Algebraic MultiGrid Preconditioners for Sparse Linear Solvers at Extreme Scale

Salvatore Filippone

Università degli studi di Roma Tor Vergata, Dep. Civil and Computer Eng.
Consiglio Nazionale delle Ricerche, Istituto per le Applicazioni del Calcolo "M. Picone"
Cranfield University, School of Aerospace, Transport and Manufacturing
Lawrence Berkeley Laboratory

http://www.ce.uniroma2.it/~sfilippone/ salvatore.filippone@uniroma2.it

Collaborators, funding and acknowledgments

Pasqua D'Ambra,
Consiglio Nazionale delle Ricerche
Istituto per le Applicazioni del Calcolo "M. Picone"

Fabio Durastante,
Università di Pisa
CNR
Istituto per le Applicazioni del Calcolo "M. Picone"

Daniele Bertaccini,
Università di Roma "Tor Vergata"
CNR
Istituto per le Applicazioni del Calcolo "M. Picone"

Stefan Kollet,
Jülich Forschungszentrum,
Institute of Bio- and Geosciences.

Herbert Owen
Barcelona Super Computing Center

E·C·E
Project ID: 824158
European Commission
Horizon 2020
European Union funding for Research & Innovation

What we want to solve

$$\text{Solve}: \ A\mathbf{x} = \mathbf{b},$$

where

- $A \in \mathbb{R}^{n \times n}$ is a very large and sparse matrix $\text{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$,

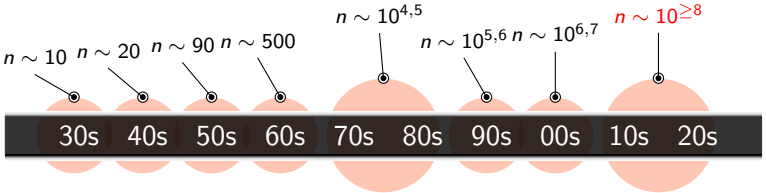
is often the most time consuming computational kernel in many areas of computational science and engineering problems.

What we want to solve

$$\text{Solve}: \ A\mathbf{x} = \mathbf{b},$$

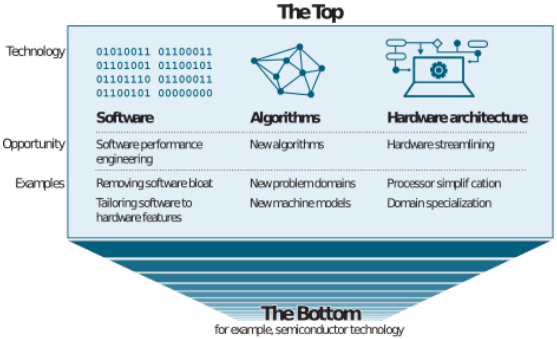
where

- $A \in \mathbb{R}^{n \times n}$ is a very large and sparse matrix $\text{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



The exascale challenge, using computer that perform 10^{15} Flops, targeting next-gen systems performing 10^{18} Flops to solve problems with tens of billions of unknowns.

The philosophy behind the effort



Leisersoneaam9744

"As miniaturization wanes, the silicon-fabrication improvements at the Bottom will no longer provide the predictable, broad-based gains in computer performance that society has enjoyed for more than 50 years. Software performance engineering, development of algorithms, and hardware streamlining at the Top can continue to make computer applications faster in the post-Moore era."

Applications

Wind Models

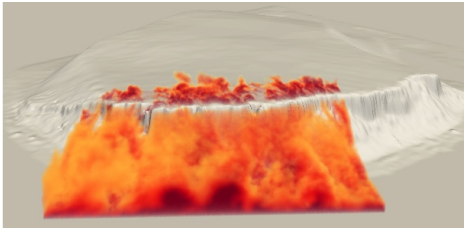
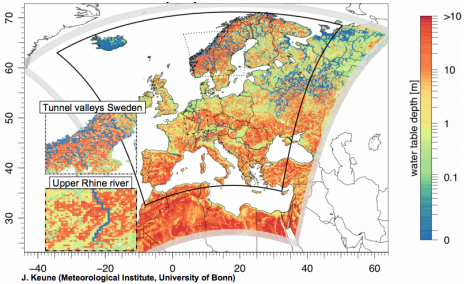


Image credits H. Owen and G. Marin, Barcelona Supercomputing Centre

- Navier-Stokes equations,
- Euler equations,
- Large Eddy Simulations,
- ...

Regional Hydrological Models



- Darcy equation,
- Richards' equation,
- Equations for overland flow
- ...

DoFs: $n \sim 10^{10}$, Processors(cores): $np \sim 10^6$

Where we want to solve it

	System	Cores	Rmax (PFlops/s)
1	Frontier	8,730,112	1,102.00
2	Fugaku	7,630,848	442.01
3	Lumi	1,110,144	151.90
4	Leonardo	1,463,616	174.70
...	...	...	...
24	Marconi-100	347,776	21.64
26	Piz Daint	387,872	21.23

Leonardo

Marconi - CINECA

Piz Daint - CSCS

MareNostrum IV - BSC

- Machines with thousands of MPI cores,
- Hybrid form of parallelism: MPI, OpenMP, CUDA/OpenCL, ...
- but how do we want to solve it?

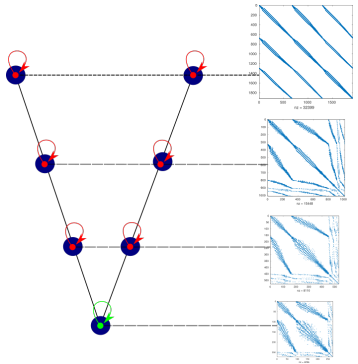
Algebraic Multigrid Preconditioners

Given Matrix $A \in \mathbb{R}^{n \times n}$ SPD

Wanted Iterative method B to precondition the CG method:

- Hierarchy of systems
$$A_l \mathbf{x}_= \mathbf{b}_l, l = 0, \ldots, \text{nlev}$$
- Transfer operators:
$$P_{l+1}^l : \mathbb{R}^{n_{l+1}} \rightarrow \mathbb{R}^{n_l}$$

Missing Structural/geometric infos



Smoother

$$M_l : \mathbb{R}^{n_l} \rightarrow \mathbb{R}^{n_l}$$

"High frequencies"

Prolongator

$$P_{l+1}^l : \mathbb{R}^{n_l} \rightarrow \mathbb{R}^{n_{l+1}}$$

"Low frequencies"

Complementarity of Smoother and Prolongator

What do we ask to it?

Solve the preconditioned system:

$$B^{-1}Ax = B^{-1}b,$$

with matrix $B^{-1} \approx A^{-1}$ (left preconditioner) such that:

Algorithmic scalability $\max_i \lambda_i(B^{-1}A) \approx 1$ being independent of n,

Linear complexity the action of B^{-1} costs as little as possible, the best being $\mathcal{O}(n)$ flops,

Implementation scalability in a massively parallel computer, B^{-1} should be composed of local actions, performance should depend linearly on the number of processors employed.

What is our *recipe*?

- The smoother M is a standard iterative solver with good parallel properties, e.g., ℓ_1–Jacobi, Hybrid-FBGS, Hybrid-SGS, CG method, etc.

What is our *recipe*?

- The smoother M is a standard iterative solver with good parallel properties, e.g., ℓ_1–Jacobi, Hybrid-FBGS, Hybrid-SGS, CG method, etc.
- The prolongator P is built by dofs *aggregation based on matching* in the weighted (adjacency) graph of A.

What is our *recipe*?

- The smoother M is a standard iterative solver with good parallel properties, e.g., ℓ_1–Jacobi, Hybrid-FBGS, Hybrid-SGS, CG method, etc.
- The prolongator P is built by dofs *aggregation based on matching* in the weighted (adjacency) graph of A.
- The coarse solver is again a preconditioned CG method.

What is our *recipe*?

- The smoother M is an iterative solver with good parallel properties:

What is our *recipe*?

- The smoother M is an iterative solver with good parallel properties:
  - GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \text{diag}(A)$ and $L = \text{tril}(A)$ is intrinsically sequential!

What is our *recipe*?

- The smoother M is an iterative solver with good parallel properties:
  - GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \text{diag}(A)$ and $L = \text{tril}(A)$ is intrinsically sequential!
  - HGS Inexact block-Jacobi version of GS, in the portion of the row-block local to each process the method acts as the GS method.

What is our *recipe*?

- The smoother M is an iterative solver with good parallel properties:

  GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \mathrm{diag}(A)$ and $L = \mathrm{tril}(A)$ is intrinsically sequential!

  HGS Inexact block-Jacobi version of GS, in the portion of the row-block local to each process the method acts as the GS method.

  ℓ_1-HGS On process $p = 1, \ldots, np$ relative to the index set Ω_p we factorize $A_{pp} = L_{pp} + D_{pp} + L_{pp}^T$ for $D_{pp} = \mathrm{diag}(A_{pp})$ and $L_{pp} = \mathrm{trilu}(A_{pp})$ and select:

$$
\begin{aligned}
M_{\ell_1-HGS} &= \mathrm{diag}((M_{\ell_1-HGS})_p)_{p=1,\ldots np}, \\
(M_{\ell_1-HGS})_p &= L_{pp} + D_{pp} + D_{\ell_1 p}, \\
(d_{\ell_1})_{i=1}^{nb} &= \sum_{j \in \Omega_p^{nb}} |a_{ij}|.
\end{aligned}
$$

What is our *recipe*?

- The smoother M is an iterative solver with good parallel properties:

  GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \mathrm{diag}(A)$ and $L = \mathrm{tril}(A)$ is intrinsically sequential!

  HGS Inexact block-Jacobi version of GS, in the portion of the row-block local to each process the method acts as the GS method.

  ℓ_1-HGS On process $p = 1, \ldots, np$ relative to the index set Ω_p we factorize $A_{pp} = L_{pp} + D_{pp} + L_{pp}^T$ for $D_{pp} = \mathrm{diag}(A_{pp})$ and $L_{pp} = \mathrm{trilu}(A_{pp})$ and select:

$$M_{\ell_1 - HGS} = \mathrm{diag}((M_{\ell_1 - HGS})_p)_{p=1,\ldots np},$$

  AINV Block-Jacobi with an approximate inverse factorization on the block $\Rightarrow$ suitable for GPU application!

