(Sparse) Linear Algebra at the Extreme Scales

Fabio Durastante

Università di Pisa – December 1, 2020

Consiglio Nazionale delle Ricerche
Istituto per le Applicazioni del Calcolo “M. Picone”
f.durastante@na.iac.cnr.it

Collaborators and Funding

Joint work with

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

Salvatore Filippone,
Università degli Studi di Roma "Tor Vergata"
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Consiglio Nazionale delle Ricerche
Istituto per le Applicazioni del Calcolo "M. Picone"

Funded by

EoCoE
Project ID: 824158
European Commission
Horizon 2020
European Union funding
for Research & Innovation

Solving Large Linear Systems

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$,

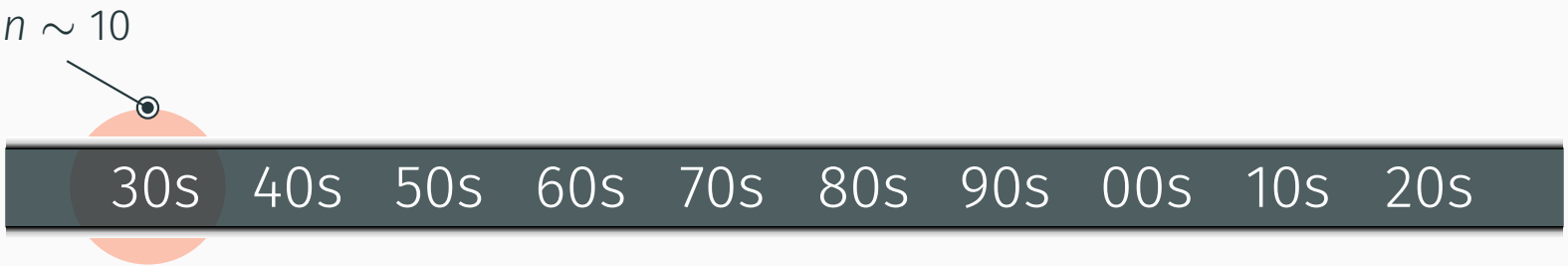
is often the most time consuming computational kernel in many areas of computational science and engineering problems.

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



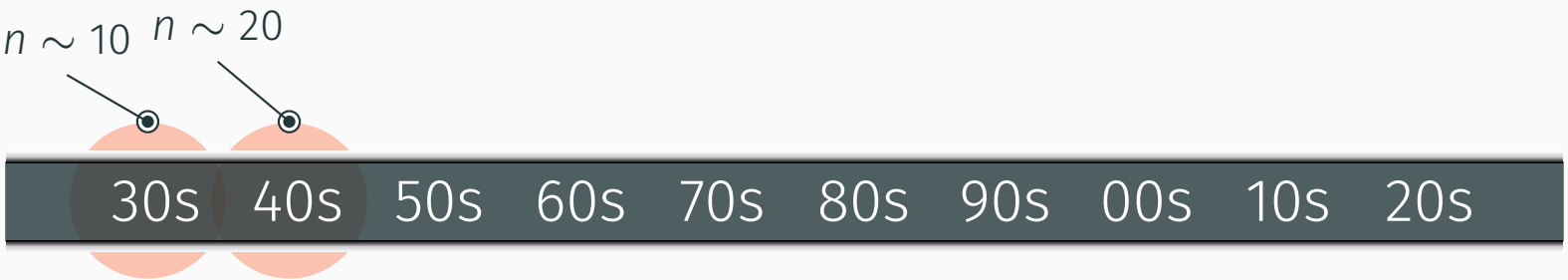
"In a ground wire problem involving a large number of ground conductors, 13 simultaneous equations were solved..." – Dwight (1930)"
"The second machine, now in operation, was designed for the direct solution of nine or fewer simultaneous equations." – Wilbur, J. B. (1936)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



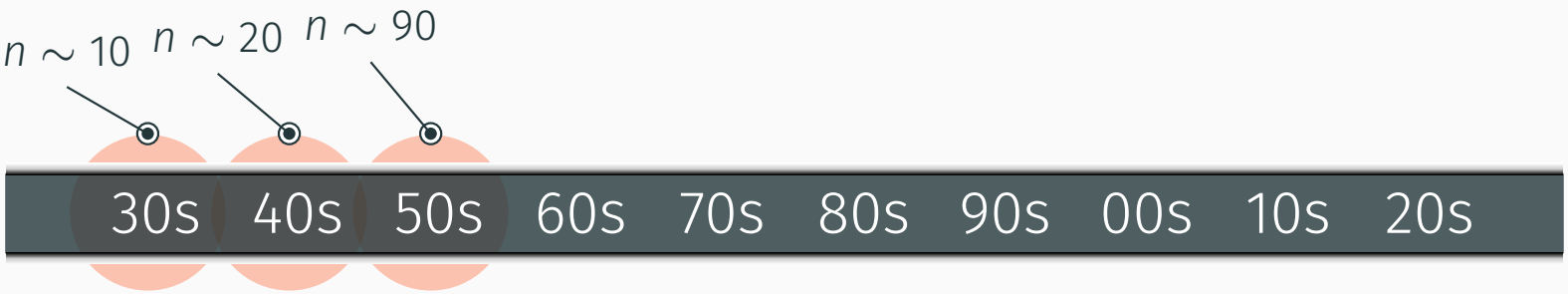
"Finally, though the labour of relaxation in three dimensions is prohibitively great, the future use of the new electronic calculating machines in this connexion is a distinct possibility" – Fox, L. (1947)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



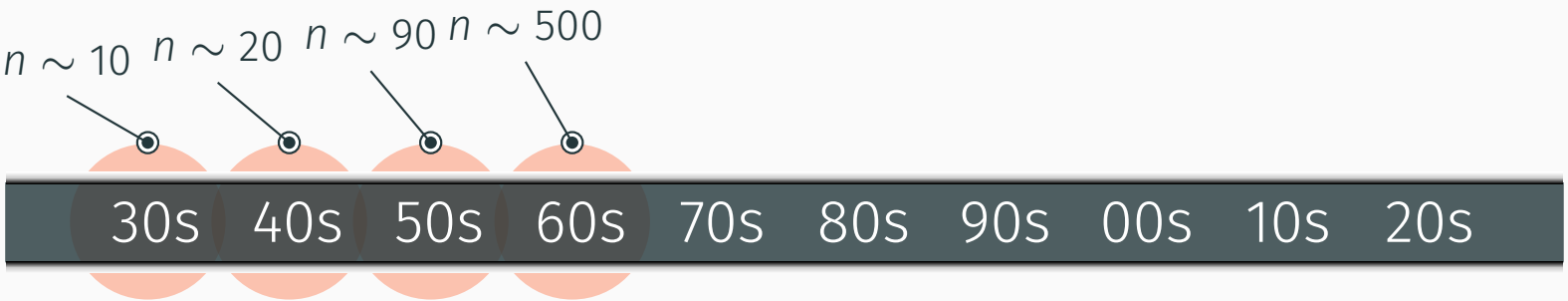
"The Ferranti PEGASUS computer, with a main store of 4096 words, can solve a maximum of 86 simultaneous equations by its standard subroutine and takes about 45 minutes to complete this calculation." – Wilson, L. B. (1959)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



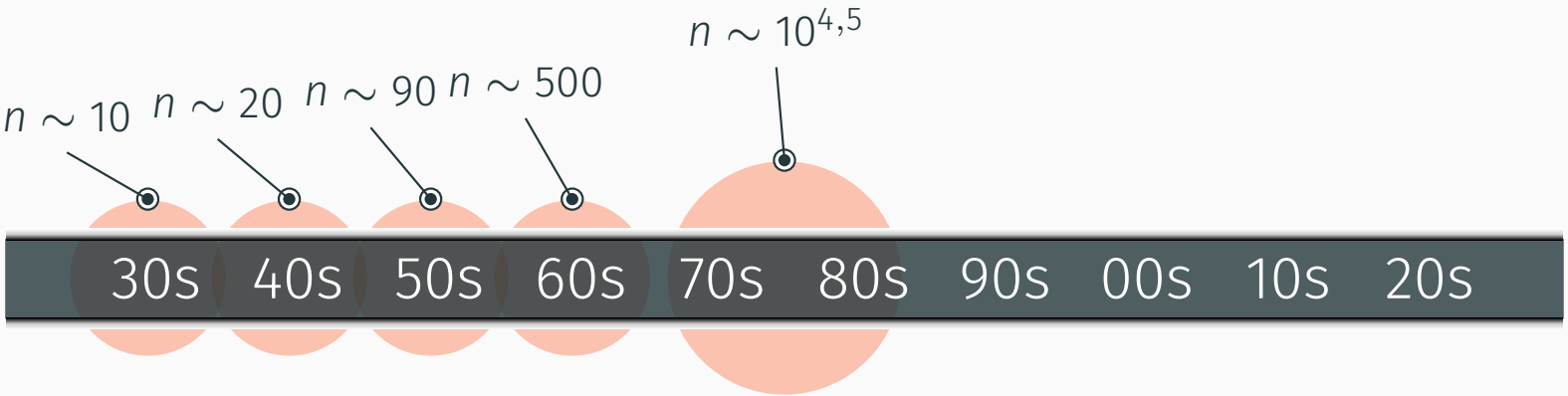
"...the bound imposed by this is $m + n \leq 474$. In addition, this number of equations would fill one standard (1.800ft) reel of magnetic tape, and the fifty-odd hours taken in the calculation might be thought excessive." – Barron, Swinnerton-Dyer (1960)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



