ` (symmetric BCSR)
    - `MatNest` (nested submatrices)
- runtime composability (different types and solvers)

Features since MUD 2017 in PETSc interface

- version 5.1.1 to 5.6.0

Features since MUD 2017 in PETSc interface

- version 5.1.1 to 5.6.0
- support for BLR via `-mat_mumps_icntl_35`

Features since MUD 2017 in PETSc interface

- version 5.1.1 to 5.6.0
- support for BLR via `-mat_mumps_icntl_35`
- transpose solve and sparse distributed block of RHS

Features since MUD 2017 in PETSc interface

- version 5.1.1 to 5.6.0
- support for BLR via `-mat_mumps_icntl_35`
- transpose solve and sparse distributed block of RHS
- `-mat_mumps_use_omp_threads` to convert processes into threads

Features since MUD 2017 in PETSc interface

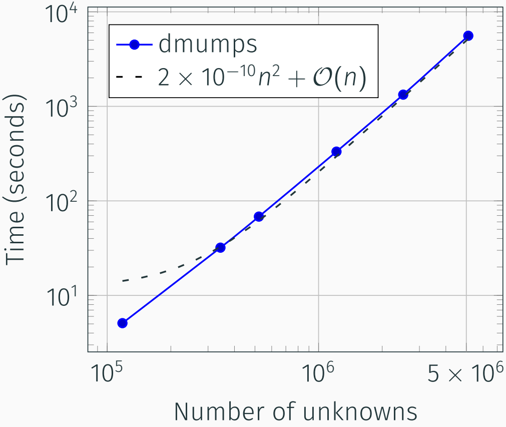
- version 5.1.1 to 5.6.0
- support for BLR via `-mat_mumps_icntl_35`
- transpose solve and sparse distributed block of RHS
- `-mat_mumps_use_omp_threads` to convert processes into threads
- better performance for block matrices via `ICNTL(15)`
- automatic handling of `MatSBAIJ` and `MatNest`

Features since MUD 2017 in PETSc interface

- version 5.1.1 to 5.6.0
- support for BLR via `-mat_mumps_icntl_35`
- transpose solve and sparse distributed block of RHS
- `-mat_mumps_use_omp_threads` to convert processes into threads
- better performance for block matrices via `ICNTL(15)`
- automatic handling of `MatSBAIJ` and `MatNest`
- different PETSc and MUMPS precision (WIP)

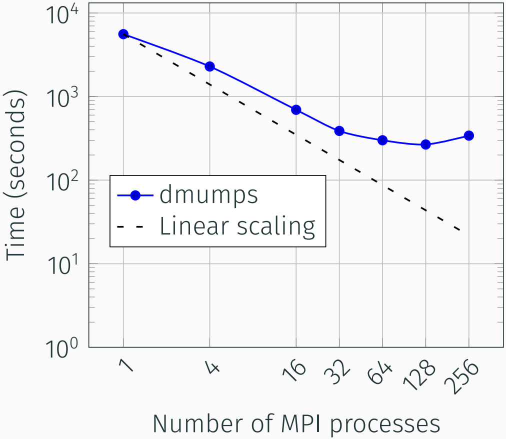
Complexity study, case #1

- 3D linear elasticity, piecewise linear FE
- sequential, double-precision, exact LDL^T factorization