What is our *recipe*?

- The prolongator P is built by dofs *aggregation based on matching* in the weighted (adjacency) graph of A.

Given $\mathbf{w} \in \mathbb{R}^n$, let $P \in \mathbb{R}^{n \times n_c}$ and $P_f \in \mathbb{R}^{n \times n_f}$ be a prolongator and a complementary prolongator, such that:

$$\mathbb{R}^n = \text{Range}(P) \oplus^{\perp} \text{Range}(P_f), \quad n = n_c + n_f$$

$\mathbf{w} \in \text{Range}(P)$: coarse space $\qquad$ $\text{Range}(P_f)$: complementary space

$$[P, P_f]^T A [P, P_f] = \begin{pmatrix} P^T A P & P^T A P_f \\ P_f^T A P & P_f^T A P_f \end{pmatrix} = \begin{pmatrix} A_c & A_{cf} \\ A_{fc} & A_f \end{pmatrix}$$

A_c: coarse matrix $\qquad$ A_f: hierarchical complement

Sufficient condition for efficient coarsening

$A_f = P_f^T A P_f$ as well conditioned as possible, i.e.,
Convergence rate of *compatible relaxation*: $\rho_f = \|I - M_f^{-1} A_f\|_{A_f} \ll 1$

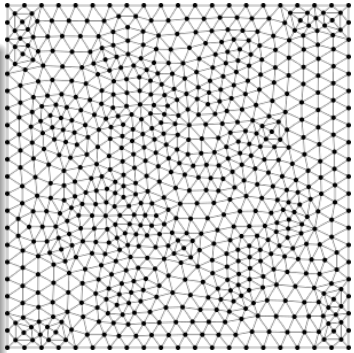
But *how* we achieve it?

Weighted graph matching

Given a graph $G = (\mathcal{V}, \mathcal{E})$ (with adjacency matrix A), and a weight vector $\mathbf{w}$ we consider the weighted version of G obtained by considering the weight matrix $\hat{A}$:

$$(\hat{A})_{i,j} = \hat{a}_{i,j} = 1 - \frac{2a_{i,j}w_i w_j}{a_{i,i}w_i^2 + a_{j,j}w_j^2},$$

- a *matching* $\mathcal{M}$ is a set of pairwise non-adjacent edges, containing no loops;
- a maximum product matching if it maximizes the product of the weights of the edges $e_{i \mapsto j}$ in it.



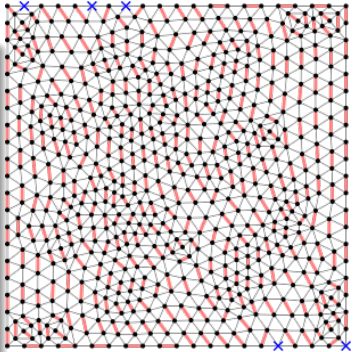
But *how* we achieve it?

Weighted graph matching

Given a graph $G = (\mathcal{V}, \mathcal{E})$ (with adjacency matrix A), and a weight vector $\mathbf{w}$ we consider the weighted version of G obtained by considering the weight matrix $\hat{A}$:

$$(\hat{A})_{i,j} = \hat{a}_{i,j} = 1 - \frac{2a_{i,j}w_i w_j}{a_{i,i}w_i^2 + a_{j,j}w_j^2},$$

- a *matching* $\mathcal{M}$ is a set of pairwise non-adjacent edges, containing no loops;
- a maximum product matching if it maximizes the product of the weights of the edges $e_{i \mapsto j}$ in it.



We divide the index set into matched vertexes $\mathcal{I} = \bigcup_{i=1}^{n_p} \mathcal{G}_i$, with $\mathcal{G}_i \cap \mathcal{G}_j = \emptyset$ if $i \neq j$, and unmatched vertexes, i.e., n_s singletons G_i.

From the matching to the prolongator

We can formally define a *prolongator*:

$$P = \left[\begin{array}{cc} \underbrace{\begin{bmatrix} \mathbf{w}_{e_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathbf{w}_{e_{n_p}} \end{bmatrix}}_{n_p} 2n_p & \mathbf{0} \\ \mathbf{0} & \underbrace{\begin{bmatrix} w_1/|w_1| & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & w_{n_s}/|w_{n_s}| \end{bmatrix}}_{n_s} n_s \end{array}\right] n = 2n_p + n_s$$

$$\underbrace{}_{n_c = n_p + n_s = J}$$

$$= \begin{bmatrix} \tilde{P} & O \\ O & W \end{bmatrix} = [\mathbf{p}_1, \dots, \mathbf{p}_J], \qquad \mathbf{w}_e = \frac{1}{\sqrt{w_i^2 + w_j^2}} \begin{bmatrix} w_i \\ w_j \end{bmatrix}.$$

$\Rightarrow$ The $\mathcal{M}$ on $\hat{A}$ produces A_f with diagonal entries $\hat{a}_{ij}$ for $(i,j) \in \mathcal{M}$ of maximal product.

We can formally define a *prolongator*:

$$P = \begin{bmatrix} \tilde{P} & O \\ O & W \end{bmatrix} = [\mathbf{p}_1, \ldots, \mathbf{p}_J].$$

Then the preconditioner is the linear operator corresponding to the multiplicative composition of

$$I - B_l A_l = (I - (M_l)^{-T} A_l)(I - P_l B_{l+1} (P_l)^T A_l)(I - M_l^{-1} A_l) \quad \forall l < nl,$$

where $A_{l+1} = (P_l)^T A_l P_l$ for $l = 0, \ldots, nl - 1$.

From the matching to the prolongator

We can formally define a *prolongator*:

$$P = \begin{bmatrix} \tilde{P} & O \\ O & W \end{bmatrix} = [\mathbf{p}_1, \ldots, \mathbf{p}_J].$$

Then the preconditioner is the linear operator corresponding to the multiplicative composition of

$$I - B_l A_l = (I - (M_l)^{-T} A_l)(I - P_l B_{l+1} (P_l)^T A_l)(I - M_l^{-1} A_l) \quad \forall l < nl,$$

where $A_{l+1} = (P_l)^T A_l P_l$ for $l = 0, \ldots, nl - 1$.

- To increase dimension reduction we can perform more than one sweep of matching per step,

From the matching to the prolongator

We can formally define a *prolongator*:

$$P = \begin{bmatrix} \tilde{P} & O \\ O & W \end{bmatrix} = [\mathbf{p}_1, \ldots, \mathbf{p}_J].$$

Then the preconditioner is the linear operator corresponding to the multiplicative composition of

$$I - B_l A_l = (I - (M_l)^{-T} A_l)(I - P_l B_{l+1} (P_l)^T A_l)(I - M_l^{-1} A_l) \quad \forall l < nl,$$

where $A_{l+1} = (P_l)^T A_l P_l$ for $l = 0, \ldots, nl - 1$.

- To increase dimension reduction we can perform more than one sweep of matching per step,
- To increase regularity of P_l we can consider a smoothed prolongator by applying a Jacobi smoother,

$$P_l^s = (I - \omega D_l^{-1} A_l) P_l, \text{ for } D_l = \text{diag}(A_l).$$

From the matching to the prolongator

We can formally define a *prolongator*:

$$P = \begin{bmatrix} \tilde{P} & O \\ O & W \end{bmatrix} = [\mathbf{p}_1, \ldots, \mathbf{p}_J].$$

Then the preconditioner is the linear operator corresponding to the multiplicative composition of

$$I - B_l A_l = (I - (M_l)^{-T} A_l)(I - P_l B_{l+1} (P_l)^T A_l)(I - M_l^{-1} A_l) \quad \forall l < nl,$$

where $A_{l+1} = (P_l)^T A_l P_l$ for $l = 0, \ldots, nl - 1$.

- To increase dimension reduction we can perform more than one sweep of matching per step,
- To increase regularity of P_l we can consider a smoothed prolongator by applying a Jacobi smoother,
- To increase the robustness we can use a non stationary solver as smoother.

1. What is the best matching algorithm from the computational point of view?

Maximum weight matching

MC64 algorithm (HSL library) based on Hungarian method, it seeks **optimal solution** for the maximum cardinality/weight matching but has a large computational complexity, $\mathcal{O}(n(n + nnz)\log n)$, and it is **i**ntrinsically sequential

Therefore, we look for

- Sub-optimal algorithms,
- quality guarantee of the computed matching, generally $1/2-$approximation to a maximum weight matching
- linear-time $\mathcal{O}(nnz)$ complexity

Parallel Matching Algorithms

1. What is the best matching algorithm from the computational point of view?

- input matrix distributed by blocks of contiguous rows,
- asynchrounous algorithm using message-aggregation to optimize communication and improve scalability
- variant available for GPU (GPU-Suitor)

Algorithm: Locally Dominant Edge

Input: Graph $G = (\mathcal{V}, \mathcal{E})$, Weights $\hat{A}$

$\mathcal{M} \leftarrow \emptyset$;

while $\mathcal{E} \neq \emptyset$ **do**

  Take a locally dominant edge $(i, j) \in \mathcal{E}$, i.e., such that

$$\arg\max_k \hat{a}_{ik} = \arg\max_k \hat{a}_{jk} = \hat{a}_{ij}$$

  Add $(i, j) \in \mathcal{M}$;

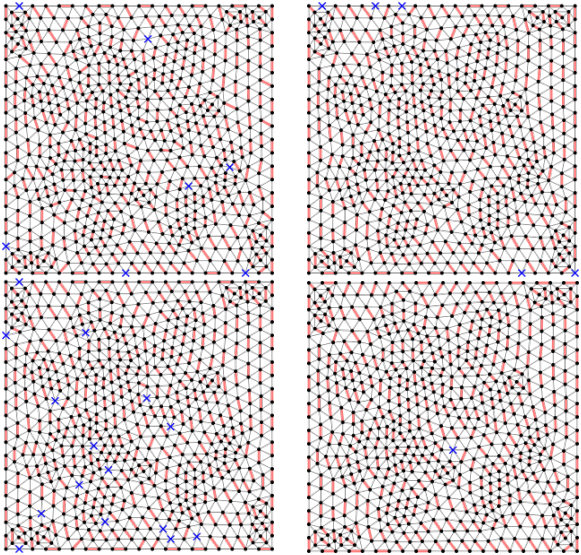
  Remove all edges incident to i and j from $\mathcal{E}$;

end

Output: Matching $\mathcal{M}$

Parallel Matching Algorithms

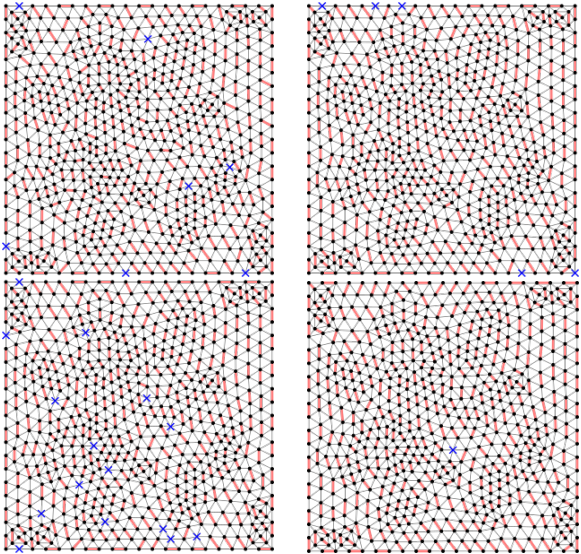
1. What is the best matching algorithm from the computational point of view?
2. How can we evaluate the quality (in term of the AMG algorithm) of the resulting matching?



