

COSC329 Parallel Algorithms

2005

1 Cost of Parallel Algorithm

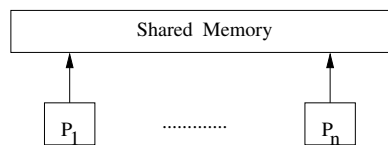
$C = PT$, where

- P : number of processors
- T : time

2 Computational Models

2.1 PRAM

PRAM stands for Parallel Random Access Machine with shared memory.



2.1.1 Memory Access conditions

As PRAM has shared memory, concurrency issue must be taken care of.

- EREW: Exclusive-Read, Exclusive-Write
- CREW: Concurrent-Read, Exclusive-Write
- CRCW: Concurrent-Read, Concurrent-Write

Exclusive R/W: If P_i wants to access some x in M , P_j ($i \neq j$) must wait to access x until P_i finishes.

If P_j wants to access y ($y \neq x$), no problem.

Concurrent R/W: If some P_i 's want to write such as $x = a[i]$ by P_i , x will get $a[i]$ for some i .

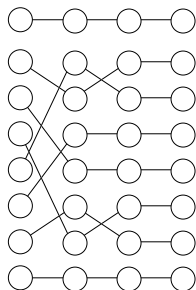
2.2 Classified by Instruction and data system (Flynn)

- SIMD: Single Instruction Stream Multiple Data stream
- MIMD: Multiple Instruction Stream Multiple Data stream

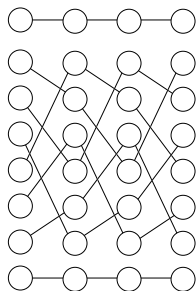
2.3 More restricted topology

2.3.1 Network type

Irregular

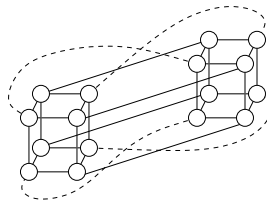


Regular: Shuffle computer



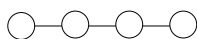
2.3.2 Cube connected computer: N-Cube

N=16

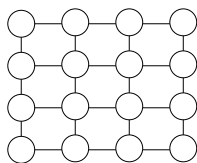


2.3.3 Mesh computer

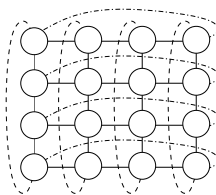
one-dimensional



two-dimensional



two-dimensional torus



3 Semiring and matrices

$S = \langle S, +, \cdot, 0, 1 \rangle$ is a semiring.

3.1 Properties

- $a + b = b + a$
- $a(b + c) = ab + ac$
- $(a + b)c = ac + bc$
- $(a + b) + c = a + (b + c)$
- $(ab)c = a(bc)$
- $a + 0 = 0 + a = a$
- $1 \cdot a = a \cdot 1 = a$
- Closure $a^* = 1 + a + a^2 + \dots + a^{n-1}$

Define matrices over S , $M(S)$

Let A, B be (n, n) matrices over S . $A + B$, AB are defined in a usual way.

$$I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}, O = \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

Theorem 3.1 $\langle M(S), +, \cdot, O, I \rangle$ is a semiring.

3.2 Closure A^*

Define closure A^* by $A^* = I + A + A^2 + \dots + A^{n-1}$

Distance semiring	Boolean semiring	Semiring			
min	0	\vee	1	+	1
+	∞	\wedge	0	\cdot	0

For distance and Boolean semiring, the meaning of A^* is:

$$A^* = I + A + A^2 + \dots + A^{n-1} = (I + A)^{n-1}$$

- distance semiring: All shortest distances
- Boolean semiring: Reflexive-transitive closure for

3.2.1 Repeated Squaring

We can compute $(I + A)^{n-1}$ more efficiently by repeated squaring

$$(I + A) \rightarrow (I + A)^2 \rightarrow (I + A)^4 \rightarrow \dots \rightarrow (I + A)^k$$

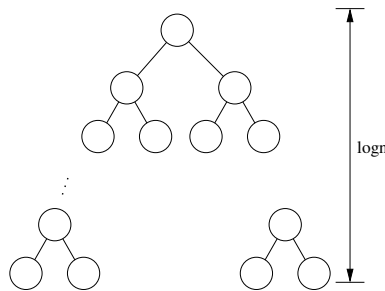
By squaring the previous product until $k \geq n$ for the first time where k is a power of 2.

