Università della Svizzera italiana | Institute of Computing CI

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

Student: Alessio Barletta

# 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.

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

Our task is to solve the following boundary value problem:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega = (0,1) \times (0,1), \\ u = 0, & \text{on } \partial\Omega, \end{cases} \tag{1}$$

where $f = f(x_1, x_2) = 20$ in $\Omega$.

First, we need to discretize the domain $\Omega$ with a uniform grid of size $n_x \times n_y$, where $n_x$ and $n_y$ are the number of grid points in the $x_1$ and $x_2$ directions, respectively. The grid spacing in each direction is given by:

$$dx = \frac{1}{n_x - 1}, \tag{2}$$

$$dy = \frac{1}{n_y - 1}. \tag{3}$$

Taking into consideration the internal grid points, we have:

$$-\Delta u = -\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}\right) \tag{4}$$

$$f = 20 \tag{5}$$

Using Taylor series expansion, we write the next and previous points in the x direction as:

$$u(x_1 + dx, x_2) = u(x_1, x_2) + \frac{\partial u}{\partial x_1} dx + \frac{\partial^2 u}{\partial x_1^2} \frac{dx^2}{2} + O(dx^3) \tag{6}$$

$$u(x_1 - dx, x_2) = u(x_1, x_2) - \frac{\partial u}{\partial x_1} dx + \frac{\partial^2 u}{\partial x_1^2} \frac{dx^2}{2} + O(dx^3) \tag{7}$$

Adding the two equations, we get:

$$u(x_1 + dx, x_2) + u(x_1 - dx, x_2) = 2u(x_1, x_2) + \frac{\partial^2 u}{\partial x_1^2} dx^2 + O(dx^3) \tag{8}$$

Rearranging gives us the second derivative approximation:

$$\frac{\partial^2 u}{\partial x_1^2} \approx \frac{u(x_1 + dx, x_2) - 2u(x_1, x_2) + u(x_1 - dx, x_2)}{dx^2} \tag{9}$$

Similarly, for the y direction:

$$\frac{\partial^2 u}{\partial x_2^2} \approx \frac{u(x_1, x_2 + dy) - 2u(x_1, x_2) + u(x_1, x_2 - dy)}{dy^2} \tag{10}$$

Combining these, we have:

$$\begin{aligned} -\Delta u \approx -\Bigg[ & \frac{u(x_1 + dx, x_2) - 2u(x_1, x_2) + u(x_1 - dx, x_2)}{dx^2} \\ & + \frac{u(x_1, x_2 + dy) - 2u(x_1, x_2) + u(x_1, x_2 - dy)}{dy^2} \Bigg] \end{aligned} \tag{11}$$

We can put this equal to $f$ and rearrange to get the final discrete equation:

$$\begin{aligned} \left( \frac{2}{dx^2} + \frac{2}{dy^2} \right) u(x_1, x_2) - \frac{1}{dx^2} u(x_1 + dx, x_2) - \frac{1}{dx^2} u(x_1 - dx, x_2) \\ - \frac{1}{dy^2} u(x_1, x_2 + dy) - \frac{1}{dy^2} u(x_1, x_2 - dy) = f \end{aligned} \tag{12}$$

## 1.1. Boundary problem above in Python [25 points]

To discretize the boundary value problem in Python, we can use the coefficients derived from the finite difference approximation.

For the Right-Hand-Side (RHS) vector, we can create a 2D array filled with the value of $f$ (which is 20) for all internal grid points. The boundary points will be set to zero as per the Dirichlet boundary conditions.

Regarding the coefficient matrix, we can use the ‘scipy.sparse‘ library. The main diagonal will contain the coefficient $\left( \frac{2}{dx^2} + \frac{2}{dy^2} \right)$, while the off-diagonals will contain the coefficients $-\frac{1}{dx^2}$ and $-\frac{1}{dy^2}$ corresponding to the neighboring grid points.

