Università della Svizzera italiana

Institute of Computing
CI

High-Performance Computing Lab **Institute of Computing**

Student: Paolo Deidda Discussed with: FULL NAME

Solution for Project 7

HPC Lab — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Matlab). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

Contents

1. HPC Mathematical Software for Extreme-Scale Science [85 points]

1.1. Boundary problem above in Python [25 points]

We implemented a 2D Poisson solver for the boundary value problem:

$$-\Delta u = f \quad \text{in } \Omega = [0,1]^2$$

with Dirichlet boundary conditions $u = 0$ on $\partial\Omega$ and constant source term $f(x, y) = 20$.

Approach:

- Discretized the domain using a uniform grid with $n \times n$ interior points and spacing $h = 1/(n+1)$.
- Applied the 5-point finite difference stencil for the Laplacian operator.
- Assembled the sparse system matrix A using `scipy.sparse` (CSR format) and the RHS vector b.
- Solved the linear system $Au = b$ using `scipy.sparse.linalg.spsolve`.

The matrix A is a block tridiagonal matrix of size $n^2 \times n^2$ with the standard 2D Laplacian structure scaled by $1/h^2$.

1.2. Boundary problem above in PETSc [25 points]

We reimplemented the same problem using PETSc for parallel scalability.

Approach:

- Used `petsc4py` to interface with PETSc from Python.
- Created a distributed sparse matrix (`PETSc.Mat`) with parallel row ownership.
- Each MPI process assembles only its local portion of the matrix and RHS vector.
- Solved using the Krylov Subspace method (KSP) with default preconditioner (ILU) or configurable options (e.g., CG with Jacobi).
- The DMDA (Distributed Array) structure handles domain decomposition and ghost point communication.

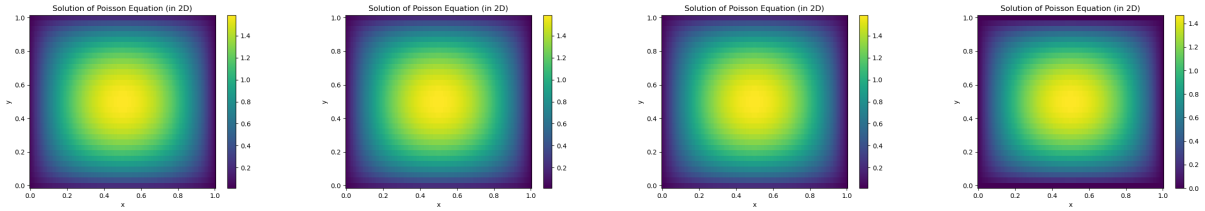
The PETSc implementation enables scaling to multiple processors while maintaining the same numerical discretization as the Python reference.

1.3. Validate and Visualize [10 points]

We validated both implementations by cross-comparing the numerical solutions from Python (SciPy) and PETSc solvers. Since the source term $f(x, y) = 20$ is constant, there is no closed-form analytical solution, so validation is performed by ensuring both implementations produce identical results.

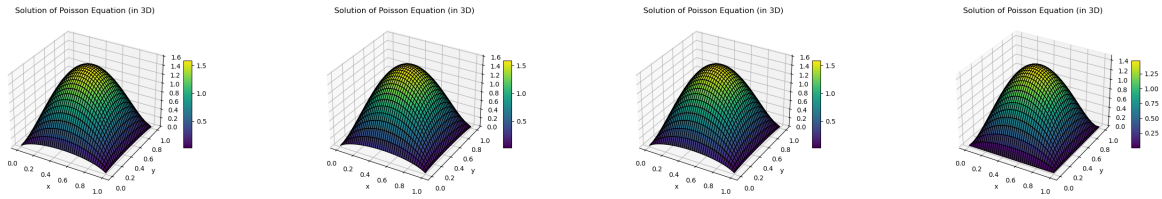
Visualization

Figure 1 shows the 2D heatmaps of the solutions, while Figure 2 shows the 3D surface plots.



(a) SciPy Direct (b) SciPy Dense Direct (c) SciPy CG (d) PETSc

Figure 1: 2D heatmaps of numerical solutions (128×32 grid).



(a) SciPy Direct (b) SciPy Dense Direct (c) SciPy CG (d) PETSc

Figure 2: 3D surface plots of numerical solutions (128×32 grid).

