EoCoE²

European Commission | Horizon 2020
European Union funding
for Research & Innovation

MLD2P4
a Package of Parallel Algebraic MultiGrid Preconditioners for Scalable Linear Solvers

Pasqua D'Ambra – pasqua.dambra@cnr.it

Energy Oriented Center of Excellence: toward exascale for energy

The MLD2P4 Team

Developers:

- Salvatore Filippone
- Pasqua D'Ambra
- Fabio Durastante

Past Contributors:

- Ambra Abdullahi Hassan
- Daniela di Serafino
- Alfredo Buttari

MLD2P4

Multi-Level Domain Decomposition Parallel Preconditioners Package based on PSBLAS

Main Ref.:
P. D'Ambra, D. di Serafino, S. Filippone, MLD2P4: a package of parallel algebraic multilevel domain decomposition preconditioners in Fortran 95, ACM TOMS, 37, 2010

Freely available from
`https://github.com/sfilippone/mld2p4-2`

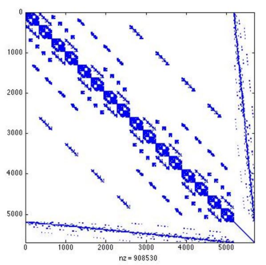
Table of Contents

Main Kernel in Computational/Data Science

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathcal{R}^{n\times n} \text{ (s.p.d.)} \quad \mathbf{x}, \mathbf{b} \in \mathcal{R}^n$$

$$n >> 10^9$$

sparsity degree $\approx 99,9\%$



Applications

numerical simulations: high-resolution models of subsurface flows in water/hydrocarbons/gas resource management require discretization meshes with more than ten billions ($> 10^{10}$) dofs

network analysis: community detection in communication/social networks, e.g., the mobile operator Vodaphone has about 200 million (2×10^8) customers and Google indexes several billion ($> 10^9$) web-pages

Table of Contents

Krylov methods

A matrix is sparse when there are so many zeros (nonzeros are typically $\mathcal{O}(n)$) that it pays off to take advantage of them in the computer representation. James Wilkinson

Krylov methods

A matrix is sparse when there are so many zeros (nonzeros are typically $\mathcal{O}(n)$) that it pays off to take advantage of them in the computer representation. James Wilkinson

Methods of choice: Search for a solution by projection

$$\begin{aligned} \mathbf{x}_m &\in \mathcal{K}_m(A, \mathbf{r}_0) \\ \mathbf{r}_m = \mathbf{b} - A\mathbf{x}_m &\perp \mathcal{K}_m(A, \mathbf{r}_0) \end{aligned}$$

$$\mathcal{K}_m(A, \mathbf{r}_0) = Span\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{m-1}\mathbf{r}_0\}$$

Krylov subspace (growing with iteration until $\mathbf{x}_m$ is good enough)

Krylov methods

A matrix is sparse when there are so many zeros (nonzeros are typically $\mathcal{O}(n)$) that it pays off to take advantage of them in the computer representation. James Wilkinson

Methods of choice: Search for a solution by projection

$$\begin{aligned} \mathbf{x}_m &\in \mathcal{K}_m(A, \mathbf{r}_0) \\ \mathbf{r}_m = \mathbf{b} - A\mathbf{x}_m &\perp \mathcal{K}_m(A, \mathbf{r}_0) \end{aligned}$$

$$\mathcal{K}_m(A, \mathbf{r}_0) = Span\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{m-1}\mathbf{r}_0\}$$

Krylov subspace (growing with iteration until $\mathbf{x}_m$ is good enough)

Conjugate Gradient (CG) for s.p.d. matrices (1952)

CG Convergence

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \leq 2\left(\frac{a-1}{a+1}\right), \quad a = \sqrt{\mu(A) = \lambda_{max}/\lambda_{min}}$$

$\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k$ error at iteration k, λ eigenvalue of A

Preconditioning

Solve the system $B^{-1}A\mathbf{x} = B^{-1}\mathbf{b}$, with matrix $B \approx A^{-1}$ (left preconditioner) such that:

