Scalable Two-Level Additive Schwarz Preconditioning for Krylov Solvers of Poisson-Type Problems

This is the report for the final Domain Decomposition project developed for the AMSC course (2025–26) held by prof. L. Formaggia

Riccardo Selis
HPC Engineering
Politecnico di Milano
Milan, Italy
10707433@polimi.it

***Abstract*—Many discretizations of elliptic partial differential equations lead to large sparse symmetric positive definite linear systems whose solution dominates runtime. This report studies a diffusion–reaction model posed in one, two, and three spatial dimensions, discretized by second-order finite differences on structured grids, and solved with Conjugate Gradient (CG). We quantify how convergence and time-to-solution evolve when progressively strengthening the solver: (i) baseline CG (identity preconditioning), (ii) one-level overlapping Additive Schwarz (AS) with inexact local SSOR block solves, and (iii) two-level Additive Schwarz (AS2) that adds a coarse correction and is executed with MPI parallelism. Sequential experiments on the 1D Poisson benchmark highlight the classical trade-off between iteration reduction and per-iteration preconditioner cost, and show that the optimal block partition depends on problem size. Across dimensions, AS2 yields consistent iteration reductions, often by one to two orders of magnitude, and when combined with MPI it delivers large end-to-end time reductions on the largest cases (with the strongest improvements observed on fine 1D grids and on challenging 2D/3D variable-coefficient tests). Strong and weak scaling results show that AS2 controls iteration growth effectively, while parallel efficiency at high ranks is increasingly limited by global synchronizations and coarse-level collectives.**

I. Introduction

Many PDE-based models lead, after discretization, to large sparse linear systems whose solution dominates the overall runtime. This report considers a parametric elliptic boundary value problem posed in a spatial dimension $d \in \{1, 2, 3\}$, discretized by finite differences on structured grids, and solved with Krylov iterative methods. The main goal is to quantify how solver performance (iteration counts, time-to-solution, and time-per-iteration) changes when moving from a baseline Conjugate Gradient (CG) solver to preconditioned variants, with particular emphasis on domain-decomposition (Schwarz-type) preconditioning.

A. Physical problem in generalized form

Let $d \in \{1, 2, 3\}$ and let $\Omega \subset \mathbb{R}^d$ be a bounded domain with boundary $\partial\Omega$. Given a diffusion coefficient $\mu > 0$, a reaction coefficient $c \geq 0$, a forcing term $f : \Omega \to \mathbb{R}$, and boundary data $g : \partial\Omega \to \mathbb{R}$, we consider the diffusion–reaction model:

$$\begin{cases} -\mu\,\Delta u(\mathbf{x}) + c\,u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \partial\Omega, \end{cases} \tag{1}$$

Under standard assumptions (e.g., $\mu > 0$, $c \geq 0$, and Dirichlet boundary conditions), problem (1) is well-posed and its discrete counterpart yields an SPD linear system, making it a natural candidate for CG and preconditioned CG. Equation (1) is representative of elliptic operators occurring in many applications. From a numerical standpoint, it captures the key computational features we aim to study: sparse operators, mesh-dependent conditioning, and strong sensitivity of solver performance to the choice of preconditioner.

B. Mathematical modeling via finite differences

In the implementation, Ω is an axis-aligned box

$$\Omega = \prod_{\ell=1}^{d} [a_\ell, b_\ell] \subset \mathbb{R}^d,$$

discretized by a structured grid with n_ℓ *interior* points in direction $\ell = 1, \ldots, d$. The mesh spacings are $h_\ell > 0$; when not provided explicitly, they are set from the domain length as

$$h_\ell = \frac{b_\ell - a_\ell}{n_\ell + 1}. \tag{2}$$

Interior nodes are indexed by a multi-index

$$\mathbf{i} = (i_1, \ldots, i_d), \qquad 1 \leq i_\ell \leq n_\ell,$$

and we denote by $u_{\mathbf{i}} \approx u(\mathbf{x}_{\mathbf{i}})$ the discrete unknown at the interior node with coordinates

$$\mathbf{x}_{\mathbf{i}} = \big(a_1 + i_1 h_1,\ a_2 + i_2 h_2,\ a_3 + i_3 h_3\big),$$

with inactive components omitted when $d < 3$. Dirichlet boundary values are associated with the (implicit) boundary layers $i_\ell = 0$ and $i_\ell = n_\ell + 1$.

The codebase used in this project is publicly available at github.com/riccardoselis00/domain-decompositions.

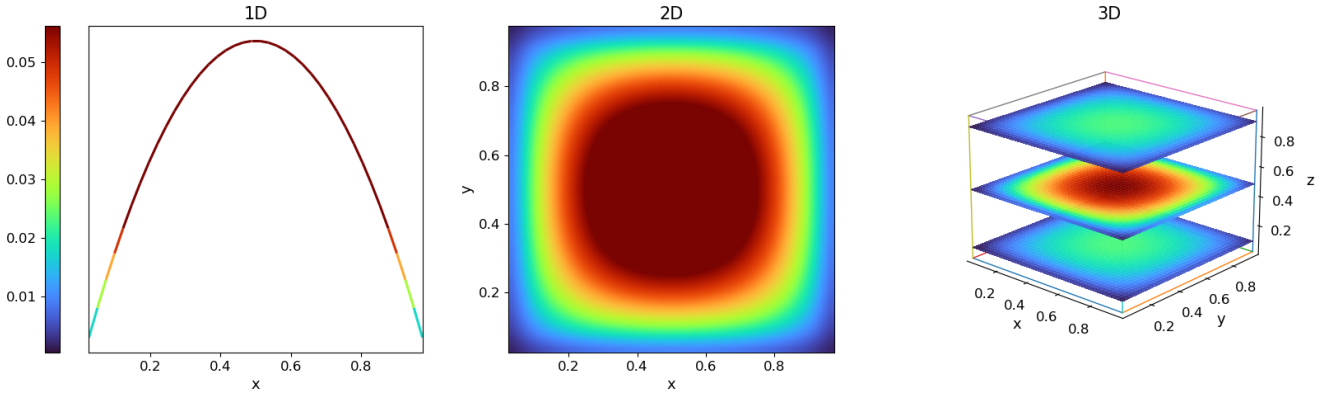

Fig. 1. Solution of 1,2,3 D poisson problem $\mu = 1, c = 0, f(\mathbf{x}) = 1.0, g(\mathbf{x}) = 0$

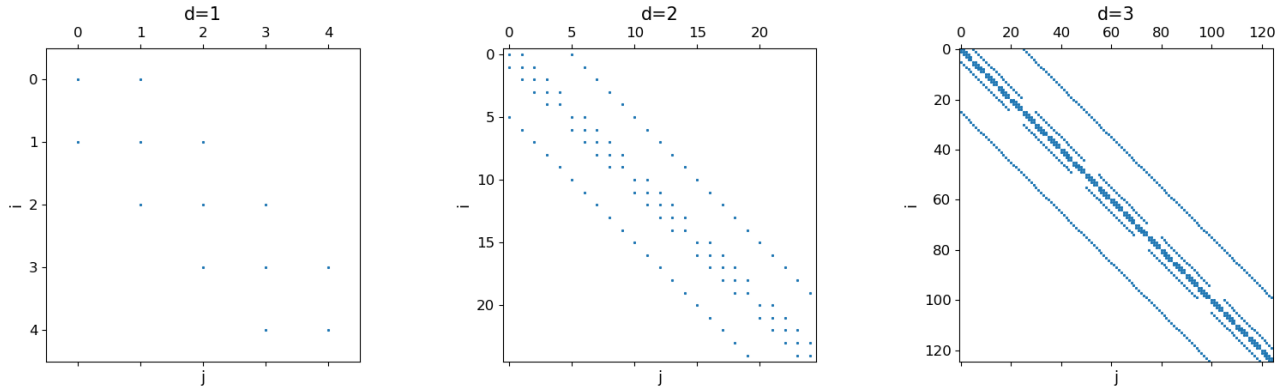

Fig. 2. Matrices of 1,2,3 D Poisson problem $\mu = 1, c = 0, f(\mathbf{x}) = 1.0, g(\mathbf{x}) = 0$, with $N_x = 5$

a) Dimension-agnostic Laplacian stencil.: Let $\mathbf{e}_\ell$ be the ℓ-th canonical unit vector in $\mathbb{R}^d$. At an interior node, the second-order central finite-difference approximation of the Laplacian is:

$$\Delta u(\mathbf{x_i}) \approx \sum_{\ell=1}^{d} \frac{u_{\mathbf{i}+\mathbf{e}_\ell} - 2u_{\mathbf{i}} + u_{\mathbf{i}-\mathbf{e}_\ell}}{h_\ell^2}. \tag{3}$$

Substituting (3) into (1) yields, for interior nodes, linear equations of the form:

$$-\mu \sum_{\ell=1}^{d} \frac{u_{\mathbf{i}+\mathbf{e}_\ell} - 2u_{\mathbf{i}} + u_{\mathbf{i}-\mathbf{e}_\ell}}{h_\ell^2} + c\, u_{\mathbf{i}} = f_{\mathbf{i}}. \tag{4}$$

Depending on d, (4) corresponds to the familiar 3-point ($d = 1$), 5-point ($d = 2$), or 7-point ($d = 3$) stencil on Cartesian grids.

b) Boundary conditions.: Dirichlet conditions are imposed by elimination: whenever a neighbor of an interior node lies on the boundary, its contribution is shifted to the right-hand side using the prescribed boundary value $g(\mathbf{x})$. If the Dirichlet function is not provided, the implementation defaults to homogeneous data ($g \equiv 0$).

c) Algebraic system.: Collecting interior unknowns in a vector $\mathbf{u} \in \mathbb{R}^n$ (with n the number of degrees of freedom) via a consistent ordering (with the x-index varying fastest), the discretization yields the sparse linear system:

$$A\mathbf{u} = \mathbf{b}, \tag{5}$$

where $A \in \mathbb{R}^{n \times n}$ is sparse and SPD for $\mu > 0$ and $c \geq 0$ with Dirichlet BC. The degree-of-freedom count (interior nodes) is

$$n = \begin{cases} n_x, & d = 1, \\ n_x n_y, & d = 2, \\ n_x n_y n_z, & d = 3, \end{cases} \tag{6}$$

and the sparsity pattern depends on d and on the grid connectivity induced by (4).