Parallel Matching Algorithms

1. What is the best matching algorithm from the computational point of view?
2. How can we evaluate the quality (in term of the AMG algorithm) of the resulting matching?

With the formalism from (Xu and Zikatanov, 2017) and using a technique from (Napov and Notay, 2011) we associate a quality measure of the aggregates in terms of the convergence properties of the whole AMG method! Better aggregates give better convergence properties.



Convergence Theorem (D'Ambra, Durastante, Filippone, Zikatanov)

The exact TL–AMG with convergent smoother M, and prolongator P based on the maximum weight matching applied on a SPD matrix A has a convergence rate of

$$\|I - B^{-1}A\|_A \leq 1 - \frac{\mu_c}{c^D}, \text{ for } \mu_c = \min_{1\leq j\leq J} \mu_j(V_j^c) = \min_{1\leq j\leq J} \left[\max_{\mathbf{v}_j \in V_j} \min_{\mathbf{v}_j^c \in V_j^c} \frac{\|\mathbf{v}_j - \mathbf{v}_j^c\|_{D_j}^2}{\|\mathbf{v}_j\|_{A_j}^2} \right].$$

and c^D the continuity constant of the smoother. Moreover, the $\mu_j^{-1}(V_j^c)$ are such that

$$\lambda_2^{-1}(D_j^{-1}A_j) \leq \mu_j^{-1}(V_j^c) \leq \lambda_1^{-1}(D_j^{-1}A_j).$$

Furthermore, if either $(\mathbf{w}_{e_{i\to j}}, \lambda_1(D_j^{-1}A_j)$, or $(\mathbf{w}_{e_{i\to j}}^{\perp}, \lambda_2(D_j^{-1}A_j)$ are eigencouples of $D_j^{-1}A_j$, then

$$\mu_j^{-1}(V_j^c) = \lambda_2^{-1}(D_j^{-1}A_j)$$

- The constants c^D depends on the symmetrized $\overline{M}$ convergent smoother

$$c_D\|\mathbf{v}\|_D^2 \leq \|\mathbf{v}\|_{\overline{M}^{-1}}^2 \leq c^D\|\mathbf{v}\|_D^2.$$

Convergence Theorem (D'Ambra, Durastante, Filippone, Zikatanov)

The exact TL–AMG with convergent smoother M, and prolongator P based on the maximum weight matching applied on a SPD matrix A has a convergence rate of

$$\|I - B^{-1}A\|_A \leq 1 - \frac{\mu_c}{c^D}, \text{ for } \mu_c = \min_{1\leq j\leq J} \mu_j(V_j^c) = \min_{1\leq j\leq J} \left[\max_{\mathbf{v}_j \in V_j} \min_{\mathbf{v}_j^c \in V_j^c} \frac{\|\mathbf{v}_j - \mathbf{v}_j^c\|_{D_j}^2}{\|\mathbf{v}_j\|_{A_j}^2} \right].$$

and c^D the continuity constant of the smoother. Moreover, the $\mu_j^{-1}(V_j^c)$ are such that

$$\lambda_2^{-1}(D_j^{-1}A_j) \leq \mu_j^{-1}(V_j^c) \leq \lambda_1^{-1}(D_j^{-1}A_j).$$

Furthermore, if either $(\mathbf{w}_{e_{i\to j}}, \lambda_1(D_j^{-1}A_j)$, or $(\mathbf{w}_{e_{i\to j}}^{\perp}, \lambda_2(D_j^{-1}A_j)$ are eigencouples of $D_j^{-1}A_j$, then

$$\mu_j^{-1}(V_j^c) = \lambda_2^{-1}(D_j^{-1}A_j)$$

- The local constants $\mu_j^{-1}(V_j^c)$ are then a quality measure for the single aggregates

Convergence Theorem (D'Ambra, Durastante, Filippone, Zikatanov)

The exact TL–AMG with convergent smoother M, and prolongator P based on the maximum weight matching applied on a SPD matrix A has a convergence rate of

$$\|I - B^{-1}A\|_A \leq 1 - \frac{\mu_c}{c^D}, \text{ for } \mu_c = \min_{1\leq j\leq J} \mu_j(V_j^c) = \min_{1\leq j\leq J} \left[\max_{\mathbf{v}_j \in V_j} \min_{\mathbf{v}_j^c \in V_j^c} \frac{\|\mathbf{v}_j - \mathbf{v}_j^c\|_{D_j}^2}{\|\mathbf{v}_j\|_{A_j}^2} \right].$$

and c^D the continuity constant of the smoother. Moreover, the $\mu_j^{-1}(V_j^c)$ are such that

$$\lambda_2^{-1}(D_j^{-1}A_j) \leq \mu_j^{-1}(V_j^c) \leq \lambda_1^{-1}(D_j^{-1}A_j).$$

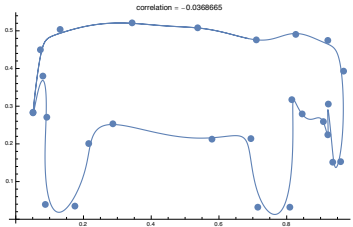
Furthermore, if either $(\mathbf{w}_{e_{i\to j}}, \lambda_1(D_j^{-1}A_j)$, or $(\mathbf{w}_{e_{i\to j}}^{\perp}, \lambda_2(D_j^{-1}A_j)$ are eigencouples of $D_j^{-1}A_j$, then

$$\mu_j^{-1}(V_j^c) = \lambda_2^{-1}(D_j^{-1}A_j)$$

- The global constant μ_c is a quality measure for the whole aggregation process

Fixing the parameters

We can fix the weight vector **w**, and evaluate the performance of the matching algorithms.



"With four parameters I can fit an elephant, and with five I can make him wiggle his trunk." – J. von Neumann

Fixing the parameters

We can fix the weight vector $\mathbf{w}$, and evaluate the performance of the matching algorithms.

Theorem (Optimal prolongator)

Let $\{\lambda_j, \mathbf{\Phi}_j\}_{j=1}^{n}$ be the eigenpairs of $\overline{T} = \overline{M}A$ for the symmetrized smoother $\overline{M}$. Let us also assume that $\mathbf{\Phi}_j$ are orthogonal w.r.t. $(\cdot,\cdot)_{\overline{M}^{-1}}$. The convergence rate $\|E(P)\|_A$ is minimal for P such that

$$\text{Range}(P) = \text{Range}(P^{opt}),$$

where $P^{opt} = \{\mathbf{\Phi}_1, \ldots, \mathbf{\Phi}_{n_c}\}$. In this case,

$$\|E\|_A^2 = 1 - \lambda_{n_c+1}$$

For our choice of P we know that:

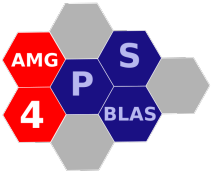
- There exists $\mathbf{h} \in \mathbb{R}^{n_c}$ such that $P\mathbf{h} = \mathbf{w}$

A good candidate can be obtained by exploiting the symmetrized smoother $\overline{M}$ to select as a weight vector an ε–smooth algebraic vector, i.e., for a given $\epsilon \in (0,1)$, $\mathbf{v}$ an algebraically ϵ-smooth with respect to A if

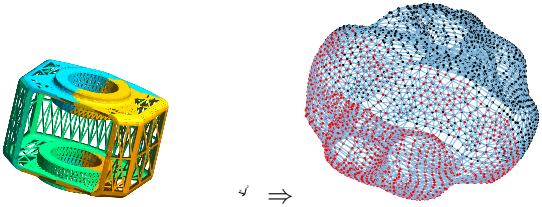
$$\|\mathbf{v}\|_A^2 \leq \epsilon \|\mathbf{v}\|_{\overline{M}^{-1}}^2.$$

Three central libraries PSBLAS, AMG4PSBLAS and PSBLAS-EXT:

- Existing software standards:
  - MPI, OpenMP, CUDA
  - Serial sparse BLAS,
  - (Par)Metis,
  - AMD
- Attention to performance;
- Research on new preconditioners;
- Data structures are essential, but design for ease of use;
- Tools for large mesh handling: the essential kernel is halo data exchange;
- Krylov subspace solvers;



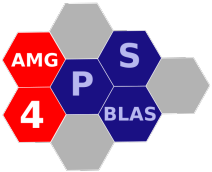
Three central libraries PSBLAS, AMG4PSBLAS and PSBLAS-EXT: Large mesh handling support



S. Filippone and A. Buttari, Object-oriented techniques for sparse matrix computations in Fortran 2003. ACM Trans. Math. Software **38** (2012), no. 4, 1–20 pp.