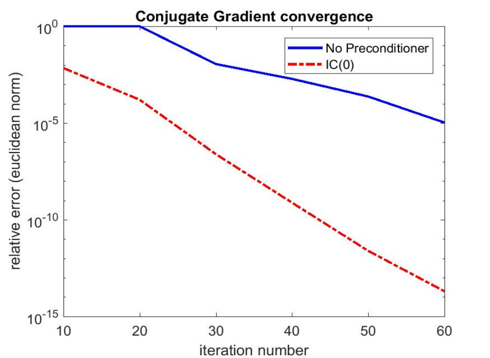
$$\mu(B^{-1}A) << \mu(A)$$

Preconditioning

Solve the system $B^{-1}A\mathbf{x} = B^{-1}\mathbf{b}$, with matrix $B \approx A^{-1}$ (left preconditioner) such that:

$$\mu(B^{-1}A) << \mu(A)$$

Solving 2D Poisson eq. (2500 dofs, $\mu(A) \approx 1.5 \times 10^3$)



IC(0): $B = LL^T$ with L incompl. Cholesky factor, $\mu(B^{-1}A) \approx 2.2 \times 10^2$

Scalable (optimal) preconditioners

- $\mu(B^{-1}A) \approx 1$, being independent of n (algorithmic scalability)
- the action of B^{-1} costs as little as possible, the best being $\mathcal{O}(n)$ flops (linear complexity)
- in a massively parallel computer, B^{-1} should be composed of local actions, (implementation scalability, i.e., parallel execution time increases linearly with n)

Scalable (optimal) preconditioners

- $\sim\$$ $\mu(B^{-1}A) \approx 1$, being independent of n (algorithmic scalability)
- $\sim\$$ the action of B^{-1} costs as little as possible, the best being $\mathcal{O}(n)$ flops (linear complexity)
- $\sim\$$ in a massively parallel computer, B^{-1} should be composed of local actions, (implementation scalability, i.e., parallel execution time increases linearly with n)

MultiGrid (MG) Preconditioners

show optimal behaviour for many s.p.d. matrices,
e.g., matrices coming from scalar elliptic PDEs

optimal preconditioner $\neq$ fastest preconditioner

Main Issues for effective parallel MG preconditioners

∼ $ single-processor performance

∼ $ memory occupation

∼ $ balance between computation and communication costs

∼ $ robustness

∼ $ flexibility and wide applicability

∼ $ preconditioner setup time vs. solve time

∼ $ re-use and efficient updating for varying matrices

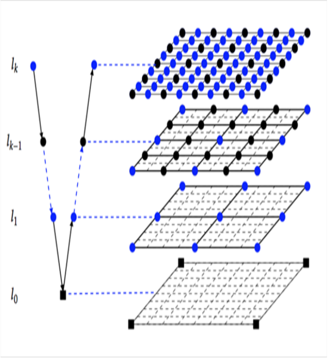
∼ $ ease of use, including interfacing with (legacy) application codes

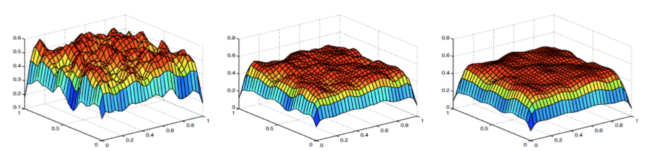
Table of Contents

MG Methods

Example: (symmetrized) V-cycle

- Pre-smoothing:
  $x = x + M^{-1}(b - Ax)$
- Residual restriction:
  $r_c = P^T(b - Ax)$
- Solution on coarse grid:
  $A_c e = r_c$, applying recursion
- Error interpolation and
  solution update: $x = x + Pe$
- Post-smoothing:
  $x = x + (M^T)^{-1}(b - Ax)$





Algebraic MultiGrid (AMG) Methods

AMG (Brandt, McCormick and Ruge, 1984)

Algebraic MultiGrid methods do not explicitly use the (eventual) problem geometry but rely only on matrix entries to generate coarse-grids by using characterizations of *algebraic smoothness*

