Parallelization of Jacobi Iteration

Solving 2-D Laplace equation

Abolfazl. Ziaeemehr\textsuperscript{1}

\textsuperscript{1}Department of Physics
Institute for Advanced Studies in Basic Sciences (IASBS)

Introductory School on Parallel Programming and Parallel Architecture for High-Performance Computing (Oct, 2016)
Outline

1. **Background**
   - Laplace equation

2. **Exercise 1: Starting Out**
   - Serial version

3. **Exercise 2: Feet a Little Wet - OpenMP**

4. **Exercise 3: MPI - 1D Decomposition**
Outline

1. Background
   - Laplace equation

2. Exercise 1: Starting Out
   - Serial version

3. Exercise 2: Feet a Little Wet- OpenMP

4. Exercise 3: MPI - 1D Decomposition
Laplace equation

\[
\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0
\]

1. Initialise \( \phi \) to some initial guess.
2. Apply the boundary conditions.
3. For each internal mesh point set

\[
\phi_{ij}^{new} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4}
\]

4. Replace old solution \( F \) with new estimate \( \phi \).
5. If solution does not satisfy tolerance, repeat from step 2.
Outline

1. Background
   - Laplace equation

2. Exercise 1: Starting Out
   - Serial version

3. Exercise 2: Feet a Little Wet- OpenMP

4. Exercise 3: MPI - 1D Decomposition
Serial

1. Download the serial version of the code in your language of choice.
2. Compile the code with optimization level -O3.
3. Test the code on a very small matrix.
4. Make a plot of matrix dimension vs. time reported to determine the scaling of the algorithm.
Serial

Abolfazl. Ziaeemehr (IASBS)
Parallelization of Jacobi Iteration
1. Insert an OpenMP pragma at the appropriate spot to parallelize the loop.

2. Test and plot the performance of the code over 1, 2, 4, 8 and 16 threads, with matrix sizes of 128, 256, 512, 1024, 2048 and 4096.
Abolfazl Ziaeemehr (IASBS)  Parallelization of Jacobi Iteration
```c
#pragma omp parallel
{
    // Iterate
    double TimeStart = seconds();
    for (int iCount = 1; iCount <= Iterations; iCount++) {
#pragma omp for private(i, j)
        for (i = 1; i <= Dimension; i++)
            for (j = 1; j <= Dimension; j++)
                SurfaceMatrix_t[i][j] = (0.25) * (SurfaceMatrix[i - 1][j] +
                    SurfaceMatrix[i][j + 1] +
                    SurfaceMatrix[i + 1][j] +
                    SurfaceMatrix[i][j - 1]);
    // PrintSurfaceMatrix(SurfaceMatrix_t, Dimension);

        double ** tmp;
        tmp = SurfaceMatrix;
        SurfaceMatrix = SurfaceMatrix_t;
        SurfaceMatrix_t = tmp;
    }
    double TimeEnd = seconds();
#pragma omp master
{
    cout << Dimension << " \t" << TimeEnd - TimeStart;
    cout << " \n";
}
```
rm -f omp*.txt

g++ -o jacobi_omp jacobi_omp.cpp -fopenmp

for i in 1 2 4 8 16
do
    export OMP_NUM_THREADS=${i}
    for j in 128 256 512 1024 2048 4096
do
        ./jacobi_omp ${j} 100 5 5 >> omp${i}.txt
    done
done

gnuplot plot.gp
display scaling.png

rm -f *.txt
The grid matrix must be completely distributed.

The whole process must be parallel.

Only asynchronous **MPI-Isend** and **MPI-Irecv** can be used for communication between processors.

Only use a 1 dimensional decomposition

---

**Figure 2:** A diagram of the 1-D decomposition of the Jacobi Relaxation for Solving the Laplace's Equation, showing that the boundary elements that need to be communicated between processors.
MPI - 1D Decomposition

Boundary point
Interior point

Process 0
Process 1
Process 2
Process 3
MPI - 1D Decomposition

/* Send up unless I’m at the top, then receive from below */
/* Note the use of xlocal[i] for &xlocal[i][0] */
for (time loop for 100 cycle) {
    if (rank < size - 1)
        MPI_Send( xlocal[maxn/size], maxn, MPI_DOUBLE, rank + 1, 0,
                  MPI_COMM_WORLD );
    if (rank > 0)
        MPI_Recv( xlocal[0], maxn, MPI_DOUBLE, rank - 1, 0,
                   MPI_COMM_WORLD , &status );
    /* Send down unless I’m at the bottom */
    if (rank > 0)
        MPI_Send(xlocal[1],maxn,MPI_DOUBLE,rank-1,1,MPI_COMM_WORLD);
    if (rank < size - 1)
        MPI_Recv(xlocal[maxn/size+1],maxn,MPI_DOUBLE,rank+1,1,
                  MPI_COMM_WORLD , &status );
    /* Compute new values (but not on boundary) */
    for (i=i_first; i<=i_last; i++)
        for (j=1; j<maxn -1; j++) {
            xnew[i][j]= (xlocal[i][j+1] + xlocal[i][j-1] +
                         xlocal[i+1][j] + xlocal[i-1][j]) / 4.0; }
    /* Only transfer the interior points */
    for (i=i_first; i<=i_last; i++)
        for (j=1; j<maxn -1; j++)
            xlocal[i][j] = xnew[i][j];  }

Abolfazl. Ziaeemehr (IASBS)
Introductory School on Parallel Programming and Parallel Architecture for High-Performance Computing(Oct,2016) 14
MPI - 1D Decomposition

![Graph showing time (s) vs. N grids for 2 core and 4 core configurations.](image-url)