High Performance Linear Algebra

Lecture 2.5: Preconditioning

Ph.D. program in High Performance Scientific Computing

Salvatore Filippone **Pasqua D'Ambra** **Fabio Durastante**

March 26 2026 — 09.00:13.00

Table of Contents

Iterative solvers: preconditioning

Transform a problem

$$Ax = b \Leftrightarrow M^{-1}Ax = M^{-1}b$$

in such a way that:

Iterative solvers: preconditioning

Transform a problem

$$Ax = b \Leftrightarrow M^{-1}Ax = M^{-1}b$$

in such a way that:

1. M is easy to compute;

Iterative solvers: preconditioning

Transform a problem

$$Ax = b \Leftrightarrow M^{-1}Ax = M^{-1}b$$

in such a way that:

1. M is easy to compute;
2. M^{-1} is easy to apply;

Iterative solvers: preconditioning

Transform a problem

$$Ax = b \Leftrightarrow M^{-1}Ax = M^{-1}b$$

in such a way that:

1. M is easy to compute;
2. M^{-1} is easy to apply;
3. $M^{-1}A \approx I$;

Iterative solvers: preconditioning

Transform a problem

$$Ax = b \Leftrightarrow M^{-1}Ax = M^{-1}b$$

in such a way that:

1. M is easy to compute;
2. M^{-1} is easy to apply;
3. $M^{-1}A \approx I$;

These three requisites are contradictory!

Iterative solvers: preconditioning

Transform a problem

$$Ax = b \Leftrightarrow M^{-1}Ax = M^{-1}b$$

in such a way that:

1. M is easy to compute;
2. M^{-1} is easy to apply;
3. $M^{-1}A \approx I$;

These three requisites are contradictory!

But a good preconditioner guarantees convergence in $j \ll n$ steps.

Preconditioning

How does it work? Assume $x_0 = 0 \Rightarrow r_0 = b$. Hence:

$$x \in \mathcal{K}_m \Rightarrow r = b - Ax \in \mathcal{K}_{m+1},$$

which is equivalent to

$$x_k = P_k(A)r_0; \quad (1)$$
$$r_k = Q_{k+1}(A)r_0. \quad (2)$$

Convergence is measured on the basis of $\|r_k\|$: hence we are searching for a polynomial $Q(\lambda)$ that takes small (absolute) value (ideally zero) on the spectrum of A. If the eigenvalues are clustered, a small degree polynomial satisfies the requirements.

Krylov methods: Preconditioned CG

Compute $r^{(0)} \leftarrow b - Ax^{(0)}$
while $r_i \neq 0$ **do**
    Solve $Mz_k = r_k$
    $i \leftarrow i + 1$
    if $i = 1$ **then**
        $p^{(1)} \leftarrow z^{(0)}$
    else
        $\beta_i \leftarrow -r_{i-1}^T z_{i-1} / r_{i-2}^T z_{i-2}$
        $p^{(i)} \leftarrow z^{(i-1)} + \beta_i p^{(i-1)}$
    end if
    $\alpha_i \leftarrow r_{i-1}^T z_{i-1} / p_i^T A p_i$
    $x^{(i)} \leftarrow x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)} \leftarrow r^{(i-1)} - \alpha_i A p_i$
end while

Krylov methods: preconditioned BiCGSTAB

Compute $r^{(0)} \leftarrow b - Ax^{(0)}$; choose $\tilde{r}_0$
Set $p_0 = r_0$; $i = 0$
for $i = 1, 2, \ldots imax$, until convergence **do**
    Solve $Mf_i = p_i$
    $\alpha_i \leftarrow (r_i, \tilde{r}_i)/(Af_i, \tilde{r}_i)$
    $s_i \leftarrow r_i - \alpha_i Af_i$
    Solve $Mz_i = s_i$
    $\omega_i \leftarrow (Az_i, z_i)/(Az_i, Az_i)$
    $x_{i+1} \leftarrow x_i + \alpha_i p_i + \omega_i s_i$
    $r_{i+1} \leftarrow s_i - \omega_i Az_i$
    $\beta_i \leftarrow (\alpha_i/\omega_i) \times (r_{i+1}, \tilde{r}_0)/(r_i, \tilde{r}_0)$
    $p_{i+1} \leftarrow r_{i+1} + \beta_i(p_i - \omega_i Af_i)$
end for

Preconditioning Techniques

1 Preconditioning

A brief timeline:

1937	Cesari: first idea of polynomial preconditioning at the time. Reappeard in 1952 by Lanczos
1977	Meijerink & van der Vorst: theory of incomplete factorizations and convergence analysis;
mid-1990s	Cai, Widlund, Saad, Sarkis: domain decomposition and Schwarz preconditioners; (theoretical explanation of RAS: Efstathiou and Gander, 2003)
1977-1986	A. Brandt: multigrid and algebraic multigrid foundations
1996-2003	Vanek, Brezina, Mandel: algebraic multigrid preconditioners with smoothed aggregation
1999-2003	van Duijn, Benzi & Tuma: approximate inverses;
2000-2010	Heroux, Tuminaro & Sala, Filippone & D'Ambra & di Serafino: packages of multilevel preconditioners, non-symmetric extensions
2002-2008	Benzi, Bertaccini, Golub: Spectral analysis of preconditioned iterations, preconditioner updates;
2010-2012	Bertaccini, Filippone: Approximate inverses and preconditioner updates;
2016	D'Ambra, Filippone and Vassilevski: Algebraic Multigrid by Weighted Matching
2021	D'Ambra, Durastante, and Filippone: AMG Preconditioners for Linear Solvers towards Extreme Scale
2023	D'Ambra, Durastante, and Filippone: Parallel Sparse Computation Toolkit
2025	D'Ambra, Durastante, Filippone, Massei, Thomas: Optimal polynomial smoothers for parallel AMG.

Table of Contents

2 Preconditioning techniques

Diagonal Scaling:

$$M(i,i) = (a_{ii})$$

very simple.

Equivalent to 1 step of (Point) Jacobi. More generally

> *Use a fixed number of sweeps of a simple iteration, such as Jacobi, Gauss-Seidel, SSOR, etc ...*

Simple to build, (relatively) simple to use.

Incomplete Factorizations:

$$A = \hat{L}\hat{U} - R$$

Strategies:

- $ILU(0)$: factors $\hat{L}$ and $\hat{U}$ have the same pattern of A;
- $ILU(n)$: we accept fill-in up to n levels;
- $ILU(\epsilon)$: we accept fill-in when its value is above a threshold ϵ.