Three central libraries PSBLAS, AMG4PSBLAS and PSBLAS-EXT:

- Domain decomposition preconditioners
- Algebraic multigrid with aggregation schemes
  - Parallel coupled Weighted Matching Based Aggregation
  - Smoothed Aggregation (Vaněk, Mandel, Brezina)
- Parallel Smoothers (Block-Jacobi, Hybrid-GS/SGS/FBGS, ℓ_1 variants) that can be coupled with specialized block (approximate) solvers MUMPS, SuperLU, Incomplete Factorizations (AINV, INVK/L, ILU-type)
- V-Cycle, W-Cycle, K-Cycle

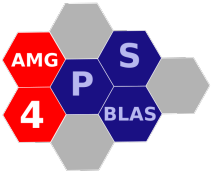


P. D'Ambra, F. Durastante, and S. Filippone. "AMG preconditioners for linear solvers towards extreme scale." SIAM J. Sci. Comp. 43.5 (2021): S679-S703.

Three central libraries PSBLAS, AMG4PSBLAS and PSBLAS-EXT:

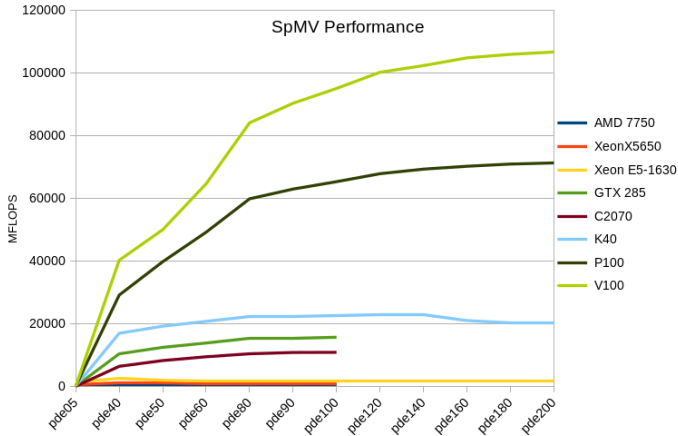
- GPU Plugin PSBLAS-EXT
- Support for NVIDIA devices;
- Many data storage formats;
- Fully integrated in PSBLAS, MPI enabled;
- Transparent use from PSBLAS/AMG4PSBLAS

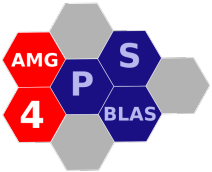
S. Filippone et al., Sparse matrix-vector multiplication on GPGPUs, ACM Trans. Math. Software **43** (2017), no. 4, Art. 30



Three central libraries PSBLAS, AMG4PSBLAS and PSBLAS-EXT.

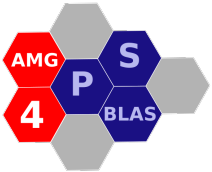
- GPU Plugin PSBLAS-EXT





Three central libraries PSBLAS, AMG4PSBLAS and PSBLAS-EXT

- Freely available from: `https://psctoolkit.github.io`,
- Open Source, released under BSD 3 Clause License,
- Interfaced with the Alya multi-physics solver, ParFlow solver, KINSOL non-linear solvers, collaborations with: Barcelona Supercomputing Centre and Jülich Forshungszentrum



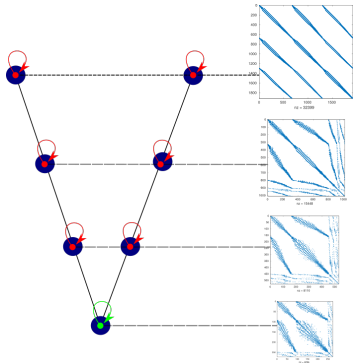
Algebraic Multigrid Preconditioners

Given Matrix $A \in \mathbb{R}^{n \times n}$ SPD

Wanted Iterative method B to precondition the CG method:

- Hierarchy of systems
$$A_l \mathbf{x}_= \mathbf{b}_l, l = 0, \ldots, \text{nlev}$$
- Transfer operators:
$$P_{l+1}^{l} : \mathbb{R}^{n_{l+1}} \rightarrow \mathbb{R}^{n_l}$$

Missing Structural/geometric infos



Smoother

$$M_l : \mathbb{R}^{n_l} \rightarrow \mathbb{R}^{n_l}$$

"High frequencies"

Prolongator

$$P_{l+1}^{l} : \mathbb{R}^{n_l} \rightarrow \mathbb{R}^{n_{l+1}}$$

"Low frequencies"

Complementarity of Smoother and Prolongator

What are we looking for?

Solve the preconditioned system:

$$B^{-1}Ax = B^{-1}b,$$

with matrix $B^{-1} \approx A^{-1}$ (left preconditioner) such that:

Algorithmic scalability $\max_i \lambda_i(B^{-1}A) \approx 1$ being independent of n,

Linear complexity the action of B^{-1} costs as little as possible, the best being $\mathcal{O}(n)$ flops,

Implementation scalability in a massively parallel computer, B^{-1} should be composed of local actions, performance should depend linearly on the number of processors employed.

What is our *recipe*?

- The smoother M is a standard iterative solver with good parallel properties, e.g., ℓ_1–Jacobi, Hybrid-FBGS, Hybrid-SGS, CG method, etc.
- The prolongator P is built by dofs *aggregation based on matching* in the weighted (adjacency) graph of A.
- The coarse solver can be (again) a preconditioned CG method.

What is our *recipe*?

- The smoother M is an iterative solver with good parallel properties:

  GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \text{diag}(A)$ and $L = \text{tril}(A)$ is intrinsically sequential!

  HGS Inexact block-Jacobi version of GS, in the portion of the row-block local to each process the method acts as the GS method.

  ℓ_1-HGS On process $p = 1, \ldots, np$ relative to the index set Ω_p we factorize $A_{pp} = L_{pp} + D_{pp} + L_{pp}^T$ for $D_{pp} = \text{diag}(A_{pp})$ and $L_{pp} = \text{tril}(A_{pp})$ then:

$$
\begin{aligned}
M_{\ell_1-HGS} &= \text{diag}((M_{\ell_1-HGS})_p)_{p=1,\ldots np}, \\
(M_{\ell_1-HGS})_p &= L_{pp} + D_{pp} + D_{\ell_1 p}, \\
(d_{\ell_1})_{i=1}^{nb} &= \sum_{j \in \Omega_p^{nb}} |a_{ij}|.
\end{aligned}
\qquad
M_{\ell_1-HGS} = \text{diag}((M_{\ell_1-HGS})_p)_{p=1,\ldots np},
$$

  AINV Block-Jacobi with an approximate inverse factorization on the block $\Rightarrow$ suitable for GPUs

What is our *recipe*?

- The prolongator P is built by dofs *aggregation based on matching* in the weighted (adjacency) graph of A.

Given $\mathbf{w} \in \mathbb{R}^n$, let $P \in \mathbb{R}^{n \times n_c}$ and $P_f \in \mathbb{R}^{n \times n_f}$ be a prolongator and a complementary prolongator, such that:

$$\mathbb{R}^n = \text{Range}(P) \oplus^{\perp} \text{Range}(P_f), \quad n = n_c + n_f$$

$\mathbf{w} \in \text{Range}(P)$: coarse space

$\text{Range}(P_f)$: complementary space

$$[P, P_f]^T A [P, P_f] = \begin{pmatrix} P^T A P & P^T A P_f \\ P_f^T A P & P_f^T A P_f \end{pmatrix} = \begin{pmatrix} A_c & A_{cf} \\ A_{fc} & A_f \end{pmatrix}$$

A_c: coarse matrix

A_f: hierarchical complement

Sufficient condition for efficient coarsening

$A_f = P_f^T A P_f$ as well conditioned as possible, i.e.,
Convergence rate of *compatible relaxation*: $\rho_f = \|I - M_f^{-1} A_f\|_{A_f} \ll 1$

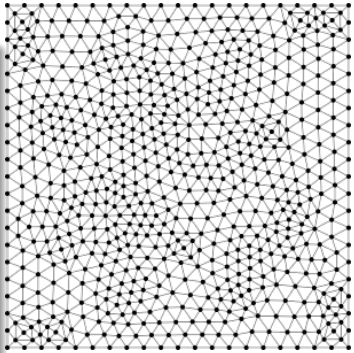
But *how* we achieve it?

Weighted graph matching

Given a graph $G = (\mathcal{V}, \mathcal{E})$ (with adjacency matrix A), and a weight vector $\mathbf{w}$ we consider the weighted version of G obtained by considering the weight matrix $\hat{A}$:

$$(\hat{A})_{i,j} = \hat{a}_{i,j} = 1 - \frac{2a_{i,j}w_i w_j}{a_{i,i}w_i^2 + a_{j,j}w_j^2},$$

- a *matching* $\mathcal{M}$ is a set of pairwise non-adjacent edges, containing no loops;
- a maximum product matching if it maximizes the product of the weights of the edges $e_{i \mapsto j}$ in it.



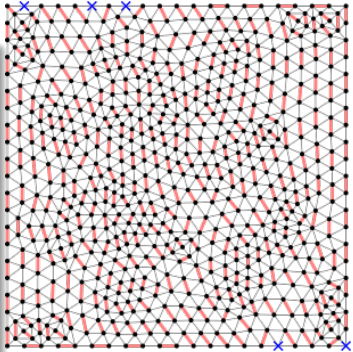
But *how* we achieve it?

Weighted graph matching

Given a graph $G = (\mathcal{V}, \mathcal{E})$ (with adjacency matrix A), and a weight vector $\mathbf{w}$ we consider the weighted version of G obtained by considering the weight matrix $\hat{A}$:

$$(\hat{A})_{i,j} = \hat{a}_{i,j} = 1 - \frac{2a_{i,j}w_i w_j}{a_{i,i}w_i^2 + a_{j,j}w_j^2},$$

- a *matching* $\mathcal{M}$ is a set of pairwise non-adjacent edges, containing no loops;
- a maximum product matching if it maximizes the product of the weights of the edges $e_{i \mapsto j}$ in it.



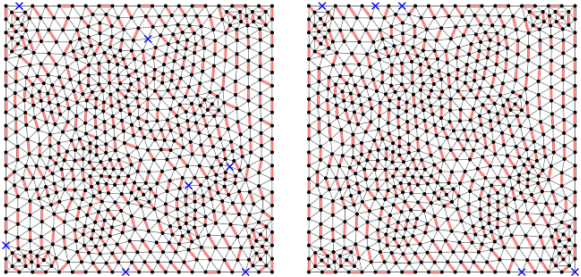
We divide the index set into matched vertices $\mathcal{I} = \bigcup_{i=1}^{n_p} \mathcal{G}_i$, with $\mathcal{G}_i \cap \mathcal{G}_j = \emptyset$ if $i \neq j$, and unmatched vertices, i.e., n_s singletons G_i.