C. Baseline results: CG on the constant-coefficient Poisson benchmark

We first establish a reference baseline by solving the constant-coefficient diffusion problem

$$-\mu \Delta u + c\, u = f \quad \text{in } \Omega = (0,1)^d, \qquad u = g \quad \text{on } \partial\Omega, \tag{7}$$

with $\mu = 1$, $c = 0$, constant forcing $f \equiv 1$, and homogeneous Dirichlet data $g \equiv 0$. A second-order finite-difference discretization on a uniform grid yields an SPD linear system $A\mathbf{u} = \mathbf{b}$ with $n = N_x^d$ interior unknowns and a sparse

stencil structure (tridiagonal in 1D, five-point in 2D, seven-point in 3D). As a baseline solver we use PCG with the identity preconditioner ($M = I$), i.e., unpreconditioned CG, starting from the zero initial guess and stopping with the relative residual criterion using $\varepsilon = 10^{-12}$ Table I reports representative measurements extracted from the CSV logs.

TABLE I
BASELINE (PCG WITH $M = I$): CONSTANT-COEFFICIENT POISSON BENCHMARK ACROSS DIMENSIONS. HERE $n = N_x^d$ COUNTS INTERIOR UNKNOWNS; T_{SOLVE} IS THE KRYLOV-LOOP TIME.

d	N_x	n	nnz	iters	T_{solve} [s]	$T_{\text{iter}}[ms]$
1	2000	2000	5998	1000	4.8e-02	0.048
1	10000	10000	29998	5000	3.49e-01	0.070
1	40000	40000	119998	20000	3.886	0.194
2	50	2500	12300	112	5.0e-03	0.043
2	200	40000	199200	450	1.78e-01	0.395
2	300	90000	448800	681	6.68e-01	0.980
3	10	1000	6400	29	3.10e-04	0.011
3	25	15625	105625	80	2.97e-02	0.371
3	35	42875	292775	111	6.21e-02	0.560

a) Observed trends.: Unpreconditioned CG exhibits the expected degradation under mesh refinement: the condition number of the discrete Laplacian grows as $\kappa(A) = \mathcal{O}(h^{-2})$, hence the iteration count to reach a fixed tolerance grows approximately as $k_{\text{final}} = \mathcal{O}(\sqrt{\kappa}) = \mathcal{O}(h^{-1}) = \mathcal{O}(N_x)$. This explains why the iteration count is primarily controlled by the grid resolution per direction (N_x), rather than by the total number of nonzeros. In our runs, the ratio k_{final}/N_x is roughly constant in 2D and 3D, while the 1D case shows $k_{\text{final}} \approx N_x/2$ for this specific right-hand side (constant forcing), which excites only a structured subset of eigenmodes.

b) Preconditioning.: Krylov methods such as CG do not change the matrix A explicitly; instead, they repeatedly apply A to vectors and build a sequence of approximations whose convergence rate depends on the spectrum (and in particular on the condition number) of A. For elliptic operators discretized on fine meshes, A becomes increasingly ill-conditioned, so CG may require many iterations.

To accelerate convergence, we introduce a *preconditioner* $M \approx A$ that is *cheaper to invert* than A itself, and we solve an equivalent system in which the linear operator is better conditioned:

$$M^{-1}A\mathbf{u} = M^{-1}\mathbf{b}. \tag{8}$$

Intuitively, M^{-1} acts as a *computationally affordable approximate inverse* of A: it rescales the error components associated with "difficult" directions of A (typically those linked to small eigenvalues), so that the eigenvalues of $M^{-1}A$ are more tightly clustered. Since the CG/PCG convergence rate improves when eigenvalues are clustered and $\kappa(M^{-1}A)$ is smaller, a good preconditioner reduces the iteration count k_{final} for a fixed tolerance.

When A is SPD, M is chosen SPD as well so that PCG preserves symmetry and guarantees convergence. In practice, the effectiveness of preconditioning must be assessed end-to-end: stronger M usually decreases iterations, but it also increases (i) *setup cost* to build M and (ii) *apply cost* to compute $M^{-1}\mathbf{v}$ inside each iteration. The best preconditioner is therefore the one that minimizes the overall time-to-solution, not necessarily the iteration count alone.

II. SEQUENTIAL ADDITIVE SCHWARZ PRECONDITIONING

This chapter presents the first implementation of an *overlapping Additive Schwarz* (AS) preconditioner in a purely sequential setting (no MPI, no coarse operator). The goal is twofold: (i) validate correctness and parameter sensitivity on a controlled benchmark, and (ii) establish a quantitative reference (iterations and time-to-solution) before introducing parallelism and multilevel components in later chapters.
To study preconditioning effects in a controlled and inexpensive setting, we use the 1D Poisson problem as the main reference case for parameter sweeps (block size, overlap, and stopping tolerance). In 1D we can push N_x to very large values at modest memory cost, obtain stable timing measurements, and isolate how the preconditioner changes iteration counts and time-to-solution without the additional complexity of higher-dimensional assemblies.

A. From domain decomposition to a preconditioner

We consider the SPD linear system

$$A\mathbf{u} = \mathbf{b}, \tag{9}$$

arising from the finite-difference discretization of the constant-coefficient Poisson model introduced in Section I-C. Unpreconditioned CG converges, but as the mesh is refined the number of iterations becomes large. Additive Schwarz addresses this by decomposing the set of unknowns into (overlapping) subdomains and combining inexpensive *local* solves into a global preconditioner.

a) Overlapping decomposition (algebraic view).: Let the index set of interior unknowns be $\mathcal{I} = \{1, \ldots, n\}$. In the sequential baseline we construct a partition into B contiguous blocks in index space. Let

$$0 = s_1 < s_2 < \cdots < s_{B+1} = n$$

define a non-overlapping partition. For a prescribed overlap $\delta \geq 0$ (in the experiments $\delta = 1$), each block is extended to the overlapped index set

$$\mathcal{I}_i \;=\; \{\, s_i - \delta,\; s_i - \delta + 1,\; \ldots,\; s_{i+1} + \delta - 1 \,\} \cap \mathcal{I}.$$

By construction,

$$\bigcup_{i=1}^{B} \mathcal{I}_i = \mathcal{I}, \qquad \mathcal{I}_i \cap \mathcal{I}_{i+1} \neq \emptyset \text{ if } \delta > 0.$$

For each subdomain define a restriction operator R_i that extracts the entries of a global vector on $\mathcal{I}_i$, and the corresponding local matrix (principal submatrix)

$$A_i \;=\; R_i A R_i^T.$$

In the implementation, each A_i is assembled by extracting from A only the nonzeros whose row and column indices both belong to $\mathcal{I}_i$.

b) Local correction with inexact block solves (SSOR sweeps).: Given a residual $\mathbf{r}$, the preconditioner computes a local correction $\mathbf{e}_i$ on each block that approximates the solution of

$$A_i \mathbf{e}_i = R_i \mathbf{r}.$$

Rather than solving this system exactly, we apply a fixed number m of Symmetric Successive Over-Relaxation (SSOR) sweeps on the local matrix A_i. Denoting by $\mathcal{S}_i$ the linear operator corresponding to "m SSOR sweeps with relaxation ω" on block i, we compute

$$\mathbf{e}_i \;=\; \mathcal{S}_i\,(R_i \mathbf{r}).$$

Operationally, one SSOR sweep consists of a forward SOR sweep followed by a backward SOR sweep on A_i, starting from the zero initial guess, repeated m times.

c) What SSOR does (matrix splitting view).: Let A_i be written as the standard splitting

$$A_i = D_i + L_i + U_i,$$

where D_i is diagonal, L_i strictly lower triangular and U_i strictly upper triangular. A *forward* SOR step for $A_i \mathbf{x} = \mathbf{b}$ with relaxation ω is

$$(D_i + \omega L_i)\,\mathbf{x}^{(t+\frac{1}{2})} = \omega \mathbf{b} - \big[\omega U_i + (\omega - 1) D_i\big]\mathbf{x}^{(t)}.$$

A *backward* SOR step applies the analogous update to the transposed splitting (sweeping from the last unknown to the first):

$$(D_i + \omega U_i)\,\mathbf{x}^{(t+1)} = \omega \mathbf{b} - \big[\omega L_i + (\omega - 1) D_i\big]\mathbf{x}^{(t+\frac{1}{2})}.$$

Combining forward and backward sweeps yields one SSOR sweep. In preconditioning terms, SSOR defines an *approximate inverse action* of A_i by efficiently damping error components through inexpensive triangular-like sweeps that exploit the local coupling encoded in L_i and U_i.

d) Impact of the relaxation parameter ω.: The parameter ω controls how aggressively each SOR update moves along the Gauss–Seidel direction. At the level of a single scalar update, SOR can be interpreted as a convex combination of the previous iterate and the Gauss–Seidel update:

$$x^{\text{new}} \;=\; (1-\omega)\,x^{\text{old}} \;+\; \omega\,x^{\text{GS}}.$$

Hence:

- $\omega = 1$ gives the symmetric Gauss–Seidel (SGS) sweep (no over-relaxation).
- $0 < \omega < 1$ under-relaxes: updates are more conservative and typically damp high-frequency errors more smoothly but with weaker per-sweep progress.
- $1 < \omega < 2$ over-relaxes: updates are more aggressive and can reduce the number of sweeps needed to achieve a comparable local correction, strengthening the effective action of $\mathcal{S}_i$.