LU factorization, IKJ variant

1: **for** $i = 2, \ldots, n$ **do**
2: **for** $k = 1, \ldots, i-1$ **do**
3: $a_{ik} \leftarrow a_{ik}/a_{kk}$
4: **for** $j = k+1, \ldots, n$ **do**
5: $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}$
6: **end for**
7: **end for**
8: **end for**

We mentioned in passing the concept of *level* of a coefficient, which is defined recursively as follows:

1. Initially we have
$$lev_{ij} = \begin{cases} 0 & \text{if } a_{ij} =\neq 0 \text{ or } i = j; \\ \infty & \text{otherwise.} \end{cases}$$
2. Each time an element is modified by the LU factorization, update as
$$lev_{ij} = \min(lev_{ij}, lev_{ik} + lev_{kj} + 1).$$

In other words, every time you create a new coefficient, you assign a level which is one up from the level of its “parents”.

Incomplete LU with pattern P:

1: **for** $i = 2, \ldots, n$ **do**
2: **for** $k = 1, \ldots, i-1$ **do**
3: **if** $(i,k) \in P$ **then**
4: $a_{ik} \leftarrow a_{ik}/a_{kk}$
5: **for** $j = k+1, \ldots, n$ **do**
6: **if** $(i,j) \in P$ **then**
7: $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}$
8: **end if**
9: **end for**
10: **end if**
11: **end for**
12: **end for**

Existence and stability issues

Incomplete factorizations need not exist even when LU does for the originating matrix.
There are sufficient conditions which ensure that we can do our job.
However, these conditions may be not very useful in practice because, e.g., $||\tilde{L}||$ and/or $||\tilde{U}||$ can be very large compared with $||A||$.
Recall: M admits the LU factorization if it can be written as $M = LU$, and it exists if and only if all leading submatrices of M are nonsingular / $\exists\ \Pi$ permutation such that ΠM admits an LU factorization.

An example (from [6]): A positive definite 2×2 and its (complete) factorization. Let $0 < \epsilon \ll a$:

$$A = \begin{pmatrix} \epsilon & a \\ -a & \epsilon \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -a/\epsilon & 1 \end{pmatrix} \cdot \begin{pmatrix} \epsilon & a \\ 0 & \epsilon + a^2/\epsilon \end{pmatrix} = L \cdot U.$$

A is positive definite nonsymmetric if $\epsilon > 0$.
$\rightarrow$ factors L and U have norm arbitrarily larger than $||A||$.
This cannot happen if A is, e.g., diagonally dominant.
Note: pivoting in incomplete factorizations is not normally used because of the fill-in.
Unfortunately, an incomplete factorization may fail due to zero pivots even if pivoting for stability is used [8].

M-Matrices

Definition

A is an M-matrix if $a_{ii} > 0$, $a_{i,j} \leq 0$ for $i \neq j$ and $A^{-1} \geq 0$ (i.e., the inverse is nonnegative).

Definition

A is an M-matrix $\iff$ $A = sI - B, \quad B \geq 0 \quad s > \rho(B)$.

If A is an M-matrix, then $Re(\lambda) > 0$ for all $\lambda \in \lambda(A)$.

Theorem

The two definitions above are equivalent

This is a corollary of the next theorem

Definition

Let

$$A = M - N.$$

The pair M, N is called a regular splitting if M is nonsingular and both M^{-1} and N are nonnegative.

Theorem

Let M, N be a regular splitting of A. We have $\rho(M^{-1}N) < 1$ if and only if A is nonsingular and A^{-1} is nonnegative.

This implies that the two M-matrix definitions are equivalent.

Proof, part 1.

Define $G = M^{-1}N$. Assume $\rho(G) < 1$; we have

$$A = M(I - G),$$

and A is nonsingular because

$$(I - G)^{-1} = \sum_{k=0}^{\infty} G^k$$

where the series converges since $\rho(G) < 1$; moreover $G \geq 0 \Rightarrow (I - G)^{-1} \geq 0$ but $M^{-1} \geq 0$ therefore

$$A^{-1} = (I - G)^{-1} M^{-1} \geq 0.$$

□

Proof, part 2.

Let A,M nonsingular, $A = M(I - G)$; then $(I - G)$ nonsingular

$$\begin{aligned} A^{-1}N &= (M(I - M^{-1}N))^{-1}N \\ &= (I - M^{-1}N)^{-1}M^{-1}N = (I - G)^{-1}G. \end{aligned}$$

$G \geq 0$; Perron-Frobenius theorem: $\exists x \geq 0$ s.t.$Gx = \rho(G)x$.

$$A^{-1}Nx = \frac{\rho(G)}{1 - \rho(G)}x,$$

but $A^{-1}N \geq 0$ then

$$\frac{\rho(G)}{1 - \rho(G)} \geq 0,$$

which implies $0 \leq \rho(G) \leq 1$; but $I - G$ is nonsingular, hence $\rho(G) < 1$. □

Theorem

Let A be an M-matrix and P a given nonzero pattern; then the incomplete factorization

$$A = \hat{L}\hat{U} - R$$

exists and both $\hat{L}$ and $\hat{U}$ are nonsingular M-matrices; this is also a regular splitting.

First proven by Meijrink and van der Vorst in 1977 for symmetric M-matrices.

Lemma

Gaussian elimination preserves the M-matrix property.

Proof.

Consider $L^{(1)}$ an elementary Gauss transformation on A, then we can restrict ourselves to prove

$$A^{(1)} = L^{(1)}A$$

is still an M-matrix, and then apply recursively. Now, we have

$$L^{(1)} = \begin{pmatrix} 1 & & & \\ -\frac{a_{21}}{a_{11}} & 1 & & \\ \dots & & \ddots & \\ -\frac{a_{n1}}{a_{11}} & & & 1 \end{pmatrix} \geq 0.$$

□

Proof (cont.)

$A^{(1)}$ is invertible, and $a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}$ for $i > 1$, and $a_{11}^{(1)} = a_{11}$. Consider $(A^{(1)})^{-1}$: first column of $A^{(1)}$ has only one nonzero $a_{11}^{(1)} = a_{11}$, hence $(A^{(1)})^{-1}$ also has only one nonzero element $1/a_{11}$. Thus

$$(A^{(1)})^{-1} e_1 = \frac{1}{a_{11}} e_1 > 0.$$

We also have

$$(A^{(1)})^{-1} e_j = A^{-1} (L^{(1)})^{-1} e_j = A^{-1} e_j \geq 0, \quad j > 1$$

therefore

$$(A^{(1)})^{-1} \geq 0,$$

that is, $A^{(1)}$ is an M-matrix. □

Hence, in $A = LU$, U is also an M-matrix

Lemma

The inverse of the elementary Gauss transformation $L^{(i)}$ is an M-matrix.

Proof.