Algebraic MultiGrid (AMG) Methods

AMG (Brandt, McCormick and Ruge, 1984)

Algebraic MultiGrid methods do not explicitly use the (eventual) problem geometry but rely only on matrix entries to generate coarse-grids by using characterizations of *algebraic smoothness*

Key issue in effective AMG for general matrices

error not reduced by the (chosen) smoother are called algebraic smoothness:

$$(Aw)_i = r_i \approx 0 \Longrightarrow w_{i+1} \approx w_i$$

Algebraic MultiGrid (AMG) Methods

AMG (Brandt, McCormick and Ruge, 1984)

Algebraic MultiGrid methods do not explicitly use the (eventual) problem geometry but rely only on matrix entries to generate coarse-grids by using characterizations of *algebraic smoothness*

Key issue in effective AMG for general matrices

error not reduced by the (chosen) smoother are called algebraic smoothness:

$$(Aw)_i = r_i \approx 0 \Longrightarrow w_{i+1} \approx w_i$$

effective AMG requires that algebraic smoothness is well represented on the coarse grid and well interpolated back $\mathbf{w} = (w_i) \in \mathcal{R}ange(P)$

Table of Contents

Algebraic MultiGrid (AMG) Setup

Recursive application of a two-grid scheme

- setup of a convergent iterative solver M (the smoother)
- setup of a coarse vector space $\mathcal{R}^{n_c}$ from $\mathcal{R}^n$
- build the prolongation P from A
- compute coarse grid matrix $A_c = P^T A P$

Algebraic MultiGrid (AMG) Setup

Recursive application of a two-grid scheme

∼ \$ setup of a convergent iterative solver M (the smoother)

∼ \$ setup of a coarse vector space $\mathcal{R}^{n_c}$ from $\mathcal{R}^n$

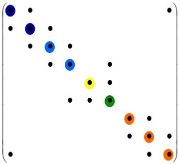
∼ \$ build the prolongation P from A

∼ \$ compute coarse grid matrix $A_c = P^T A P$

AMG based on Aggregation of dofs

Group the dofs into disjoint sets of aggregates G_j; each aggregate G_j corresponds to 1 coarse dof

Associated prolongation:



$$P := P_{ij} = \begin{cases} w_i & \text{if } i \in G_j \\ 0 & \text{otherwise} \end{cases}$$

$$i = 1, \ldots, n, \; j = 1, \ldots, n_c,$$

or smoothed version of P (Vaněk 1996)

Parallel AMG Setup: decoupled aggregationâĂŔ

Given a user-defined threshold ε

Repeat

- Pick a new root point not adjacent to any existing aggregate
- Add neighbours which are strongly connected $\left(|a^k_{ij}| \ge \varepsilon\sqrt{|a^k_{ii}a^k_{jj}|}\right)$
- Mark all points adjacent to the aggregate