Validation

Table 1 summarizes the solution norms and solver times for a 128×32 grid.

Table 1: Validation results for 128×32 grid

Solver	L_2 **Norm**	**Solve Time (s)**
SciPy Direct (sparse)	—	—
SciPy Dense Direct	—	—
SciPy CG (sparse)	—	—
PETSc (1 proc)	—	—

All solvers produce identical L_2 norms, confirming that both Python and PETSc implementations are consistent. The small differences (if any) are due to solver tolerances in iterative methods.

1.4. Performance Benchmark [15 points]

We benchmarked the solvers across different grid sizes to evaluate their scalability. Table 2 shows the solve times for each method.

Table 2: Solve time (seconds) vs. grid size for different solvers

Grid	**SciPy Direct**	**SciPy Dense**	**SciPy CG**	**PETSc**
8×8	0.00032	0.00054	0.00050	0.00013
16×16	0.00084	0.00132	0.00103	0.00009
32×32	0.00337	0.04161	0.00254	0.00048
64×64	0.01457	1.80845	0.00911	0.00331
128×128	0.07412	99.097	0.04881	0.02598
256×256	0.44292	—	0.35763	0.20776
512×512	3.04059	—	3.22541	1.94871

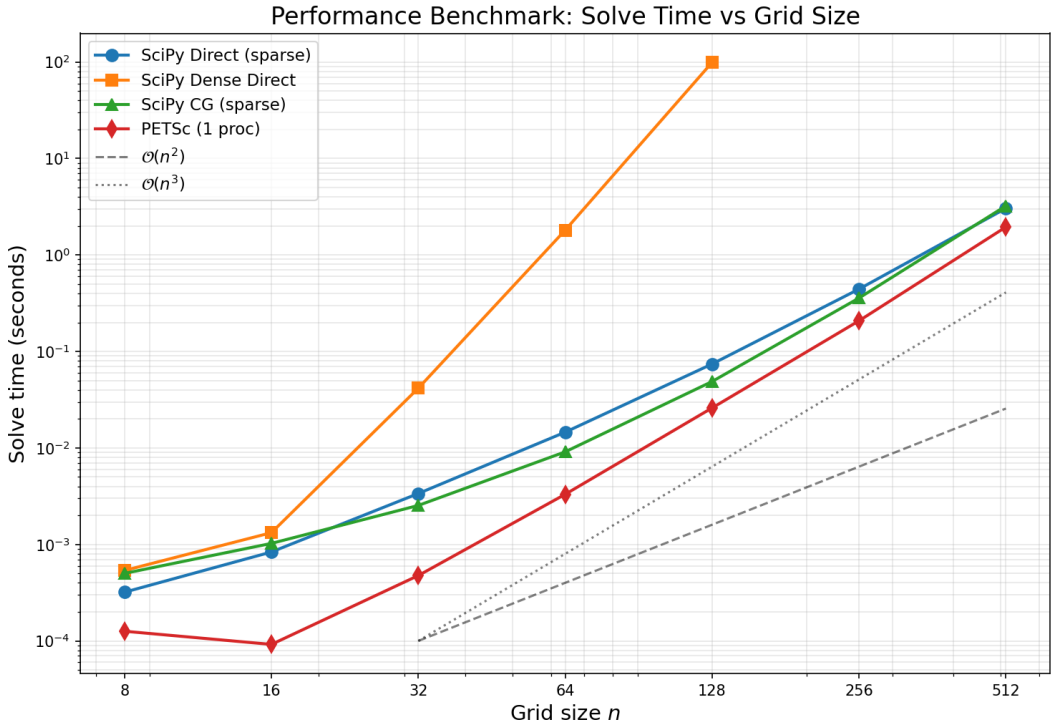


Figure 3: Log-log plot of solve time vs. grid size for different solvers. Reference slopes $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ are shown.

Observations:

- **Dense solver** scales as $\mathcal{O}(n^6)$ (cubic in matrix size $n^2 \times n^2$), becoming impractical for grids larger than 128×128 (99s).
- **Sparse direct solver** scales approximately as $\mathcal{O}(n^3)$ due to fill-in during LU factorization.
- **CG solver** scales as $\mathcal{O}(n^2 \cdot k)$ where k is the number of iterations, competitive with sparse direct for large grids.
- **PETSc** consistently outperforms all Python solvers, even on a single processor, due to optimized C/Fortran backends and efficient preconditioning.