Consider $L^{(1)}$; it differs from the identity only in the first column, where

$$l_{i1}^{(1)} = -\frac{a_{i1}}{a_{11}} \geq 0;$$

hence $L^{(1)} \geq 0$. The inverse of $L^{(1)}$ is obtained by switching the signs of its off-diagonal elements, which then become nonpositive. Thus it is a matrix with a positive diagonal, nonpositive offdiagonal entries, it is nonsingular and its inverse is nonnegative, hence an M-matrix. □

It follows that L is an M-matrix

Lemma

Dropping an off-diagonal entry from an M-matrix preservers the M-matrix property.

Proof.

This requires the following (see e.g. Horn & Johnson):

$$|C| \leq E \Longrightarrow \rho(C) \leq \rho(|C|) \leq \rho(E)$$

but then dropping an off-diagonal element means

$$A' = sI - B' \qquad B' \leq B, \Longrightarrow s > \rho(B) \geq \rho(B')$$

therefore A' is still an M-matrix. □

Therefore both $\hat{L}$ and $\hat{U}$ are M-matrices.

Approximate Inverses

Seek a matrix $G = M^{-1}$ approximating **the inverse of the matrix**, e.g.

$$\min \|I - GA\|$$

Direct search attractive but expensive.

Alternative strategies:

- Approximate Biconjugation [3]
- Inversion of incomplete factors [5]

Work by Rafiei, Bollhöfer (2011) on biconjugation. All of them compute

$$A^{-1} \approx ZD^{-1}W^T$$

Development directions: efficient and robust implementation, embed in multilevel and preconditioner update frameworks.

- Approximate inverse preconditioners in factored form require just matrix-vector multiply, no triangular system solve (see [1, 7])
- The construction phase of Approximate INVerse preconditioners is expensive: lots of work to be done.
- Many issues (computational cost, stability, approximation wrt A^{-1}) of approximate inverse preconditioners like AINV and *inversion-and-sparsification* still not completely understood.
- Updates of preconditioners by Benzi and B. et al. (see [2]) require approximate inverse factors.

Approximate inverses: Biconjugation

Given two A-biconjugate sets of vectors W and Z we have

$$W^T A Z = D = \begin{pmatrix} p_1 & 0 & \dots & 0 \\ 0 & p_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & p_n \end{pmatrix};$$

it follows easily that

$$A^{-1} = \sum (1/p_i) z_i w_i^T.$$

A possible idea is then (Benzi & Tuma):

> *Apply a biconjugation process with W and Z triangular, and sparsify while building.*

Critical issue: how many dot products do we actually perform.

Approximate inverses: Biconjugation

Right looking variant, stabilized (Benzi, Cullum, Tuma)

1: $w_i^{(0)} \leftarrow z_i^{(0)} \leftarrow e_i \qquad 1 \leq i \leq n$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad v_i \leftarrow A z_i^{(i-1)}$
4: $\quad$ **for** $j = i, i+1, \ldots, n$ **do**
5: $\quad\quad p_j^{(i-1)} \leftarrow v_i^T z_j^{(i-1)};$
6: $\quad$ **end for**
7: $\quad$ **for** $j = i+1, \ldots, n$ **do**
8: $\quad\quad z_j^{(i)} \leftarrow z_j^{(i-1)} - \left(\frac{p_j^{(i-1)}}{p_i^{(i-1)}} \right) z_i^{(i-1)};$
9: $\quad$ **end for**
10: **end for**
11: $z_i \leftarrow z_i^{(i-1)},\ p_i \leftarrow p_i^{(i-1)},\ 1 \leq i \leq n$

Good behaviour in difficult cases.

Approximate inverses: Biconjugation

Left looking variant

1: Let $z_1^{(0)} \leftarrow e_1$, $p_1^{(0)} \leftarrow a_{11}$
2: **for** $i = 2, \ldots, n$ **do**
3: $\quad z_i^{(0)} \leftarrow e_i$
4: $\quad$ **for** $j = 1, \ldots, i-1$ **do**
5: $\quad\quad p_i^{(j-1)} \leftarrow a_j^T z_i^{(j-1)}$
6: $\quad\quad z_i^{(j)} \leftarrow z_i^{(j-1)} - (\frac{p_i^{(j-1)}}{p_j^{(j-1)}}) z_j^{(j-1)}$
7: $\quad$ **end for**
8: $\quad p_i^{(i-1)} \leftarrow a_i^T z_i^{(i-1)}$
9: **end for**

Note: the drop strategy is applied just once per column, to the end result of all updates; this is better from a numerical point of view than unstabilized right looking.

New Left-Looking biconjugation

Full vector z_i during execution of loop 4; two dropping rules:

1. At 6 update only for a sufficiently large p_i/p_j.
2. After 8 a dropping to z_i

Key observation here:

> *Execution of statement 5 computes the dot product* (a_j, z_i) *but in most cases it will be exactly zero*

hence, a useless quadratic cost.

Make sure loop 4 is executed only as necessary, i.e.:

> *Execute iteration j only when the dot product at 5 is nonzero: j the next lowest index such that row* a_j^T *has at least one nonzero in a column corresponding to a nonzero in* $z_i^{(j-1)}$.

Proposed in [4]

New Left-Looking biconjugation

2 Preconditioning techniques

1: A_{i*} *i*th row of A, A_{*j} *j*th column;
2: $C_{A_{i*}} = \{j : a_{ij} \neq 0\}$; $R_{A_{*j}} = \{i : a_{ij} \neq 0\}$; $z_1^{(0)} \leftarrow e_1$; $\quad p_1^{(0)} \leftarrow a_{11}$
3: **for** $i = 2, \ldots, n$ **do**
4: $\quad zw \leftarrow e_i$; $\mathcal{S} \leftarrow C_{A_{i*}}$;
5: $\quad$ **while** $\mathcal{S} \neq \emptyset$ **do**
6: $\quad\quad j \leftarrow \text{first}(\mathcal{S})$; $\mathcal{S} \leftarrow \mathcal{S} - \{j\}$; $p(i) \leftarrow A_{j*}^T zw$; $\alpha \leftarrow (p(i)/p(j))$
7: $\quad\quad$ **if** $|\alpha| > \epsilon$ (drop rule) **then**
8: $\quad\quad\quad zw \leftarrow zw - \alpha Z_{*j}$
9: $\quad\quad\quad$ **for** $k \in R_{Z_{*j}}$ **do**
10: $\quad\quad\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{l \in R_{A_{*k}} : l > j\}$
11: $\quad\quad\quad$ **end for**
12: $\quad\quad$ **end if**
13: $\quad$ **end while**
14: $\quad p(i) \leftarrow A_{i*}^T zw$;
15: $\quad$ Apply a drop rule to zw;
16: $\quad Z_{*i} \leftarrow zw$;
17: **end for**

Computational cost:

Theorem

Let nz_a *be the average number of nonzeros per row in* a *and similarly* nz_z*. Assume:*

$$|S| \leq \gamma\, \mathrm{nz}_a, \tag{3}$$
$$\mathrm{nz}_z \leq \beta\, \mathrm{nz}_a, \tag{4}$$

with $|S|$ *maximum cardinality of the sets of entries in any of the* z_i *before drop rule 15. Then the computational complexity of left-looking biconjugation is*

$$O(\gamma n\, \mathrm{nz}_a^2(1 + \beta(1 + \log(\gamma\, \mathrm{nz}_a)))). \tag{5}$$

Proof.

