Parallel Gaussian Elimination

Gaussian Elimination

- We work with the rowwise decomposition:
  - Each processor receives n/p rows of the extended matrix (A,b)
- We parallelize the individual pivoting steps
  - In step i we find the maximum A(j,i) for $i \leq j \leq n$
  - Swap row j with row i
  - Broadcast row i to processors holding rows i+1…n
  - Each processor computes the new $row_j = row_j - A(j,i)/A(i,i) \cdot row_i$

Gaussian Elimination

- Time: Finding the maximum A(j,i) and j can be done by tournaments in time O(n/p+log p)
- Broadcasting the pivot row takes communication time O(n$\cdot$ log p)
- Computation time for the remaining operations is O(n^2/p) per phase
- In n phases the computation time is thus O(n^3/p) and the communication time is O(n^2 log p)
- Hence the algorithm has constant efficiency once n=Ω(p log p)

Gaussian Elimination

- The disadvantage of this algorithm is that during the broadcasting step no computations take place, so effectively processors are idle during these communications, for time $O(n^2p)$

A different algorithm

- Again we work with the rowwise decomposition
- In the previous algorithm we were using tournaments to find the maximum A(j,i)
- Consequently we couldn‘t predict the pivot row j ahead of time
- And must then broadcast that row

A different algorithm

- Again we parallelize the pivoting step
- This time we always use row i in phase i
- Find the maximum A(i,j)
- We then want to eliminate all A(k,j) for k>i
- When we do this for i=1,…,n the resulting matrix is a columnwise permuted triangular matrix
- So we can afterwards solve via backsubstitution

A different algorithm

- The big advantage is that now we can organize computation and communication more efficiently
  - Processors are arranged as a linear array
  - We know that row i is the pivoting row
  - The processor holding row i sends row i to its nearest neighbor in the array
  - Every processor immediately passes the row on after reception
  - Then starts to replace $row_k = row_k - A(k,j)/A(i,j) \cdot row_i$ for all rows k in its possesion

A different algorithm

- Time: The computation time is again $O(n^3/p)$
- The communication time:
  - The last processor can start working after $(p-1)n$ communication time
  - However, this delay happens only once, when the computation time n^2/p per phase exceeds pn, i.e., if $n=\Omega(p^2)$, because
  - In that case the overall extra communication time is only $O(pn)$
  - This effect is called pipelining