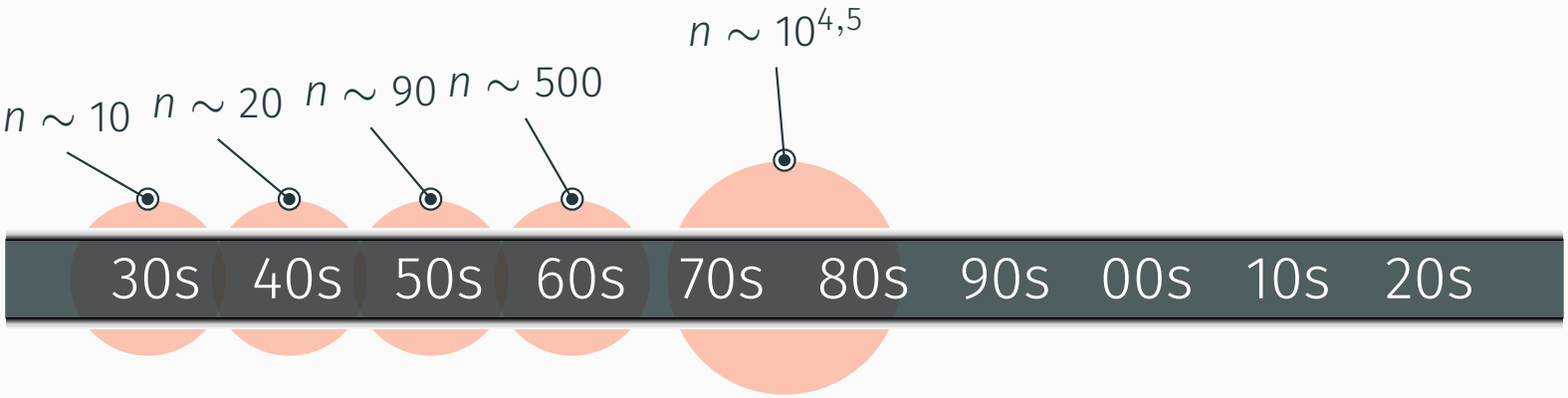
"...handling problems involving sets of simultaneous equations of two-thousandth order, and SAMIS available through "Cosmic" at the University of Georgia, which can treat up to 10,000 simultaneous equations."
– Melosh, Schmele (1969)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



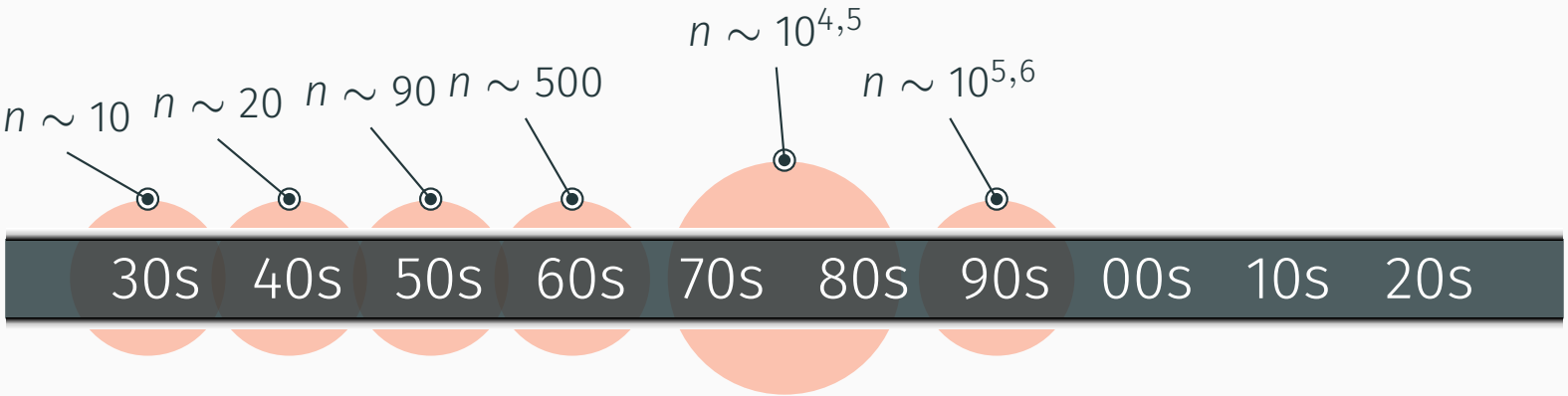
"The mini-computer cost algorithm is applied to the same complex shell problem used previously, with 9120 degrees of freedom [...]. The running times, however, are 40 and 70 hr, respectively! It would appear that improvement of mini-computer speeds is required..." – Kamel, McCabe (1978)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



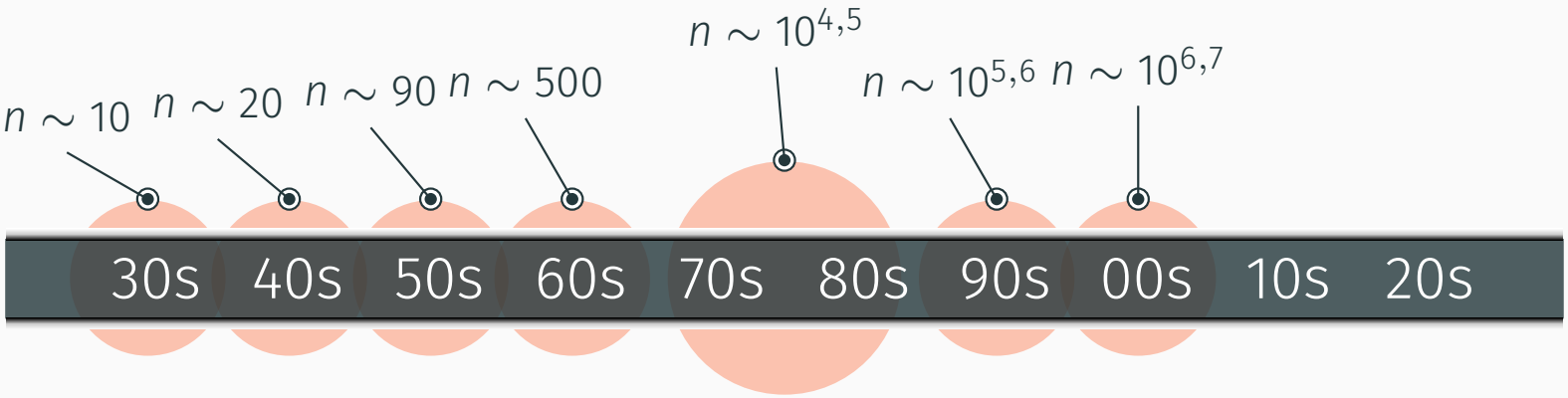
"For instance, Pomerell in 1994 reports on successful application of preconditioned Krylov methods for very ill-conditioned unstructured finite element systems of order up to 210,000 that arise in semiconductor device modeling." – Saad Y., van der Vorst, H.A. (2000)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



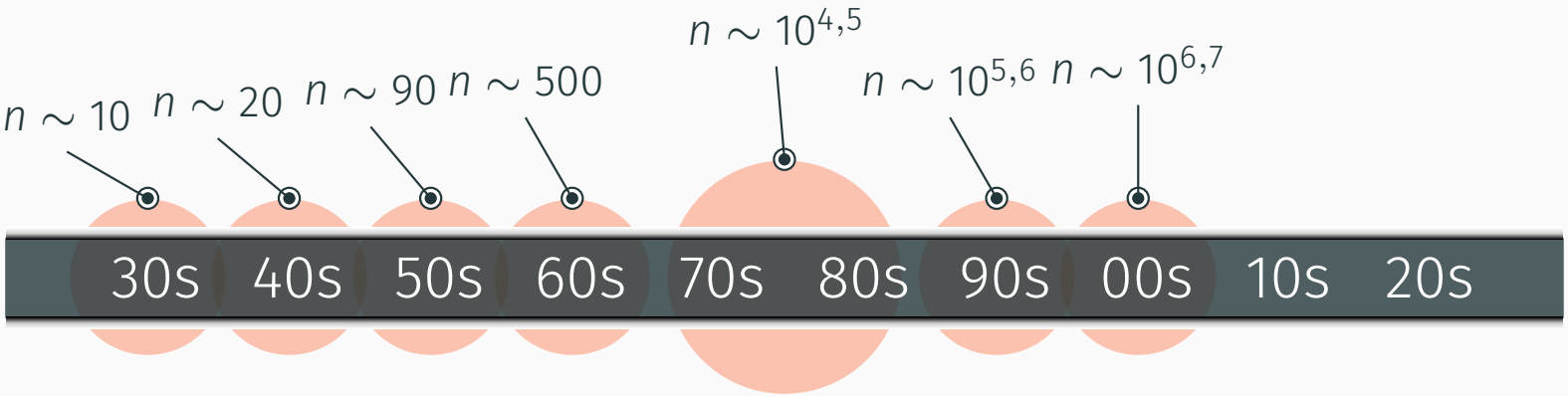
"As a second example, we show results (Table VIII) for a problem arising in ocean modeling (barotropic equation) with $n = 370,000$ unknowns and approximately 3.3 million nonzero entries..." – Benzi, M. (2002)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



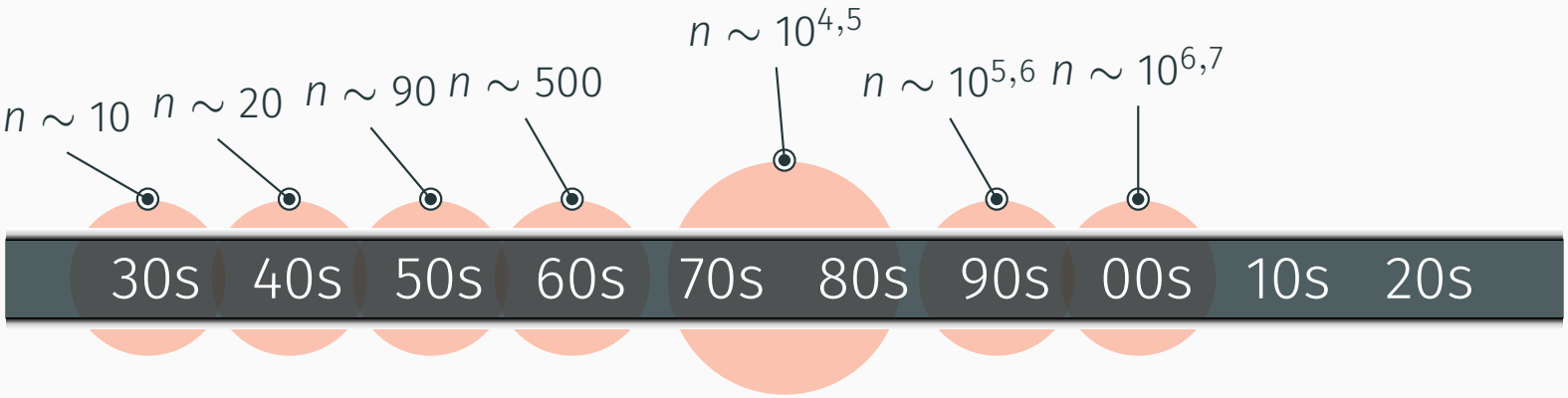
"Problem: Large, mesh size: $180 \times 60 \times 30$, ♯ unknowns (in simulation): 1,010,160, Solution time 45.7 h" – Wang, de Sturler, Paulino (2006)

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 $\mathbf{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