Until all points are marked

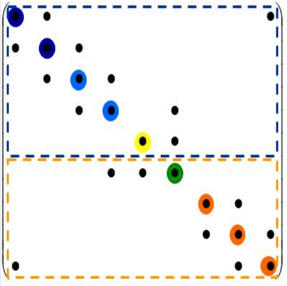
For all leftover points

- Add to an aggregated neighbour over threshold; if multiple ones, choose

$$j : |a^k_{ij}| \ge |a^k_{il}| \quad \forall l$$

- If no neighbour is above threshold, start a new aggregate

Endfor



∼ $ embarrassingly parallel but it may produce non-uniform aggregates

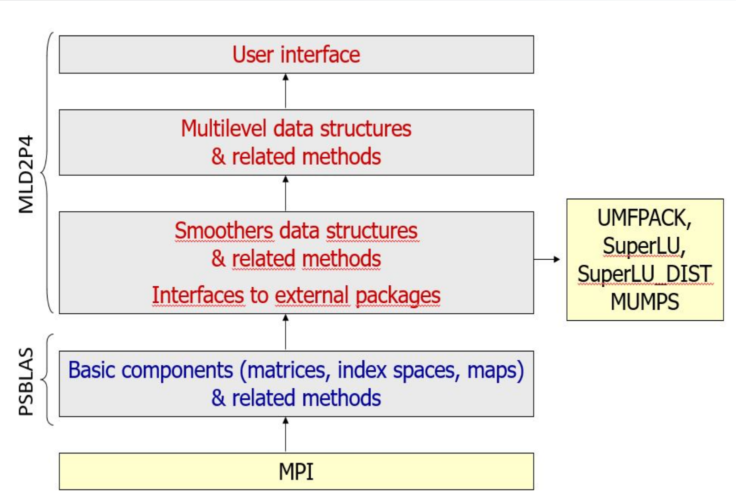
∼ $ generally it yields good results in practice on scalar elliptic problems (Tuminaro and Tong, 2000)

Table of Contents

MLD2P4: Parallel Preconditioners based on PSBLAS

- Initially developed as a package of algebraic multigrid Schwarz preconditioners, extended to more general AMG preconditioning within EoCoE
- Object-oriented design in Fortran 2003, layered sw architecture on top of PSBLAS
  $\Longrightarrow$ modularity and flexibility
- Clear separation between interface and implementation of methods
  $\Longrightarrow$ performance and extensibility
- Separated users' interface for setup of the multigrid hierarchy and setup of the smoothers and solvers to have large flexibility at each level
- Plugin for GPU exploitation (work in progress)
- C and Octave interfaces (work in progress)

MLD2P4 Software Architecture



Current version of MLD2P4 preconditioners

setup phase: **GPU implementation is work in progress**

- decoupled smoothed aggregation
- distributed or replicated coarsest matrix

solve phase: **already available on GPU** for some methods

- cycles: V, W, K
- smoothers: l_1-Jacobi, hybrid (F/B) Gauss-Seidel, block-Jacobi / additive Schwarz with LU, ILU factorizations or sparse approximate inverses for the blocks
- coarsest-matrix solvers: sparse LU, l_1-Jacobi, hybrid (F/B) Gauss-Seidel, block-Jacobi with LU, ILU factorizations or sparse approximate inverses of the blocks, iterative PCG
- LU factorizations for smoothers & coarsest-level solvers: UMFPACK, MUMPS, SuperLU, SuperLU_Dist

User's interface for preconditioner setup

- ~ $ `p%init(icontx,ptype,info)`: allocates and initializes the preconditioner p, according to the preconditioner type chosen by the user
- ~ $ `p%set(what,val,info [,ilev, ilmax, pos, idx])`: sets the parameters defining the preconditioner p, i.e., the value contained in `val` is assigned to the parameter identified by `what`
- ~ $ `p%hierarchy_build(a,desc_a,info)`: builds the hierarchy of matrices and restriction/prolongation operators for the multilevel preconditioner p
- ~ $ `p%smoothers_build(a,desc_a,p,info[,am,vm,im])`: builds the smoothers and the coarsest-level solvers for the multilevel preconditioner p
- ~ $ `p%build(a,desc_a,info[,am,vm,im])`: builds the preconditioner p (it is internally implemented by invoking the two previous methods)

User's interface for preconditioner apply

- ~ $ `p%apply(x,y,desc_a,info [,trans,work])`: computes $y = op(B^{-1})\, x$, where B is a previously built preconditioner, stored into `p`, and *op* denotes the preconditioner itself or its transpose, according to the value of `trans`.
  `p%apply` is called within the PSBLAS method `psb_krylov` and hence it is completely transparent to the user.