Given bound (3), the term $\gamma n \operatorname{nz}_a$ follows from the nesting of the two outer loops. The body of loop 5 contains the following terms:

- The dot product at 6 adds a term nz_a;
- The cost of 8 is given by $\beta \operatorname{nz}_a$, by assumption (4);
- The update of the set S at 10 adds another cost $\beta \operatorname{nz}_a \log(\gamma \operatorname{nz}_a))$.

The result follows because by drop rule 7 statements 8-10 are executed at most as many times as statement 6. □

Approximate inverses: sparse inversion of sparse factors

Given one factor of an LDU decomposition

$$U = I + \sum e_i u_i^T = \prod_{i=n-1}^{1} (I + e_i u_i^T)$$

its inverse is also triangular; thus

$$U^{-1} = \prod_{i=1}^{n-1} (I - e_i u_i^T) = I + \sum e_i \hat{u}_i^T$$

and therefore

$$\hat{u}_i^T = -u_i^T \prod_{j=i+1}^{n-1} (I - e_j u_j^T)$$

Approximate inverses: sparse inversion of sparse factors

It is natural to

1. Start from a sparse factor (e.g., from ILUt)
2. Apply a drop strategy to avoid a dense inverse

Effective practical schemes for computing $\tilde{Z}$ and $\tilde{W}$: dropping fill as soon as it occurs within a sparse vector update

- To control the number of fills: impose a restriction on the allowed number of nonzero entries.
- Effectiveness on shared memory machines: the only concurrent access to the same memory location is a read type access.

There is potential for parallelization, to be explored further.

Approximate inverses: sparse inversion of sparse factors

Numerical drop strategy:

1: **for** $j = 1$ to $n - 1$ **do**
2: $\quad \hat{u}_i^T \leftarrow -u_i^T$; j location of first nonzero in $\hat{u}_i^T$;
3: $\quad$ **while** $j < n$ **do**
4: $\quad\quad \alpha \leftarrow -\hat{u}_i^T e_j$
5: $\quad\quad$ **if** $|\alpha| > \epsilon$ **then**
6: $\quad\quad\quad \hat{u}_i^T \leftarrow \hat{u}_i^T + \alpha u_j^T$
7: $\quad\quad$ **else**
8: $\quad\quad\quad \hat{u}_i^T(j) \leftarrow 0$
9: $\quad\quad$ **end if**
10: $\quad\quad$ j location of next nonzero in $\hat{u}_i^T$
11: $\quad$ **end while**
12: $\quad$ Drop elements in $\hat{u}_i$ to achieve desired number of nonzeros.
13: **end for**

Easy to implement reusing all the building blocks of ILUT.

Theorem

Let nz_u be the average number of nonzeros per row in u and similarly $\mathrm{nz}_{\hat{u}}$; assume moreover the bounds

$$|S| \leq \gamma\, \mathrm{nz}_u, \tag{6}$$
$$\mathrm{nz}_{\hat{u}} \leq \beta\, \mathrm{nz}_u, \tag{7}$$

where $|S|$ is the maximum size of the set of entries in any of the $\hat{u}_i$ before the application of the drop rule 12. Then the cost of algorithm for inversion of sparse factors is

$$O(\gamma\beta n \cdot \mathrm{nz}_u^2(1 + \log(\gamma\, \mathrm{nz}_u))). \tag{8}$$

Proof.

Given the bound (6), the term $\gamma n \cdot \mathrm{nz}_u$ follows easily from the nesting of the two outer loops. For the remaining factor $\beta\,\mathrm{nz}_u(1+\log(\gamma\,\mathrm{nz}_u))$, consider statement 6: this is executed whenever the nonzero in $\hat{u}_i$ is above threshold, and we would expect this to happen a number of times within a (moderate) factor times the size of u_i. On each execution, statement 6 requires nz_u floating-point operations to execute the sparse AXPY, plus $\mathrm{nz}_u \log(|\mathcal{S}|)$ operations to update the set $\mathcal{S}$ with the (possibly new) nonzero entries. Using again bound (6) gives the desired result. □

Approximate inverses: computational load

Crucial assumption: both both β and γ are small constants.

- β small: in many applications we like a preconditioner with nonzeros of the same order as A; number of nonzeros is enforced at step 12;
- γ is more complex: relies on the interaction between the profile of u and $\hat{u}$. It follows from an exponential decay argument by Demko & Moss.

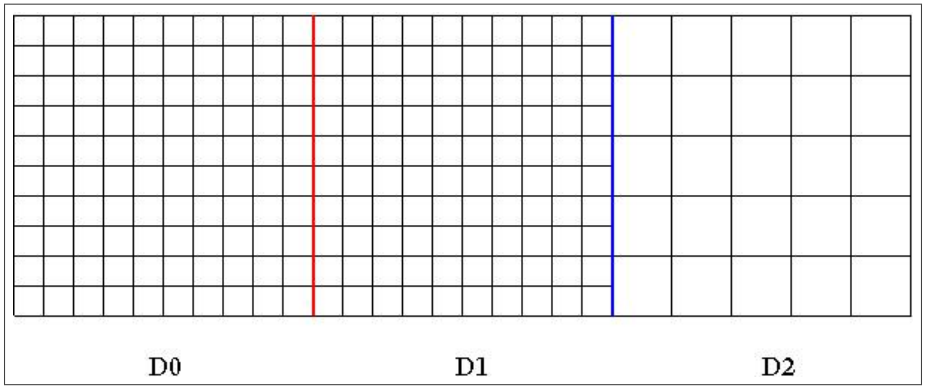
From Van Duin:

$$C_{\text{invrt}} = O\left(nz_{\hat{U}} \frac{nz_U}{n}\right)$$

nz_U number of nonzeros above the main diagonal in U. First term is $n\beta\,\mathrm{nz}_u$, second term is exactly nz_u; equivalent under the mild assumption $\log(\gamma\,\mathrm{nz}_u)$ bounded by a small constant.

