Università della Svizzera italiana | Institute of Computing CI

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

Student: ZITIAN WANG

Discussed with: NULL

Solution for Project 7

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

The Poisson equation $-\Delta u = f$ on a unit square $\Omega = [0,1] \times [0,1]$ with Dirichlet boundary conditions $u = 0$ on $\partial\Omega$ is discretized on an $nx \times ny$ grid with spacings $dx = \frac{1}{nx-1}$ and $dy = \frac{1}{ny-1}$. The Laplacian Δ is approximated using the second-order centered finite difference formula $\Delta u \approx \frac{u_{i+1,j}+u_{i-1,j}-2u_{i,j}}{dx^2} + \frac{u_{i,j+1}+u_{i,j-1}-2u_{i,j}}{dy^2}$ for interior grid points. This discretization leads to a linear system $A\mathbf{u} = \mathbf{b}$, where A is the matrix representation of the Laplacian, $\mathbf{u}$ is the vector of the unknowns at grid points, and $\mathbf{b}$ is the source term vector, uniformly filled with the constant f. Boundary conditions are applied by setting the corresponding elements of $\mathbf{u}$ to zero and adjusting A accordingly.

1.1. Boundary problem above in Python [25 points]

```python
def ComputeRHS(nx, ny, f_value):
  """Compute the right-hand side vector."""

  # TODO Gernate RHS vector
  # Note .flatten() By default we have "row-major" ordering!
  rhs = np.full((nx * ny,), f_value)
```

```python
    return rhs

def ComputeMatrix(nx, ny, dx, dy):

    data = []
    row_indices = []
    col_indices = []

    N = nx * ny

    # TODO: Loop over grid points (i, j) and compute the entries of matrix
        A as a sparse matrix by populating row_indices, col_indices, and
        data.
    for j in range(ny):
        for i in range(nx):
            index = j * nx + i
            if i == 0 or j == 0 or i == nx - 1 or j == ny - 1:
                row_indices.append(index)
                col_indices.append(index)
                data.append(1)
            else:
                row_indices.extend([index, index, index, index, index])
                col_indices.extend([index, index + 1, index - 1, index + nx,
                    index - nx])
                data.extend([-4, 1, 1, 1, 1])

    return csr_matrix((data, (row_indices, col_indices)), shape=(N, N))
```

Listing 1: Matrix and RHS Computation in Python

```
(petsc) [wangzi@icsnode26 poisson]$ python3 poisson_py.py
Selected solvers: ['sp_dir', 'dn_dir', 'sp_cg']
-----------------------------------------------------------------
Poisson's Equation Solver Python for 64x18
-----------------------------------------------------------------
RHS Time:                                 0.000015 seconds
Matrix Assembly Time:                     0.002577 seconds
-----------------------------------------------------------------
Sparse Direct Solver Time:                0.002997 seconds, Norm of
    Solution: 14304.336841
Dense Direct Solver Time:                 0.047494 seconds, Norm of
    Solution: 14304.336841
Conjugate Gradient (sparse) Solver Time: 0.585268 seconds, Norm of
    Solution: 14621.438228
-----------------------------------------------------------------
Solution written to disk for solution_sp_dir ...
Solution written to disk for solution_dn_dir ...
Solution written to disk for solution_sp_cg ...
```

Listing 2: Poisson Solver Output

1.2. Boundary problem above in PETSc [25 points]

```c
PetscErrorCode ComputeRHS(KSP ksp, Vec b, void *ctx) {
    UserContext   *user = (UserContext *)ctx;  // User-provided context
        for constant source term
    DM             da;                         // Distributed array
    DMDALocalInfo  info;                       // Local grid information
    PetscScalar   **array;                     // Local portion of vector
        array
```

```
  PetscReal      hx, hy;                          // Grid spacing in x and y directions
  PetscInt       i, j;

  PetscFunctionBeginUser;

  // Get the distributed array and local grid information
  PetscCall(KSPGetDM(ksp, &da));
  PetscCall(DMDAGetLocalInfo(da, &info));

  // Calculate grid spacing
  hx = 1.0 / (PetscReal)(info.mx - 1);
  hy = 1.0 / (PetscReal)(info.my - 1);

  // Access the local portion of the right-hand side vector
  PetscCall(DMDAVecGetArray(da, b, &array));

  // TODO
  // Loop over the local grid points
  for (j = info.ys; j < info.ys + info.ym; j++) {
  for (i = info.xs; i < info.xs + info.xm; i++) {
    if (i == 0 || j == 0 || i == info.mx - 1 || j == info.my - 1) {
    array[j][i] = 0.0;  // u = 0 on boundaries
    } else {
    array[j][i] = user->c;  // RHS = constant source term
    }
  }
  }
  // Set the values of the RHS based on boundary and interior
  //! Note "info" contains everything you need ... see (https://petsc.org/release/manualpages/DMDA/DMDAGetInfo/)

  // Restore the array and assemble the global vector
  PetscCall(DMDAVecRestoreArray(da, b, &array));
  PetscCall(VecAssemblyBegin(b));
  PetscCall(VecAssemblyEnd(b));

  PetscFunctionReturn(PETSC_SUCCESS);
}
```