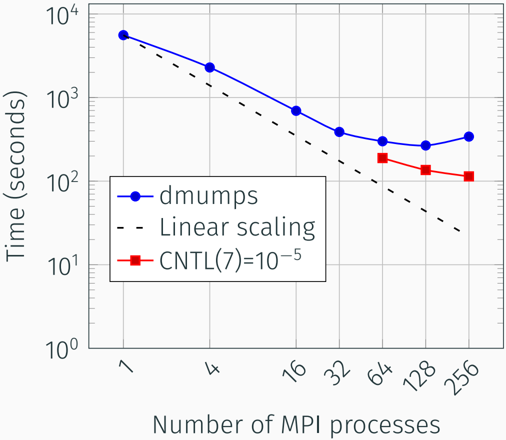
→ fighting an uphill battle

Performance study for the 5M unknowns



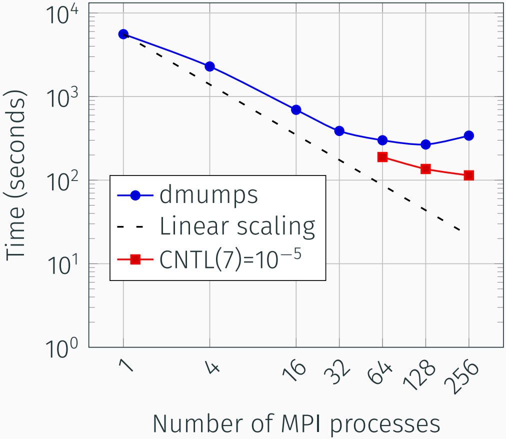
- 92% of the time (average) in numerical factorization

Performance study for the 5M unknowns



- 92% of the time (average) in numerical factorization
- FGMRES with a 10^{-5} tolerance, 15 iterations

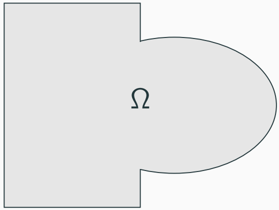
Performance study for the 5M unknowns



- 92% of the time (average) in numerical factorization
- FGMRES with a 10^{-5} tolerance, 15 iterations
- still not quite ideal, 20% efficiency

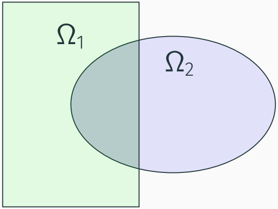
Domain decomposition preconditioning

- global linear system $Ax = b \in \mathbb{R}^n$



Domain decomposition preconditioning

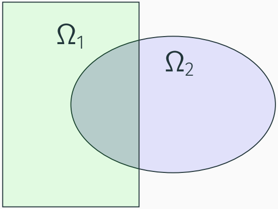
- global linear system $Ax = b \in \mathbb{R}^n$
- 2-way surjection of $[\![1; n]\!] \rightsquigarrow$ restriction operators



Domain decomposition preconditioning

- global linear system $Ax = b \in \mathbb{R}^n$
- 2-way surjection of $[\![1; n]\!]$ ⇝ restriction operators

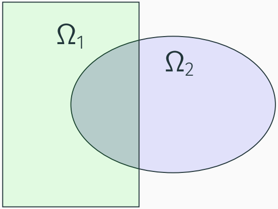
$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N=2} R_i^T (R_i A R_i^T)^{-1} R_i$$



Domain decomposition preconditioning

- global linear system $Ax = b \in \mathbb{R}^n$
- 2-way surjection of $[\![1; n]\!]$ ⇝ restriction operators

$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N=2} R_i^T (R_i A R_i^T)^{-1} R_i$$



→ not so easy, M_{ASM}^{-1} doesn't scale (numerically) as $N \to +\infty$

HPDDM

- `https://github.com/hpddm/hpddm`
- spectral coarse correction $M^{-1}_{\text{additive}} = ZA_C^{-1}Z^T + M^{-1}_{\text{ASM}}$ with $A_C = Z^TAZ$

HPDDM

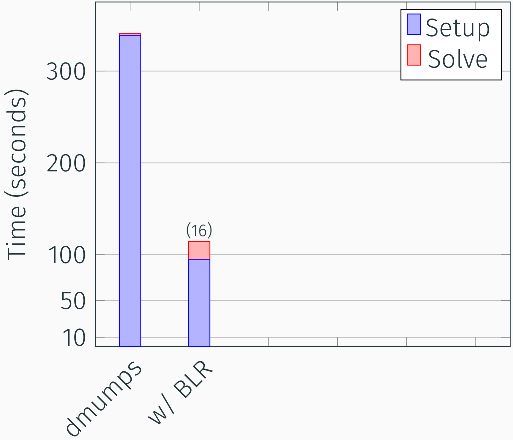
- `https://github.com/hpddm/hpddm`
- spectral coarse correction $M^{-1}_{\text{additive}} = ZA_C^{-1}Z^T + M^{-1}_{\text{ASM}}$
  with $A_C = Z^TAZ$
- three instances of MUMPS in a typical preconditioner
  - local eigensolver (computation of local "Z_i")
  - local subdomain solver $(R_iAR_i^T)^{-1}$
  - distributed coarse operator solver $(Z^TAZ)^{-1}$