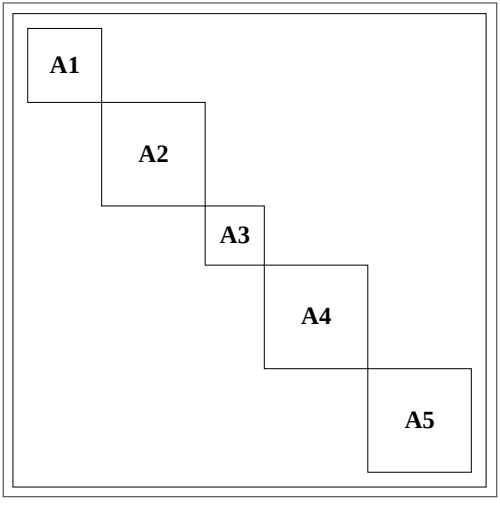
Domain decomposition

Basic idea: decompose the computational domain in subdomains, solve locally, then combine the solutions



Block Jacobi

Approximate local inverse of the diagonal blocks; it is possible to have multiple sweeps



Block Jacobi

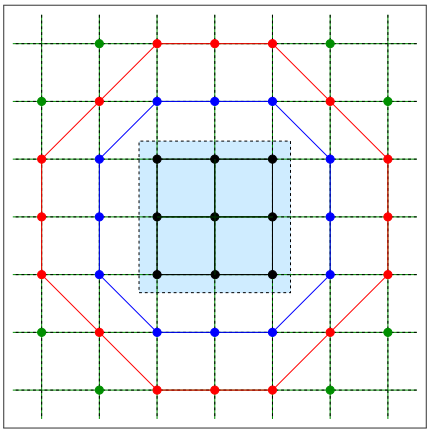
Let D^{-1} be the approximate inverse of $BD(A)$, the block diagonal of A.
We are computing $y \leftarrow y + \hat{y}$ wher $yhat$ is computed from x as follows

Initialize $\hat{y}_0$
for $i = 0 \ldots$ **do**
  $\hat{y}_{i+1} \leftarrow D^{-1}(x - ND * \hat{y}_i)$
end for

where $ND = A - BD(A)$ is the part of matrix A outside the block diagonal.
This is a Block Jacobi smoother; it is commonly used with D^{-1} implemented through ILU, but any other approximated solver could be used.

Domain decomposition: Additive Schwarz

Basic idea: solve on a support “extended” with respect to the local domain:



$$Ax = b, \qquad A \in \mathbb{R}^{n \times n} \qquad \text{symmetric pattern}$$
$$G = (W, E) \qquad \text{adjacency graph of } A$$
$$W = \bigcup W_i^0, \qquad W_i^0 \cap W_j^0 = \varnothing \quad i \neq j \quad \text{a partition of G}$$

Definition of δ-overlap:

$$W_i^\delta \supset W_i^{\delta-1} \quad j \in W_i^\delta \iff \exists k \in W_i^{\delta-1} : (j,k) \in E$$

Restriction operator R_I^δ:

$$R_i^\delta = (e_{j_1}, e_{j_2} \dots e_{j_m})^T, \quad j_k \in W_i^\delta$$

Prolongation operator $P_i = (R_i^\delta)^T$. Restriction of A :

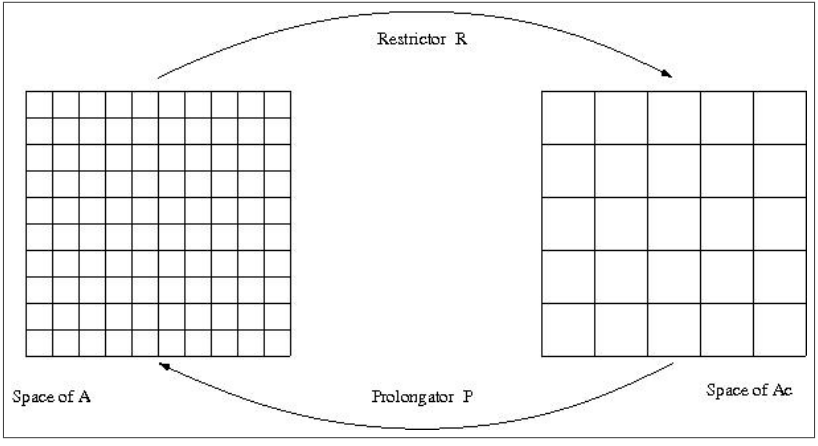
$$A_i^\delta = R_i^\delta A (R_i^\delta)^T$$

Table of Contents

3 Multilevel preconditioners

Multilevel Corrections

3 Multilevel preconditioners



Project the problem onto a new space with a coarser discretization, then use its solution onto the original fine space.

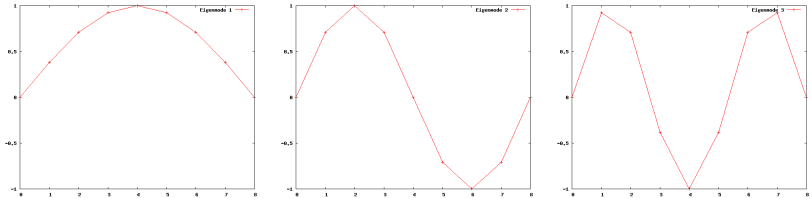
Multilevel Corrections

Basic idea:

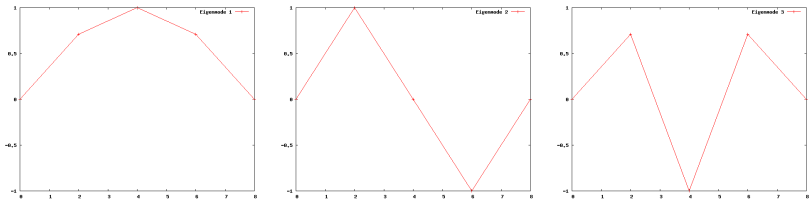
- "Smoothers" (e.g. Jacobi) are efficient in damping high-frequency components of the error;
- A low-frequency component appears as a high-frequency component when subsampled (i.e. projected onto a coarse grid);
- Therefore: project onto the coarse grid, apply a "smoother", project back onto the fine grid;
- Generalize to $n > 2$ levels.

Algebraic Multigrid: extension to a purely "topological" situation, i.e. no assumptions on the underlying geometry.