In our baseline AS implementation, ω therefore tunes the *strength* of the inexact local solves: larger ω (up to the usual SOR stability range) tends to yield a correction closer to the exact block solve for a fixed number of sweeps m, potentially reducing the outer PCG iteration count, while also changing the cost and numerical behavior of each local sweep. In the experiments we use a small number of sweeps (default $m = 1$) and an aggressive relaxation value (default $\omega = 1.95$) to obtain a lightweight yet effective block correction.

e) Additive assembly of the global preconditioned residual.: The global preconditioned vector $\mathbf{z}$ is obtained by prolongation and summation of the block corrections:

$$\mathbf{z} \;=\; \sum_{i=1}^{B} R_i^T \mathbf{e}_i \;=\; \left(\sum_{i=1}^{B} R_i^T \mathcal{S}_i R_i\right)\mathbf{r}.$$

This defines the baseline one-level Additive Schwarz preconditioner used in this work:

$$M_{\text{AS}}^{-1} \;=\; \sum_{i=1}^{B} R_i^T \mathcal{S}_i R_i,$$

where each $\mathcal{S}_i$ represents a fixed SSOR-based approximate inverse action on the local block matrix.

f) PCG with Additive Schwarz.: We employ left-preconditioned Conjugate Gradient. At each iteration k we form the residual

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{u}^{(k)},$$

apply the preconditioner

$$\mathbf{z}^{(k)} = M_{\text{AS}}^{-1}\mathbf{r}^{(k)},$$

and update the standard PCG recurrences using $\rho_k = (\mathbf{r}^{(k)})^T\mathbf{z}^{(k)}$. Convergence is assessed through the relative residual norm $\|\mathbf{r}^{(k)}\|_2/\|\mathbf{r}^{(0)}\|_2$.

B. Sequential implementation and parameters

a) Reference problem: 1D Poisson: For the first evaluation of AS we focus on the one-dimensional Poisson operator with homogeneous Dirichlet boundary conditions. This choice allows us to sweep parameters over large problem sizes at low memory cost, measure timings reliably, and isolate preconditioner behavior without additional complexity from 2D/3D assemblies. These results serve as the reference baseline for subsequent extensions (MPI distribution and coarse levels).

b) Subdomains ("blocks") and overlap: We partition the 1D set of unknowns into B contiguous blocks and enlarge each block by a fixed overlap of one grid point on each side (when available). In the experiments below the overlap is fixed to `OVERLAP=1`, and we vary the number of blocks $B \in \{4, 8, 16, 32\}$. Increasing B reduces the average subdomain size (cheaper local solves), but can weaken the preconditioner (more iterations). This is exactly the trade-off we want to quantify.

C. Results and discussion: effect of the number of blocks

Figure 3 summarizes the impact of B on both iteration counts and times. A compact numerical snapshot is reported in Table II, where we compare the best AS configuration (in total time) against the identity-preconditioned baseline (unpreconditioned CG).

TABLE II
SEQUENTIAL AS ON 1D POISSON (OVERLAP=1, $\varepsilon = 10^{-16}$). FOR EACH n, WE REPORT THE BEST NUMBER OF BLOCKS B (MINIMUM T_{TOT}) AND COMPARE AGAINST UNPRECONDITIONED CG.

n	B	iters$_{\text{CG}}$	T_{CG} [s]	iters$_{\text{AS}}$	T_{AS} [s]	speedup
10000	8	5001	0.377	215	0.071	5.27
20000	8	10001	1.13	387	0.250	4.52
40000	32	20001	3.90	778	0.724	5.38
80000	32	40001	14.6	1472	2.76	5.30
160000	32	80001	106.3	2912	16.37	6.49

a) Iteration reduction vs per-iteration overhead: Across all sizes, Additive Schwarz reduces the number of PCG iterations by roughly one order of magnitude (typically a factor $\approx$ 23–28 in our measurements). However, the time-to-solution speedup is smaller (roughly $4.5\times$ to $6.5\times$) because each PCG iteration with AS is more expensive: applying M_{AS}^{-1} requires solving B local subproblems per iteration. A useful way to see this trade-off is the *time per iteration*: unpreconditioned CG has a cheap iteration dominated by one SpMV, whereas AS adds the cost of local solves, so $T_{\text{solve}}/k_{\text{final}}$ increases even though k_{final} decreases substantially. In other words, AS improves convergence (iterations) but shifts some work inside each iteration.

b) Effect of the number of blocks B.: Varying B changes both convergence and cost:

- **Convergence effect:** increasing B (smaller subdomains) generally yields a slightly weaker preconditioner and thus slightly higher iteration counts. This is visible most clearly at smaller n.
- **Cost effect:** increasing B reduces the size of each local problem and can improve cache locality, so the preconditioner application becomes cheaper. For larger n, this reduction in per-iteration cost dominates the mild iteration increase.

This explains why the optimal choice shifts with n: for smaller problems ($n = 10^4$ and $2 \cdot 10^4$) the best total time is achieved around $B = 8$, while for larger problems ($n \geq 4 \cdot 10^4$) the best total time is obtained with $B = 32$.

c) Setup cost becomes negligible at scale.: In the sequential experiments, T_{setup} is already small compared to T_{solve}, and its relative weight decreases as n grows. For the largest case ($n = 160000$), setup is a tiny fraction of the total time, indicating that optimization efforts should prioritize the cost of applying the preconditioner inside the Krylov loop.

d) Implications for the project roadmap.: These results validate the early AS implementation and highlight the key trade-off that will guide the rest of the project: a good preconditioner must reduce iterations enough to compensate for its per-iteration overhead In a sequential setting, the overhead is paid serially. In the next chapter, we introduce MPI-based distribution of subdomains so that the local solves (the dominant AS work) can be performed concurrently. Finally, to target mesh-independent convergence for higher dimensions and large-scale runs, we will extend the one-level method with a coarse correction (two-level Schwarz), which addresses global low-frequency error components that one-level AS struggles to eliminate efficiently.

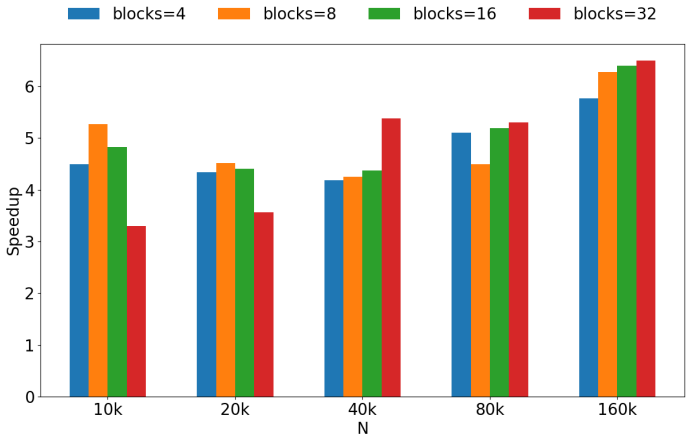

Fig. 3. Sequential Additive Schwarz on the 1D Poisson benchmark (OVERLAP=1, $\varepsilon = 10^{-16}$). (Left) PCG iterations vs number of blocks B. (Right) total time (or solve time) vs B for multiple problem sizes. The optimal B shifts toward larger values as n increases due to reduced per-iteration preconditioner cost.

III. PARALLEL IMPLEMENTATIONS AND COMPARISON WITH OTHER PRECONDITIONERS

Moving from the sequential experiments to MPI introduces two additional, practical dimensions: (i) the interaction between solver algorithms and hardware resources (cores, cache hierarchy, NUMA layout), and (ii) the impact of communication and synchronization on time-to-solution. Before presenting the MPI formulation of Additive Schwarz and the related experiments, we summarize the hardware platform used to collect the results, since cache and core topology strongly influence the cost balance between sparse matrix–vector products, local block solves, and global collectives.

A. Experimental platform and hardware topology

All MPI experiments were executed on a single-socket x86_64 workstation equipped with an Intel(R) Core(TM) Ultra 7 155H CPU and 31 GiB of main memory. The processor exposes 16 physical CPUs (threads), with one NUMA domain (`NUMA node(s)=1`), meaning that all ranks allocate memory from the same NUMA node and do not incur remote-NUMA access penalties.

a) Core and cache hierarchy.: Figure 4 highlights the cache hierarchy that is most relevant for sparse linear algebra. Each core has private L1 instruction/data caches (tens of KiB) and private or small-group L2 caches, while the machine provides a shared last-level cache (L3) of 24 MiB. In our solver, the dominant kernels (SpMV, dot products, and SSOR sweeps inside the Schwarz blocks) have low arithmetic intensity and are therefore sensitive to cache reuse and memory bandwidth. Although MPI is typically used across nodes, running multiple MPI ranks on a single socket is a controlled way to study algorithmic and communication effects

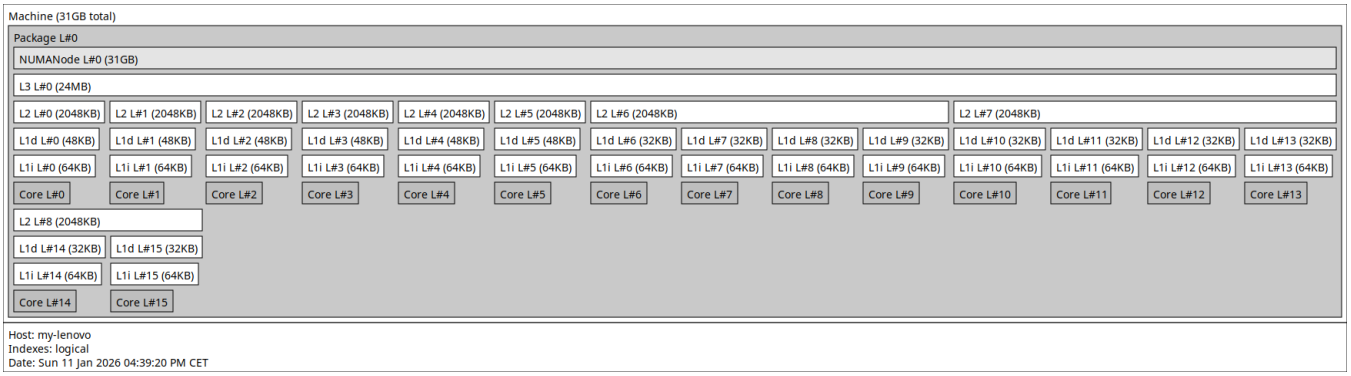

Fig. 4. Hardware topology of the experimental platform (output of `lstopo`). The machine exposes a single NUMA node with a shared last-level cache, and multiple cores with private L1/L2 resources. This layout influences the scaling of both SpMV and local preconditioner applications, and determines when memory bandwidth and synchronization dominate runtime.

while keeping interconnect variability out of the picture. On this platform, MPI collectives (e.g., `MPI_Allreduce`) still introduce synchronization points and data movement, but the costs are dominated by intra-node messaging and shared-cache/memory contention rather than network latency. This helps interpret the scalability results: strong scaling is initially excellent when computation dominates, while weak scaling efficiency degrades at larger P because (i) CG requires global reductions every iteration and (ii) AS2 adds extra collectives for coarse residual/operator aggregation.