HPDDM

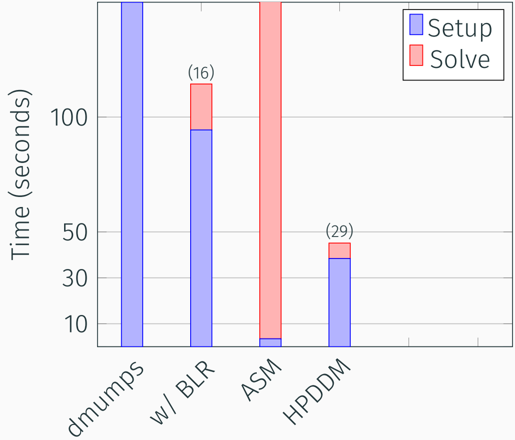
- `https://github.com/hpddm/hpddm`
- spectral coarse correction $M^{-1}_{\text{additive}} = ZA_C^{-1}Z^T + M^{-1}_{\text{ASM}}$

  with $A_C = Z^T A Z$
- three instances of MUMPS in a typical preconditioner
  - local eigensolver
  - local subdomain solver (reuse symbolic factorization)
  - distributed coarse operator solver
- runtime flexibility
  - `-pc_hpddm_levels_1_sub_mat_mumps_…`
  - `-pc_hpddm_coarse_mat_mumps_…`

5M unknowns on 256 processes

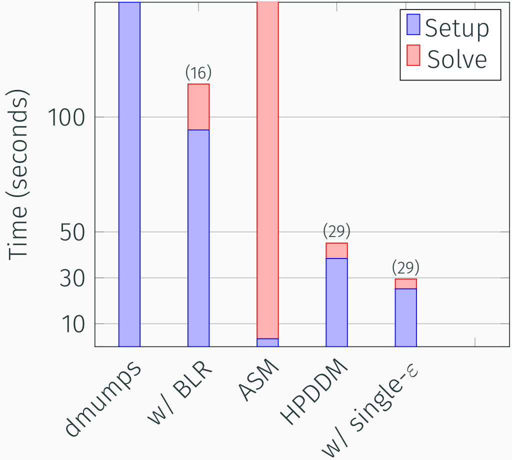


5M unknowns on 256 processes



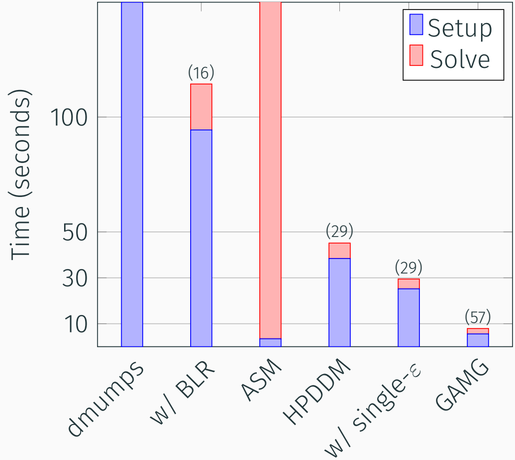
- breaking the complexity of the exact factorization

5M unknowns on 256 processes



- ○ breaking the complexity of the exact factorization
- ○ low-precision subdomain/coarse solvers

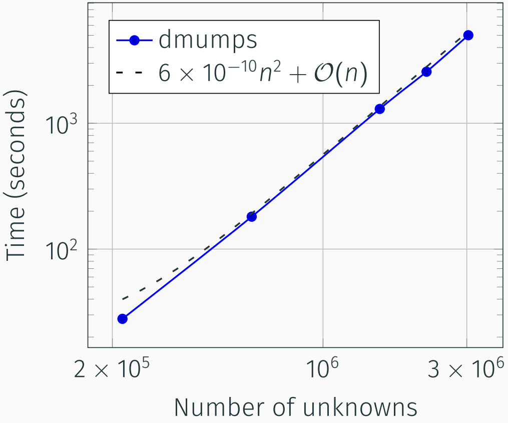
5M unknowns on 256 processes



- breaking the complexity of the exact factorization
- low-precision subdomain/coarse solvers
- disclaimer: difficult to beat AMG (when it converges)