Eigenvectors of laplacian on 1D fine grid



Eigenvectors of laplacian on 1D coarse grid



Multilevel Corrections

- Algebraic coarsening: set of indices of coarse grid W^0 is obtained by aggregation;
- Definition of restriction/prolongation operatoris:
  1. Building the prolongator $\tilde{P} : W^0 \to W$

$$\tilde{P}_{ij} = \begin{cases} 1 & if \quad (vertex\ i) \in (aggregate\ j) \\ 0 & otherwise \end{cases}$$

  2. Application of a smoother (Jacobi)

$$P = SP = (I - \omega D^{-1} A)\tilde{P}$$

- Building the coarse matrix

$$A^0 = R^0 A P^0$$

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:

  Jacobi/ℓ_1–Jacobi Scaling with a diagonal, simple but suitable for GPUs

  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 $L_{pp} = \mathrm{tril}(A_{pp})$ then:

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

  AINV Block-Jacobi with an approximate inverse factorization on the block applied via matrix-vector product $\Rightarrow$ suitable for GPUs

What is our *recipe*?

3 Multilevel preconditioners

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

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

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

$\mathbf{w} \in \text{Range}(P)$: 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.,

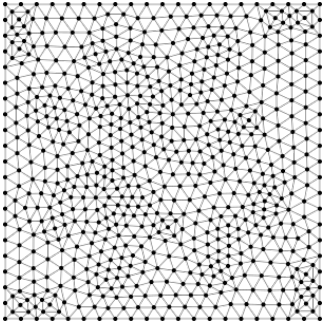
Convergence rate of *compatible relaxation*: $\rho_f = \|I - M_f^{-1} A_f\|_{A_f} \ll 1$

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.

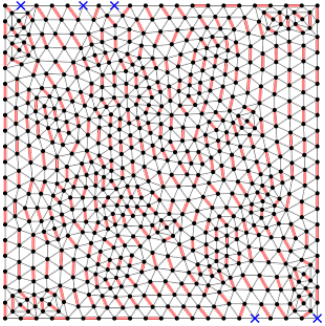


Weighted graph matching

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.

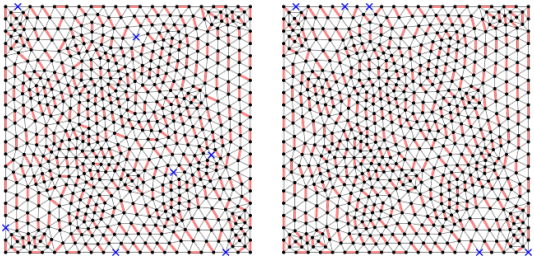


We divide the index set into matched vertices $= \bigcup_{i=1}^{n_p} \mathcal{I}_i$, with $\mathcal{I}_i \cap \mathcal{I}_j = \emptyset$ if $i \neq j$, and unmatched vertices, i.e., n_s singletons $\mathcal{G}_i$.

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, CAMWA, 2023



[H]Locally Dominant Edge Graph $G = (,)$, Weights $\hat{A}$
$\mathcal{M} \leftarrow \emptyset$

while $\neq \emptyset$ **do** Take a locally dominant edge $(i,j) \in$, 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 Matching $\mathcal{M}$

Table of Contents

4 Numerical Examples

Weak Scalability - CPU/GPU Runs - Piz Daint

4 Numerical Examples

- 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

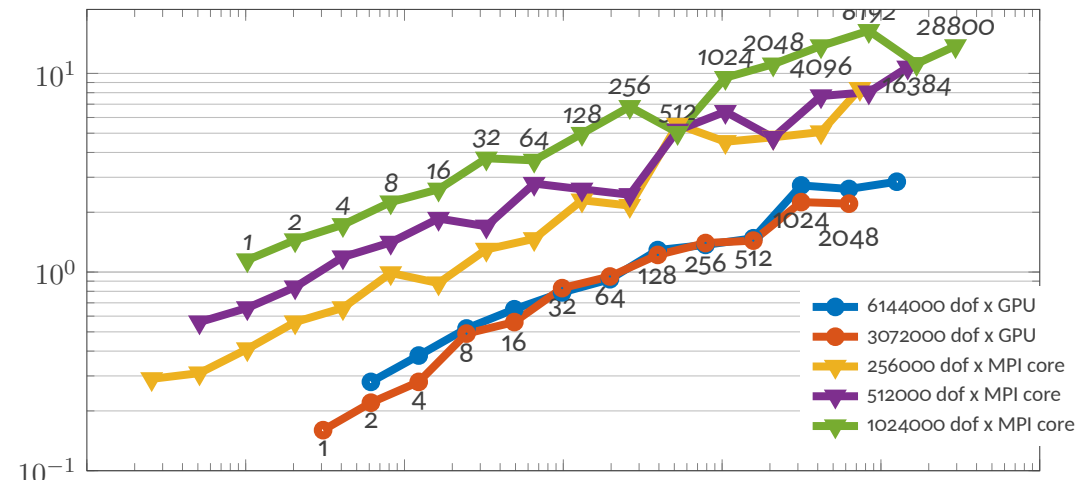
There are two common notions of scalability:

- 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.

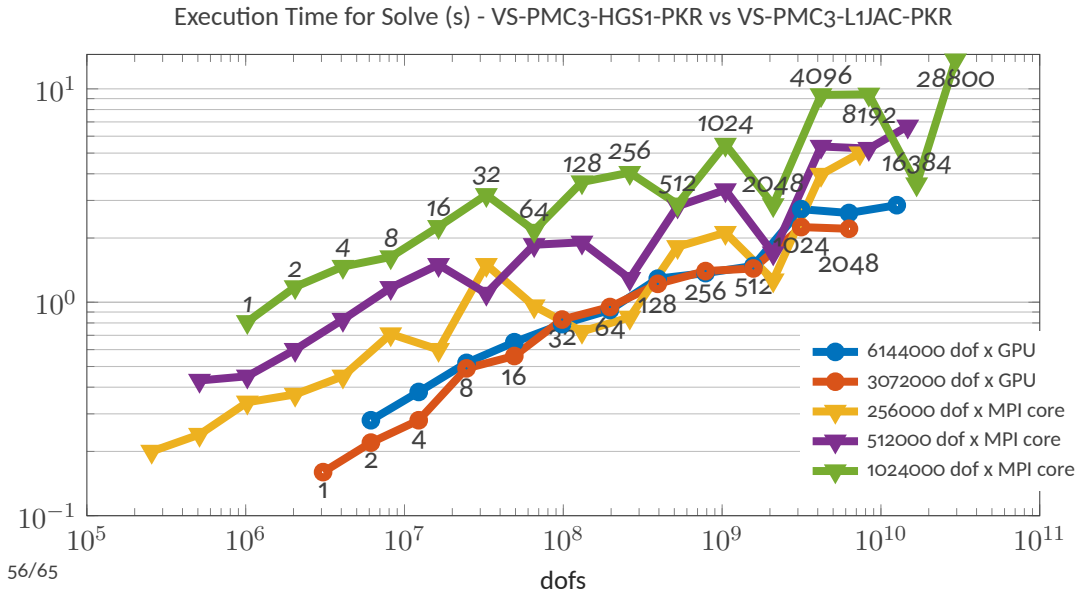
Weak Scalability - CPU/GPU Runs - Piz Daint

4 Numerical Examples



Weak Scalability - CPU/GPU Runs - Piz Daint

4 Numerical Examples



Weak Scaling — Runs on Leonardo

- In **weak scaling**, both the **number of computing units** and the **problem size** are **increased**: *constant workload per computing unit*.
- We use 8×10^6 unknowns per GPU, *i.e.*, 3.2×10^7 unknowns per node.