Parallel Matching Algorithms

1. What is the best matching algorithm from a computational point of view?
2. How can we evaluate the quality (in term of the AMG algorithm) of the resulting matching?

With the formalism from (Xu and Zikatanov, 2017) and using a technique from (Napov and Notay, 2011) we associate a quality measure of the aggregates in terms of the convergence properties of the whole AMG method! Better aggregates give better convergence properties.

- P. D'Ambra, F. Durastante, S. Filippone, L. Zikatanov, Automatic coarsening in Algebraic Multigrid utilizing quality measures for matching-based aggregations, ArXiV:2001.09969



Algorithm: Locally Dominant Edge

Input: Graph $G = (\mathcal{V}, \mathcal{E})$, Weights $\hat{A}$

$\mathcal{M} \leftarrow \emptyset$;

while $\mathcal{E} \neq \emptyset$ **do**

  Take a locally dominant edge $(i, j) \in \mathcal{E}$, i.e., such that

$$\arg\max_k \hat{a}_{ik} = \arg\max_k \hat{a}_{jk} = \hat{a}_{ij}$$

  Add $(i, j) \in \mathcal{M}$;

  Remove all edges incident to i and j from $\mathcal{E}$;

end

Weak Scalability - CPU/GPU Runs - Piz Daint

- Run on the Piz Daint machine up to 28800 cores
- Test: 3D Constant coefficient Poisson Problem with FCG
- DoF: 256k/512k/1M unknowns $\times$ MPI core
- Measure: Solve Time (s).

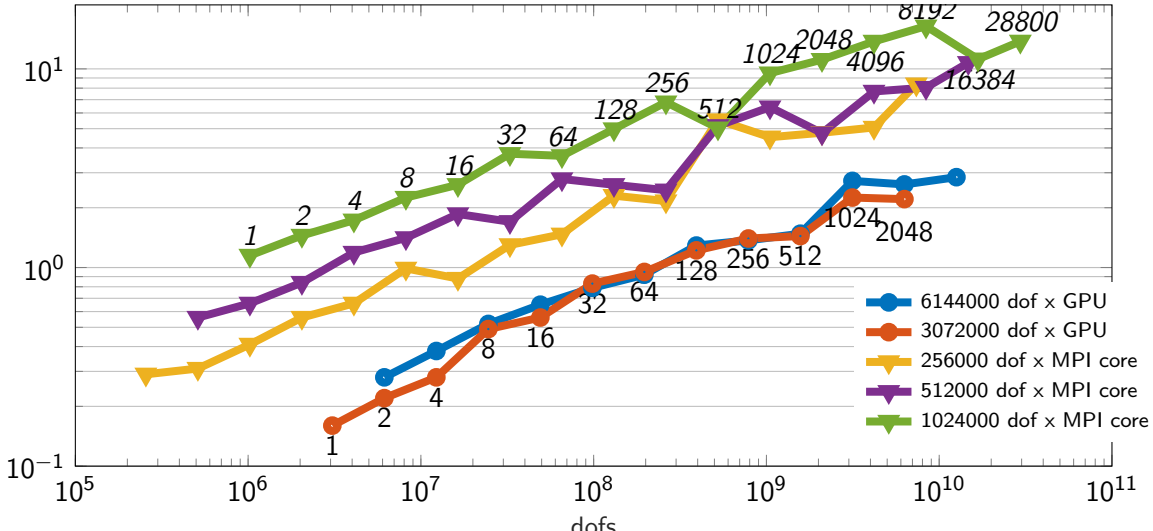
Scaling

There are two common notions of scalability:

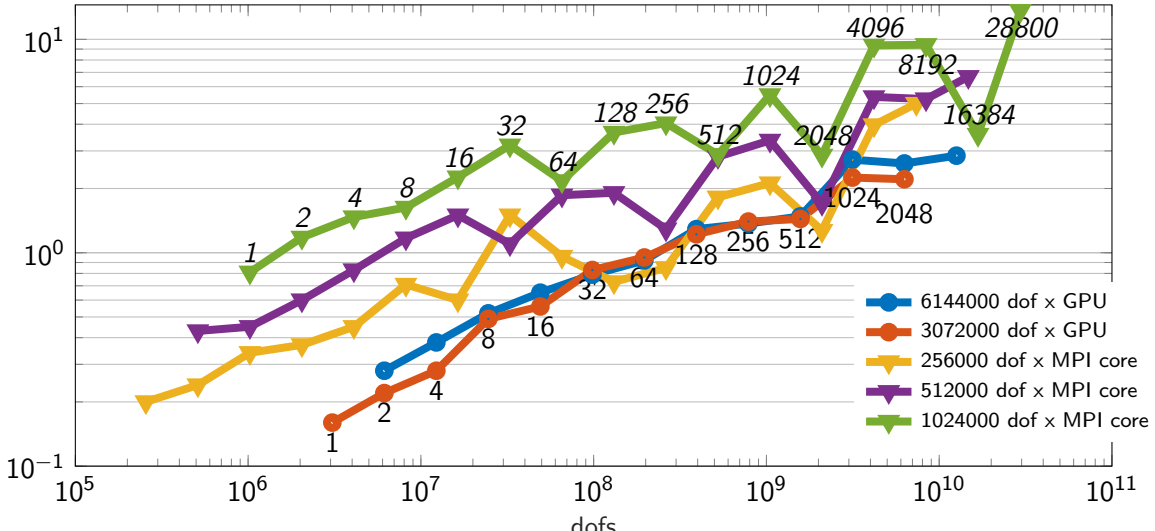
- Strong scaling analysis studies as how the solution time varies with the number of processors for a fixed total problem size.
- Weak scaling analysis studies as how the solution time varies with the number of processors for a fixed problem size per processor.

P. D'Ambra, F. Durastante, and S. Filippone. "AMG preconditioners for linear solvers towards extreme scale." SIAM J. Sci. Comp. 43.5 (2021): S679-S703.

Weak Scalability - CPU/GPU Runs - Piz Daint



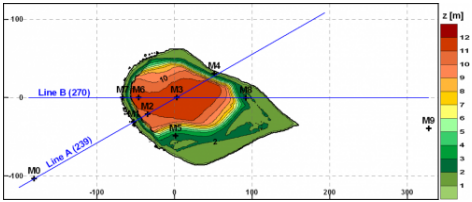
Weak Scalability - CPU/GPU Runs - Piz Daint



A CFD application inside Alya

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

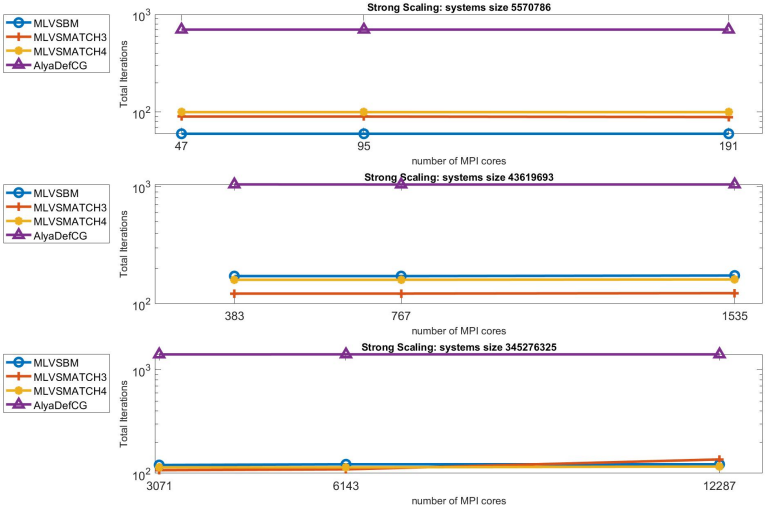
Joint work with
Herbert Owen
Barcelona Super Computing Center



Bolund is an isolated hill situated in Roskilde Fjord, Denmark. An almost vertical escarpment in the prevailing W-SW sector ensures flow separation in the windward edge resulting in a complex flow field.

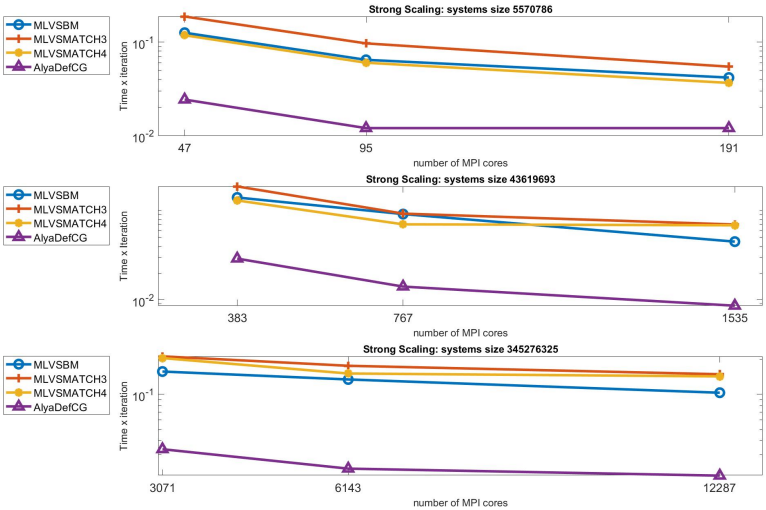
- **Model**: 3D incompressible unsteady Navier-Stokes equations for the Large Eddy Simulations of turbulent flows,
- **Discretization**: low-dissipation mixed FEM (linear FEM both for velocity and pressure),
- **Time-Stepping**: non-incremental fractional-step for pressure, explicit fourth order Runge-Kutta method for velocity.