B. Two-level Additive Schwarz and MPI parallelization

We consider the SPD linear system

$$A\mathbf{u} = \mathbf{b},$$

and extend the one-level Additive Schwarz preconditioner by adding a global *coarse correction*. This yields a two-level method that targets both local (high-frequency) and global (low-frequency) error components, while enabling parallel execution across MPI ranks.

1) Coarse operator and two-level additive formulation: Let the global set of unknown indices be $\mathcal{I} = \{1, \ldots, n\}$ and assume it is partitioned into M (possibly overlapping) subdomains $\mathcal{I}_1, \ldots, \mathcal{I}_M$ (in the implementation, $M = \texttt{nparts}$). Denote by R_i the restriction to subdomain i and by $A_i = R_i A R_i^T$ the corresponding local matrix. As in the one-level baseline, the local action is not an exact inverse: we apply a fixed number of SSOR sweeps on each A_i. Let $\mathcal{S}_i$ denote the linear operator corresponding to "m SSOR sweeps with relaxation ω" on block i. The one-level Schwarz preconditioner action is

$$P_S^{-1}\mathbf{r} \;=\; \sum_{i=1}^{M} R_i^T \,\mathcal{S}_i\, R_i\, \mathbf{r}.$$

a) Piecewise-constant coarse space (agglomeration with one DOF per subdomain).: To build the coarse correction we introduce a coarse space V_H of dimension $n_H = M$, with one coarse degree of freedom per subdomain. Algebraically, this corresponds to choosing coarse basis vectors that are *piecewise constant* on each subdomain:

$$\boldsymbol{\beta}^{(i)} \in \mathbb{R}^n, \qquad \beta_k^{(i)} = \begin{cases} 1, & k \in \mathcal{I}_i \text{ (non-overl. partition)}, \\ 0, & \text{otherwise.} \end{cases}$$

Stacking these vectors as columns defines the coarse prolongation matrix

$$R_H^T = \begin{bmatrix} \boldsymbol{\beta}^{(1)} & \boldsymbol{\beta}^{(2)} & \cdots & \boldsymbol{\beta}^{(M)} \end{bmatrix} \in \mathbb{R}^{n \times n_H},$$

while R_H acts as restriction from fine to coarse by aggregating (summing) fine-grid values on each subdomain:

$$(\mathbf{r}_H)_i = (R_H \mathbf{r})_i = \sum_{k \in \mathcal{I}_i} r_k.$$

With this choice, the coarse (agglomerated) operator is defined by the Galerkin projection

$$A_H = R_H A R_H^T, \qquad \mathbf{r}_H = R_H \mathbf{r}.$$

In the code, A_H is assembled without forming R_H explicitly: each nonzero a_{kj} contributes to the coarse entry (i, m) where $i = \text{part}(k)$ and $m = \text{part}(j)$, i.e., $A_H(i, m)$ is obtained by a weighted sum of entries of A consistent with the definition above.

b) Two-level additive Schwarz preconditioner.: The implemented two-level preconditioner is the *additive* combination

$$P_{2L}^{-1} \;=\; P_S^{-1} \;+\; R_H^T A_H^{-1} R_H.$$

Given a residual $\mathbf{r}$, the two-level preconditioning step computes

$$\mathbf{z} = P_{2L}^{-1}\mathbf{r} = \underbrace{\sum_{i=1}^{M} R_i^T \,\mathcal{S}_i\, R_i\, \mathbf{r}}_{\text{one-level (local) AS/SSOR}} \;+\; \underbrace{R_H^T \mathbf{y}_H}_{\text{coarse correction}}$$

In the implementation, A_H is small ($n_H = M$), and the coarse system is solved by a dense Cholesky-like factorization routine.

$$A_H \mathbf{y}_H = \mathbf{r}_H = R_H \mathbf{r}$$

2) MPI parallelization: partitioning, AS application, and PCG coupling:

a) Distributed data layout.: Under MPI, the unknowns are distributed across ranks. Each rank p owns a contiguous slice of global indices

$$\mathcal{I}^{(p)} = \{\ell_p, \ell_p + 1, \ldots, r_p\}, \qquad n_p = |\mathcal{I}^{(p)}|,$$

and stores the corresponding local residual $\mathbf{r}^{(p)} \in \mathbb{R}^{n_p}$ and local sparse matrix $A^{(p)}$ (its local rows). Global indices are recovered from local ones via $g = \ell_p + i_{\text{loc}}$, mirroring the code variables `m_ls` and `m_n_loc`. The mapping

$$\text{part}(g) \in \{1, \ldots, M\}$$

is built consistently across ranks (in the code: `m_rowToPart`), and is used both for coarse aggregation and for interpreting the coarse degrees of freedom.

b) Parallel one-level AS/SSOR.: The one-level preconditioner action on each rank is computed locally by applying SSOR sweeps on the extracted local blocks (submatrices). Denoting by $P_S^{-1,(p)}$ the local contribution on rank p, the global action is the sum of rank-local contributions assembled through the distributed vector representation. Conceptually, the dominant cost—the local block applications $\mathcal{S}_i$—is parallelized across MPI ranks since each rank works on its own local portion of the preconditioner application.

c) Parallel coarse operator construction and application.: Each rank forms a local contribution $A_H^{(p)}$ by accumulating coarse entries from its locally stored nonzeros:

$$A_H^{(p)}(i, m) \mathrel{+}= a_{kj} \quad \text{whenever} \quad i = \text{part}(k),\ m = \text{part}(j),$$

with (k, j) ranging over nonzeros of $A^{(p)}$ interpreted in global indexing. The global coarse matrix is obtained via an MPI reduction:

$$A_H = \sum_p A_H^{(p)} \qquad \text{(implemented with } \texttt{MPI\_Allreduce}\text{)}.$$

Similarly, each rank forms a local coarse residual

$$(\mathbf{r}_H^{(p)})_i = \sum_{g \in \mathcal{I}^{(p)} \cap \mathcal{I}_i} r_g,$$

and the global coarse residual is

$$\mathbf{r}_H = \sum_p \mathbf{r}_H^{(p)} \qquad \text{(again via } \texttt{MPI\_Allreduce}\text{)}.$$

After solving $A_H \mathbf{y}_H = \mathbf{r}_H$ (performed redundantly on each rank since A_H is small), each rank prolongs the coarse correction back to its local fine vector by adding, for each local DOF with global index g, the coarse value $\mathbf{y}_H(\text{part}(g))$.

d) MPI-coupled PCG.: The two-level preconditioner is used inside PCG through the standard update

$$\mathbf{z}^{(k)} = P_{2L}^{-1} \mathbf{r}^{(k)}.$$

In an MPI setting, the key PCG scalars require global reductions, e.g.

$$\rho_k = (\mathbf{r}^{(k)})^T \mathbf{z}^{(k)}, \qquad (\mathbf{p}^{(k)})^T A \mathbf{p}^{(k)}, \qquad \|\mathbf{r}^{(k)}\|_2,$$

which are computed as sums of rank-local dot products followed by `MPI_Allreduce`. Therefore, the per-iteration time is the combination of (i) local sparse operations (SpMV and preconditioner application) and (ii) global synchronization steps due to collective reductions. The two-level coarse operator adds additional collectives during preconditioning (coarse residual reduction) but aims to reduce the number of outer iterations by providing a global correction mechanism.

C. Results: AS vs AS2 under MPI

This subsection compares the *one-level* Additive Schwarz preconditioner (AS) with the *two-level* variant including the coarse correction (AS2), using the MPI implementation described in Section III-B. Unless otherwise stated, all runs solve the same reference PDE discretization (1D Poisson testbed), use the same overlap (here $\delta = 1$), the same stopping criterion based on the relative residual, and identical PCG tolerances and maximum iteration counts.

a) Experimental factors.: We vary: (i) the global problem size n (number of unknowns), (ii) the number of MPI ranks P, and (iii) the Schwarz partition parameter denoted as *number of blocks* M (shown as `nblocks` in plots; in the implementation it corresponds to `nparts`). For each configuration we report the number of outer PCG iterations needed to reach the target tolerance. Solve times and total times are discussed in the scalability subsection.

b) AS behavior: iterations depend on both n and the partition granularity.: Figure 5 shows that one-level AS is effective at reducing iterations relative to an unpreconditioned baseline, but its iteration counts remain sensitive to the partition granularity. At fixed n and P, moving from a coarse partition (small M) to a finer partition (large M) changes the preconditioner structure and the balance between local and global error components. In particular, for large problem sizes the one-level method exhibits significantly larger iteration counts for larger M, a classical motivation for introducing a coarse level in Schwarz-type methods.

c) AS2 effect: coarse correction becomes increasingly beneficial as n and M grow.: The two-level preconditioner AS2 adds a global correction step via the coarse operator $R_H^T A_H^{-1} R_H$ (Section III-B1), which targets slowly-varying (global) error components that are not efficiently eliminated by purely local block corrections. This effect is visible in the iteration curves: the advantage of AS2 becomes more pronounced for larger n and for larger block counts M.