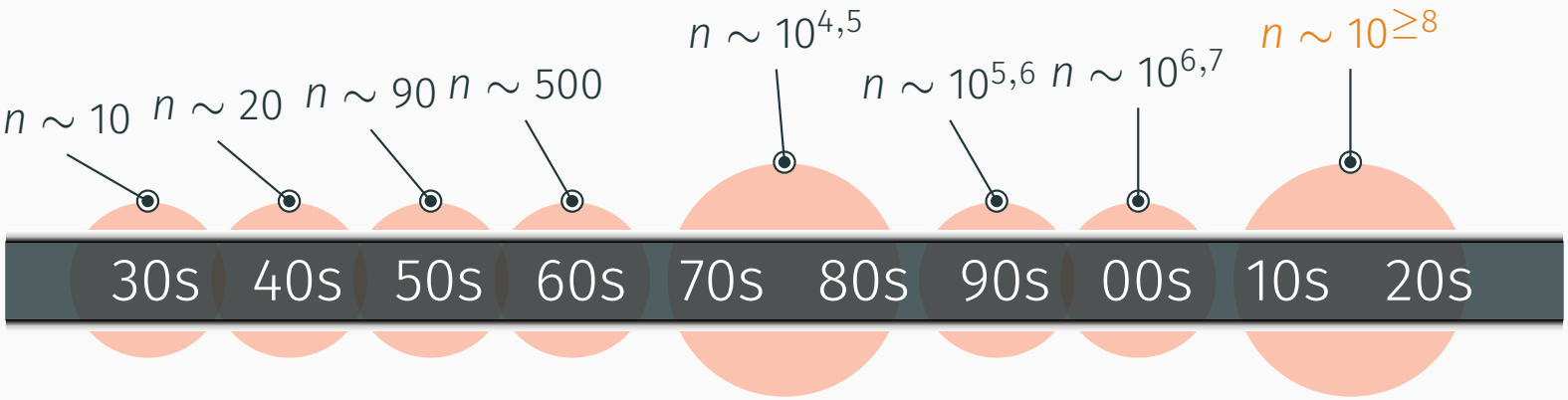
"The parallel GMRES was tested on the Tesla T10P GPU using a set of matrix data from the oil field simulation data of Conoco Phillips. The order of the system ranges from $\sim$ 2000 to $\sim$ 1.1 million." – M. Wang, H. Klie, M. Parashar, H. Sudan (2009)

What we want to solve

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

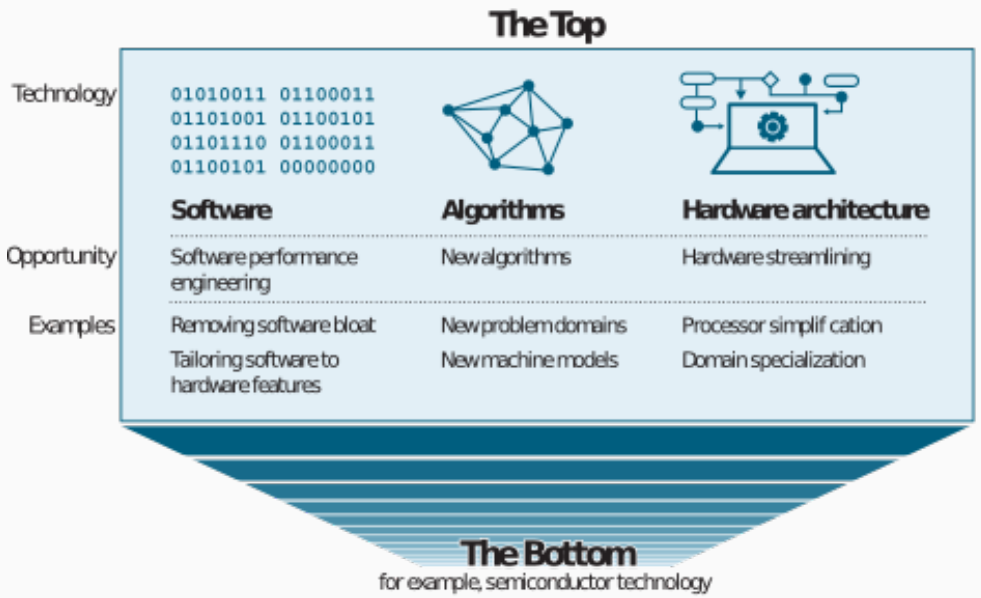
where

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



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

The philosophy behind the effort



C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the Top: What will drive computer performance after Moore's law?", *Science* (2020)

"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."

Target Applications

Wind Models

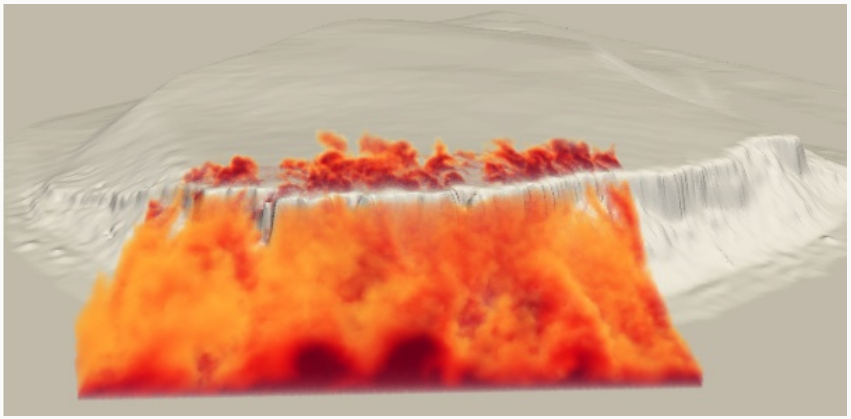
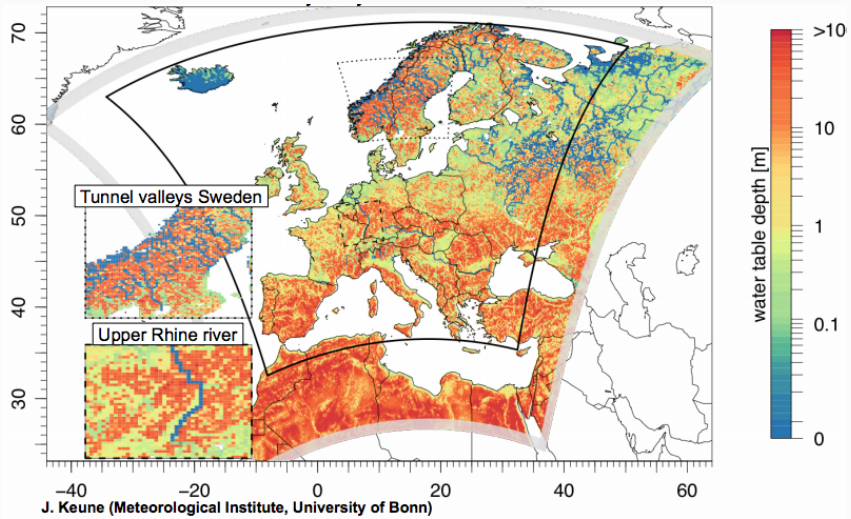


Image credits H. Owen and G. Marin,
Barcelona Supercomputing Centre

DoFs: $n \sim 10^{10}$

Regional Hydrological Models



Processors: $np \sim 10^6$

Where we want to solve it[1]

	System	Cores	Rmax (TFlops/s)
1	Fugaku	7,630,848	442,010.0
2	Summit	2,414,592	148,600.0
3	Sierra	1,572,480	94,640.0
⋮	⋮	⋮	⋮
11	Marconi-100	347,776	21,640.0
12	Piz Daint	387,872	21,230.0
⋮	⋮	⋮	⋮
42	MareNostrum	153,216	6,470.8

MareNostrum IV - BSC

Piz Daint - CSCS

- Machines with hundreds of MPI cores,
- Hybrid form of parallelism: MPI, OpenMP, CUDA/OpenCL, ...

[1] TOP500 list, November 2020 – `https://www.top500.org`

Where we want to solve it[1]

	System	Cores	Rmax (TFlops/s)
1	Fugaku	7,630,848	442,010.0
2	Summit	2,414,592	148,600.0
3	Sierra	1,572,480	94,640.0
⋮	⋮	⋮	⋮
11	Marconi-100	347,776	21,640.0
12	Piz Daint	387,872	21,230.0
⋮	⋮	⋮	⋮
42	MareNostrum	153,216	6,470.8

MareNostrum IV - BSC

Piz Daint - CSCS

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

[1] TOP500 list, November 2020 – `https://www.top500.org`

Parallel AMG Algorithms

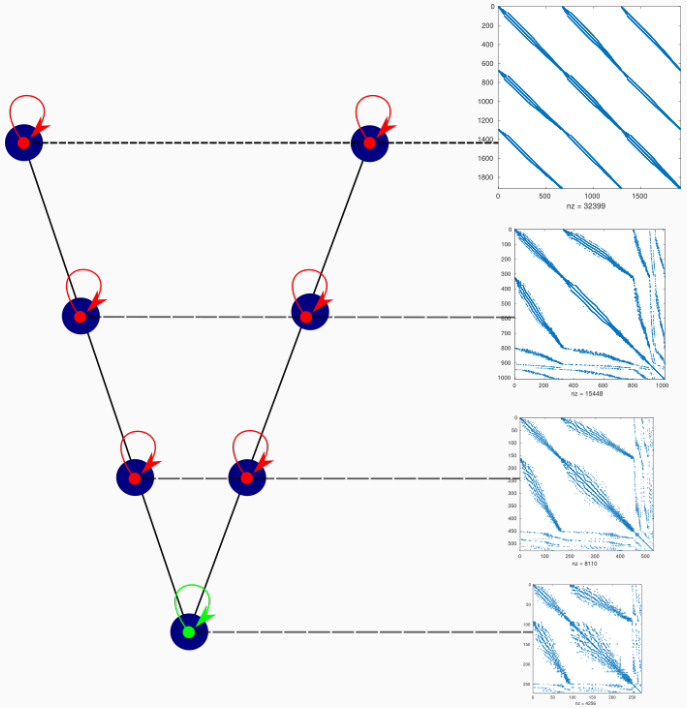
Algebraic Multigrid Algorithms

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

Missing Structural/geometric infos



Smoother

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

"High frequencies"

Prolongator

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

"Low frequencies"

Complementarity of Smoother and Prolongator

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 a standard iterative solver with good parallel properties:

What is our *recipe*?

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

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

What is our *recipe*?