Bolund Test Case - Strong Scaling - Pressure Equation



- Total number of linear iterations is smaller and stable for increasing number of cores,

Bolund Test Case - Strong Scaling - Pressure Equation



- The time needed per each iteration decreases for increasing number of cores,

Collaboration with JÜLICH Forschungszentrum

Richards equation models fluid flow in the *unsaturated* (vadose) zone, it is

- **non-linear** the parameters that control the flow are dependent on the saturation of the media,
- a combination of **Darcy's law** and the principle of **mass conservation**

$$\frac{\partial (\rho \phi s(p))}{\partial t} + \nabla \cdot q = 0,$$

- $s(p)$ is the saturation at pressure head p of a fluid with density ρ and terrain porosity ϕ,

The Richards Equation: constitutive equations

- q is the volumetric water flux, using Darcy's law it is written as

$$q = -K(p)\left(\nabla p + c\hat{z}\right),$$

  - $K(p)$ the hydraulic conductivity,
  - c the cosine of the angle between the downward z-axis $\hat{z}$ and the direction of the gravity force

To complete the model we need equations for both $s(p)$ and $K(p)$, we use the Van Genuchten formulation [Celia et al. 1990; Van Genuchten, 1980]

$$s(p) = \frac{\alpha(s_s - s_r)}{\alpha + |p|^\beta} + s_r, \text{ and } K(p) = K_s \frac{a}{a + |p|^\gamma},$$

where

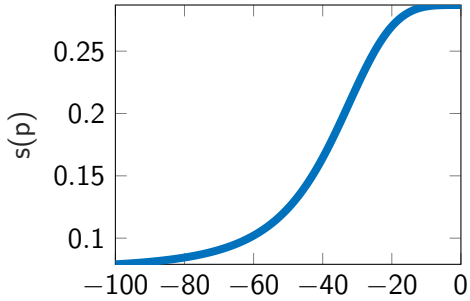
- all the parameters $(\alpha, \beta, \gamma, a)$ are fitted on real data and *assumed* to be *constant* in the media;
- K_s is the saturated hydraulic conductivity.

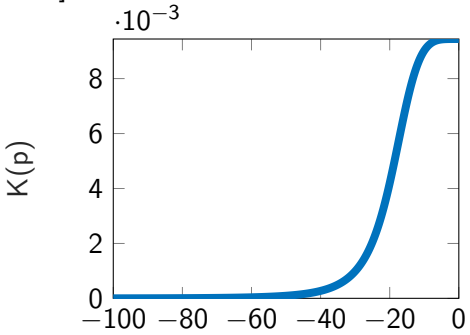
- q is the volumetric water flux, using Darcy's law it is written as

$$q = -K(p)\left(\nabla p + c\hat{z}\right),$$

  - $K(p)$ the hydraulic conductivity,
  - c the cosine of the angle between the downward z-axis $\hat{z}$ and the direction of the gravity force

To complete the model we need equations for both $s(p)$ and $K(p)$, we use the Van Genuchten formulation [Celia et al. 1990; Van Genuchten, 1980]





Cell-centered finite difference discretization

We use a cell-centered finite difference **tensor mesh** on

- a parallelepiped discretized with $\mathbf{N} = (N_x, N_y, N_z)$ nodes,
- the cell centers $\{x_{i,j,k} = (ih_x, jh_y, kh_z)\}_{i,j,k=0}^{N-1}$, for $\mathbf{h} = (h_x, h_y, h_z) = (L_x, L_y, L_z)/(\mathbf{N} - 1)$;
- the relative interfaces located at midpoints between adjacent nodes;
- N_t uniform time steps, i.e., the grid $\{t_l = l\Delta\, t\}_{l=0}^{N_t-1}$ for $\Delta\, t = 1/(N_t - 1)$.

This gives the **non-linear equations**:

$$\begin{aligned}\mathbf{\Phi}(p_{i,j,k}^{(l)}) = &\frac{\rho\phi}{\Delta t}\left(s\left(p_{i,j,k}^{(l)}\right) - s\left(p_{i,j,k}^{(l-1)}\right)\right) + q_{i+1/2,j,k}^{(l)} - q_{i-1/2,j,k}^{(l)} + q_{i,j+1/2,k}^{(l)} \\ &- q_{i,j-1/2,k}^{(l)} + q_{i,j,k+1/2}^{(l)} - q_{i,j,k-1/2}^{(l)} + f_{i,j,k} \equiv 0, \\ &\text{for } i, j, k = 1, \ldots, \mathbf{N} - 2,\end{aligned}$$

with

$$q^{(l)}_{i+1/2,j,k} = -{}^{\mathrm{AV}}K^{(l)}_{i+1,i}\left(\frac{p^{(l)}_{i+1,j,k}-p^{(l)}_{i,j,k}}{h_x^2}\right), \qquad q^{(l)}_{i-1/2,j,k} = -{}^{\mathrm{AV}}K^{(l)}_{i-1,i}\left(\frac{p^{(l)}_{i,j,k}-p^{(l)}_{i-1,j,k}}{h_x^2}\right),$$

$$q^{(l)}_{i,j+1/2,k} = -{}^{\mathrm{AV}}K^{(l)}_{j+1,j}\left(\frac{p^{(l)}_{i,j+1,k}-p^{(l)}_{i,j,k}}{h_y^2}\right), \qquad q^{(l)}_{i,j-1/2,k} = -{}^{\mathrm{AV}}K^{(l)}_{j-1,j}\left(\frac{p^{(l)}_{i,j,k}-p^{(l)}_{i,j-1,k}}{h_y^2}\right),$$

$$q^{(l)}_{i,j,k+1/2} = -{}^{\mathrm{AV}}K^{(l)}_{k+1,k}\left(\frac{p^{(l)}_{i,j,k+1}-p^{(l)}_{i,j,k}}{h_z^2}\right) - \frac{K(p_{i,j,k+1})}{2h_z},$$

$$q^{(l)}_{i,j,k-1/2} = -{}^{\mathrm{AV}}K^{(l)}_{k-1,k}\left(\frac{p^{(l)}_{i,j,k}-p^{(l)}_{i,j,k-1}}{h_z^2}\right) - \frac{K(p_{i,j,k-1})}{2h_z},$$

The Newton method and the sequence of the Jacobians

- Newton step for the solution, at each time step, of the nonlinear systems,
- The Jacobian matrix $J = J_{\mathbf{\Phi}}$ can then be computed in closed form,
- At the core of the (distributed) parallel solution we perform the solution of the (right) preconditioned linear system

$$JM^{-1}(M\mathbf{d}_k) = -\mathbf{\Phi}(\mathbf{p}^{(k,l)}),$$

What did we do in `https://arxiv.org/abs/2112.05051`:

- Describe the asymptotic spectral properties of the sequence $\{J_{\mathbf{N}}\}_{\mathbf{N}}$,
- Analyze the impact of (some) of the different **choices for the interface mean**,
- Use this information to get a matrix sequence $\{M_{\mathbf{N}}\}_{\mathbf{N}}$ for preconditioning $\{J_{\mathbf{N}}\}_{\mathbf{N}}$,
- Approximate such a matrix sequence by a (parallel) AMG method to efficiently solve the systems.

The Newton method and the sequence of the Jacobians

- Newton step for the solution, at each time step, of the nonlinear systems,
- The Jacobian matrix $J = J_{\mathbf{\Phi}}$ can then be computed in closed form,
- At the core of the (distributed) parallel solution we perform the solution of the (right) preconditioned linear system

$$JM^{-1}(M\mathbf{d}_k) = -\mathbf{\Phi}(\mathbf{p}^{(k,l)}),$$

What did we do in `https://arxiv.org/abs/2112.05051`:

- Use this information to get a matrix sequence $\{M_{\mathbf{N}}\}_{\mathbf{N}}$ for preconditioning $\{J_{\mathbf{N}}\}_{\mathbf{N}}$,
- Approximate such a matrix sequence by a (parallel) AMG method to efficiently solve the systems.

We **focus** here on the **implementation aspects**, for the spectral analysis and the other mathematical information: `https://arxiv.org/abs/2112.05051`

The Theoretical sequence of preconditioners (cont'd)

- The theoretical analysis suggests that we can use the discretization of the diffusion operator to precondition. This is *somewhat natural*, see, e.g., [Jones & Woodward, 2001], **but** now we **have a theoretical underpinning** of why it works,
- The organization of the proof works for **different choices of the fluxes** at the interfaces,
- We use the **G**eneralized **L**ocally **T**oeplitz machinery to achieve the formal result; see the books/papers by [Serra & Garoni 2017], [Barbarino, Serra, Garoni 2020].

But

- We still need to find a way to apply $\{M_{\mathbf{N}}^{-1}\}_{\mathbf{N}}$ sequence. Even if the sequence is simpler.
- Use an Algebraic Multigrid Algorithm to generate a $\{\tilde{M}_{\mathbf{N}}^{-1}\}_{\mathbf{N}}$ sequence.

What do we ask to it?

Solve the preconditioned system:

$$J\tilde{M}^{-1}(\tilde{M}\mathbf{d}_k) = -\mathbf{\Phi}(\mathbf{p}^{(k,l)}),$$

with matrix $\tilde{M}^{-1} \approx J^{-1}$ (right preconditioner) such that:

Algorithmic scalability $\max_i \lambda_i(\tilde{M}^{-1}J) \approx 1$ being independent of $\mathbf{N}$,

Linear complexity the action of $\tilde{M}^{-1}$ costs as little as possible, the best being $\mathcal{O}(\mathbf{N})$ flops,

Implementation scalability in a massively parallel computer, $\tilde{M}^{-1}$ should be composed of local actions, performance should depend linearly on the number of processors employed.

Observe that by the GLT analysis, we know that $\max_i \lambda_i(M^{-1}J) \approx 1$, thus if our multigrid hierarchy is "good enough" we can achieve a "near enough" result with it.

D. Bertaccini, P. D'Ambra, F. Durastante, S. Filippone, Why diffusion-based preconditioning of Richards equation works: spectral analysis and computational experiments at very large scale, arXiv preprint (2022), arXiv:2112.05051.