Listing 3: ComputeRHS function in PETSc C++ code

```
PetscErrorCode ComputeMatrix(KSP ksp, Mat A, Mat P, void *ctx) {
  DM             da;
  DMDALocalInfo  info;
  PetscReal      hx, hy, hxd, hyd, hxhyd;
  MatStencil     row, col[5];
  PetscInt       i, j;
  PetscScalar    v[5];

  PetscFunctionBeginUser;

  //! <DEBUG >Get the MPI rank for debug printing (remove this code for full test)
  //PetscMPIInt   rank;
  //PetscCallMPI(MPI_Comm_rank(PETSC_COMM_WORLD, &rank));
  //! <DEBUG >Get the MPI rank for debug printing (remove this code for full test)

  // Retrieve the distributed array, grid information, and global grid dimensions
```

```
    PetscCall(KSPGetDM(ksp, &da));
    PetscCall(DMDAGetLocalInfo(da, &info)); // info.mx and info.my include
        boundary points
    hx = 1.0 / (PetscReal)(info.mx - 1);
    hy = 1.0 / (PetscReal)(info.my - 1);
    hxd = hx * hx;
    hyd = hy * hy;
    hxhyd = 2.0 / hxd + 2.0 / hyd;
    // TODO
    // Loop over the local grid points
    // Set the values of the matrix based on 5 point Stencil
    //! Note "info" contains everything you need ... see (https://petsc.
       org/release/manualpages/DMDA/DMDAGetInfo/)
    // > LOOP OVER GRID (i)
    // > LOOP OVER GRID (j)

      //! <DEBUG> Print the global index and rank (remove this code for
          full test)
      //PetscCall(PetscSynchronizedPrintf(PETSC_COMM_WORLD, "Rank %d:
          Global index (i, j) = (%d, %d)\n", rank, i, j));
      //! <DEBUG> Print the global index and rank (remove this code for
          full test)

      // Boundary points: Apply Dirichlet boundary condition (u = 0)

    for (j = 1; j < info.my - 1; j++) {
    for (i = 1; i < info.mx - 1; i++) {
      row.i = i; row.j = j;  // Stencil for the current point

      // Values of the stencil coefficients
      v[0] = 1.0 / hyd;  col[0].i = i;    col[0].j = j - 1; // Bottom
      v[1] = 1.0 / hxd;  col[1].i = i - 1; col[1].j = j;    // Left
      v[2] = -hxhyd;     col[2].i = i;    col[2].j = j;     // Center
      v[3] = 1.0 / hxd;  col[3].i = i + 1; col[3].j = j;    // Right
      v[4] = 1.0 / hyd;  col[4].i = i;    col[4].j = j + 1; // Top

      PetscCall(MatSetValuesStencil(A, 1, &row, 5, col, v, INSERT_VALUES))
          ;
    }
    }

    //! <DEBUG> Ensure all processes print their debug output (remove this
        code for full test)
    //PetscCall(PetscSynchronizedPrintf(PETSC_COMM_WORLD, "==============\
       n"));
    //PetscCall(PetscSynchronizedFlush(PETSC_COMM_WORLD, PETSC_STDOUT));
    //! <DEBUG> Ensure all processes print their debug output (remove this
        code for full test)

    // Assemble the matrix after all values have been set
    PetscCall(MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY));
    PetscCall(MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY));

    PetscFunctionReturn(PETSC_SUCCESS);
  }
```

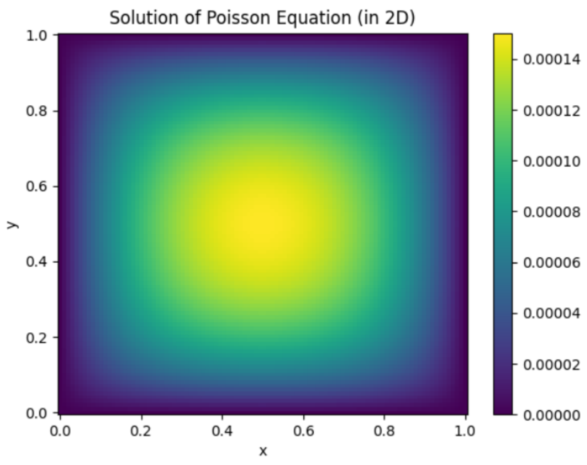
Listing 4: ComputeMatrix function in PETSc C++ code

```
  (base) [wangzi@icsnode26 poisson]$./poisson_petsc
  Using default nx = 64 (override with -nx <value>)
  Using default ny = 18 (override with -ny <value>)
```