- ~ $ `call p%free(p,info)`: deallocates the preconditioner data structure `p`
- ~ $ `call p%descr(info, [iout])`: prints a description of the preconditioner `p`

Table of Contents

Example of use for CPU/GPU

```
! sparse matrix
type(psb_dspmat_type) :: A

! variable declaration needed for GPU running
type(psb_d_hlg_sparse_mat), target   :: ahlg
type(psb_d_vect_gpu)  :: vgm
type(psb_i_vect_gpu)  :: igm

! sparse matrix descriptor
type(psb_desc_type)   :: DESC_A
! preconditioner data
type(mld_dprec_type)  :: P

...
! inizialize parallel environment
  call psb_init(ictxt)
  call psb_info(ictxt,iam,np)

...
! read and assemble matrix A and rhs b using PSBLAS facilities
...
```

Example of Use for CPU/GPU (cont'd)

```
! setup AMG preconditioner
  call P%init('ML', info)
  call P%set(<attribute>, value, info)
...
  call P%set(<attribute>, value, info)
...
! build preconditioner
  call P%hierarchy_build(A,DESCA,info)
! last three optional parameters needed for GPU unning
  call P%smoothers_build(A,DESCA,info,am=ahlg, vm=vgm, im=igm)
! print description of the built preconditioner
  call P%descr(info)

! conversions and vector assembly needed for GPU running
  call DESCA%cnv(mold=igm)
  call A%cscnv(info,mold=ahlg)
  call psb_geasb(x,DESC_A,info,mold=vgm)
  call psb_geasb(b,DESC_A,info,mold=vgm)
```

Example of Use for CPU/GPU (cont'd)

```
! set solver parameters and initial guess
  ...
! solve Ax=b with precond CG
  call psb_krylov('CG',A,P,b,x,tol,DESC_A,info,...)
  ...
! cleanup storage
  call P%free(info)
  ...
!
! leave PSBLAS
  call psb_exit(ictxt)
```

Parameter Setting for Preconditioner Setup

```
  ...
! build a V-cycle preconditioner with 1 block-Jacobi sweep
! (with ILU(0) on the blocks) as pre- and post-smoother,
! and 8 block-Jacobi sweeps (with ILU(0) on the blocks)
! as coarsest solver
call P%init('ML',info)
call_P%set('SMOOTHER_TYPE','BJAC',info)
call P%set('COARSE_SOLVE','BJAC',info)
call P%set('COARSE_SWEEPS',8,info)
call P%hierarchy_build(A,desc_A,info)
call P%smoothers_build(A,desc_A,info)
  ...
```

Parameter Setting for Preconditioner Setup (cont'd)

```
  ...
! build a W-cycle preconditioner with 2 hybrid Gauss-Seidel sweeps
! as pre- and post-smoother, a distributed coarsest
! matrix, and MUMPS as coarsest-level solver
call P%init('ML',info)
call P%set('ML_CYCLE','WCYCLE',info)
call P%set('SMOOTHER_TYPE','FBGS',info)
call P%set('SMOOTHER_SWEEPS',2,info)
call P%set('COARSE_SOLVE','MUMPS',info)
call P%set('COARSE_MAT','DIST',info)
call P%hierarchy_build(A,desc_A,info)
call P%smoothers_build(A,desc_A,info)
  ...
```

Parameter Setting for Preconditioner Setup (cont'd)

```
  ...
! set 1-lev Restricted Additive Schwarz
! with overlap 2 and ILU(0) on the local blocks
call P%init('AS',info)
call P%set('SUB_OVR',2,info)
call P%bld(A,desc_A,info)
  ...
```

Example tests directories are available in the library both for reading data from file and for solving a classic scalar elliptic PDE

Table of Contents

Parflow Model

Simulations of subsurface flow for regional hydrology studies

Richard's equation

Filtration through variably saturated porous media for incompressible flows (3D model based on Darcy's law):

$$
\begin{aligned}
\frac{\partial(\Phi s(p))}{\partial t} + \nabla \cdot \mathbf{u} &= f \\
\mathbf{u} &= -\mathbf{K}\nabla(p - z)
\end{aligned}
$$

- implicit time integration method
- finite difference discretization of spatial operator on a structured Cartesian mesh
- Newton-Krylov solver for non-linear algebraic equation coupled with a linear geometric preconditioner
- MPI-based parallel code written in C