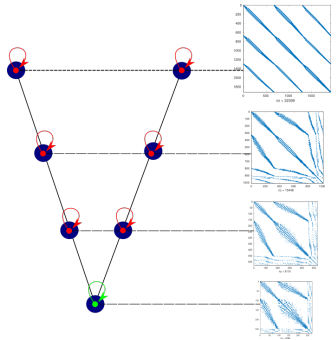
An Algebraic Multigrid Approximation of $\{M_{\mathbf{N}}^{-1}\}_{\mathbf{N}}$

Given Matrix $M_{\mathbf{N}} \in \mathbb{R}^{\mathbf{N}\times\mathbf{N}}$ SPD

Wanted Iterative method $\tilde{M}$ to precondition a Krylov iterative method:

- Hierarchy of systems
$$R_l \mathbf{x}_= \mathbf{b}_l, l = 0, \ldots, \text{nlev}$$
- Transfer operators:
$$P_{l+1}^{l} : \mathbb{R}^{n_{l+1}} \to \mathbb{R}^{n_l}$$

Missing Structural/geometric infos



Smoother

$$R_l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$$

"High frequencies"

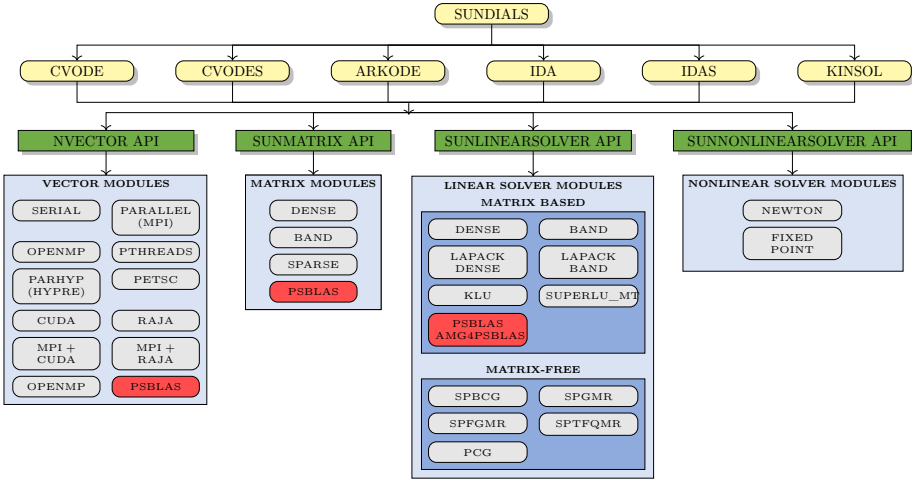
Prolongator

$$P_{l+1}^{l} : \mathbb{R}^{n_l} \to \mathbb{R}^{n_{l+1}}$$

"Low frequencies"

Complementarity of Smoother and Prolongator

To implement the Newton part of the Newton-Krylov solver we implemented an extension to the SUNDIALS KINSOL package.



- Wrapping of PSCToolkit *distributed sparse linear algebra* in KINSOL
  - NVECTOR: distributed vectors with all relevant operations (`axpy`, `norms`, `dot`, integrated actions for group of vectors, ...)
  - SUNMatrix: distributed matrix for all the formats in PSBLAS (`CSR`, `CSC`, `COO`, `HYB`, ...) and all the relevant operators (`spmv`, `matrix shift`, ...)
  - SUNLinSol: interface to *all* the Krylov **linear solvers** in PSBLAS (`CG`, `GMRES`, `BiCGStab`, ...) and all the **preconditioner** that can be used (or added in future) to AMG4PSBLAS (Algebraic Multigrid with different aggregation strategies, Domain Decomposition techniques)

- Wrapping of PSCToolkit *distributed sparse linear algebra* in KINSOL
  - NVECTOR: distributed vectors with all relevant operations (`axpy`, `norms`, `dot`, integrated actions for group of vectors, ...)
  - SUNMatrix: distributed matrix for all the formats in PSBLAS (`CSR`, `CSC`, `COO`, `HYB`, ...) and all the relevant operators (`spmv`, `matrix shift`, ...)
  - SUNLinSol: interface to *all* the Krylov **linear solvers** in PSBLAS (`CG`, `GMRES`, `BiCGStab`, ...) and all the **preconditioner** that can be used (or added in future) to AMG4PSBLAS (Algebraic Multigrid with different aggregation strategies, Domain Decomposition techniques)
- (PSCToolkit) $\Rightarrow$ KINSOL $\Rightarrow$ ParFlow

From KINSOL to ParFlow

- Wrapping of PSCToolkit *distributed sparse linear algebra* in KINSOL
  - NVECTOR: distributed vectors with all relevant operations (`axpy`, `norms`, `dot`, integrated actions for group of vectors, ...)
  - SUNMatrix: distributed matrix for all the formats in PSBLAS (`CSR`, `CSC`, `COO`, `HYB`, ...) and all the relevant operators (`spmv`, `matrix shift`, ...)
  - SUNLinSol: interface to *all* the Krylov **linear solvers** in PSBLAS (`CG`, `GMRES`, `BiCGStab`, ...) and all the **preconditioner** that can be used (or added in future) to AMG4PSBLAS (Algebraic Multigrid with different aggregation strategies, Domain Decomposition techniques)
- (PSCToolkit) $\Rightarrow$ KINSOL $\Rightarrow$ ParFlow
- KINSOL is used in many codes as the supplier of both linear and nonlinear solvers, this first integration is portable for other problems.

Problem and Machine

- Parallelepipedal domain Ω of size $[0, L_x] \times [0, L_y] \times [0, L]$,
- Water at height $z = L$ such that the pressure head becomes zero in a square region at the center of the top layer

$$p(x, y, L, t) = \frac{1}{\alpha} \ln \left[\exp(\alpha h_r) + (1 - \exp(\alpha h_r)) \chi_{[\frac{a}{4}, \frac{3a}{4}] \times [\frac{b}{4}, \frac{3b}{4}]}(x, y, z) \right],$$

- Initial condition is given by $p(x, y, z, 0) = h_r$,
- In all cases we run the simulation for $t \in [0, 2]$ and $N_t = 10$.

Marconi 100

(21th in 06/2022 TOP500)

IBM Power System AC922 nodes

2×16 *IBM POWER93* 3.1 GHz,

256 GB of RAM.

Dual-rail *Mellanox EDR*

Infiniband network by *IBM* 220/300 GB/s.

Preconditioners

	Multigrid		**One-Level**	
Cycle	1 sweep of V-cyle		Additive Schwarz	Type
Aggregation	Parallel **D**ecoupled smoothed aggregation [Vaněk. Mandel, Brezina, 1996]	Parallel **Coupled** smoothed aggregation based on graph matching aggregate size: 8 [D'Ambra, Filippone, Vassilevski, 2018]	1 layer of mesh points in each grid direction	Overlap
Pre/post-smoother	1 iteration of hybrid backward/forward Gauss-Seidel		ILU(0)	Local solver
Coarsest solver	preconditioned CG method with ILU(1)-block-Jacobi preconditioner			
Label	VDSVMB	VSMATCH	AS	Label

Preconditioners

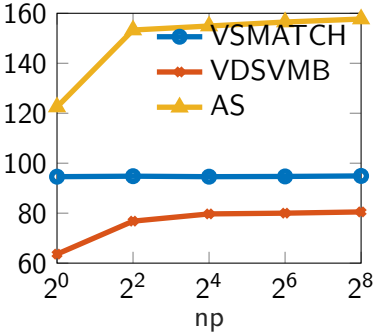
	Multigrid		**One-Level**	
Cycle	1 sweep of V-cyle		Additive Schwarz	Type
Aggregation	Parallel **Decoupled** smoothed aggregation [Vaněk. Mandel, Brezina, 1996]	Parallel **Coupled** smoothed aggregation based on graph matching aggregate size: 8 [D'Ambra, Filippone, Vassilevski, 2018]	1 layer of mesh points in each grid direction	Overlap
Pre/post-smoother	1 iteration of hybrid backward/forward Gauss-Seidel		ILU(0)	Local solver
Coarsest solver	preconditioned `CG` method with ILU(1)-block-Jacobi preconditioner			
Label	VDSVMB	VSMATCH	AS	Label

Preconditioners

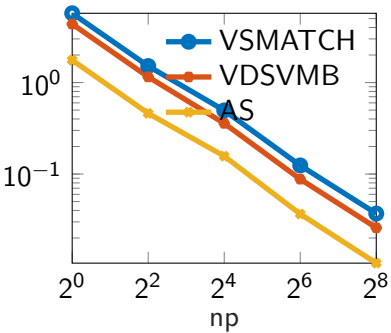
	Multigrid		**One-Level**	
Cycle	1 sweep of V-cyle		Additive Schwarz	Type
Aggregation	Parallel **Decoupled** smoothed aggregation [Vaněk. Mandel, Brezina, 1996]	Parallel **Coupled** smoothed aggregation based on graph matching aggregate size: 8 [D'Ambra, Filippone, Vassilevski, 2018]	1 layer of mesh points in each grid direction	Overlap
Pre/post-smoother	1 iteration of hybrid backward/forward Gauss-Seidel		ILU(0)	Local solver
Coarsest solver	preconditioned `CG` method with ILU(1)-block-Jacobi preconditioner			
Label	VDSVMB	VSMATCH	AS	Label

Strong Scalability Analysis

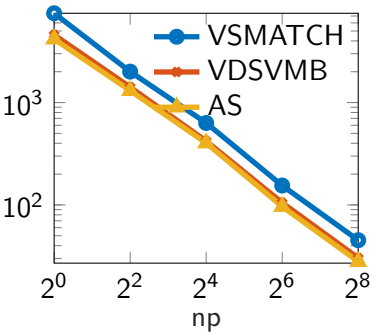
- Parallelepiped $[0,64] \times [0,64] \times [0,1]$, discretized with $N_x = N_y = 800$, and $N_z = 40 \Rightarrow$ 20 millions of dofs,
- Computational cores from 1 to 256, i.e., $np = 4^p$, $p = 0, \ldots, 4$,



Average number of linear iterations



Average time per linear iteration T (s)



Total solution time T (s)

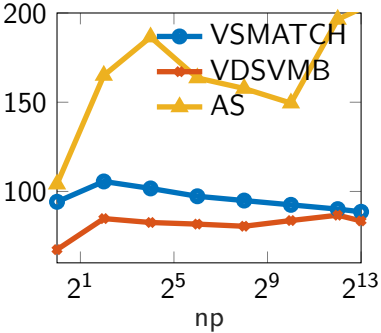
Strong Scalability Analysis

	VDSVBM		VSMATCH		AS	
np	N Jac.s	NLin It.s	N Jac.s	NLin It.s	N Jac.s	NLin It.s
1	3	36	3	38	3	43
4	3	37	3	38	4	39
16	3	37	3	38	4	39
64	3	37	3	38	4	39
256	3	37	3	38	4	39

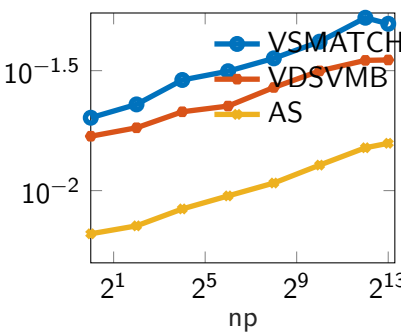
Number of **nonlinear iterations** (NLin It.s), and number of **computed Jacobians** (N Jac.s) for the three preconditioners.