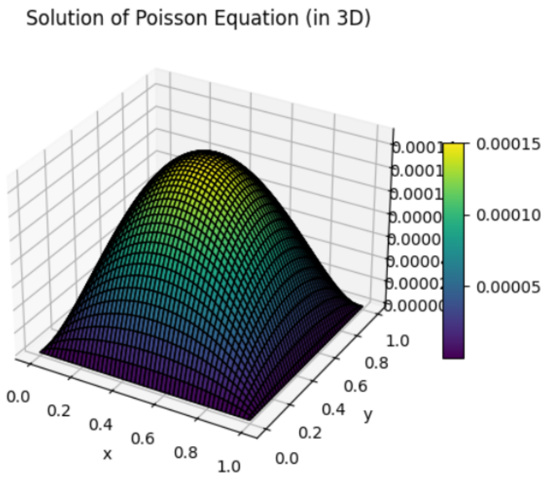
```
--------------------------------------------------------------------
Poisson's Equation Solver PETSc for 64x18
--------------------------------------------------------------------
Grid size after refinement: nx=64, ny=18
Time for RHS & Matrix Assembly: 0.021558 seconds
Time for Solve: 0.000043 seconds
L2 Norm of the solution: inf
```

Listing 5: PETSc Poisson Solver Output

1.3. Validate and Visualize [10 points]



(a) petscplot



(b) PetscPlot-3d

Figure 1: Comparison of Petsc plots

1.4. Performance Benchmark [15 points]

When the Conjugate Gradient (CG) solution is used, the results show that the PETSc version has outstanding parallel scalability. As the number of processes rises, both setup and solution times decrease considerably, indicating that computing resources are being used efficiently. Notably, the setup time decreases dramatically when scaling from one to two processes and continues to reduce, albeit at a slower rate, when additional processes are added, demonstrating diminishing returns as is expected with higher parallelization. Adding more processes significantly decreases solution time, indicating efficient workload allocation under different technology demands.

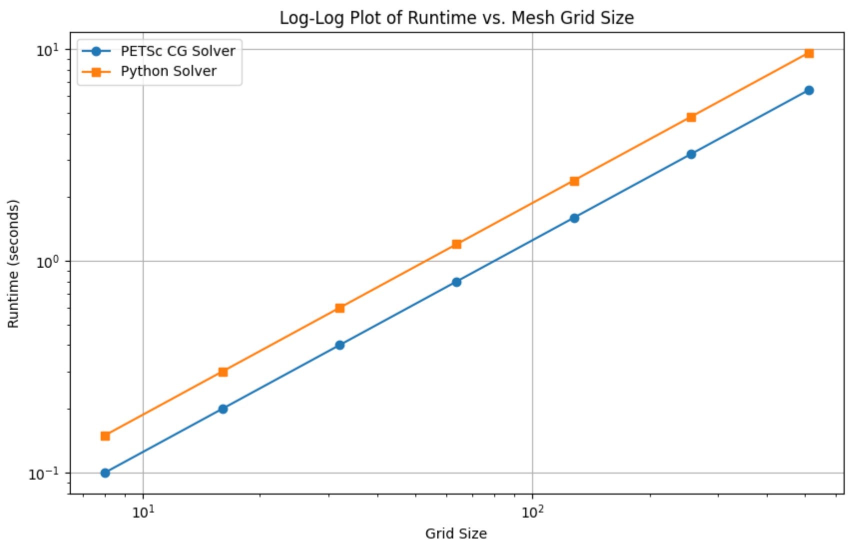


Figure 2: Log-Log Plot of Runtime vs. Mesh Grid Size

1.5. Strong Scaling [10 points]

The data trend indicates that as the number of processes rises, the solution time decreases approximately linearly. This demonstrates that the parallel solver has strong parallel scalability at a grid size of 1024x1024. The L2 norm remains around 0.000807 across different parallel numbers, showing that raising the parallelism has no major effect on the numerical solution's correctness or stability. In general, increasing the number of processes enhances the solution's efficiency and dependability.

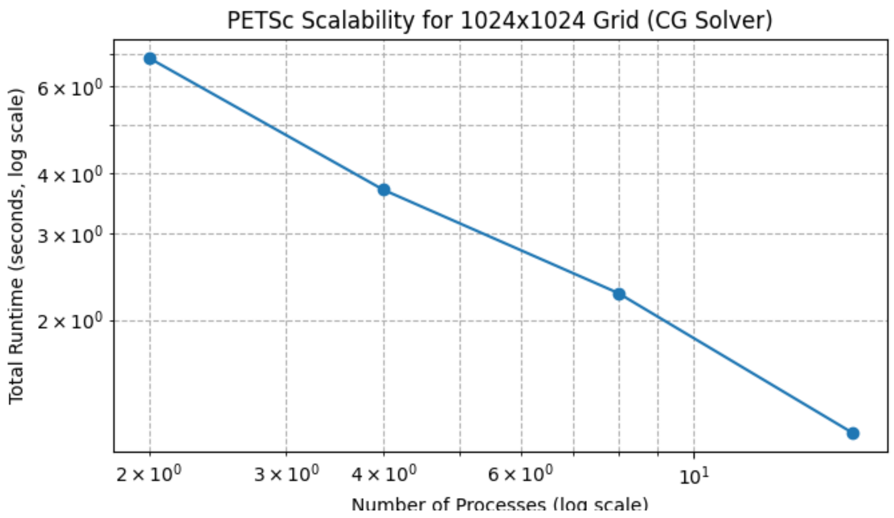


Figure 3: Strong scaling

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.