We use the following resources:

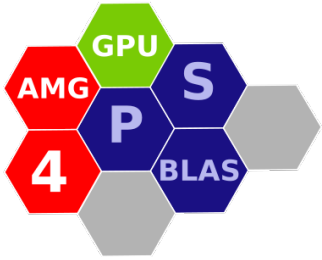
- Number of GPUs from 1 to 8192,
- GPUs x Node 4 (1 MPI Task x GPU, 8 CPUs per Task)
- Pure MPI: 32 MPI Tasks per Node

Within the software framework:

- Compilers: `gcc/11.3.0`
- MPI: `openmpi/4.1.4`
- CUDA `compilation tools, release 11.8, V11.8.89`

D'Ambra, Durastante, Filippone: PSCTOOLKIT, Minisymposium on Sparse Linear Solvers for Computational Science at Extreme Scales, ICIAM 2023, Tokyo

- **Aggregation:** VBM, **Cycle:** V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver:** PCG + ℓ_1-Jacobi,
- **Aggregation:** Smoothed Matching, **Cycle:** V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver:** PCG + ℓ_1-Jacobi,
- **Aggregation:** Matching, **Cycle:** Variable V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver:** PCG + ℓ_1-Jacobi,
- **Coarsening:** Classical Algebraic Multigrid, **Cycle:** V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver:** ℓ_1-Jacobi, 40 sweeps
- **Aggregation:** (Iterative) Parallel Graph Matching, **Cycle:** V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver:** ℓ_1-Jacobi, 40 sweeps



NVIDIA/**AMGX**

Distributed multigrid linear solver library on GPU

Operator Complexity

A first measure of the **theoretical computational cost** and of the **memory footprint** of the different algorithms is given by the **operator complexity**:

$$\text{opc} = \frac{\sum_{l=0}^{n_{\text{lev}}} \text{nnz}(A_l)}{\text{nnz}(A)} =$$

"the total number of nonzeroes in the linear operators on all grids divided by the number of nonzeroes in the fine grid operator"

Computing Units	VBM	Matching Smoothed	Matching Unsmoothed	AMGX Classical	AMGX Matching
1	1,575	1,894	1,142	4,45456	1,27979
2	1,578	1,905	1,142	4,43576	1,31187
4	1,58	1,915	1,143	4,51377	1,33117
8	1,583	1,917	1,142	4,52376	1,33162
16	1,584	1,925	1,143	4,51239	1,32133

Operator Complexity

4 Numerical Examples

A first measure of the **theoretical computational cost** and of the **memory footprint** of the different algorithms is given by the **operator complexity**:

$$\text{opc} = \frac{\sum_{l=0}^{n_{\text{lev}}} \text{nnz}(A_l)}{\text{nnz}(A)} =$$

"the total number of nonzeroes in the linear operators on all grids divided by the number of nonzeroes in the fine grid operator"

Computing Units	VBM	Matching Smoothed	Matching Unsmoothed	AMGX Classical	AMGX Matching
32	1,584	1,93	1,143	4,49595	1,31887
64	1,587	1,93	1,143	4,50135	1,31914
128	1,588	1,936	1,143	4,49925	1,31421
256	1,587	1,905	1,144	4,49252	1,31314
512	1,589	1,937	1,143	4,4952	1,31329
1024	1,588	1,942	1,144	4,49503	1,31091

Operator Complexity

A first measure of the **theoretical computational cost** and of the **memory footprint** of the different algorithms is given by the **operator complexity**:

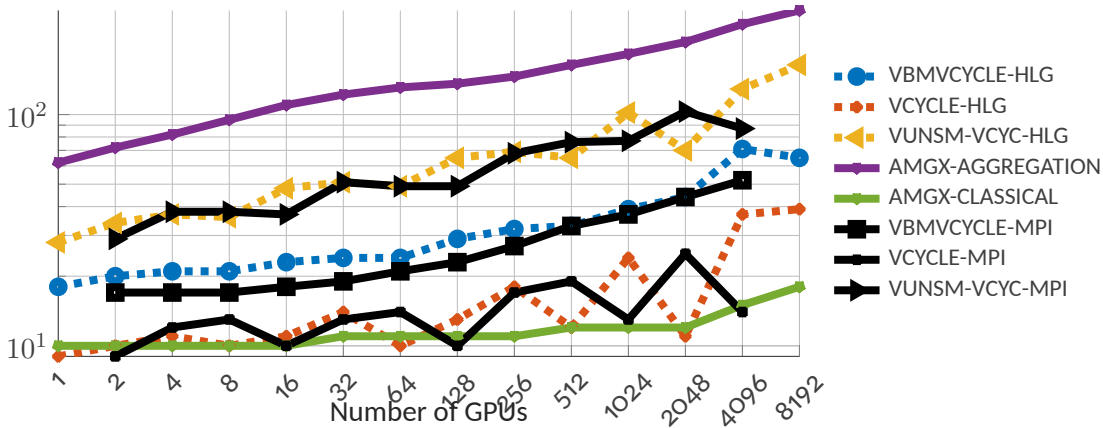
$$\text{opc} = \frac{\sum_{l=0}^{n_{\text{lev}}} \text{nnz}(A_l)}{\text{nnz}(A)} =$$

"the total number of nonzeroes in the linear operators on all grids divided by the number of nonzeroes in the fine grid operator"

Computing Units	VBM	Matching Smoothed	Matching Unsmoothed	AMGX Classical	AMGX Matching
2048	1,59	1,939	1,143	4,4921	1,31041
4096	1,588	1,906	1,144	4,49354	1,31049
8192			1,144	4,49371	1,30932