Weak scalability analysis

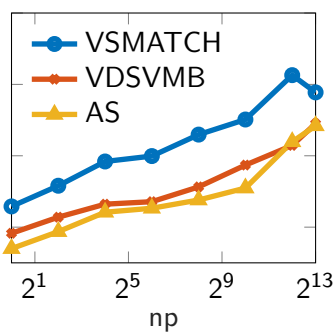
- $N_x = N_y = 50$, and $N_z = 40$, $\Omega(np) = [0, 2^p \times 4.0] \times [0, 2^q \times 4.0] \times [0, 1.0]$
- $np = p \times q$ processes, $p = 0, \ldots, 7$, $q = 0, \ldots, 6$, and a corresponding mesh $N(p \times q) = (2^p N_x, 2^q N_y, N_z) \Rightarrow$ 820 millions of dofs.



Average number of linear iterations



Average time per linear iteration T (s)



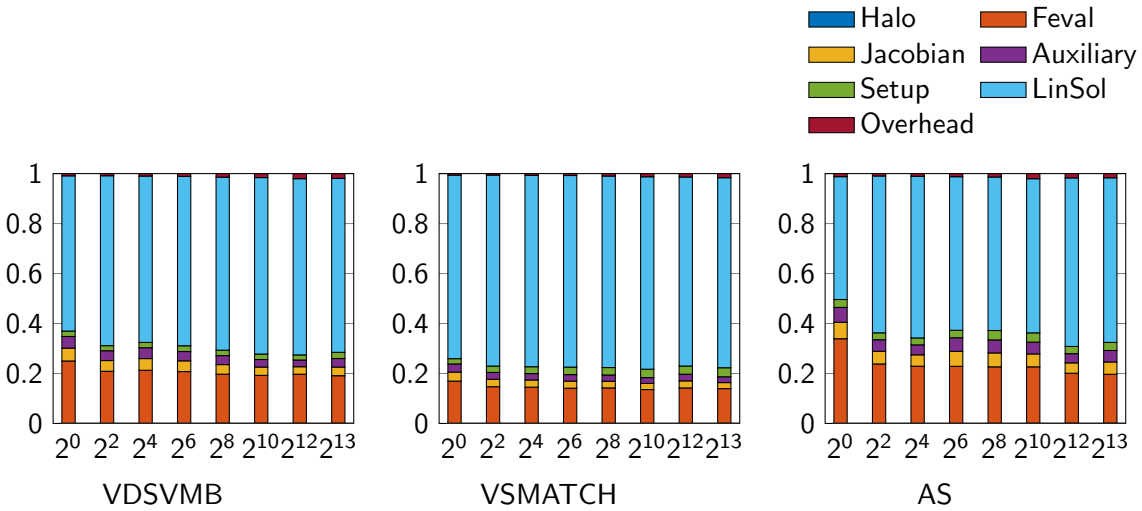
Total solution time T (s)

Weak scalability analysis

	VDSVBM		VSMATCH		AS	
np	N Jac.s	NLin It.s	N Jac.s	NLin It.s	N Jac.s	NLin It.s
1	3	37	3	36	3	40
4	3	38	3	38	3	36
16	3	38	3	38	3	40
64	3	37	3	38	4	37
256	3	37	3	38	4	39
1024	3	39	3	38	4	41
4096	3	41	3	38	4	47
8192	3	40	3	38	4	48

Number of **nonlinear iterations** (NLin It.s), and number of **computed Jacobians** (N Jac.s) for the three preconditioners.

Weak scalability analysis - Time Fractions



Conclusions and future perspectives

We focused on **two main objectives**

- ✔ prove some asymptotic spectral properties of the sequence of Jacobian matrices generated discretizing the Richards equation;
- ✔ prove the efficiency, flexibility and robustness of a software framework for parallel sparse matrix computations.

Our plans for the future

- 🔧 extension of the PSCToolkit interface to KINSOL, in order to use the ability of the PSCToolkit linear solvers in exploiting GPU architectures;
- 🔧 integration of the software stack into the ParFlow code for realistic simulations in hydrological applications.

Conclusions and Future Directions

We have proved

- ✔ Aggregation procedure with certified quality,
- ✔ Scalability results on tens of thousands of cores,
- ✔ Comparable results with state of the art libraries,
- ✔ Interfacing with large scale scientific applications,
- ✔ Multi-GPU support.

Algorithmic and software extensions to AMG4PSBLAS (future work)

- ⚙ Multi-objective matching to increase coarsening ratio,
  - 👥 Collaboration with Pacific Northwest National Laboratory (Richland, WA), and Purdue University (IN)
- ⚙ Process remapping for coarse grid solutions,
  - 👥 Collaboration with Centre national de la recherche scientifique (Toulouse)
- ⚙ GPU data and preconditioner setup improvements,
- ⚙ Communication avoiding Krylov methods,
- ⚙ Mixed-precision arithmetic.

Essential bibliography

- Multigrid based on matching
  - P. D'Ambra and P. S. Vassilevski, Adaptive AMG with coarsening based on compatible weighted matching, Comput. Vis. Sci. **16** (2013), no. 2, 59–76.
  - P. D'Ambra, S. Filippone and P. S. Vassilevski, BootCMatch: a software package for bootstrap AMG based on graph weighted matching, ACM Trans. Math. Software **44** (2018), no. 4, Art. 39, 25 pp.
  - M. Bernaschi, P. D'Ambra and D. Pasquini, AMG based on compatible weighted matching for GPUs, Parallel Comput. **92** (2020), 102599, 13 pp.
  - P. D'Ambra, F. Durastante, S. Filippone and L. Zikatanov, Automatic coarsening in Algebraic Multigrid utilizing quality measures for matching-based aggregations. arXiv preprint (2022), `arXiv:2001.09969`.
  - D. Bertaccini, P. D'Ambra, F. Durastante, S. Filippone, Why diffusion-based preconditioning of Richards equation works: spectral analysis and computational experiments at very large scale, arXiv preprint (2022), `arXiv:2112.05051`.
- Scalability results
  - P. D'Ambra, F. Durastante, and S. Filippone. "AMG preconditioners for linear solvers towards extreme scale." SIAM J. Sci. Comp. 43.5 (2021): S679-S703.
- PSBLAS
  - S. Filippone and A. Buttari, Object-oriented techniques for sparse matrix computations in Fortran 2003. ACM Trans. Math. Software **38** (2012), no. 4, 1–20 pp.
  - S. Filippone et al., Sparse matrix-vector multiplication on GPGPUs, ACM Trans. Math. Software **43** (2017), no. 4, Art. 30, 49 pp.

Thank You!

Convergence Theorem (D'Ambra, Durastante, Filippone, Zikatanov)

The exact TL–AMG with convergent smoother M, and prolongator P based on the maximum weight matching applied on a SPD matrix A has a convergence rate of

$$\|I - B^{-1}A\|_A \leq 1 - \frac{\mu_c}{c^D}, \text{ for } \mu_c = \min_{1\leq j\leq J} \mu_j(V_j^c) = \min_{1\leq j\leq J} \left[\max_{\mathbf{v}_j \in V_j} \min_{\mathbf{v}_j^c \in V_j^c} \frac{\|\mathbf{v}_j - \mathbf{v}_j^c\|_{D_j}^2}{\|\mathbf{v}_j\|_{A_j}^2} \right].$$

and c^D the continuity constant of the smoother. Moreover, the $\mu_j^{-1}(V_j^c)$ are such that

$$\lambda_2^{-1}(D_j^{-1}A_j) \leq \mu_j^{-1}(V_j^c) \leq \lambda_1^{-1}(D_j^{-1}A_j).$$

Furthermore, if either $(\mathbf{w}_{e_{i\to j}}, \lambda_1(D_j^{-1}A_j)$, or $(\mathbf{w}_{e_{i\to j}}^{\perp}, \lambda_2(D_j^{-1}A_j)$ are eigencouples of $D_j^{-1}A_j$, then

$$\mu_j^{-1}(V_j^c) = \lambda_2^{-1}(D_j^{-1}A_j)$$

- The local constants $\mu_j^{-1}(V_j^c)$ are then a quality measure for the single aggregates

Fixing the parameters

We can fix the weight vector $\mathbf{w}$, and evaluate the performance of the matching algorithms.

Theorem (Optimal prolongator)

Let $\{\lambda_j, \mathbf{\Phi}_j\}_{j=1}^{n}$ be the eigenpairs of $\overline{T} = \overline{M}A$ for the symmetrized smoother $\overline{M}$. Let us also assume that $\mathbf{\Phi}_j$ are orthogonal w.r.t. $(\cdot,\cdot)_{\overline{M}^{-1}}$. The convergence rate $\|E(P)\|_A$ is minimal for P such that

$$\text{Range}(P) = \text{Range}(P^{opt}),$$

where $P^{opt} = \{\mathbf{\Phi}_1, \ldots, \mathbf{\Phi}_{n_c}\}$. In this case,

$$\|E\|_A^2 = 1 - \lambda_{n_c+1}$$

For our choice of P we know that:

- There exists $\mathbf{h} \in \mathbb{R}^{n_c}$ such that $P\mathbf{h} = \mathbf{w}$

⚠ A good candidate can be obtained by exploiting the symmetrized smoother $\overline{M}$ to select as a weight vector an ε–smooth algebraic vector, i.e., for a given $\epsilon \in (0,1)$, $\mathbf{v}$ an algebraically ϵ-smooth with respect to A if

$$\|\mathbf{v}\|_A^2 \leq \epsilon \|\mathbf{v}\|_{\overline{M}^{-1}}^2.$$