3.2.2 Parallel Computation of a_{ij}

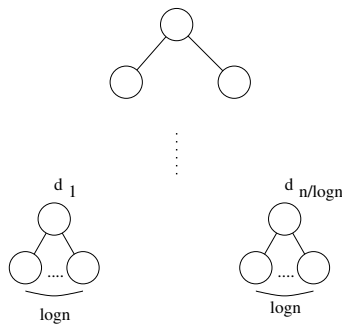
For distance semiring,

$$\begin{aligned} a_{ij} &= (a_{i1} + b_{1j}) \min(a_{i1} + b_{1j}) \min(a_{i2} + b_{2j}) \dots \min(a_{in} + b_{nj}) \\ &= d_1 \min \dots \min d_n \end{aligned}$$

We compute a_{ij} in parallel by



At each node, we take the minimum if there are n processors. We can compute a_{ij} in $O(\log n)$ time. Then the cost is $O(n \log n)$ time. We can do better than this! (Consider that finding the minimum among n elements can be done $O(n)$ time sequentially.)



We prepare $P = n/\log n$ processors. Let $P_1, \dots, P_{n/\log n}$ compute minimum spending $\log n$ time in parallel. Then we go up the tree in parallel with $P_1, \dots, P_{n/\log n}$.

- $T = 2 \log n$
- $P = n/\log n$
- Total cost= $PT = O(n)$

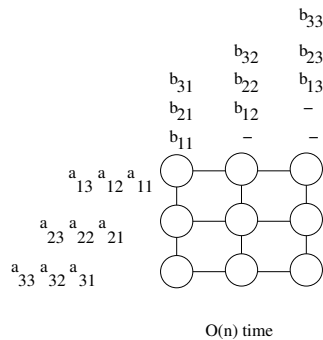
The total time by PRAM (Section 2.1) for A^* is $O(\log^2 n)$ with $P = n^3/\log n$ processors. Total cost is $O(n^3 \log n)$. Note that we compute a_{ij} for all i and j in parallel. (See Tutorial for details)

4 Matrix multiplication by mesh

We use the mesh model in Section 2.3.3.

4.1 Case Study: Closure A^*

Skew the matrices.



By this mesh computer, we can compute A^* with the following costs.

- $T = O(n \log n)$
- $P = n^2$.
- Total cost: $O(n^3 \log n)$.

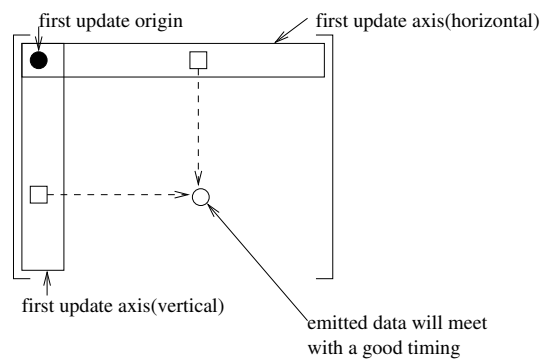
4.2 Case Study: Mesh Floyd

We can implement Floyd's algorithm (classical all-pairs shortest paths algorithm) on a mesh.

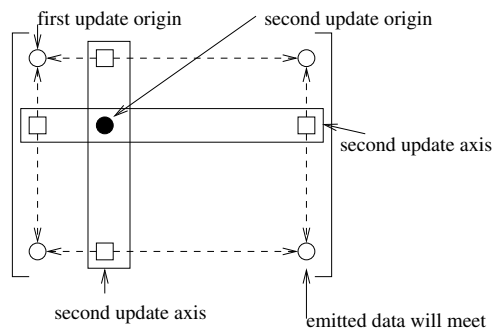
- $T = O(n)$
- $P = O(n^2)$
- Total cost= $O(n^3)$

4.2.1 Description

Stage 1



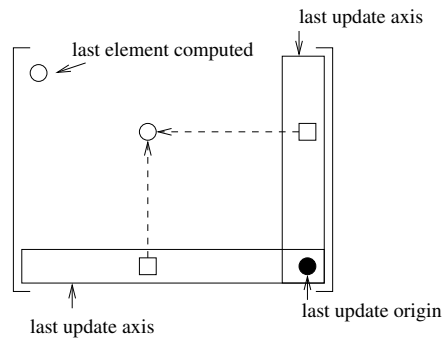
Stage 2



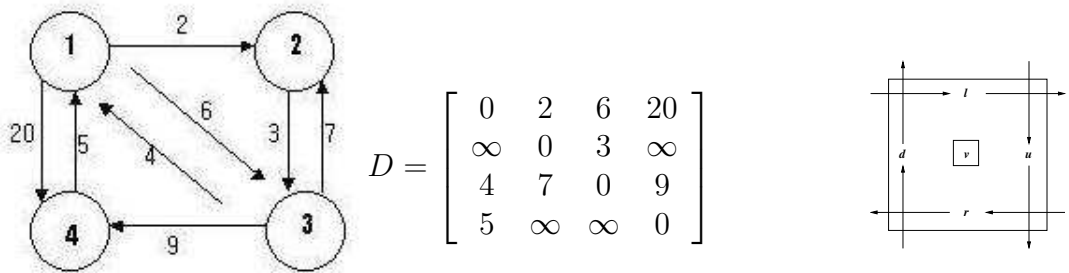
This will occur before everything is over at stage 1

....