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

  GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \mathbf{diag}(A)$ and $L = \mathbf{tril}(A)$ 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 a standard iterative solver with good parallel properties:

  GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \mathbf{diag}(A)$ and $L = \mathbf{tril}(A)$ 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 $D_{pp} = \mathbf{diag}(A_{pp})$ and $L_{pp} = \mathbf{trilu}(A_{pp})$ and select:

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

What is our *recipe*?

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

  GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \mathbf{diag}(A)$ and $L = \mathbf{tril}(A)$ 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 $D_{pp} = \mathbf{diag}(A_{pp})$ and $L_{pp} = \mathbf{trilu}(A_{pp})$ and select:

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

  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 $\quad \text{Range}(P_f)$: 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

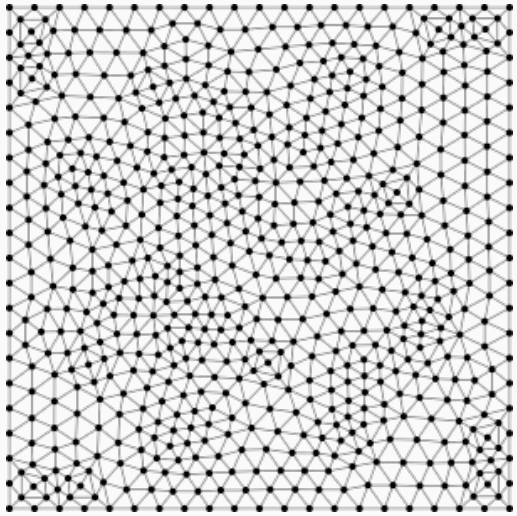
Sufficient condition for efficient coarsening

$A_f = P_f^T A P_f$ 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 $\hat{A}$:

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



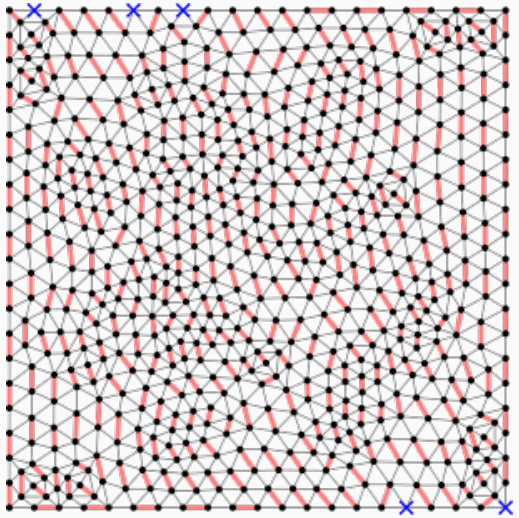
- a *matching* $\mathcal{M}$ is a set of pairwise non-adjacent edges, containing no loops;
- a maximum product matching if it maximizes the product of the weights of the edges $e_{i \mapsto j}$ in it.

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*:

$$P = \underbrace{\begin{bmatrix} \begin{bmatrix} \mathbf{w}_{e_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathbf{w}_{e_{n_p}} \end{bmatrix}}_{n_p} 2n_p & 0 \\ 0 & \begin{bmatrix} w_1/|w_1| & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & w_{n_s}/|w_{n_s}| \end{bmatrix} n_s \end{bmatrix} n = 2n_p + n_s}_{n_c = n_p + n_s = J}$$

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

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

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 $l = 0, \ldots, nl - 1$.

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,

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

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.

Parallel Matching Algorithms

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 intrinsically 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)

---

1 **Algorithm:** Locally Dominant Edge

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

2 $\mathcal{M} \leftarrow \emptyset$;

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

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

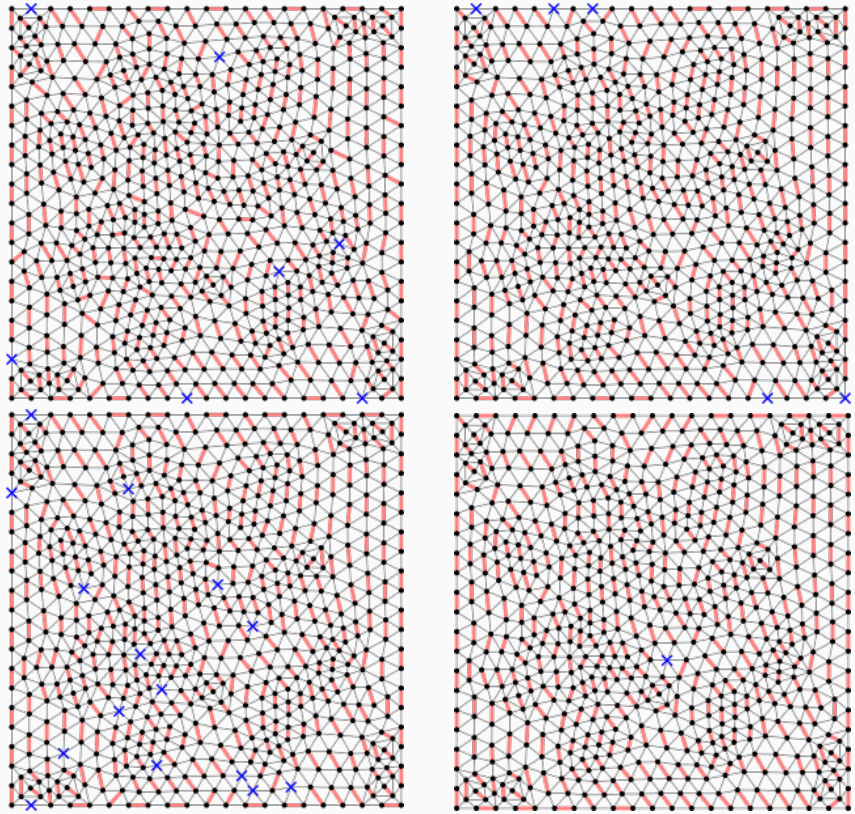
5 Remove all edges incident to i and j from $\mathcal{E}$;

6 **end**

Output: Matching $\mathcal{M}$

Parallel Matching Algorithms

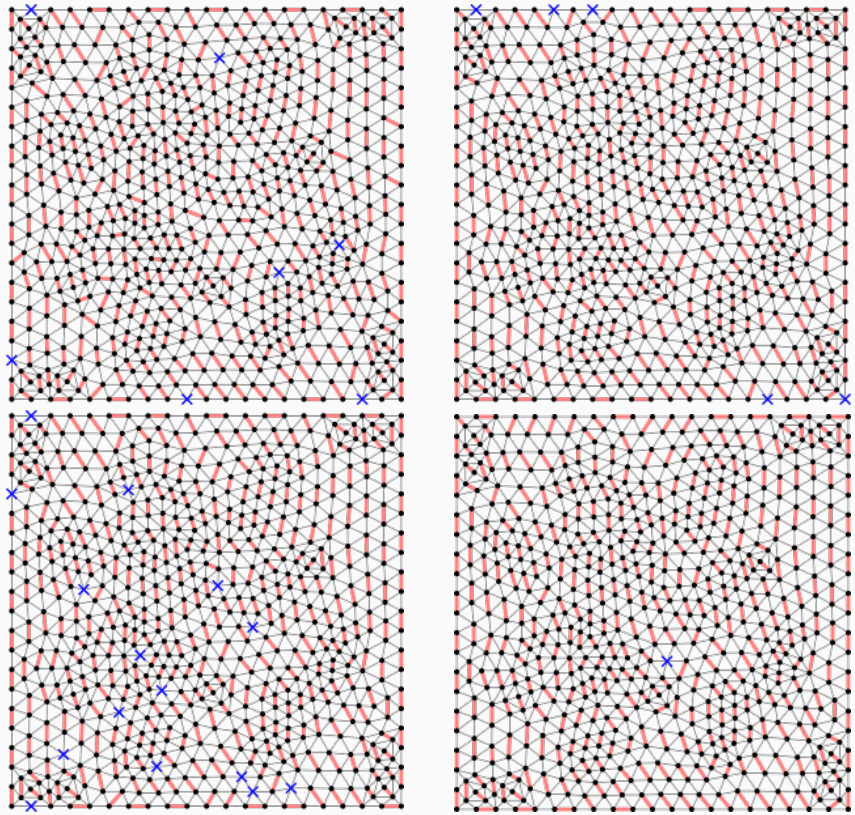
1. What is the best matching algorithm from the computational point of view?
2. How can we evaluate the quality (in term of the AMG algorithm) of the resulting matching?



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.



Quality and Convergence: *a posteriori* analysis

Convergence Theorem (D'Ambra, D., Filippone)

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

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

Quality and Convergence: *a posteriori* analysis

Convergence Theorem (D'Ambra, D., Filippone)

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

Quality and Convergence: *a posteriori* analysis

Convergence Theorem (D'Ambra, D., Filippone)

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

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

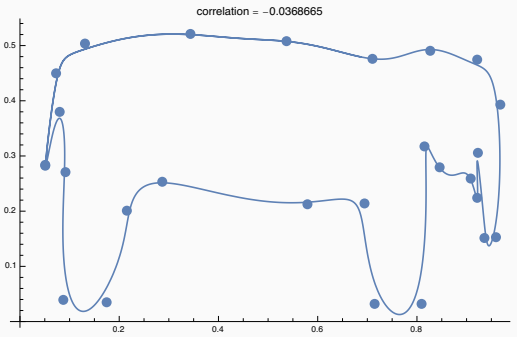
Furthermore, if either $(\mathbf{w}_{e_{i\to j}}, \lambda_1(D_j^{-1}A_j)$, or $(\mathbf{w}_{e_{i\to j}}^{\perp}, \lambda_2(D_j^{-1}A_j)$ are eigencouples of $D_j^{-1}A_j$, then

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

- 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 $\hat{M}$. 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

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

An Example of Quality Analysis

Test problems: 2D Laplace equation

The 2D Laplace equation with variable coefficients on the unit square $\Omega = [0,1]^2$, dicretized with 5–point finite differences, i.e. the equation

$$\begin{cases} -\nabla \cdot (a(x,y)\nabla u(x,y)) = f(x,y), & (x,y) \in \Omega, \\ u(x,y) = 0, & (x,y) \in \partial\Omega. \end{cases}$$