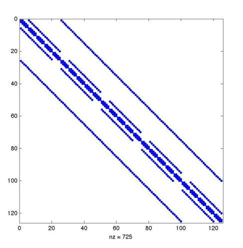
Test cases for PSBLAS and MLD2P4

Simplified steady-state model

$$-\nabla \cdot \mathsf{K} \nabla p = f$$

on unit cube, with no-flow boundary conditions

- discretization obtained by a PSBLAS code reproducing a Matlab mini-app provided by JSC
- isotropic conductivity tensor
- cartesian grid with uniform refinement along the coordinates for increasing mesh size
- hepta-diagonal spd matrices



Weak scalability on Marenostrum 4 - operated by BSC

Selected PSBLAS/MLD2P4 preconditioned iterative solvers:

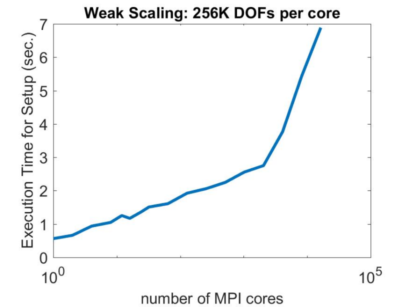
- Krylov Solver: Conjugate Gradient, with stopping criterion $\|r_k\| \leq 10^{-6}\|r_0\|$
- Preconditioner:
  - AMG based on decoupled smoothed aggregation
  - V-cycle with 1 sweep of forward/backward Hybrid Gauss-Seidel sweep as pre/post-smoother and parallel CG preconditioned with Block-Jacobi and ILU(0) at the coarsest level

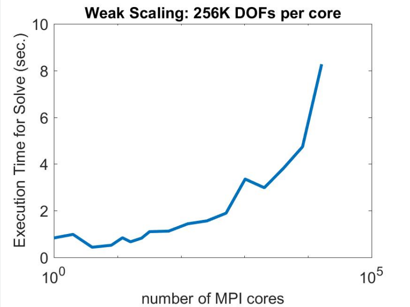
Machine Configuration: at 11,14 Petaflops, rank 29 in Top 500

- Intel Xeon Platinum 8160 CPU at 2.10GHz (Skylake); 3456 nodes, 48 cores per node
- Intel Omni-Path high-performance interconnection network

Weak scalability on Marenostrum 4 - operated by BSC

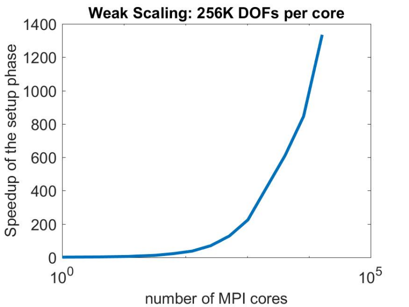
Row-block distribution of the matrix obtained
by a 3d decomposition of the grid

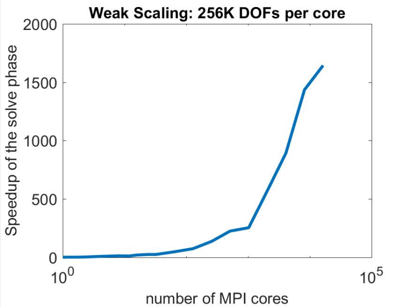


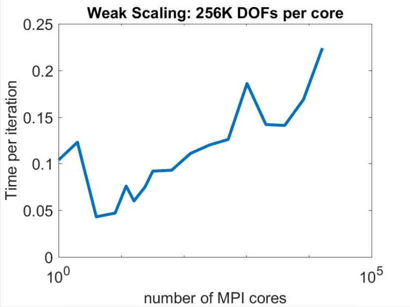


matrix with 256×10^3 rows (dofs) per core
up to 4×10^9 dofs on 16384 cores

Weak scalability on Marenostrum 4 - operated by BSC







Weak scalability on Piz Daint operated by CSCS

Selected PSBLAS/MLD2P4 preconditioned iterative solvers:

