Parallelization of a fast Poisson solver

Arne Morten Kvarving

Department of Mathematical Sciences
Norwegian University of Science and Technology

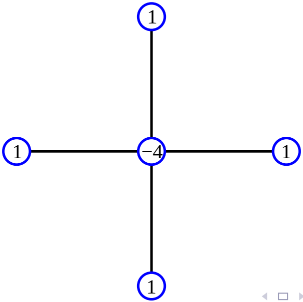
March 4, 2010

Problem overview

- We consider the Poisson problem

$$\begin{aligned} -\nabla^2 u &= f \qquad && \textit{in } \Omega \\ u &= 0 \qquad && \textit{on } \partial\Omega \end{aligned}$$

 in a square domain $\Omega = (0, 1) \times (0, 1)$.
- We discretize using finite differences, more specifically using the five-point stencil with $n + 1$ points in each spatial direction,



Problem overview contd.

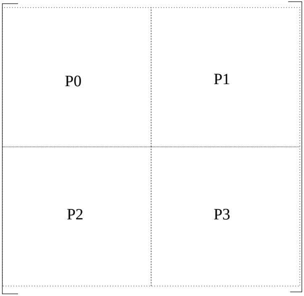
- This gives us the approximation to our problem as a linear system of equations
$$\mathbf{Au} = \mathbf{g}.$$
of dimension $(n-1)^2 \times (n-1)^2$.
- We solve this problem using diagonalization techniques as
  1. Transform
  $$\tilde{\mathbf{G}} = \mathbf{Q}^T \mathbf{G} \mathbf{Q}$$
  2. Scale
  $$\tilde{u}_{i,j} = \frac{\tilde{g}_{ij}}{\lambda_i + \lambda_j}, \qquad 1 \le i,j \le n-1$$
  3. Transform back
  $$\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}\mathbf{Q}^T.$$

Problem overview contd.

- Due to the eigenvectors of this specific operator, we can perform multiplications with $\mathbf{Q}$ and $\mathbf{Q}^T$ using fast sine transforms; see the lecture notes.
- This was a brief overview of the maths behind the serial code you have been given. We now move on to the real problem at hand; parallelization of this code.

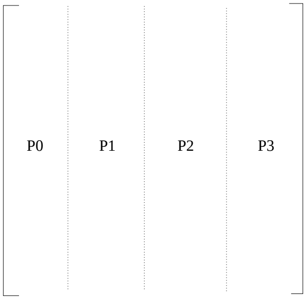
Parallelization - problem partitioning

- We have to decide how we want to divide our problem between the processes.
- We basically have two choices; either a block strategy like



Parallelization - problem partitioning

- Alternatively we can use a column/row division like



Parallelization - problem partitioning

- The *fst* operate along whole columns of the matrix.
- The block approach would require parallization within the *fst* method. This is something we need to avoid.
- Note that since we have $n - 1$ columns to divide among p processes, where n is a power of two, we cannot assign equal amounts of data to each process in general.
- Important hint: Store the designated sizes in an array!

Parallelization - transposing the matrix

- Since data is stored in several processes, transposing the matrix is nontrivial. This is where most of the effort needs to be invested.
- Since we don't have an equal amount of data in each process, we have to use `MPI_Alltoallv`.
- `MPI_Alltoallv` can only work on a vector. We hence need to pack the matrix into a suitable vector before calling the function.
- We now consider two simple examples in an attempt to give you the idea of what you have to do.

Parallelization - 3x3 using 2 processes

- Consider the case of a 3x3 matrix distributed on 2 processes.

$$\begin{array}{c|cc} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{array}$$

  The size array is $S = \begin{vmatrix} 1 & 2 \end{vmatrix}$.