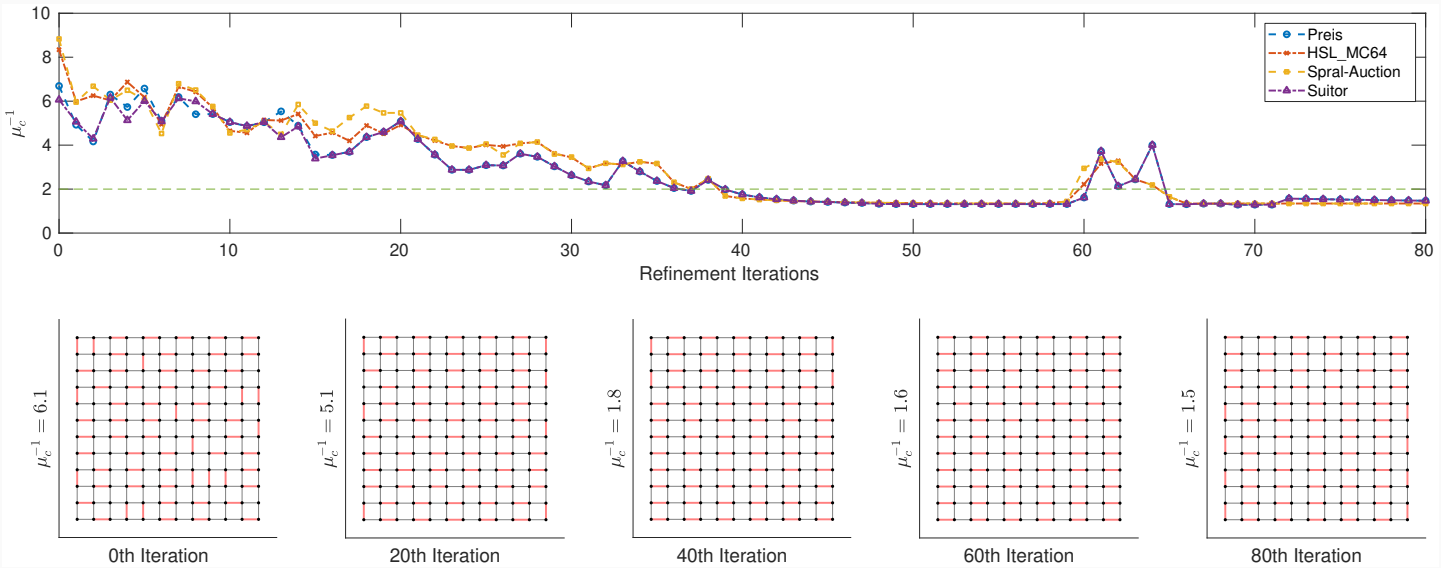
- We focus on a 2D example so that we can graphically represent the different aggregates,
- As choice of the coefficient $a(x,y)$ we take an axial oriented anistropy of modulus $\varepsilon = 100$ and angle $\theta = \pi/8$

An Example of Quality Analysis

Test problems: 2D Laplace equation

The 2D Laplace equation with variable coefficients on the unit square $\Omega = [0,1]^2$, dicretized with 5–point finite differences, i.e. the equation

$$\begin{cases} -\nabla \cdot (a(x,y)\nabla u(x,y)) = f(x,y), & (x,y) \in \Omega, \\ u(x,y) = 0, & (x,y) \in \partial\Omega. \end{cases}$$



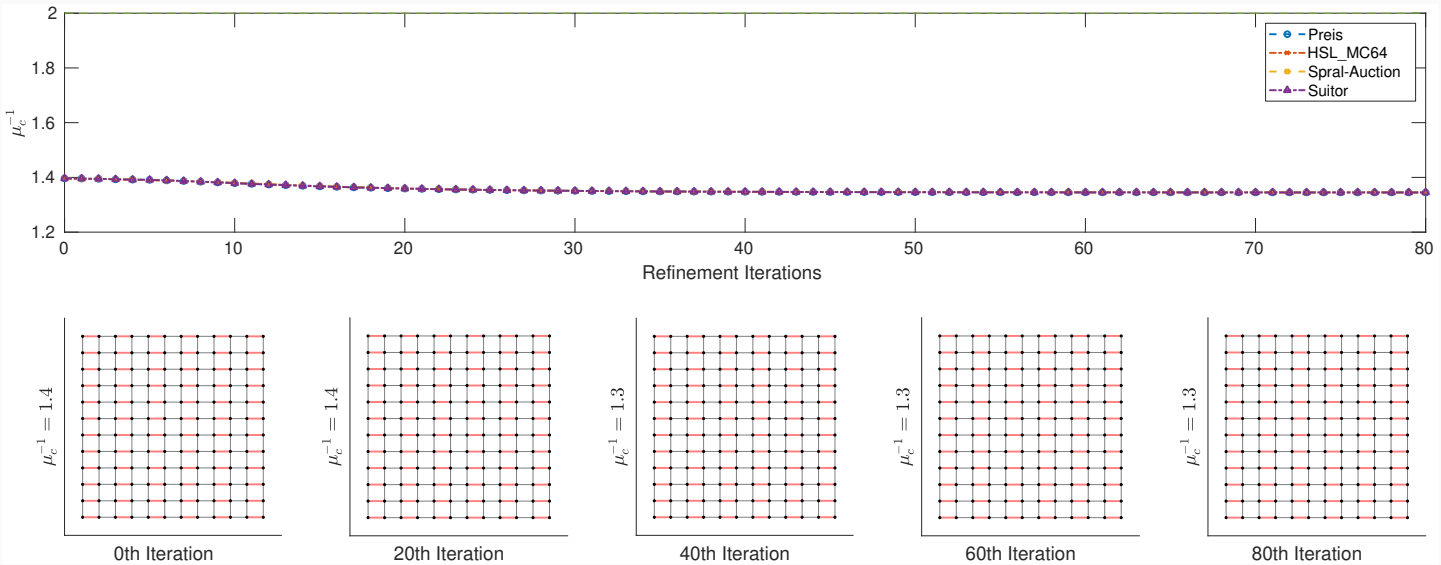
(a) Refinements from a random **w**.

An Example of Quality Analysis

Test problems: 2D Laplace equation

The 2D Laplace equation with variable coefficients on the unit square $\Omega = [0,1]^2$, dicretized with 5–point finite differences, i.e. the equation

$$\begin{cases} -\nabla \cdot (a(x,y)\nabla u(x,y)) = f(x,y), & (x,y) \in \Omega, \\ u(x,y) = 0, & (x,y) \in \partial\Omega. \end{cases}$$



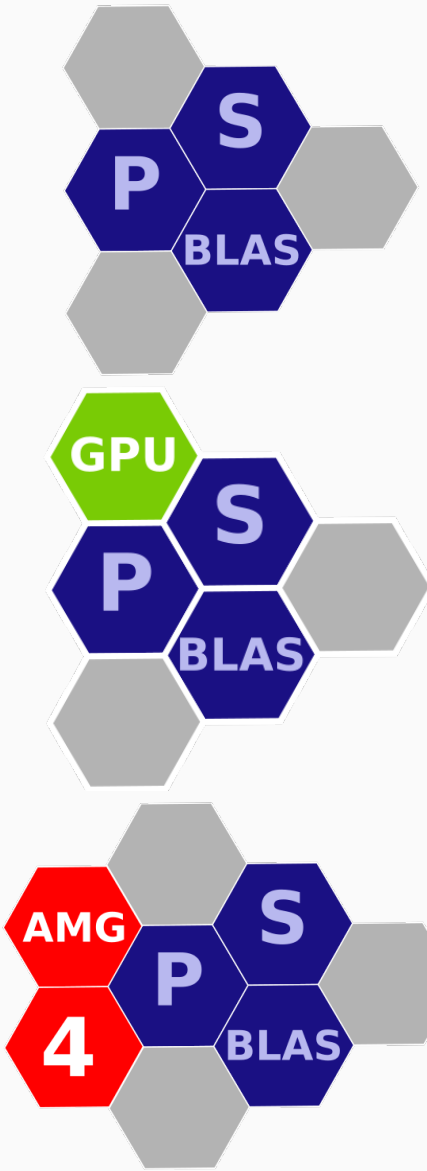
(b) Refinements from $\mathbf{w} = (1, 1, \ldots, 1)^T$

PSCToolkit

Parallel Sparse Computation Toolkit – psctoolkit.github.io

Two central libraries PSBLAS and AMG4PSBLAS:

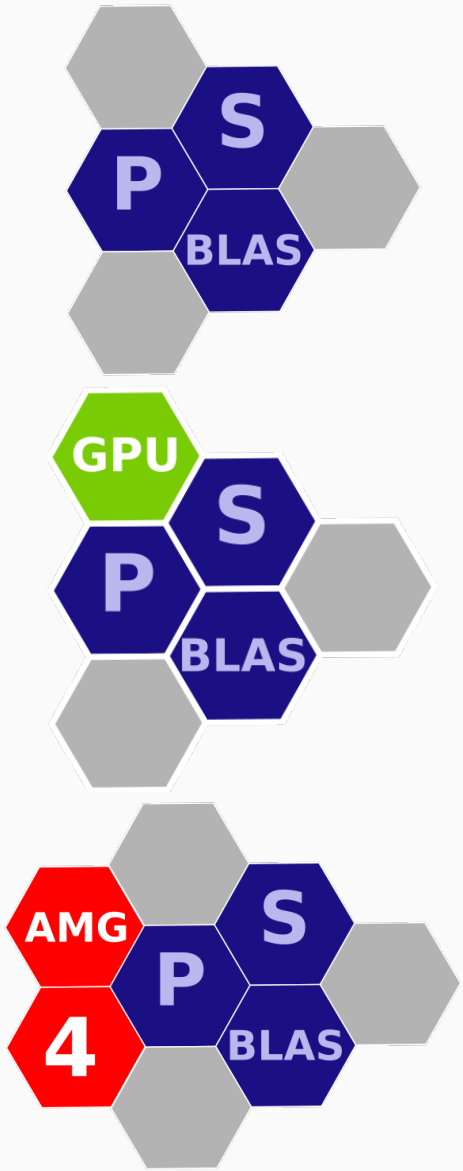
- Existing software standards:
  - MPI, OpenMP, CUDA
  - Serial sparse
  - BLAS,
  - (Par)Metis,
  - AMD
- Attention to performance using modern Fortran;
- Research on new preconditioners;
- No need to delve in the data structures for the user;
- Tools for error and mesh handling beyond simple algebraic operations;
- Standard Krylov solvers



Parallel Sparse Computation Toolkit – psctoolkit.github.io

Two central libraries PSBLAS and AMG4PSBLAS:

- Domain decomposition preconditioners
- Algebraic multigrid with aggregation schemes
  - Vaněk, Mandel, Brezina Aggregation
  - Matching Based
  - Smoothed Aggregation
- 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