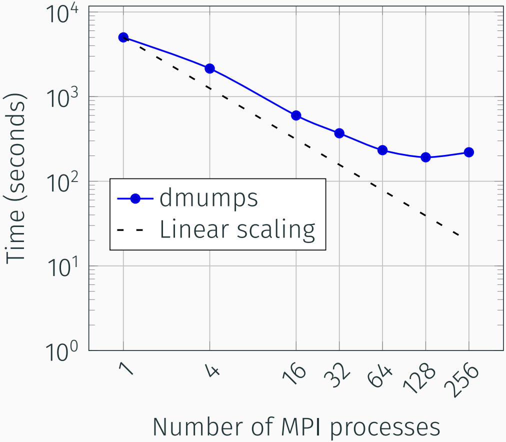
Complexity study, case #2

- 3D Stokes equation, lowest-order Taylor–Hood FE
- sequential, double-precision, exact LDL^T factorization



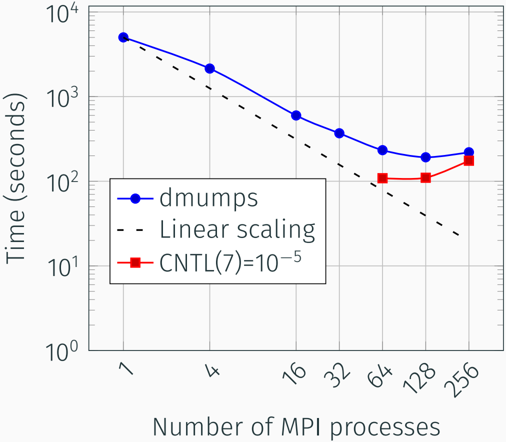
→ variable block size

Performance study for the 3M unknowns



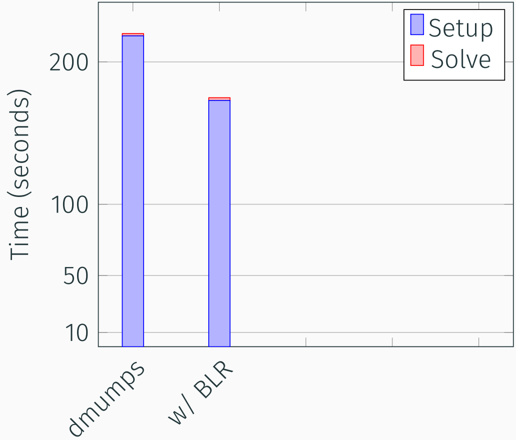
- costly symbolic factorization – no ICNTL(15)=1

Performance study for the 3M unknowns

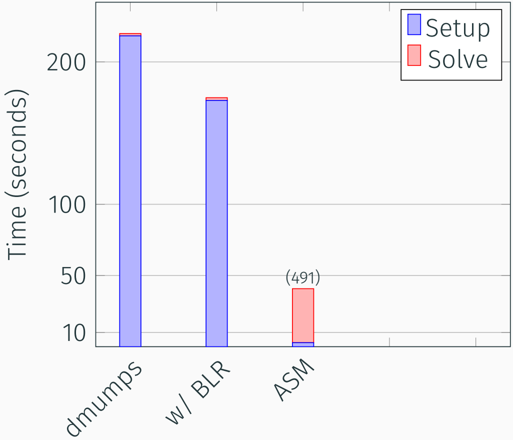


- costly symbolic factorization – no ICNTL(15)=1
- FGMRES with a 10^{-5} tolerance