- Transposed we have

$$\begin{array}{c|cc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}$$

Parallelization - 3x3 using 2 processes

- Scount arrays:
  proc 0:
  $$\begin{vmatrix} 1 & 2 \end{vmatrix} = \begin{vmatrix} S(1)S(1) & S(1)S(2) \end{vmatrix}$$
  proc 1:
  $$\begin{vmatrix} 2 & 4 \end{vmatrix} = \begin{vmatrix} S(2)S(1) & S(2)S(2) \end{vmatrix}$$
- Sdispl arrays:
  proc 0:
  $$\begin{vmatrix} 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & S(1)S(1) \end{vmatrix}$$
  proc 1:
  $$\begin{vmatrix} 0 & 2 \end{vmatrix} = \begin{vmatrix} 0 & S(2)S(1) \end{vmatrix}$$
- Rcount and Rdispl is the same arrays.

Parallelization - 3x3 using 2 processes

- Send buffers:

proc 0	proc 1
1	4
2	7
3	5
	6
	8
	9

- We then perform the call to `MPI_Alltoallv`.
- Receive buffers:

proc 0	proc 1
1	2
4	3
7	5
	6
	8
	9

Parallelization - 7x7 using 3 processes

- Consider the case of a 7x7 matrix distributed on 3 processes.

$$\left.\begin{array}{cc|cc|ccc} 1 & 8 & 15 & 22 & 29 & 36 & 43 \\ 2 & 9 & 16 & 23 & 30 & 37 & 44 \\ 3 & 10 & 17 & 24 & 31 & 38 & 45 \\ 4 & 11 & 18 & 25 & 32 & 39 & 46 \\ 5 & 12 & 19 & 26 & 33 & 40 & 47 \\ 6 & 13 & 20 & 27 & 34 & 41 & 48 \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 \end{array}\right.$$

  The size array is $S = \begin{vmatrix} 2 & 2 & 3 \end{vmatrix}$.

- Transposed we have

$$\left.\begin{array}{cc|cc|ccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 & 20 & 21 \\ 22 & 23 & 24 & 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 & 33 & 34 & 35 \\ 36 & 37 & 38 & 39 & 40 & 41 & 42 \\ 43 & 44 & 45 & 46 & 47 & 48 & 49 \end{array}\right.$$

Parallelization - 7x7 using 3 processes

- Scount arrays:

  proc 0:

  $$\begin{vmatrix}4 & 4 & 6\end{vmatrix} = \begin{vmatrix}S(1)S(1) & S(1)S(2) & S(1)S(3)\end{vmatrix}$$

  proc 1:

  $$\begin{vmatrix}4 & 4 & 6\end{vmatrix} = \begin{vmatrix}S(2)S(1) & S(2)S(2) & S(2)S(3)\end{vmatrix}$$

  proc 2:

  $$\begin{vmatrix}6 & 6 & 9\end{vmatrix} = \begin{vmatrix}S(3)S(1) & S(3)S(2) & S(3)S(3)\end{vmatrix}$$

Parallelization - 7x7 using 3 processes

- Sdispl arrays:
  proc 0:

$$\begin{vmatrix}0 & 4 & 8\end{vmatrix} = \begin{vmatrix}0 & S(1)S(1) & S(1)S(1)+S(1)S(2)\end{vmatrix}$$

  proc 1:

$$\begin{vmatrix}0 & 4 & 8\end{vmatrix} = \begin{vmatrix}0 & S(2)S(1) & S(2)S(1)+S(2)S(2)\end{vmatrix}$$

  proc 2:

$$\begin{vmatrix}0 & 6 & 12\end{vmatrix} = \begin{vmatrix}0 & S(3)S(1) & S(3)S(1)+S(3)S(2)\end{vmatrix}$$

- Again Rcount and Rdispl is the same arrays.

Parallelization - 7x7 using 3 processes

- Send buffers:

proc 0	proc 1	proc 2
1	15	29
2	16	30
8	22	36
9	23	37
3	17	43
4	18	44
10	24	31
11	25	32
5	19	38
6	20	39
7	21	45
12	26	46
13	27	33
14	28	34
		35
		40
		41
		42
		47
		48
		49

Parallelization - 7x7 using 3 processes

- Receive buffers:

proc 0	proc 1	proc 2
1	3	5
2	4	6
8	10	7
9	11	12
15	17	13
16	18	14
22	24	19
23	25	20
29	31	21
30	32	26
36	38	27
37	39	28
43	45	33
44	46	34
		35
		40
		42
		47
		48
		49