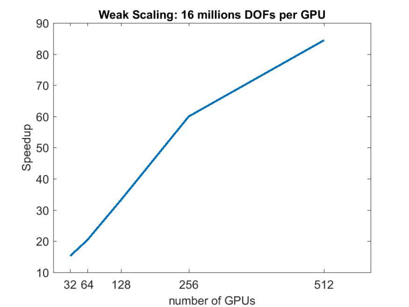
- Krylov Solver: Conjugate Gradient, with stopping criterion $\|r_k\| \leq 10^{-6}\|r_0\|$
- Preconditioner:
  - AMG based on decoupled smoothed aggregation
  - V-cycle with 2 point-wise Jacobi sweeps as pre/post-smoother and 10 sweeps of parallel Block-Jacobi, with approximate inverse applied to the blocks at the coarsest level

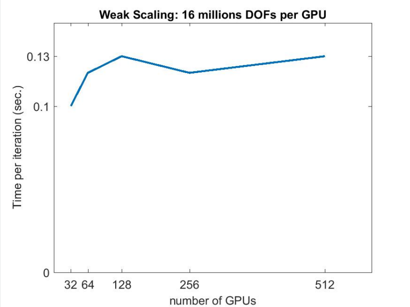
Machine Configuration (hybrid Cray XC40/XC50 system): at 21.2 petaflops, rank 6 in Top 500.

- 5704 compute nodes with Intel Xeon E5-2690 v3 CPUs per node and NVIDIA Tesla P100 16GB, 1813 compute nodes equipped with 2 Intel Xeon E5-2695 v4
- Aries routing and communications ASIC with Dragonfly network topology

Weak scalability on Piz Daint operated by CSCS

Row-block distribution of the matrix obtained
by a 3d decomposition of the grid





matrix with 16×10^6 rows (DOFs) per core
up to 8×10^9 DOFs on 512 GPUs

Work in progress within EoCoE: toward extreme scale

- ∼ $ new coupled aggregation scheme based on maximum weight matching in graphs
- ∼ $ new smoothers for efficient hybrid CPU/GPU versions
- ∼ $ efficient implementation of hybrid CPU/GPU version of preconditioners setup phase
- ∼ $ integration within KINSOL by LLNL for non-linear solvers
- ∼ $ testing within Alya from BSC and Parflow from JSC

Main References

- P. D'Ambra, F. Durastante, S. Filippone, On the Quality of Matching-based Aggregates for Algebraic Coarsening of SPD Matrices in AMG, January 2020. Available at https://arxiv.org/abs/2001.09969
- M. Bernaschi, P. D'Ambra, D. Pasquini, AMG based on compatible weighted matching for GPUs, Parallel Computing, 92, 2020.
- A. Abdullahi, V. Cardellini, P. D'Ambra, D. di Serafino, S. Filippone, Efficient Algebraic Multigrid Preconditioners on Clusters of GPUs, Parallel Processing Letters, 29, 2019
- D Bertaccini, S Filippone, Sparse approximate inverse preconditioners on high performance GPU platforms, Computers and Mathematics with Applications, 71 (3), 2016.
- A. Aprovitola, P. D'Ambra, F. M. Denaro, D. di Serafino, S. Filippone, SParC-LES: Enabling Large Eddy Simulations with Parallel Sparse Matrix Computation Tools, Computers and Mathematics with Applications, 70, 2015
- P. D'Ambra, D. di Serafino, S. Filippone, Performance Analysis of Parallel Schwarz Preconditioners in the LES of Turbulent Channel Flows, Computers and Mathematics with Applications, 65, 2013
- P. D'Ambra, D. di Serafino, S. Filippone, MLD2P4: a Package of Parallel Algebraic Multilevel Domain Decomposition Preconditioners in Fortran 95, ACM Transactions on Mathematical Software, 37 (3), 2010
- A. Buttari, P. D'Ambra, D. di Serafino, Filippone, 2LEV-D2P4: a package of high-performance preconditioners for scientific and engineering applications, Applicable Algebra in Engineering, Communication and Computing, 18 (3), 2007
- P. D'Ambra, D. di Serafino, S. Filippone, On the Development of PSBLAS-based Parallel Two-level Schwarz Preconditioners, Applied Numerical Mathematics, 57, 2007

Thanks for Your Attention