Parallel Sparse Computation Toolkit – `psctoolkit.github.io`

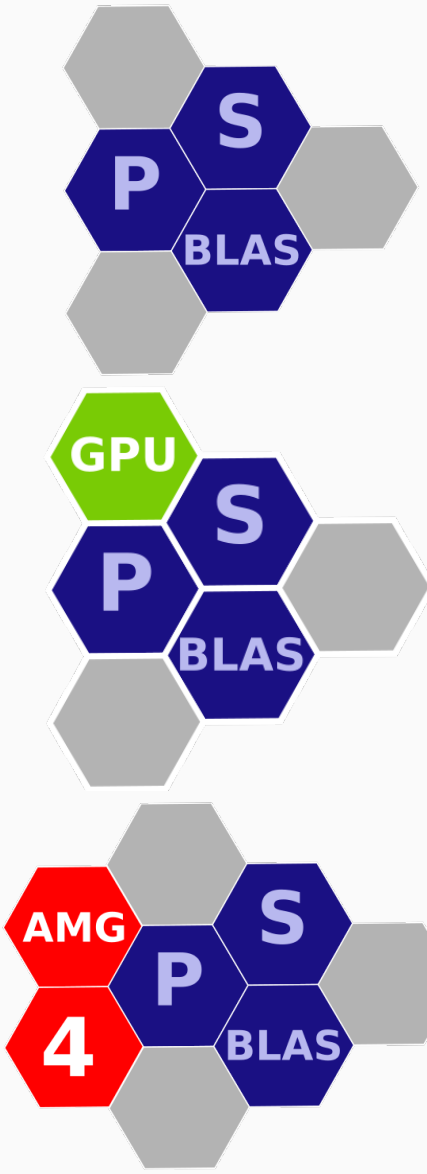
Two central libraries PSBLAS and AMG4PSBLAS.

- Freely available from: `https://psctoolkit.github.io`,
- Open Source with BSD 3 Clause License,
- Soon to be released/interfaced with the Alya multi-physics solver, and the ParFlow solver, KINSOL non-linear solvers.

  These are collaborations with:

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

JÜLICH Forschungszentrum



Parallel Sparse Computation Toolkit – Application Example

Init, setup and build the preconditioner

```
call prec%init(ictxt,ptype,info)
! Set-up calls
call prec%set('ml_cycle', 'KCYCLE', info)
call prec%set('outer_sweeps', 1, info)
! ... build
call prec%hierarchy_build(a,desc_a,info)
call prec%smoothers_build(a,desc_a,info)
```

Solve the linear system

```
call psb_krylov("FCG",a,prec,b,x,eps, &
 & desc_a,info,itmax=100,iter=iter, err=err)
```

Parallel Sparse Computation Toolkit – Application Example

Init, setup and build the preconditioner

```
call prec%init(ictxt,ptype,info)
! Set-up calls
call prec%set('ml_cycle', 'KCYCLE', info)
call prec%set('outer_sweeps', 1, info)
! ... build
call prec%hierarchy_build(a,desc_a,info)
call prec%smoothers_build(a,desc_a,info)
```

Solve the linear system

```
call psb_krylov("FCG",a,prec,b,x,eps, &
 & desc_a,info,itmax=100,iter=iter, err=err)
```

There are also C interfaces available for everything!

Numerical Examples

Comparison with Hypre - CPU Runs - MareNostrum

Comparison with the preconditioners available in the **Hypre**, a state of the art preconditioning library from LLNL.

- 👉 Run on the MareNostrum machine up to 8192 cores
- 👉 Test: 3D Constant coefficient Poisson Problem with FCG
- 👉 DoF: 256k unknown $\times$ MPI core
- ⏷ Measures: Operator Complexity opc $= \frac{\sum_{l=0}^{\text{nl}-1} \textbf{nnz}(A_l)}{\textbf{nnz}(A_0)}$ and Solve Time (s).

Comparison with Hypre - CPU Runs - MareNostrum

Comparison with the preconditioners available in the **Hypre**, a state of the art preconditioning library from LLNL.

- 👍 Run on the MareNostrum machine up to 8192 cores
- 👍 Test: 3D Constant coefficient Poisson Problem with FCG
- 👍 DoF: 256k unknown $\times$ MPI core
- ⏷ Measures: Operator Complexity opc $= \frac{\sum_{l=0}^{\text{nl}-1} \textbf{nnz}(A_l)}{\textbf{nnz}(A_0)}$ and Solve Time (s).

Scaling

There are two common notions of scalability:

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

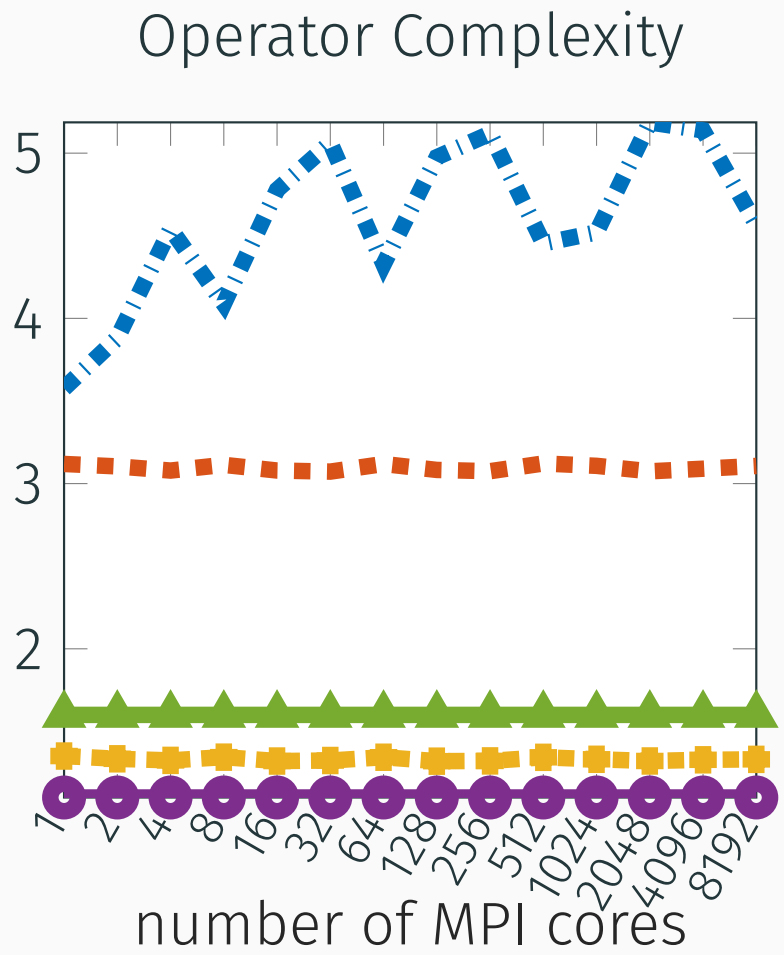
Comparison with Hypre - CPU Runs - MareNostrum

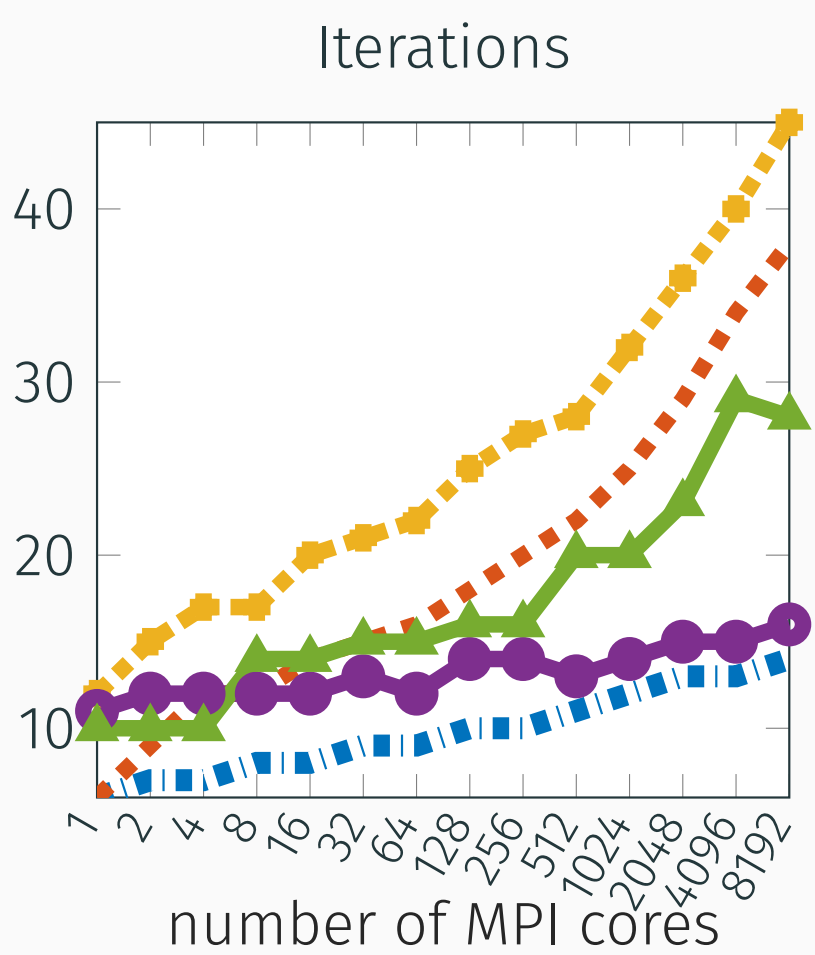
Giving a name to preconditioners with many parameters:

Cycle	Aggregation	Smoother	Coarsest solver
K	Unsmoothed **P**arallel **M**at**c**hing **3/4**	**H**ybrid **G**auss-**S**eidel	**P**reconditioned **Kr**ylov Method
V	**S**moothed **P**arallel **M**at**c**hing **3/4**	l_1–**H**ybrid **G**auss-**S**eidel	
	Smoothed **VBM**	**INVK**	
		l_1–**INVK**	
		l_1–**Jac**obi	

For **Hypre** we test HMIS and Falgout coarsening schemes.