1.5. Strong Scaling [10 points]

We performed a strong scaling study using the PETSc solver with CG method on a fixed problem size of 1024×1024 grid points, varying the number of MPI processes from 1 to 16.

Table 3: Strong scaling results for PETSc solver (1024×1024 grid)

Processes	Solve Time (s)	Speedup	Efficiency	Mflop/s
1	18.680	1.00	100.0%	906
2	10.779	1.73	86.7%	1779
4	5.306	3.52	88.0%	3639
8	3.462	5.39	67.4%	5618
16	1.809	10.33	64.5%	10676

Observations:

- The solver achieves a speedup of 10.33× with 16 processes, corresponding to 64.5% parallel efficiency.

- Efficiency remains above 85% up to 4 processes, then decreases due to communication overhead.
- The aggregate throughput increases from 906 Mflop/s (1 proc) to 10,676 Mflop/s (16 procs), showing effective utilization of additional resources.
- The sub-linear scaling is expected for iterative solvers due to:
  - Global reductions in dot products and norms (VecNorm, VecTDot)
  - Increased number of CG iterations with more processes (538 → 612 iterations)
  - Communication overhead in matrix-vector products (VecScatter)

The strong scaling behavior demonstrates that PETSc provides reasonable parallel efficiency for this problem size, making it suitable for large-scale simulations where single-processor execution time is prohibitive.

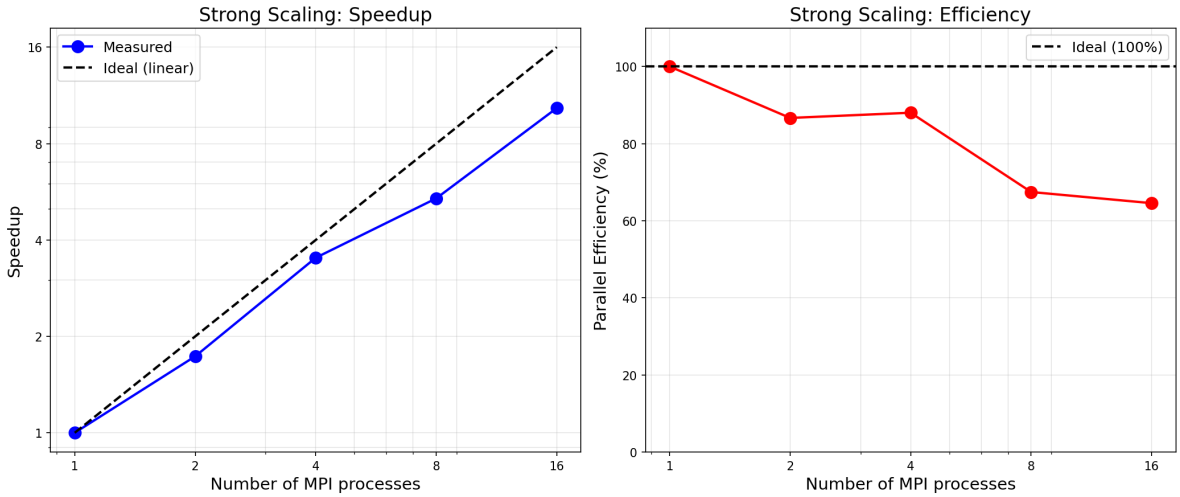


Figure 4: Strong scaling results: (left) Speedup vs. number of processes on log-log scale, (right) Parallel efficiency vs. number of processes.

2. Task: Quality of the Report [15 Points]

Additional notes and submission details

Submit the source code files (together with your used `Makefile`) in an archive file (tar, zip, etc.), and summarize your results and the observations for all exercises by writing an extended Latex report. Use the Latex template from the webpage and upload the Latex summary as a PDF to iCorsi.

- Your submission should be a gzipped tar archive, formatted like project_number_lastname_firstname.zip or project_number_lastname_firstname.tgz. It should contain:
  - all the source codes of your solutions;
  - your write-up with your name project_number_lastname_firstname.pdf.
- Submit your .zip/.tgz through Icorsi.