Distributed Computation of a Sparse Matrix Vector Product

Lecture 18
MA/CS 471
Fall 2003
Recall

• We wish to solve $A x = b$ for some sparse $A$

• In each of the three iterative schemes we need to evaluate a matrix vector product of the form: $Q v$ for some sparse matrix $Q$.

• For the Jacobi iteration:

$$x_{i}^{n+1} = \frac{1}{A_{ii}} \left( b_{i} - \sum_{j=1}^{j=N} Q_{ij} x_{j}^{n} \right)$$

$$Q_{ij} := A_{ij} - \delta_{ij} A_{ij}$$

where $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$ is the Kronecker delta
The Sparse Matrix-Vector Product

\[ \sum_{j=1}^{j=N} Q_{ij} x^n_j \]

For the homework I strongly suggest building a distributed sparse matrix and associated functions..

We will discuss possible approaches to this.
• A Distributed Sparse Matrix

Suppose we decided to assign a set of rows to each process:

Rows of the matrix owned by process 0.
A Distributed Sparse Matrix

• Suppose we decided to assign a set of rows to each process:

Sparse entries of the global matrix stored on proc 0
Divide The Unknowns

- Suppose we decided to assign a part of the vector to be multiplied to each process.
Dividing The Rows Up

• Suppose we decided to assign a part of the vector to be multiplied to each process.

• Then we can naturally group the part of the matrix sitting on each process into a set of sparse matrices.
Matrix Vector Product

• We can break up the row*vector products into separate sums

• Demo on board.
Practical Description

1) Each process has to inform the other process how many vector entries and which vector entry it **owns**

2) *perhaps* use MPI_Bcast
3) Create a routine that is given the global numerical id of a degree of freedom and returns the process id which owns it and the local identity of the vector entry on that process.

<table>
<thead>
<tr>
<th>Global number:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location and local number:</td>
<td>(0,0)</td>
<td>(0,1)</td>
<td>(0,2)</td>
<td>(1,0)</td>
<td>(1,1)</td>
<td>(1,2)</td>
<td>(2,0)</td>
<td>(2,1)</td>
<td>(2,2)</td>
</tr>
</tbody>
</table>
4) Each process should create $N\text{procs}$ empty sparse matrices with the correct number of rows (i.e. the number of rows \textit{owned} by this process).

5) Next – fill in the non-zero entries, using the vector entry locator (part 3) to determine which entry of which matrix should be set.
6) **On each process:**
   for each process p (excluding self)
   a) inspect the p’th matrix and make a list of which columns contain a non-zero entry.
   b) mpi_isend the number of entries required
   c) mpi_isend the list column ids
   end

   for each process p (excluding self)
   a) irecv and wait for the number of vector components ids which will be required to be sent to the p’th process
   b) irecv and wait for the list of vector components which will be required to be sent to the p’th process.
   end

Each process now knows which degrees of freedom it requires to receive from each of the other processes.
Each process has also informed the other processes which of their vector values it needs.
Practical Description \textit{cont}

7) Now each process will be able to request the parts of the vector it needs to compute the matrix row * vector it needs to compute.

8) Create the matrix vector routine:

\textbf{on each process:}

a) mpi\_isend the data required by the other processes to them
b) mpi\_irecv the data required for this process.
c) compute the local sparse matrix * local part of the global vector

d) \textbf{for each other process p:}
   a) mpi\_wait for the data to have arrived from the process p
   b) \textbf{compute the sparse matrix vector for process p}
   c) add this to the accumulating result vector.

\textbf{end}

e) mpi\_wait for the outgoing data to have been sent out.
Details

• There are a couple of wrinkles.
  1) To make this easier: you can create some redundant storage. i.e. create a vector of zeros and set the non-zeros coming from the p’th process.
  2) You should also consider the necessary modification required to make the matrix symmetric and diagonally dominant.
Last Stage

• Once you have a distributed matrix constructor – create a Jacobi iterative solver.

• Benchmark this code for N=102400 on 2, 4, 8, 16 processors.

• Hand in speed up (or down) curves and code print out as part of the project report.