3M unknowns on 256 processes

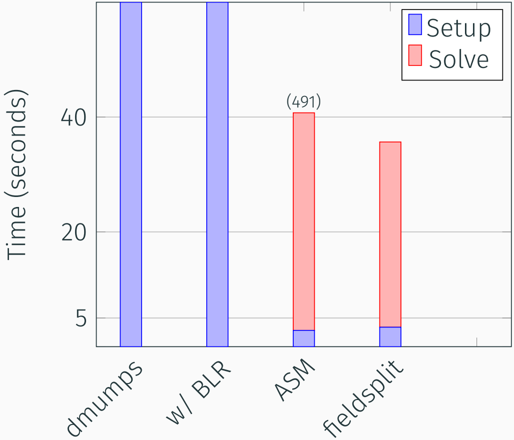


3M unknowns on 256 processes



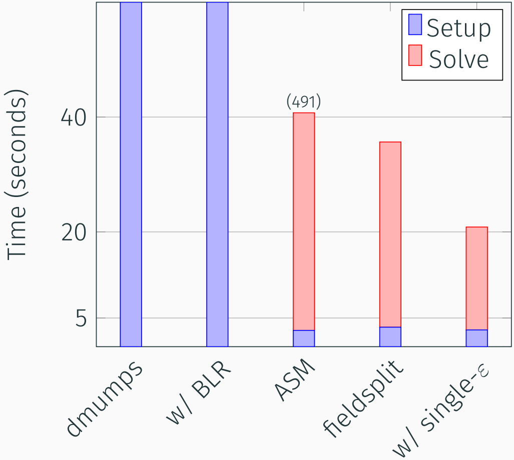
- ASM converges (albeit rather slowly)

3M unknowns on 256 processes



- ASM converges (albeit rather slowly)
- Schur complement: inner-outer iterations, ICNTL(15)=-3
- `-fieldsplit_0_`

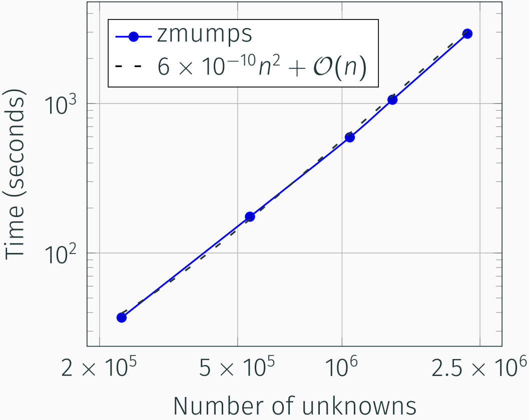
3M unknowns on 256 processes



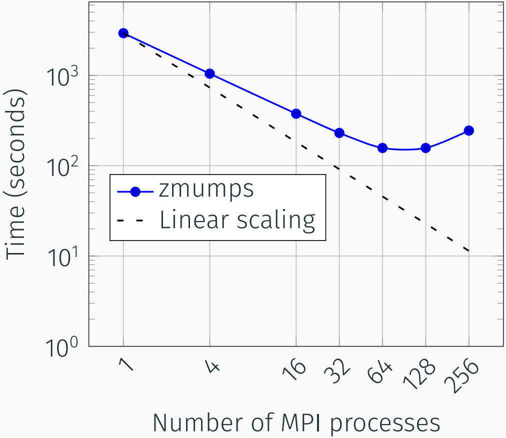
- ASM converges (albeit rather slowly)
- Schur complement: inner-outer iterations, ICNTL(15)=-3
- `-fieldsplit_0_sub_pc_precision single`

Complexity study, case #3

- 3D Maxwell equation, order-two Nédélec FE
- sequential, double-precision, exact LDL^T factorization