Stage n



4.2.2 Trace of Mesh Floyd



Dark gray cells represent the origin, where control signals are originated. Light gray cells are where the control signal arrived. When the control signal arrives, the v value is copied to the registers. If the control signal has come in horizontal direction, d and u registers copy the value of v to be sent vertically. When the signal has come in vertical direction, l and r registers copy v to be sent horizontally.

t=0

0	2	6	20
99	0	3	99
4	7	0	9
5	99	99	0

Update origin 1 initiated.

t=1

0	2 2 2	6	20
99	0	3	99
4	7	0	9
5	99	99	0

t=2

0	2	6 6 6	20
99	99	0 2	3
4	4	7	0
5	99	99	0

t=3

0	2	6	20 20 20
99	0	99	3 6
4	4	6 2	0
5	5	99	99

Update origin 2 initiated.

t=4

0	2 2 2	6	20
99 99 99	0	3 3 3	99 99 20
4	6 6 6	4 0 6	9
5	5 7 2	99	0

t=5

99 0 2	2	2 3 5	20
99	0	3	99 99 99
4 99 6	6	6 0 3	4 9 20
5	7 7 7	5 11 6	0

t=6

0	2	5	2 99 20
99	0	3	99
4	6	0	6 9 99
5 99 7	7	7 10 3	5 0 20

Update origin 3 initiated.

t=7

0	2	5	20
99	0	3 3 3	99
4	6 6 6	0	9 9 9
5	7	10 10 10	7 0 99

t=8

0	2	5 5 5	20
99	6 0 3	3	3 9 12
4 4 4	6	0	9
5	7 6 10	10	10 0 9

t=9

0	6 2 5	5	5 9 14
4 7 3	0	3	12
4	6	0	9
5 4 10	7	10	0

Update origin 4 initiated.

t=10

4 0 5	2	5	14
7	0	3	12
4	6	0	9 9 9
5	7	10 10 10	0

t=11

0	2	5	14
7	0	3	12 12 12
4	6	10 0 9	9
5	7 7 7	10	0

t=12

0	2	5	14 14 14
7	0	10 3 12	12
4	7 6 9	0	9
5 5 5	7	10	0

t=13

0	2	10 5 14	14
7	7 0 12	3	12
5 4 9	6	0	9
5	7	10	0

t=14

0	7 2 14	5	14
5 7 12	0	3	12
4	6	0	9
5	7	10	0

t=15

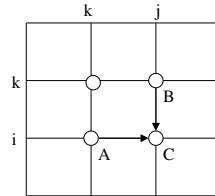
5 0 14	2	5	14
7	0	3	12
4	6	0	9
5	7	10	0

4.2.3 Analysis

The following timing function defines update at (i, j) by the k th update action.

$$T(i, j, k) = 3k + |i - k| + |j - k|$$

Case 1:



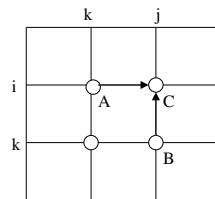
Time for A: $T(i, k, k) = 3k + i - k = 2k + i$

Time for B: $T(k, j, k) = 3k + j - k = 2k + j$

$$\left. \begin{aligned} A \rightarrow C : 2k + i + j - k &= i + j + k \\ B \rightarrow C : 2k + j + i - k &= i + j + k \end{aligned} \right\} \text{same time!}$$

Time for previous data at C: $T(i, j, k - 1) = 3(k - 1) + i - k + 1 + j - k + 1 = i + j + k - 1 \Rightarrow$ exactly one step before! Timing is tight.

Case 2:



Time for A: $T(i, k, k) = 3k + k - i = 4k - i$

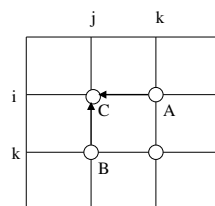
Time for B: $T(k, j, k) = 3k + j - k = 2k + j$

$$\left. \begin{aligned} A \rightarrow C : 4k - i + j - k &= 3k - i + j \\ B \rightarrow C : 2k + j + k - i &= 3k - i + j \end{aligned} \right\} \text{same time!}$$

Time for previous data at C: $T(i, j, k - 1) = 3(k - 1) + k - 1 - i + j - k + 1 = 3k - i + j - 3$

\Rightarrow 3 steps earlier...Timing is not tight.

Case 3:



Time for A : $T(i, k, k) = 3k + k - i = 4k - i$

Time for B : $T(k, j, k) = 3k + k - j = 4k - j$

$A \rightarrow C : 4k - i + k - j = 5k - i - j$
 $B \rightarrow C : 2k + j + k - i = 5k - i - j$) same time!

Time for previous data at C : $T(i, j, k - 1) = 3(k - 1) + k - 1 - i + k - 1 - j = 5k - i - j - 5$

\Rightarrow 5 steps earlier...Timing is not tight.