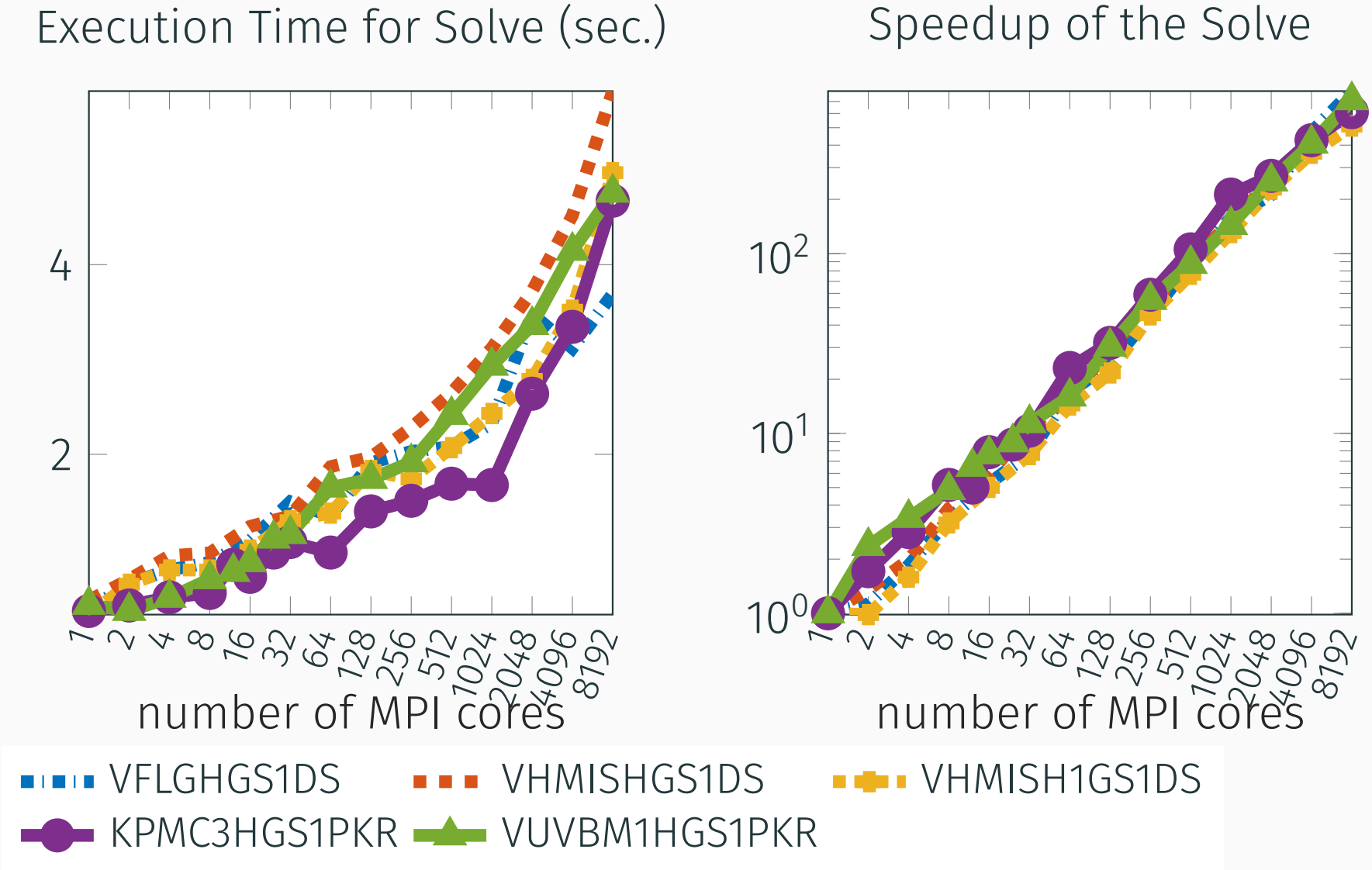
Comparison with Hypre - CPU Runs - MareNostrum





VFLGHGS1DS
VHMISHGS1DS
VHMIS1HGS1DS
KPMC3HGS1PKR
VUVBMHGS1PKR

Comparison with Hypre - CPU Runs - MareNostrum

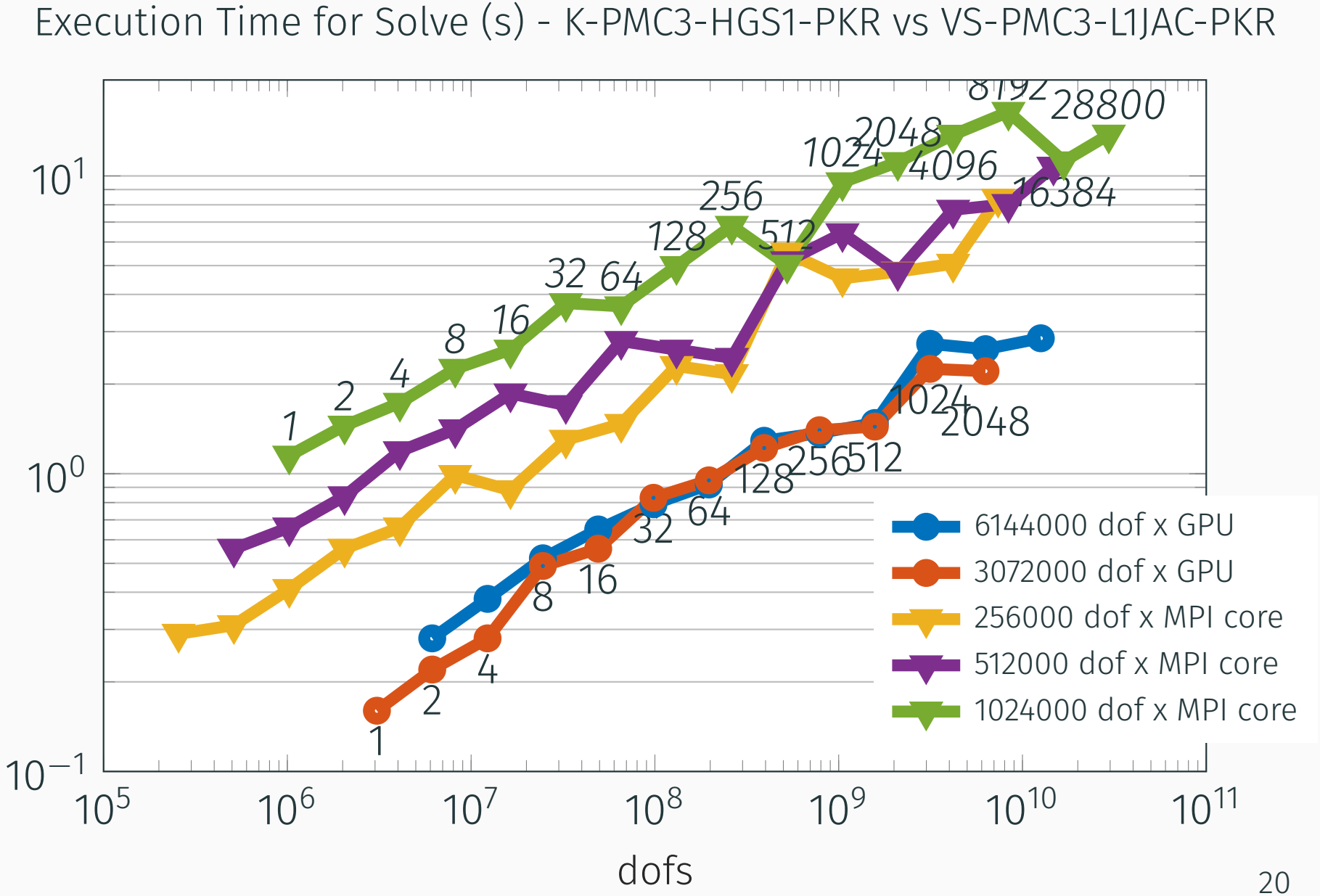


Weak Scalability - CPU/GPU Runs - Piz Daint

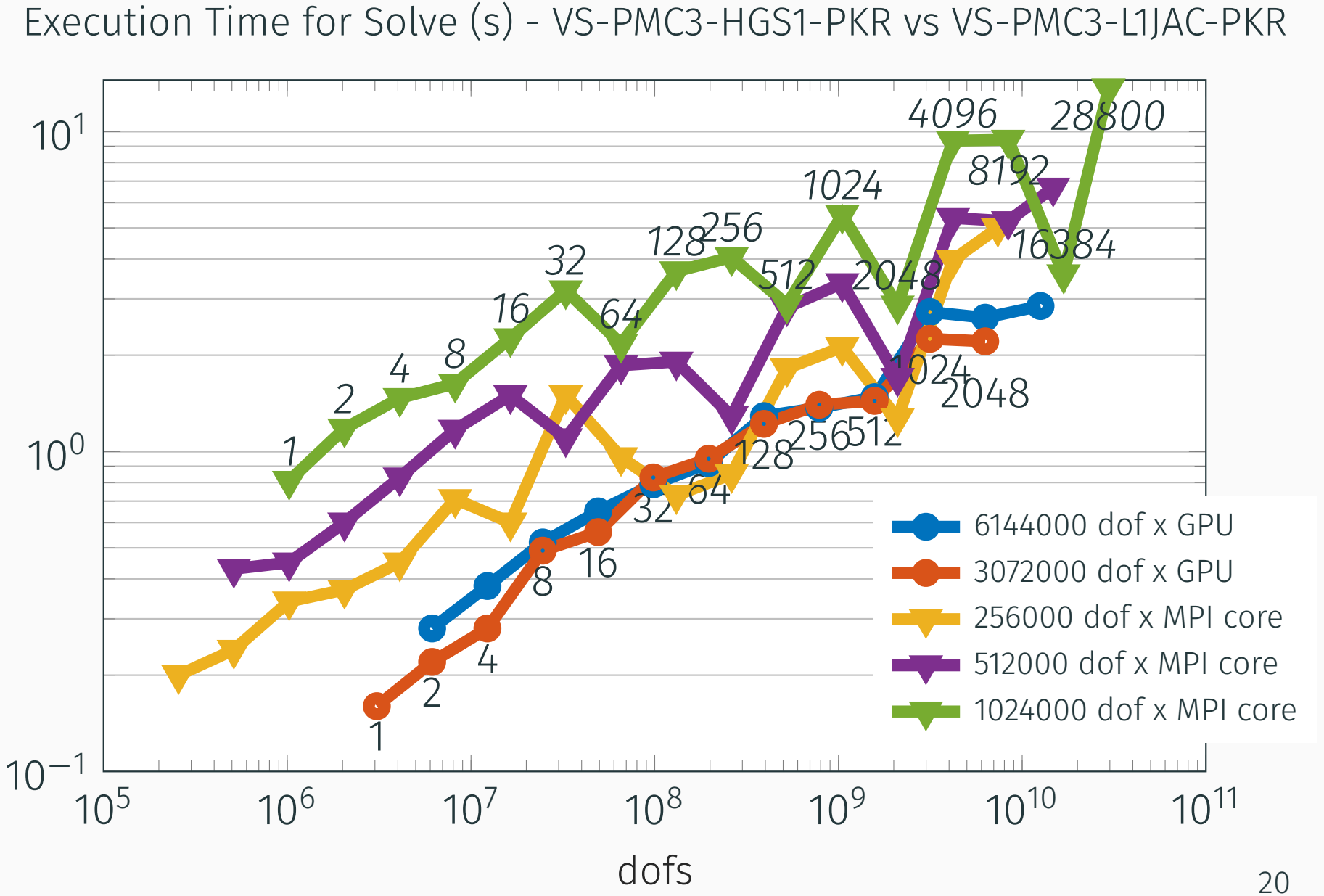
The resulting performance of the multigrid preconditioner in term of implementation scalability depends also on how effective the coarsening procedure is, and on how well balanced is the distribution of the coarsest matrix.