A similar improvement is observed for intermediate sizes (e.g., $n = 160{,}000$: 2881 vs 682 at $M = 32$). This aligns with the theory: when the partition is refined (larger M), a one-level method increasingly relies on local corrections, while the coarse solve reintroduces a global coupling mechanism and mitigates the growth of iteration counts.

d) MPI ranks: iteration trends and diminishing returns.: Across all panels in Figure 5, increasing the number of MPI ranks P generally reduces the iteration count for both AS and AS2, with diminishing returns at higher P. This is consistent with the combined effect of (i) smaller local work per rank in

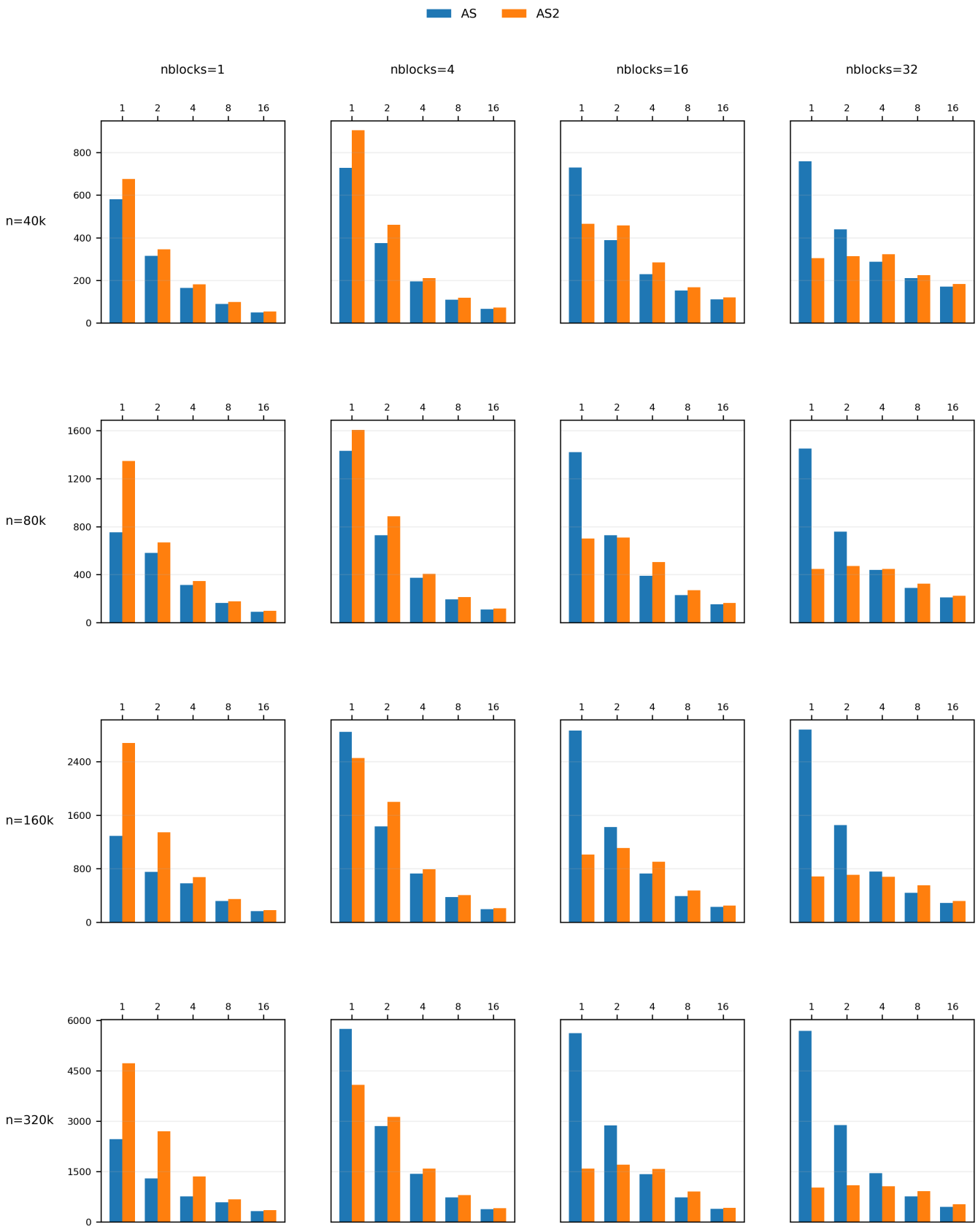

Fig. 5. PCG iteration counts for one-level AS (blue) and two-level AS2 (orange), as a function of MPI ranks P and number of Schwarz blocks M (panels). Each row corresponds to a different global problem size n.

the preconditioner application, and (ii) the fact that the global convergence is ultimately limited by the hardest error modes, which are precisely those targeted by the coarse correction in AS2.

For moderate parallelism, AS2 tends to retain a clear advantage in the most challenging regimes (large n, large M). For instance, at $n = 320{,}000$ and $M = 32$, going from $P = 1$ to $P = 4$ the iteration counts are

$$P = 4: \text{ AS } 1449 \text{ iters vs AS2 } 1061,$$

still reflecting a substantial reduction from the coarse correction. As P increases further, the iteration curves tend to converge, and the differences become smaller relative to other costs (communication and coarse collectives), which will be assessed in the scalability study.

e) Summary and link to expected theory.: Overall, the experiments support the main theoretical expectations developed in the previous chapters:

- One-level AS improves convergence by combining local block corrections, but the iteration count remains sensitive to global problem size and to the partition granularity M.
- The two-level extension AS2 introduces a coarse correction that becomes increasingly important as the problem size grows and/or the partition is refined, reducing iteration counts substantially in the regimes where one-level methods typically lose scalability.
- Under MPI, both methods benefit from increasing P, but the two-level formulation is designed to preserve convergence properties as parallelism and decomposition complexity increase.

D. Scalability: strong and weak scaling of AS2+MPI

This subsection evaluates the parallel scalability of the two-level Additive Schwarz method (AS2), i.e., Additive Schwarz augmented with the coarse correction, coupled with MPI-PCG. We report both *strong scaling* (fixed global problem size) and *weak scaling* (global size proportional to the number of MPI ranks).

a) Strong scaling (fixed n): effective speedup and iteration reduction.: For strong scaling we keep n fixed and define the speedup as

$$S(P) = \frac{T_1}{T_P},$$

where T_P is the measured solve time (time spent in the Krylov loop with preconditioning). Figure 6 (left) shows a steep reduction in solve time as P increases for all tested sizes.

These values are *larger than linear* in P because $S(P)$ reflects an *effective* speedup that combines: (i) reduced per-iteration cost due to distributed sparse operations, and (ii) a marked *decrease in PCG iteration count* as P grows (e.g., for $n = 320$k, iterations drop from 6804 at $P = 1$ to 421 at $P = 16$). This behavior is consistent with the implementation details discussed earlier: for a fixed number of SSOR sweeps, smaller local blocks (and smaller local problems per rank) make the inexact local SSOR actions closer to exact block solves, while the coarse correction mitigates global low-frequency error components. The net effect is a rapid contraction of both iteration count and time-to-solution.

b) Weak scaling ($n \propto P$): algorithmic scalability vs parallel efficiency.: For weak scaling we increase the global problem size proportionally to the number of ranks, keeping the work per rank approximately constant. We measure the weak scaling efficiency as

$$E_w(P) = \frac{T_1}{T_P},$$

where T_1 is the solve time for the base case and T_P is the solve time at P ranks for the scaled problem. The results show that AS2 preserves *algorithmic scalability*: iteration counts remain nearly flat across the sweep (about $937, 917, 813, 820, 813$ for $P = 1, 2, 4, 8, 16$ respectively), indicating that the coarse level effectively controls the global modes that typically cause iteration growth in one-level Schwarz methods.

This trend is expected for Krylov methods under MPI: each PCG iteration requires global reductions (dot products and norms), and AS2 additionally introduces collectives tied to the coarse correction (e.g., coarse residual aggregation and coarse operator assembly via `MPI_Allreduce`). As P increases, these synchronization/communication costs occupy a larger fraction of the iteration time, reducing weak scaling efficiency even when the iteration count is stable.

c) Takeaway.: Overall, the two-level formulation achieves the intended goal: it provides robust convergence as the problem size and parallelism increase (stable iteration counts under weak scaling), and it delivers strong time-to-solution improvements under strong scaling. The weak scaling results also highlight the practical bottleneck to address in future refinements: reducing synchronization overheads (e.g., fewer reductions per iteration, communication-avoiding variants, and/or more scalable coarse-level strategies) to improve efficiency at high P.

IV. Final Benchmarking and Conclusions

This section consolidates the experimental evidence collected in the previous chapters and discusses the impact of progressively stronger solver components: *(i)* baseline PCG with identity preconditioning, *(ii)* one-level Additive Schwarz (AS) as a local preconditioner, and *(iii)* two-level Additive Schwarz (AS2), i.e. AS augmented with a coarse correction and executed under MPI. All results refer to the same model operator and discretization settings presented earlier, and we report convergence and timing metrics extracted from the benchmark logs. The main quantitative summary is given in Table III.

A. Baseline vs AS2: what changes in practice

The baseline reference is **PCG+Identity** at $p = 1$, which is representative of a solver that . Baseline vs AS2: what changes in practice The baseline reference is PCG+Identity at p = 1, which is representative of a solver that exploits sparsity (SpMV- based iterations) but has no mechanism to improve

TABLE III

BASELINE PCG+IDENTITY AT $p = 1$ VS. PCG_MPI + TWO-LEVEL ADDITIVE SCHWARZ (AS2) FOR $p \in \{1, 2, 4, 8, 16\}$. REPORTED METRICS: ITERATION COUNT AND TIMINGS (SETUP, SOLVE, TOTAL). SPEED-UPS ARE COMPUTED W.R.T. THE IDENTITY BASELINE OF THE *same case* (SAME N_x, NNZ): $S_{\text{iter}} = \text{iters}_{\text{id},p=1}/\text{iters}$ AND $S_{\text{time}} = t_{\text{total,id},p=1}/t_{\text{total}}$.