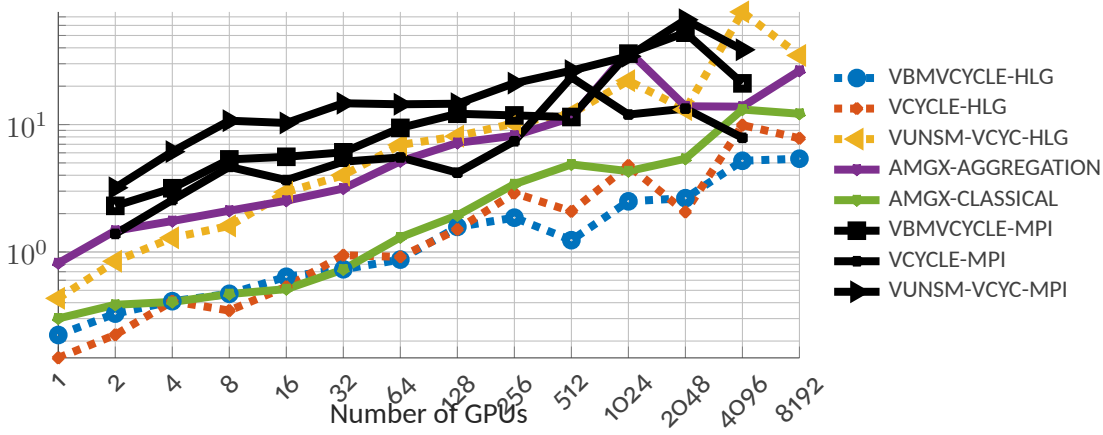
Algorithmic Scalability: Iteration Count

4 Numerical Examples

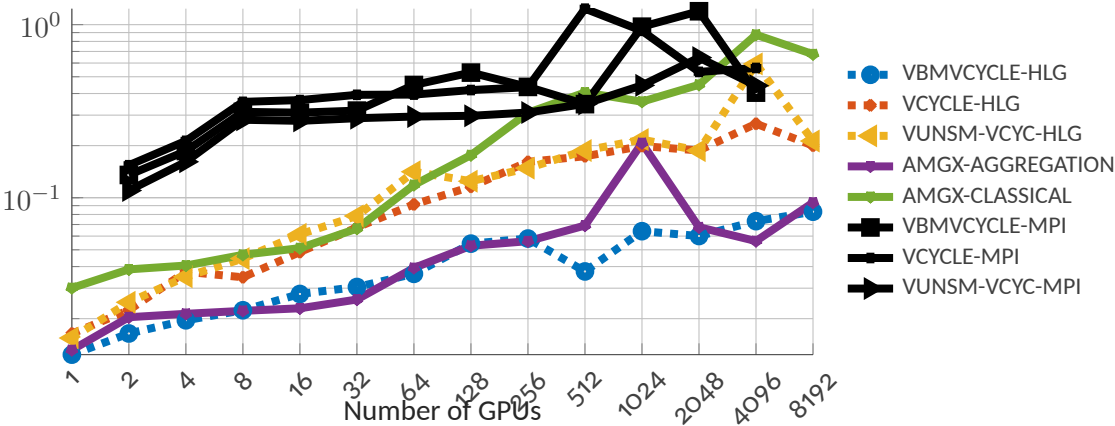


Implementation Scalability: Solve Time (s)

4 Numerical Examples

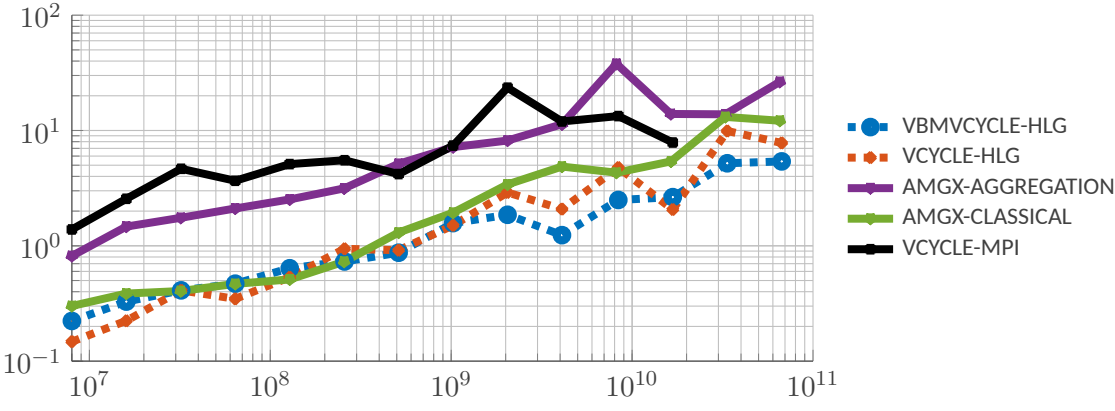


Implementation Scalability: Time × Iteration (s)



Best Solve Time vs Global System Size

4 Numerical Examples



Largest System Size is: $67121414144 \approx 7 \times 10^{10}$.

References

5 Bibliography

[1] D. Barbieri et al. “Design Patterns for Scientific Computations on Sparse Matrices”. In: *Proc. of HPSS 2011*. Springer, 2011.

[2] S. Bellavia, D. Bertaccini, and B. Morini. “Nonsymmetric Preconditioner Updates in Newton-Krylov Methods for Nonlinear Systems”. In: *SIAM J. Sci. Comput.* 33.5 (2011), pp. 2595–2619.

[3] M. Benzi, J. K. Cullum, and M. Tůma. “Robust Approximate Inverse Preconditioning for the Conjugate Gradient Method”. In: *SIAM J. Sci. Comput.* 22.4 (2000), pp. 1318–1332.

[4] D. Bertaccini and S. Filippone. “Sparse approximate inverse preconditioners on high performance GPU platforms”. In: *Computers & Mathematics with Applications* 71.3 (2016), pp. 693–711. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2015.12.008. URL: https://www.sciencedirect.com/science/article/pii/S0898122115005763.

References

5 Bibliography

[5] A. C. N. van Duin. “Scalable Parallel Preconditioning with the Sparse Approximate Inverse of Triangular Matrices”. In: *SIAM J. Matrix Anal. Appl.* 20.4 (1999), pp. 987–1006.

[6] G. H. Golub and C. F. Van Loan. *Matrix computations*. Fourth. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2013, pp. xiv+756. ISBN: 978-1-4214-0794-4.

[7] M. Naumov. *Incomplete-LU and Cholesky Preconditioned Iterative Methods Using CUSPARSE and CUBLAS*. Tech. rep. NVIDIA Corporation, June 2011.

[8] Y. Saad. “Preconditioning techniques for nonsymmetric and indefinite linear systems”. In: *J. Comput. Appl. Math.* 24.1–2 (Nov. 1988), pp. 89–105. ISSN: 0377-0427. DOI: 10.1016/0377-0427(88)90345-7. URL: https://doi.org/10.1016/0377-0427(88)90345-7.