Elementary O(logN) Step Algorithms

- Packet routing
- Sorting
- Matrix vector multiplication
- Jacobi relaxation
- Pivoting
- Convolution

Σπύρος Κομνηνός

Elementary O(logN) Step Algorithms

- Algorithms are optimal in terms of speed. Only matrix Algorithms are work efficient.
- Routing sorting convolution use N^2 processors to solve problems of size N in O(log N) steps.
- But they can be improved by Θ(N) by pipelining.
- With hypercubic networks these can be solved in O(logN) steps and with O(N) processors. Optimal in both speed and work efficiency.

Routing

- N x N mesh of trees have bisection width N
- Hence they will not be able to sort or route in less than $\Omega(N)$ steps.
- N x N mesh of trees no faster than N x N array when one needs to route N^2 packets.
- But N x N mesh of trees have a smaller diameter!

Sparse Routing

- $M \leq N$ packets stored in the row roots, can be routed in 2 log N steps to desired column roots.

Sparse Routing Algorithm

Let $0 \le p_i \le N-1$ be the desired destination of the packet stored in the i-th row tree root. (column tree root)

- Route the packet to the p_i-th leaf of its row. (*log N steps needed*)
- Route the packet to its column tree root. (*log N steps needed*)

Sparse Routing Algorithm

- Let $0 \leq p_i \leq N-1$ be the desired destination of the packet. (column tree root)
- If the destinations are mutually different the paths will never intersect.

Matrix Vector Multiplication

- Let $A=(a_{ij})$ be a N x N matrix, x an N vector, y their product.
- Enter x_i into the i-th column root. $1 \leq i \leq N$.
- Pass x_i to the leafs of its tree. (i-th tree) (*log N steps*)
- Input a_{ij} into the (i,j) leaf.
- Compute the product $a_{ij}\, x_j$
- The values are summed by the row trees. (*log N steps*)

$$y_i = \sum_{j=1}^{N} a_{i,j} x_j$$

Matrix Vector Multiplication Complexity

- The algorithm needs 2 log N steps
- By pipelining r vectors (input at the row roots new vector elements) we get with a delay 2 log N the result of each multiplication.
- log N + r complexity for pipelined multiplications.

Jacobi Relaxation

- Jacobi Relaxation can be expressed as a matrix vector product.

$$A\vec{x} = \vec{b} \quad \text{approximated by}$$

$$\begin{pmatrix} x_1(t+1) \\ x_2(t+1) \\ \vdots \\ x_N(t+1) \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -{}^{a_{12}}\!/_{a_{11}} & \cdots & -{}^{a_{1N}}\!/_{a_{11}} & {}^{b_1}\!/_{a_{11}} \\ -{}^{a_{21}}\!/_{a_{22}} & 0 & \cdots & -{}^{a_{2N}}\!/_{a_{22}} & {}^{b_2}\!/_{a_{22}} \\ \vdots & & & \vdots & \vdots \\ -{}^{a_{N1}}\!/_{a_{NN}} & -{}^{a_{N2}}\!/_{a_{NN}} & \cdots & 0 & {}^{b_N}\!/_{a_{NN}} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_N(t) \\ 1 \end{pmatrix}$$

Jacobi Relaxation Implementation

- b_i is stored on the (i,i) processor. a_{ij} is stored on the (i,j) processor
- $x_i(t)$ is stored in the i-th column root.

- [INIT]a_{ii} is inverted at the (i,i) leaf processor and passed to every leaf of the i-th row. b_i /a_{ii} is stored on the (i,i) processor. -a_{ij} /a_{ii} is stored on the (i,j) processor

Jacobi Relaxation Implementation

- Matrix vector multiplication as usual, but
- If (i=j) $/a_{ii}$ is the constant result the leaf processor passes its row parent.
- Route x_i from the row root to the column root.

- (* forall iterations 4 log N = O(log N) steps needed *)

Gaus Seidel Relaxation

- Gaus Seidel Relaxation in 3D meshes of trees in $O(\log^2 N)$ with $\Theta(N^3)$ processors.
- In 2D meshes of trees unresolved.