N_x	nnz	prec	p	iters	t_{setup} [s]	t_{solve} [s]	t_{total} [s]	S_{iter}	S_{time}
1D									
40k	119998	id	1	19999	0	3.778	3.780	1.0x	1.0x
		as2	1	605	1.013e-02	0.563	0.574	33.1x	6.6x
		as2	2	529	5.946e-03	0.247	0.254	37.8x	14.9x
		as2	4	338	3.528e-03	8.525e-02	9.269e-02	59.2x	40.8x
		as2	8	232	2.557e-03	4.914e-02	5.482e-02	86.2x	69.0x
		as2	16	188	2.164e-03	2.854e-02	3.399e-02	106.4x	111.2x
80k	239998	id	1	39999	0	14.72	14.72	1.0x	1.0x
		as2	1	945	2.773e-02	1.706	1.735	42.3x	8.5x
		as2	2	870	1.757e-02	0.830	0.848	46.0x	17.4x
		as2	4	560	6.544e-03	0.281	0.288	71.4x	51.1x
		as2	8	389	5.721e-03	0.151	0.161	102.8x	91.4x
		as2	16	235	5.639e-03	7.901e-02	9.181e-02	170.2x	160.3x
160k	479998	id	1	79999	0	110.6	110.9	1.0x	1.0x
		as2	1	1535	6.264e-02	9.506	9.566	52.1x	11.6x
		as2	2	1449	3.544e-02	4.047	4.082	55.2x	27.2x
		as2	4	759	2.188e-02	1.089	1.110	105.4x	99.9x
		as2	8	440	2.383e-02	0.474	0.498	181.8x	222.7x
		as2	16	333	2.801e-02	0.353	0.382	240.2x	290.3x
2D									
400	796800	id	1	980	0	1.788	1.997	1.0x	1.0x
		as2	1	675	0.167	4.626	5.256	1.5x	0.4x
		as2	2	644	8.520e-02	2.367	2.701	1.5x	0.7x
		as2	4	523	5.441e-02	1.204	1.415	1.9x	1.4x
		as2	8	368	5.128e-02	0.611	0.814	2.7x	2.5x
		as2	16	237	6.265e-02	0.280	0.343	4.1x	5.8x
800	3193600	id	1	2078	0	18.23	18.93	1.0x	1.0x
		as2	1	1005	0.631	28.97	29.78	2.1x	0.6x
		as2	2	1002	0.343	15.66	16.21	2.1x	1.2x
		as2	4	843	0.218	8.060	8.535	2.5x	2.2x
		as2	8	669	0.178	4.717	5.324	3.1x	3.6x
		as2	16	436	0.170	2.506	3.262	4.8x	5.8x
1600	12787200	id	1	4170	0	198.2	199.1	1.0x	1.0x
		as2	1	1454	2.659	211.0	214.1	2.9x	0.9x
		as2	2	1487	1.429	119.5	121.4	2.8x	1.6x
		as2	4	1379	0.882	67.85	69.28	3.0x	2.9x
		as2	8	1147	0.803	49.79	51.42	3.6x	3.9x
		as2	16	664	0.845	31.30	34.04	6.3x	5.9x
3D									
40	444080	id	1	125	0	0.616	0.716	1.0x	1.0x
		as2	1	186	0.140	2.299	2.618	0.7x	0.3x
		as2	2	153	7.449e-02	1.143	1.401	0.8x	0.5x
		as2	4	119	5.503e-02	0.620	0.968	1.1x	0.7x
		as2	8	67	4.154e-02	0.276	0.451	1.9x	1.6x
		as2	16	47	5.977e-02	0.153	0.248	2.7x	2.9x
80	3568640	id	1	274	0	1.881	2.195	1.0x	1.0x
		as2	1	274	0.415	7.459	7.875	1.0x	0.3x
		as2	2	257	0.154	2.583	2.775	1.1x	0.8x
		as2	4	186	8.615e-02	1.076	1.206	1.5x	1.8x
		as2	8	109	7.024e-02	0.516	0.649	2.5x	3.4x
		as2	16	58	6.216e-02	0.246	0.399	4.7x	5.5x
160	28518400	id	1	542	0	29.83	30.12	1.0x	1.0x
		as2	1	384	2.263	59.61	62.17	1.4x	0.5x
		as2	2	388	1.161	34.57	36.04	1.4x	0.8x
		as2	4	331	0.713	19.37	20.43	1.6x	1.5x
		as2	8	204	0.583	10.21	11.27	2.7x	2.7x
		as2	16	105	0.688	5.313	7.274	5.2x	4.1x

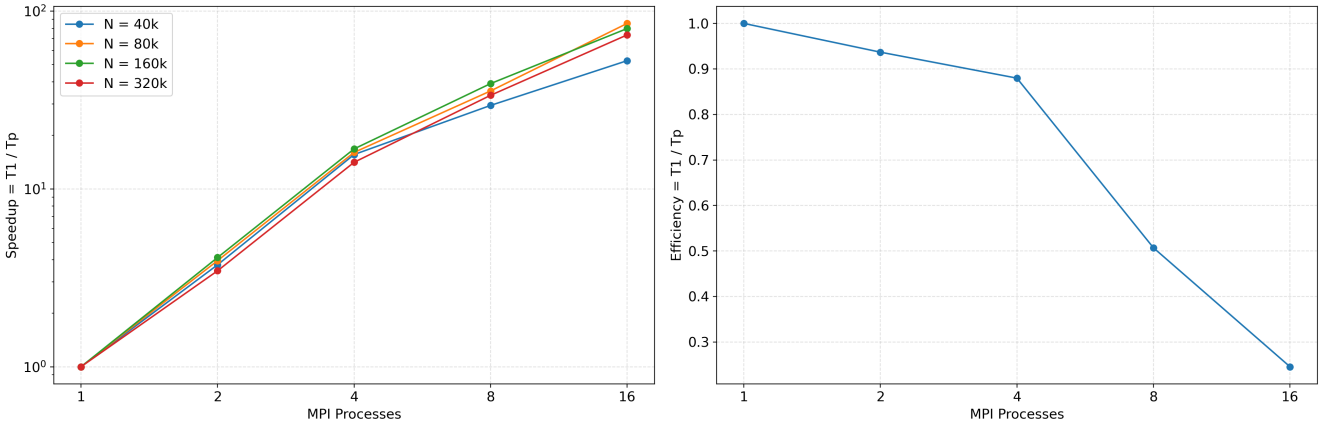

Fig. 6. Scalability of AS2+MPI. (*Left*) Strong scaling: speedup $S(P) = T_1/T_P$ versus number of MPI ranks P for multiple fixed problem sizes n. (*Right*) Weak scaling: efficiency $E_w(P) = T_1/T_P$ for $n \propto P$ (constant workload per rank).

condi- tioning. As discussed in the theory chapters exploits sparsity (SpMV-based iterations) but has no mechanism to improve conditioning. As discussed in the theory chapters, this translates into iteration counts that grow rapidly with problem size and (for elliptic operators) with mesh refinement. The two-level method **PCG_MPI+AS2** adds two key ingredients: (i) local block corrections (AS) that efficiently attenuate local error components, and (ii) a coarse correction that couples subdomains and targets global (slowly varying) modes. The combination is expected to reduce the number of Krylov iterations substantially, while MPI reduces per-iteration wall-clock time through distributed SpMV and distributed preconditioner work.

Table III makes the overall trend explicit: AS2 delivers dramatic iteration reductions in all dimensions, and when combined with MPI it yields strong reductions in solve time, even for very large problems.

B. 1D Poisson: reference problem and clean performance trends

We first consider the 1D reference problem (Ny=Nz=1), which was used throughout the project as the main testbed to validate the Additive Schwarz design choices and to study scaling without the additional geometric complexity of higher-dimensional stencils.

a) Iteration reduction.: For $N_x = 40$k, the baseline PCG+Identity requires 19,999 iterations, while AS2 reduces this to 605 iterations already at $p = 1$, i.e. a reduction of about $33\times$. As the problem size increases, the advantage of AS2 becomes even more pronounced:

$$N_x = 80\text{k}: \ 39{,}999 \to 945 \ (\approx 42\times)$$
$$N_x = 160\text{k}: \ 79{,}999 \to 1{,}535 \ (\approx 52\times)$$

This behavior matches the theoretical motivation for a two-level method: the coarse correction prevents the iteration count from growing proportionally to the global problem size, while the local SSOR-based block solves efficiently remove high-frequency error components.

b) Time-to-solution and MPI benefit.: The impact on solve time is equally strong. At $N_x = 160$k the baseline solve time is $t_{\text{solve}} = 110.6$ s, whereas AS2 at $p = 1$ reduces it to 9.506 s (about $11.6\times$). MPI then pushes the time further down: $t_{\text{solve}} = 4.047$ s at $p = 2$ and $t_{\text{solve}} = 0.353$ s at $p = 16$. The setup time is small compared to solve time in 1D (e.g., $t_{\text{setup}} = 2.8 \times 10^{-2}$ s at $N_x = 160\text{k}, p = 16$), confirming that in this regime AS2 is dominated by the Krylov loop and the distributed SpMV/preconditioner applications.

c) Takeaway (1D).: Overall, the 1D results illustrate the cleanest version of the method's benefit: AS2 provides a robust reduction in iterations and, combined with MPI, yields strong reductions in solve time. This makes 1D Poisson an ideal reference benchmark to compare one-level vs two-level behavior and to assess scalability trends.