## 1.2. Boundary problem above in PETSc [25 points]

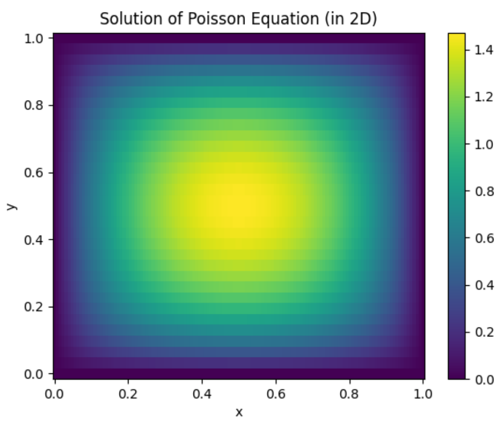
To implement the boundary value problem in PETSc, we can basicaly follow the same approach as in Python, but using PETSc's data structures and functions.

In the function `ComputeRHS`, to complete the code, we have to go over the grid points of the local domain assigned to each process. For internal grid points, we set the corresponding entry in the RHS vector to the value of $f$ (which is 20). For boundary points (with respect to the global domain), we set the value to zero.

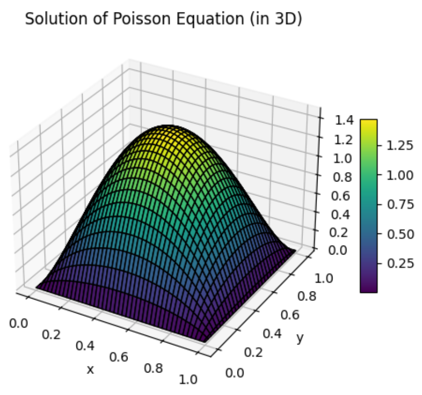
In the function `ComputeMatrix`, we need to assemble the sparse matrix using PETSc's `MatSetValuesStencil` function. For each internal grid point, we set the main diagonal entry to $\left( \frac{2}{dx^2} + \frac{2}{dy^2} \right)$ and the off-diagonal entries to $-\frac{1}{dx^2}$ and $-\frac{1}{dy^2}$ for the neighboring grid points, exactly as we did before.

## 1.3. Validate and Visualize [10 points]

Running the test in the `test_val` directory, we obtain the following results:

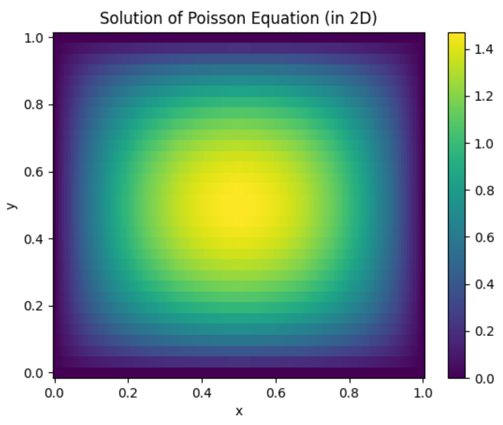


(a) 2D visualization

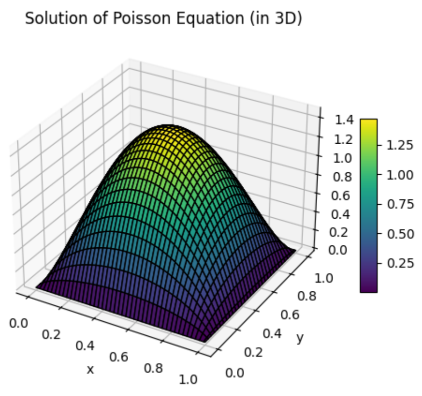


(b) 3D visualization

Figure 1: Solution derived using PETSc.