- 👉 Run on the Piz Daint machine up to 28800 cores and 2048 GPUs
- 👉 Test: 3D Constant coefficient Poisson Problem with FCG
- 👉 DoF: 256k/512k/1M unknown $\times$ MPI core and 3M/6M per GPUs
- ⏷ Measures: execution time for solve

Weak Scalability - CPU/GPU Runs - Piz Daint

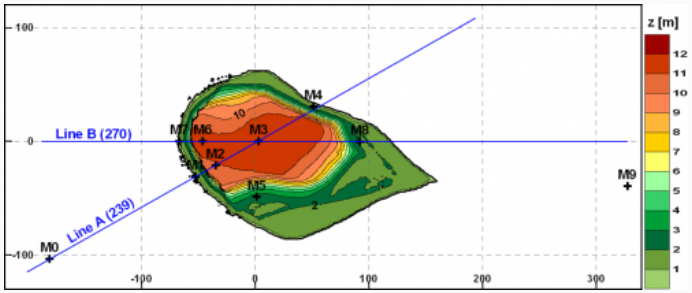


Weak Scalability - CPU/GPU Runs - Piz Daint



A CFD application inside Alya

From a Joint work with
Herbert Owen
Barcelona Super Computing Center

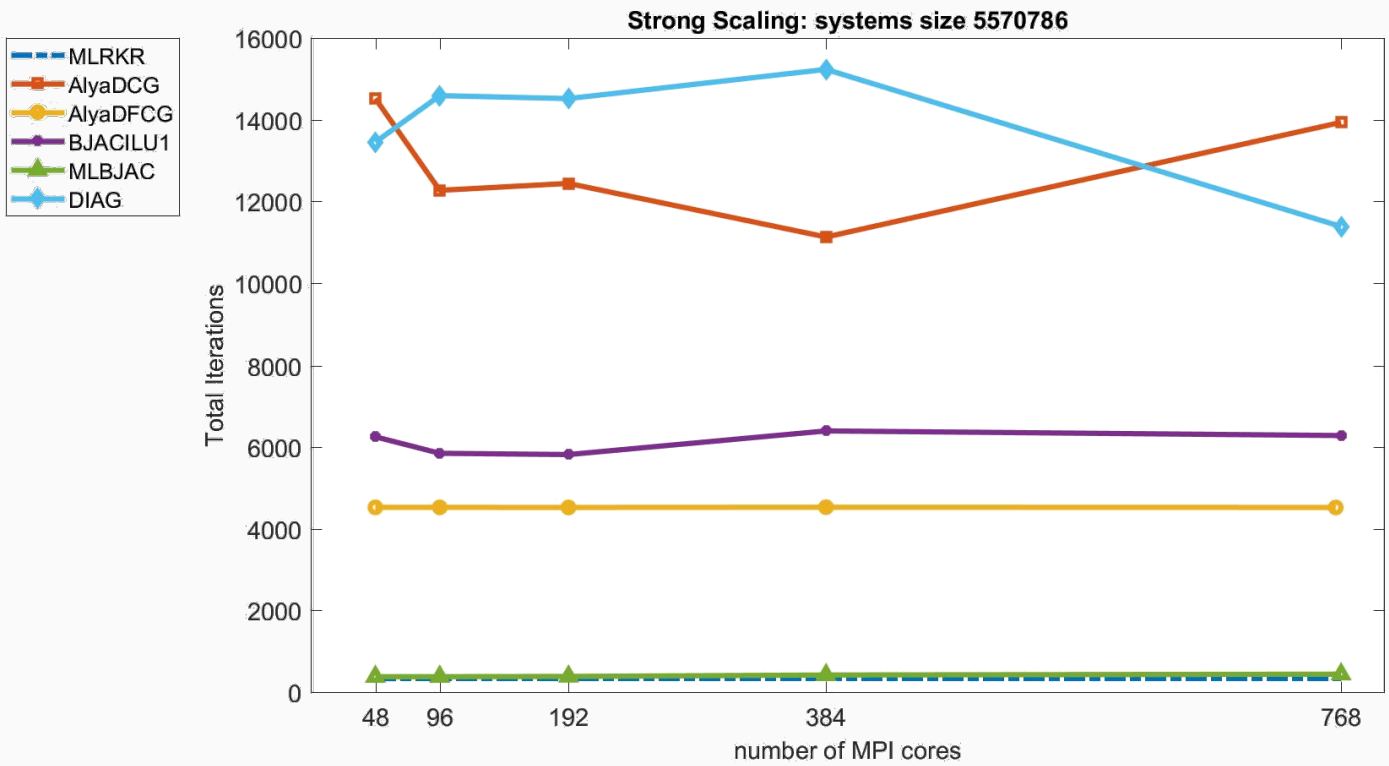


Bolund is an isolated hill situated in Roskilde Fjord, Denmark. An almost vertical escarpment in the prevailing W-SW sector ensures flow separation in the windward edge resulting in a complex flow field.

- **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

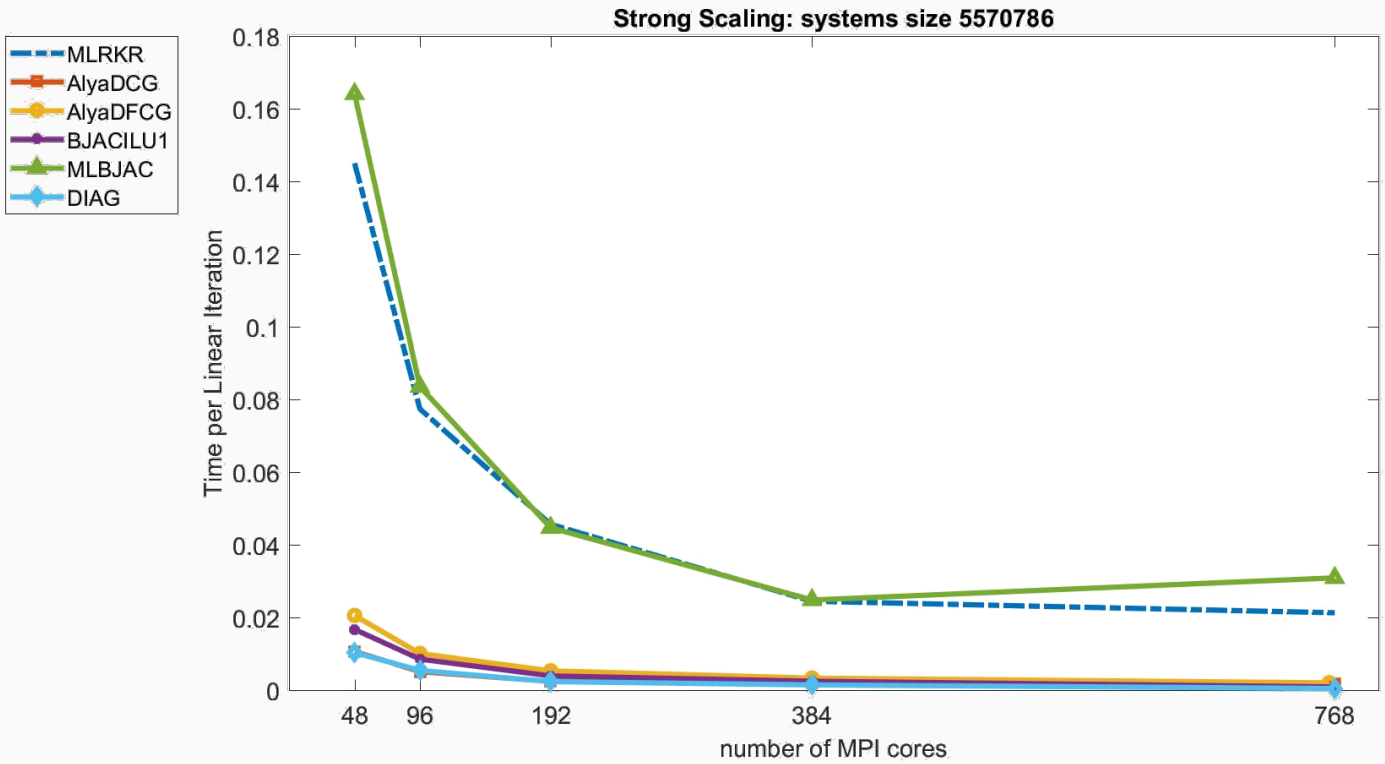
Fixed size problem with $n = 5570786$ dofs, 100 time steps



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

Bolund Test Case - Strong Scaling - Pressure Equation

Fixed size problem with $n = 5570786$ dofs, 100 time steps



- Total number of linear iterations is smaller and stable for increasing number of cores,
- The time needed per each iteration decreases for increasing number of cores.

Conclusions and Future Directions

Conclusions and Future Directions

We have proved

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

Algorithmic and software extensions to AMG4PSBLAS

- ⚙ Multi-objective matching to reduce the number of singletons,
- ⚙ Process remapping for coarse grid solutions,
- ⚙ Communication avoiding Krylov methods.

Completely new extension packages for the PSCToolkit,

- Parallel computations of $f(A)\mathbf{b}$ for general $f(x)$,
- Krylov based eigenvalue solvers.

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 and S. Filippone, On the quality of matching-based aggregates for algebraic coarsening of SPD matrices in AMG. arXiv preprint (2020), `arXiv:2001.09969`.
- Scalability results
  - P. D'Ambra, F. Durastante and S. Filippone,AMG preconditioners for Linear Solvers towards Extreme Scale. arXiv preprint (2020), `arXiv:2006.16147`.
- 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!