C. 2D Poisson: stronger coupling and increasing value of parallelism

We now consider the 2D case (Ny=Nx, Nz=1), where each unknown couples to more neighbors and the global system exhibits stronger long-range interactions than in 1D. This is reflected in larger iteration counts for the baseline identity solver and in a larger absolute cost per iteration due to increased nnz.

a) Iterations: AS2 consistently lowers Krylov work.: At $N_x = 400$ (nnz = 796,800), PCG+Identity needs 980 iterations, whereas AS2 requires 675 at $p = 1$, and drops further to 237 at $p = 16$. At $N_x = 800$ (nnz = 3,193,600), the baseline is 2,078 iterations, while AS2 requires 1,005 at $p = 1$ and reaches 436 at $p = 16$. At $N_x = 1600$ (nnz = 12,787,200), the baseline reaches 4,170 iterations, while AS2 reduces it to 1,454 at $p = 1$ and to 664 at $p = 16$. These reductions are consistent with the intended role of the two-level correction: as coupling becomes more global in higher dimension, the coarse space contribution becomes increasingly important to reduce global modes and stabilize convergence.

b) Time: distributed solve time improves strongly with p.: In 2D, solve times scale down effectively with MPI. For

example, at $N_x = 1600$ AS2 reduces t_{solve} from 211.0 s at $p = 1$ to 31.30 s at $p = 16$, while also lowering the iteration count from 1,454 to 664. Similarly, for $N_x = 800$ the solve time decreases from 28.97 s ($p = 1$) to 2.506 s ($p = 16$). Setup time remains moderate relative to solve time in the largest cases (e.g., $t_{\text{setup}} = 0.845$ s at $N_x = 1600, p = 16$), which supports the design choice of keeping the coarse solve small and amortizing setup cost across a long Krylov phase.

c) Takeaway (2D).: The 2D results show that AS2 provides both convergence improvements and substantial wall-clock improvements. The benefit of MPI grows with nnz: as the matrix becomes larger and the SpMV dominates per-iteration cost, distributed execution yields strong reductions in t_{solve} while AS2 keeps iterations under control.

D. 3D Poisson: coarse correction + MPI for the most challenging regime

Finally, we consider the 3D case (Ny=Nz=Nx), which is the most demanding regime in terms of stencil connectivity and nnz growth.

a) Iterations and robustness across sizes.: For $N_x = 40$ (nnz = 444,080), PCG+Identity converges in 125 iterations, while AS2 requires 186 at $p = 1$ and then drops to 47 iterations at $p = 16$. For $N_x = 80$ (nnz = 3,568,640), both PCG+Identity and AS2 at $p = 1$ converge in 274 iterations, and AS2 then reduces to 58 iterations at $p = 16$. For the largest 3D case $N_x = 160$ (nnz = 28,518,400), PCG+Identity requires 542 iterations, whereas AS2 converges in 384 at $p = 1$ and in 105 at $p = 16$. These results highlight the intended property of the two-level method: combining local smoothing (SSOR-based block corrections) with a global coarse solve yields stable convergence as problem complexity increases, and parallelism further improves convergence and time-to-solution.

b) Time-to-solution: large nnz benefits from MPI.: The strongest improvements appear at the largest nnz. For $N_x = 160$, the baseline solve time is 29.83 s, while AS2 at $p = 16$ reduces this to $t_{\text{solve}} = 5.313$ s, with total time $t_{\text{total}} = 7.274$ s including setup. At $N_x = 80$, AS2 reduces solve time from 7.459 s ($p = 1$) to 0.246 s ($p = 16$). This trend is consistent with the performance model discussed earlier: as nnz increases, SpMV dominates, and MPI parallelism directly reduces per-iteration cost, while AS2 reduces the number of iterations required.

c) Takeaway (3D).: In 3D, the combination of (i) two-level convergence control and (ii) distributed execution provides the most relevant end-to-end benefit for large systems. The method preserves convergence efficiency and translates it into strong improvements in time-to-solution when p increases.

E. Why the improvement is most pronounced in 1D

Table III highlights that the end-to-end gain of AS2 (and AS2+MPI) is particularly striking in 1D, while in 2D and 3D the benefit remains significant but comparatively less spectacular against the baseline. This behavior is consistent with how conditioning evolves under mesh refinement and with the structure of the current two-level preconditioner.

a) Conditioning and mesh refinement: why the baseline explodes in 1D.: For diffusion-dominated elliptic operators, refining the grid increases the effective condition number of the discrete operator. In practice, this spreads the eigenvalues of A and slows down CG, because the convergence rate depends on the spectrum (and, in a compact form, on the condition number). This effect is clearly visible in the **PCG+Identity** baseline in 1D: the iteration count grows almost linearly with N_x in Table III (19,999 at 40k, 39,999 at 80k, 79,999 at 160k). Thus, 1D provides an "extreme" but very clean setting where ill-conditioning under refinement dominates the computational cost.

b) Why AS2 is a near-ideal match for 1D error components.: In 1D, the error components that are hardest for unpreconditioned CG are predominantly *smooth, low-frequency modes* along the line. The two-level structure is designed precisely to address this split:

- the **one-level AS** component (local blocks + overlap) damps high-frequency components locally, and
- the **coarse correction** couples the blocks globally and targets the remaining low-frequency components that otherwise decay slowly.

As a result, AS2 transforms the 1D iteration counts by one to two orders of magnitude: for example, 79,999 → 1,535 at $p = 1$ for $N_x = 160$k, and further down to 333 at $p = 16$.

c) Local SSOR block solves are particularly effective in 1D.: In the current implementation, each block solve is approximated by a small, fixed number of SSOR sweeps. For a 1D stencil, the extracted block matrix is very close to a tridiagonal SPD operator, for which SSOR provides a strong smoothing/inversion effect even with few sweeps. Intuitively, the local solve is "closer" to an exact A_i^{-1} action than it would be on a more strongly coupled higher-dimensional block. This amplifies the impact of the additive Schwarz correction in 1D, and helps explain the very large iteration reductions observed.

d) Why the relative gains are smaller in 2D/3D (but still valuable).: In 2D and 3D, the slow-to-converge subspace is richer: smooth error components can vary along multiple directions, and long-range coupling becomes stronger. Two practical consequences follow:

1) At fixed overlap and fixed SSOR sweeps, the one-level component remains effective but each local correction is a less accurate proxy of a true block inverse than in 1D, because the block operator has wider coupling and a more complex spectrum.
2) A lightweight coarse space (in the present baseline, one coarse degree of freedom per partition) captures the dominant global component, but in higher dimensions it cannot represent all low-energy modes with the same efficiency as in 1D; hence the iteration reduction factor is typically smaller.

This matches the measurements in Table III: in 2D the baseline already requires thousands of iterations (e.g., 4,170 at $N_x = 1600$), and AS2 reduces this to 1,454 at $p = 1$ and 664 at $p = 16$; in 3D, the baseline iteration counts at the tested sizes

are moderate (e.g., 542 at $N_x = 160$), leaving less room for dramatic multiplicative reductions, while AS2 still produces clear drops (e.g., $542 \to 105$ at $p = 16$).

e) Why MPI preserves strong time-to-solution improvements in 2D/3D.: Even when iteration reduction factors are smaller than in 1D, the time benefit of AS2+MPI remains strong because the per-iteration cost increases quickly with nnz in higher dimensions. In 2D and especially 3D, the SpMV cost dominates each Krylov step, so distributing the matrix and vectors makes t_{solve} decrease sharply with p. This is visible in the largest cases: for 2D at $N_x = 1600$, AS2 reduces t_{solve} from 211.0 s ($p = 1$) to 31.30 s ($p = 16$); for 3D at $N_x = 160$, it drops from 59.61 s ($p = 1$) to 5.313 s ($p = 16$) (Table III).

f) Summary.: The exceptional 1D gains come from the combination of (i) a baseline whose iteration count grows sharply with refinement, (ii) local SSOR block solves that are particularly effective on 1D-like block operators, and (iii) a coarse correction that captures the dominant global (low-frequency) error mode on a line. In 2D/3D the slow subspace is broader and local blocks are more strongly coupled, so the same lightweight AS2 design yields comparatively smaller (but still substantial) iteration reductions. Nonetheless, because nnz and SpMV cost grow rapidly with dimension, MPI parallelism amplifies the end-to-end benefit and preserves strong improvements in time-to-solution at scale.

F. Variable-coefficient tests: coefficient patterns that reliably increase iterations

To stress the solver in 2D/3D without pushing the mesh size to the largest feasible values, we introduced a *variable diffusion* coefficient $\mu(\mathbf{x})$ in the discrete operator. The resulting performance summary is reported in Table III (2D–3D blocks), where the identity baseline is always run at $p = 1$ and AS2 is tested for $p \in \{1, 2, 4, 8, 16\}$.

a) Mathematical setting.: We consider the diffusion model problem

$$-\nabla \cdot \big(\mu(\mathbf{x})\nabla u(\mathbf{x})\big) = f(\mathbf{x}) \quad \text{in } \Omega, \qquad u = 0 \quad \text{on } \partial\Omega, \tag{10}$$

discretized as in the previous chapters. The coefficient $\mu(\mathbf{x})$ is bounded and strictly positive, with a prescribed contrast ratio

$$\kappa_\mu \;=\; \frac{\mu_{\max}}{\mu_{\min}} \;\gg\; 1. \tag{11}$$

In the discrete system $A(\mu)\mathbf{u} = \mathbf{b}$, large κ_μ typically deteriorates the conditioning and introduces error components tied to *interfaces* (regions where μ changes abruptly). As a consequence, baseline PCG with the identity preconditioner can require very large iteration counts. The goal of this subsection is to show that AS2 remains effective in this harder regime and that MPI reduces time-to-solution once the iteration count is controlled.

b) Coefficient patterns used in the benchmarks.: Among the available options, the experiments rely on structured patterns that are easy to specify and reproducible. A representative choice (and the one used in these results) is:

- **(A) Layered medium (2D).** We define a planar interface at $x = x_0$ and set

$$\mu(\mathbf{x}) = \begin{cases} \mu_{\min}, & x < x_0, \\ \mu_{\max}, & x \geq x_0, \end{cases} \qquad \text{with } \kappa_\mu = \frac{\mu_{\max}}{\mu_{\min}} \gg 1.$$

  This configuration creates a strong internal interface and promotes *global, low-frequency* components coupled across the discontinuity, which are particularly expensive for unpreconditioned CG.