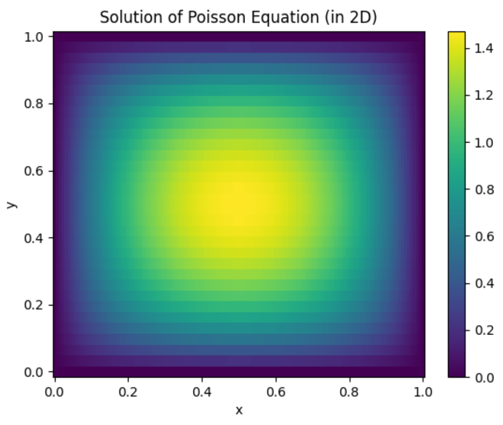


(a) 2D visualization

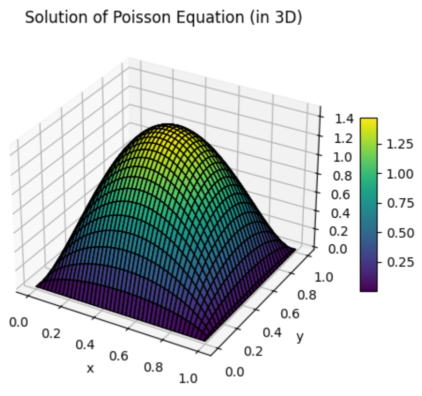


(b) 3D visualization

Figure 2: Solution derived using Dense Direct.



(a) 2D visualization



(b) 3D visualization

Figure 3: Solution derived using Sparse Direct.

As we can observe from the figures above, all the results are the same, as expected.

## 1.4. Performance Benchmark [15 points]

The graph for the performance benchmark comparing the three solvers (Dense Direct, Sparse Direct, and PETSc) and the C implementation is shown below:

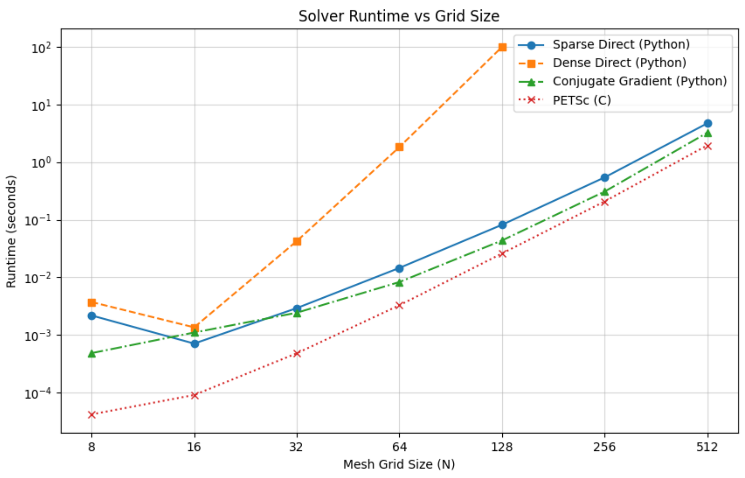


Figure 4: Performance Benchmark of Different Solvers

As expected, the C implementation using PETSc outperforms both Dense Direct and Sparse Direct solvers, however, this difference shrinks as the problem size increases.

The Dense Direct solver shows by far the worst performance, especially for larger problem sizes, being three orders of magnitude slower than PETSc for the largest tested size. This is expected, the dense solver makes no use of the sparsity of the matrix, leading to higher memory usage and computational cost.

## 1.5. Strong Scaling [10 points]

The strong scaling graph for the PETSc implementation is shown below:

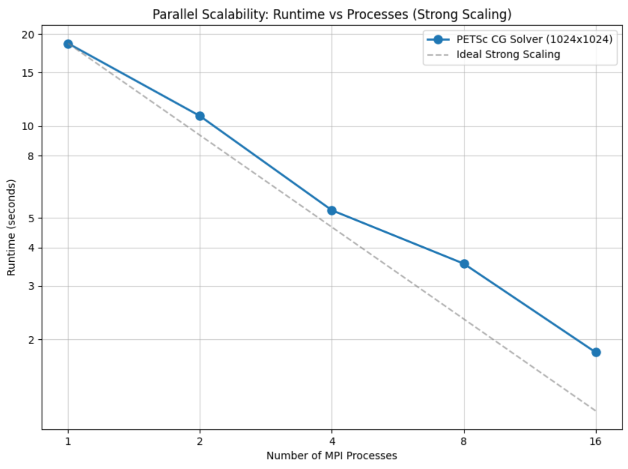


Figure 5: Strong Scaling of PETSc Implementation

As we cn see, the PETSc implementation shows good strong scaling behavior up to 16 processes, nearly halving the runtime each time the number of processes is doubled.

## 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.