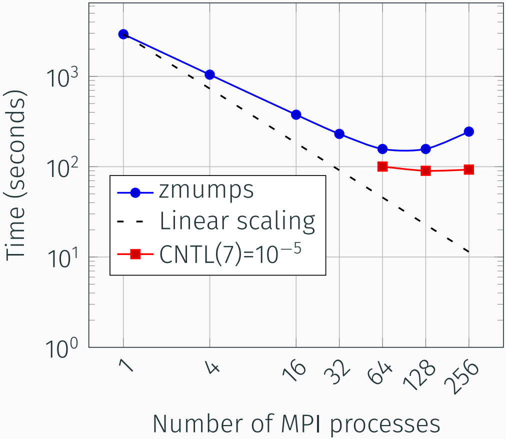


Performance study for the 2M unknowns



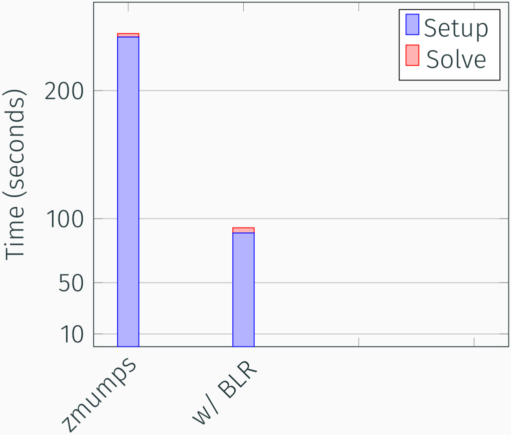
- costly symbolic factorization – no ICNTL(15)=1

Performance study for the 2M unknowns



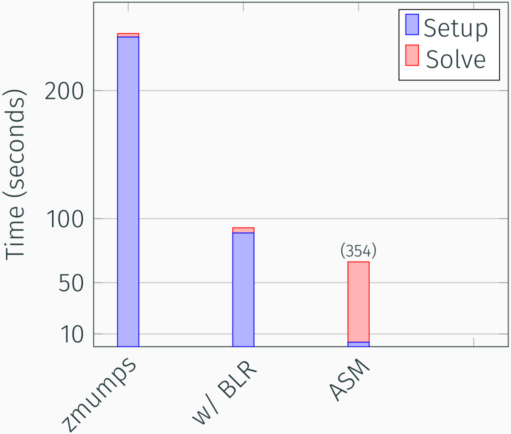
- costly symbolic factorization – no ICNTL(15)=1
- FGMRES with a 10^{-5} tolerance

2M unknowns on 256 processes



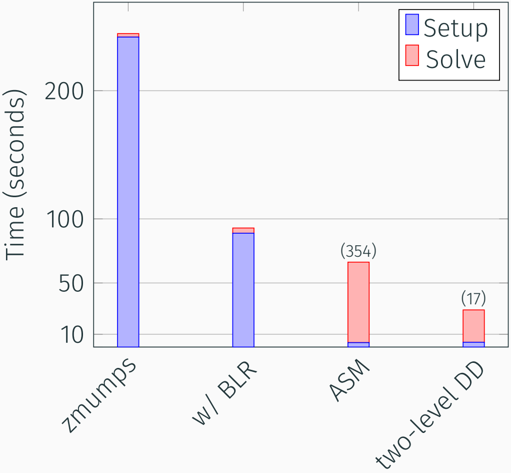
- FGMRES with a 10^{-5} tolerance

2M unknowns on 256 processes



- FGMRES with a 10^{-5} tolerance

2M unknowns on 256 processes



- FGMRES with a 10^{-5} tolerance
- better convergence with a coarse grid correction

Final words

- interoperability of MUMPS with other libraries
- runtime tuning
- composable solvers/preconditioners

Final words

- interoperability of MUMPS with other libraries
- runtime tuning
- composable solvers/preconditioners
- whish list (the one from MUD 2017 is complete!)

Final words

- interoperability of MUMPS with other libraries
- runtime tuning
- composable solvers/preconditioners
- whish list (the one from MUD 2017 is complete!)
    - ParMETIS/PT-SCOTCH on a subcommunicator

Final words

- interoperability of MUMPS with other libraries
- runtime tuning
- composable solvers/preconditioners
- whish list (the one from MUD 2017 is complete!)
    - ParMETIS/PT-SCOTCH on a subcommunicator
    - easier handling of mixed types

Final words

- interoperability of MUMPS with other libraries
- runtime tuning
- composable solvers/preconditioners
- whish list (the one from MUD 2017 is complete!)
  - ParMETIS/PT-SCOTCH on a subcommunicator
  - easier handling of mixed types

Thank you!