(Other available patterns, such as *checkerboard* and *inclusion*, follow the same idea: they increase stiffness heterogeneity and create difficult error modes, but the layered case provides the cleanest and most predictable difficulty increase.)

c) Why these patterns increase iterations (spectral perspective).: For constant μ, the discrete operator corresponds to a standard SPD diffusion matrix, whose eigenmodes are smooth and well understood. When $\mu(\mathbf{x})$ is discontinuous and highly contrasted, the operator $A(\mu)$ contains strongly varying local stiffnesses and interface coupling. A practical consequence is that the spectrum of $A(\mu)$ becomes more spread, and the energy norm

$$\|\mathbf{v}\|^2_{A(\mu)} = \mathbf{v}^T A(\mu)\mathbf{v}$$

weights gradients differently in the high/low diffusion regions. Interface-dominated error components can persist for many iterations because they are "smooth" in one region but constrained by flux continuity across the interface. This is exactly the type of behavior observed for the identity baseline in 2D: for instance, at $N_x = 100$ the baseline requires 91,755 iterations, and at $N_x = 200$ it grows to 200,907 iterations (Table III).

d) Why AS2 is effective on variable coefficients.: AS2 combines two complementary mechanisms:

- **Local block corrections (AS):** each subdomain applies a small number of SSOR sweeps on its local operator, which efficiently damps high-frequency components *within* each region (including within each side of the interface).
- **Coarse correction:** the agglomeration coarse space introduces global coupling between subdomains, which is essential when the dominant slow components are *global* and tied to the interface (as in (IV-F0b)). This limits the growth of the iteration count with respect to contrast and problem size.

The resulting improvements are substantial in 2D: at $N_x = 100$ the iteration count reduces from 91,755 (identity) to 801 (AS2, $p = 1$), and further to 346 at $p = 16$. Similarly, at $N_x = 200$ we obtain $200{,}907 \to 928$ at $p = 1$ and down to 363 at $p = 16$ (Table III). These reductions align with the theoretical expectation that two-level Schwarz methods handle heterogeneous diffusion robustly by combining local smoothing and global error correction.

e) Time-to-solution and MPI.: Once the iteration count is controlled by AS2, MPI parallelism translates the reduction in Krylov work into wall-clock gains. In 2D, for example, at

TABLE IV

BASELINE PCG+IDENTITY AT $p = 1$ VS. PCG_MPI + TWO-LEVEL ADDITIVE SCHWARZ (AS2) FOR $p \in \{1, 2, 4, 8, 16\}$. SPEEDUPS ARE COMPUTED AGAINST THE IDENTITY BASELINE (SAME CASE, $p = 1$): $S_{\text{it}} = \text{iters}_{\text{Id}}/\text{iters}$ AND $S_t = t_{\text{total,Id}}/t_{\text{total}}$.

N_x	nnz	prec	p	iters	t_{setup} [s]	t_{solve} [s]	t_{total} [s]	S_{it}	S_t
2D									
100	49600	id	1	91755	0	5.056	5.154	1.0x	1.0x
		as2	1	801	9.656e-03	0.647	0.756	114.5x	6.8x
		as2	2	785	5.834e-03	0.303	0.391	116.9x	13.2x
		as2	4	687	4.479e-03	0.140	0.215	133.6x	23.9x
		as2	8	546	2.857e-03	7.209e-02	9.751e-02	168.1x	52.9x
		as2	16	346	3.137e-03	3.255e-02	5.745e-02	265.2x	89.7x
200	198400	id	1	200907	0	23.72	24.10	1.0x	1.0x
		as2	1	928	3.708e-02	3.902	4.369	216.5x	5.5x
		as2	2	923	2.430e-02	1.794	2.091	217.6x	11.5x
		as2	4	762	1.584e-02	0.796	1.031	263.7x	23.4x
		as2	8	576	1.323e-02	0.395	0.538	348.8x	44.8x
		as2	16	363	1.481e-02	0.193	0.339	553.6x	71.1x
400	796800	id	1	91747	0	109.7	110.7	1.0x	1.0x
		as2	1	1321	1.267e-01	19.60	20.17	69.4x	5.5x
		as2	2	1259	7.847e-02	9.597	10.23	72.9x	10.8x
		as2	4	1010	4.640e-02	4.317	5.079	90.8x	21.8x
		as2	8	417	1.580e-02	0.477	0.512	220.0x	233.0x
		as2	16	261	1.398e-02	0.233	0.276	351.6x	431.9x
3D									
20	53600	id	1	380	0	2.306e-02	2.386e-02	1.0x	1.0x
		as2	1	116	4.132e-03	2.293e-02	2.797e-02	3.3x	0.9x
		as2	2	106	2.284e-03	1.047e-02	1.355e-02	3.6x	1.8x
		as2	4	80	1.289e-03	4.596e-03	8.093e-03	4.8x	2.9x
		as2	8	66	1.014e-03	4.150e-03	7.755e-03	5.8x	3.1x
		as2	16	52	8.061e-04	2.522e-03	5.351e-03	7.3x	4.5x
40	438400	id	1	1182	0	0.668	0.675	1.0x	1.0x
		as2	1	198	3.391e-02	0.385	0.425	6.0x	1.6x
		as2	2	173	2.201e-02	0.178	0.220	6.8x	3.1x
		as2	4	143	1.572e-02	9.510e-02	0.125	8.3x	5.4x
		as2	8	101	1.497e-02	6.064e-02	9.290e-02	11.7x	7.3x
		as2	16	67	1.483e-02	3.614e-02	7.829e-02	17.6x	8.6x
80	3568640	id	1	274	0	14.51	15.15	1.0x	1.0x
		as2	1	150	2.160e-01	3.446	3.862	1.8x	3.9x
		as2	2	145	1.184e-01	1.719	2.098	1.9x	7.2x
		as2	4	113	6.693e-02	0.807	1.154	2.4x	13.1x
		as2	8	85	5.352e-02	0.398	0.644	3.2x	23.5x
		as2	16	63	4.913e-02	0.210	0.441	4.3x	34.3x

$N_x = 200$ the total time decreases from 24.10 s (identity, $p = 1$) to 0.339 s (AS2, $p = 16$), while simultaneously increasing the iteration speedup and time speedup reported in the table. In 3D the same trend holds, and becomes more pronounced as nnz increases: at $N_x = 80$ the baseline total time is 15.15 s, while AS2 reaches 0.441 s at $p = 16$, with a clear reduction in iterations ($274 \rightarrow 63$).

f) Summary.: The structured coefficient patterns in the benchmark script (notably the layered diffusion (IV-F0b)) provide a reliable way to increase the difficulty of 2D/3D problems at moderate grid sizes by creating strong internal interfaces and large diffusion contrasts (11). The results in Table III show that AS2 remains highly effective in this regime: it sharply reduces iterations and enables MPI to deliver strong end-to-end time improvements.

V. CONCLUSIONS

This work studied the performance of CG/PCG for sparse SPD linear systems produced by finite-difference discretizations of diffusion–reaction problems in $d \in \{1, 2, 3\}$. The focus was on how progressively stronger preconditioners—from the identity baseline to Additive Schwarz (AS) and then to a two-level variant (AS2) with a coarse correction—affect iteration counts, time-to-solution, and scalability under MPI.

The baseline identity-preconditioned CG is a useful reference for correctness, but it rapidly becomes iteration-dominated as the mesh is refined and conditioning deteriorates. Introducing one-level AS already changes the picture: local overlapping subdomain corrections (implemented via inexact SSOR sweeps) reduce the number of outer Krylov iterations

significantly, but the total speedup is limited by the increased per-iteration cost of applying the preconditioner. As a result, the best configuration is determined by the balance between iteration reduction and preconditioner overhead, and optimal parameters (e.g., number of blocks) shift with problem size.

The most robust improvement comes from the two-level extension AS2. Adding a coarse correction mitigates the slow global modes that remain difficult for purely local preconditioners, especially for large problems, fine decompositions, and variable-coefficient cases with sharp diffusion interfaces and high contrasts. Once the iteration count is stabilized by AS2, MPI parallelism translates this algorithmic gain into strong reductions in wall-clock time, since both sparse matrix-vector products and preconditioner applications are distributed across ranks. Strong scaling benefits can therefore be very large in practice because they combine reduced work per rank with fewer iterations.

Finally, weak scaling results highlight the main practical bottleneck at higher rank counts: global reductions required by PCG (and additional collectives related to the coarse level) progressively dominate runtime and reduce parallel efficiency, even when iteration counts remain well controlled. This suggests clear directions for further improvements, such as communication-aware CG variants, more scalable coarse-level strategies, and enriched coarse spaces for the most challenging heterogeneous 2D/3D operators.

Overall, the experiments confirm that domain-decomposition preconditioning—and in particular a two-level Additive Schwarz formulation—provides an effective route to scalable time-to-solution improvements for SPD elliptic problems, with the coarse correction as the key ingredient for robustness under refinement and decomposition.

REFERENCES

[1] L. Formaggia, M. Sala, and F. Saleri, "Domain Decomposition Techniques," in *Numerical Solution of Partial Differential Equations on Parallel Computers*, A. M. Bruaset and A. Tveito, Eds.

[2] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: SIAM, 2003, doi: 10.1137/1.9780898718003.

[3] A. Toselli and O. Widlund, *Domain Decomposition Methods—Algorithms and Theory*. Berlin, Heidelberg: Springer, 2005, (Springer Series in Computational Mathematics, vol. 34), doi: 10.1007/b137868.

[4] A. Quarteroni and A. Valli, *Domain Decomposition Methods for Partial Differential Equations*. Oxford, UK: Oxford Univ. Press (Clarendon Press), 1999, ISBN: 978-0198501787.

[5] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Version 4.1, Nov. 2, 2023. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf

[6] ISO/IEC, *ISO/IEC 14882:2024 — Programming languages — C++*. Geneva, Switzerland: International Organization for Standardization, 2024. [Online]. Available: https://www.iso.org/standard/83626.html

[7] B. Stroustrup, *The C++ Programming Language*, 4th ed. Boston, MA, USA: Addison-Wesley Professional, 2013, ISBN: 978-0321563842.

[8] A. Williams, *C++ Concurrency in Action*, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2019, ISBN: 978-1617294693.

[9] A. Quarteroni, R. Sacco, F. Saleri, and P. Gervasio, *Matematica Numerica*, 4a ed. Milano, Italy: Springer, 2014, ISBN: 978-8847056435.