

1 Parallel and Distributed Computation

Parallel computation: Tightly coupled processors that can communicate almost as quickly as perform a computation

Distributed computation: Loosely couple processor for which communication is much slower than computation

2 PRAM Model

A *PRAM* machine consists of m synchronous processors with shared memory. This model ignores synchronization problems and communication issues, and concentrates on the task of parallelization of the problem. One gets various variations of this model depending on how various processors are permitted to access the same memory location at the same time.

- ER= Exclusive Read, only one processor can read a location in any 1 step
- CR= Concurrent Read, any number of processors can read a location in a step
- EW= Exclusive write, only one processor can write a location in any 1 step
- CW= Concurrent Write, any number of processors can write a location in a step. What if 2 processors try to write different values?
 - Common: All processors must be trying to write the same value
 - Arbitrary: An arbitrary processor succeeds in the case of a write conflict
 - Priority: The lowest number processor succeeds

We define $T(n, m)$ to be parallel time for the algorithm under consideration with m processors on input of size n Let $S(n)$ be the time complexity of the best *known* sequential time algorithm for a particular problem. Then the *efficiency* of a parallel algorithm is defined by

$$E(n, m) = \frac{S(n)}{mT(n, m)}$$

Efficiencies can range from 0 to 1. The best possible efficiency you can have is 1. Generally we prefer algorithm whose efficiency is $\Omega(1)$. The *folding principle* states that you can always use fewer processors and get the same efficiency.

3 OR Problem

INPUT: bits b_1, \dots, b_n

OUTPUT: The logical or of the bits

One can obtain an EREW Algorithm with $T(n, n) = \log n$ using a divide and conquer algorithm that is perhaps best understood as a binary tree. The leaves of the binary tree are the bits. Each internal node is a processor that OR's the output of its children. The efficiency of the EREW Algorithm is

$$\Theta\left(\frac{n}{n \log n}\right) = \Theta\left(\frac{1}{\log n}\right)$$

One can obtain an EREW algorithm for the OR problem with $E(n, m = n/\log n) = \Theta(1)$, by partitioning the input into $\frac{n}{\log n}$ partitions of $\log n$ bits each, and precomputing the OR of the bits in these partitions.

One can also obtain a CRCW Algorithm with $T(n, n) = \Theta(1)$. In this algorithm each processor P_i sets a variable answer to 0, then if $b_i = 1$, P_i sets answer to 1. The efficiency of this algorithm is $E(n, n) = \Theta(1)$.

4 MIN Problem

See section 10.2.1, and section 10.2.2.

INPUT: Integers x_1, \dots, x_n

OUTPUT: The smallest x_i

The results are essentially the same as for the OR problem. There is an EREW divide and conquer algorithm with $E(n, n/\log n) = 1$. Note that this technique works for any associative operator (both OR and MIN are associative).

There is an CRCW algorithm with $T(n, m = n^2) = 1$ and $E(n, m = n^2) = 1/n$. Here's code for processor $P_{i,j}$, $1 \leq i, j \leq n$ for the CRCW algorithm to compute the location of the minimum number:

```

If x[i] <= x[j] then T[i,j] = 1 else T[i, j]=0;
And[i]=1;
If T[i, j] = 0 then And[i]=0;
If And[i]=1 then Answer = i

```

What happens when the above code is run in the various CW models if there are two smallest numbers?

What happens in the various CW models if there are two smallest numbers and you just want to compute the value of the smallest number, that is if the last line is changed to

```

If And[i]=1 then Answer = x[i]

```

5 Parallel Prefix Problem

INPUT: Integers x_1, \dots, x_n . Let $S_i = \sum_{j=1}^i x_j$.

OUTPUT: S_1, \dots, S_n .

We give a divide and conquer algorithm. Solve the problem for the even x_i 's and odd x_i 's separately Then

$$S_{2i} = (x_1 + x_3 + \dots + x_{2i-1}) + (x_2 + x_4 + \dots + x_{2i})$$

and

$$S_{2i-1} = (x_1 + x_3 + \dots + x_{2i-1}) + (x_2 + x_4 + \dots + x_{2i-2})$$

This gives an algorithm with time $T(n, n) = \log n$ on EREW PRAM. This can easily be improved to $T(n, n/\log n) = \log n$, thus giving $\Omega(1)$ efficiency.

Note that divide and conquer into the first half and last half is more difficult because of the sum for the first half becomes a bottleneck that all of the last half of the processors want to access. Also note that this technique works for any associative operator.

6 All Pairs Shortest Path Problem

INPUT: A directed edge weighted graph G , with no negative cycles.

OUTPUT: The length of the shortest path $D[i, j]$ between any pair of points i and j .

First consider the following sequential code:

```

For i = 1 to n do
  For j=1 to n do
    D[i,j]=weight of edge (i, j)

Repeat log n times
  For i = 1 to n do
    For j=1 to n do
      For m=1 to n do
        D[i,j]=min{D[i,j], D[i,m]+D[m,j]}

```

The correctness of this procedure can be seen using the following loop invariant: After t times through the repeat loop, for all i and j , if the length of the shortest path between vertices i and j that has 2^t or less edges is equal to $D[i, j]$. Note that the outer loop is definitely not parallelizable. Let's try to parallelize the inner 3 loops to see what happens.

```

Repeat log n times
  For all i,j,m in parallel D[i,j]=min{D[i,j], D[i,m]+D[m,j]}

```

But note that this would require a concurrent write machine that always writes the smallest value. So we try:

```

Repeat log n times
For all i,j,m in parallel
  T[i,m, j]=min{D[i,j], D[i,m]+D[m,j]}
  D[i,j]=min{T[1,1,j]... T[1,n,j]}

```

This runs in time $T(n, n^3) = \log^2 n$ on an CREW. This gives an efficiency something like $E(n, m = n^3) = n^3 / (n^3 \log^2 n) = 1 / \log^2 n$.

7 Odd-Even Merging

See section 10.2.1.

INPUT: Sorted lists x_1, \dots, x_n , and y_1, \dots, y_n .

OUTPUT: The two lists merged into one sorted list z_1, \dots, z_{2n}

We give the following divide and conquer algorithm

Merge($x_1, \dots, x_n, y_1 \dots y_n$)

Merge($x_1, x_3, x_5, y_2, y_4, y_6 \dots$) to get $a_1 \dots a_n$

Merge($x_2, x_4, x_6, y_1, y_3, y_5 \dots$) to get $b_1 \dots b_n$

for $i=1$ to n do $z_{2i-1}=\min(a_i, b_i)$

$z_{2i} =\max(a_i, b_i)$

This can be implemented on an EREW PRAM to run in time $T(n, n) = \log n$ thus giving efficiency $\Theta(1/\log n)$. The following argument establishes the correctness of the algorithm. Each a_i is greater than or equal a_1, \dots, a_i . Each $a_i, i > 1$ is larger than b_{i-1} . Hence, $a_i \geq z_{2i-1}$. Each b_i is greater than or equal a_1, \dots, b_i . Each $b_i, i > 1$, is larger than a_{i-1} . Hence, $b_i \geq z_{2i-1}$. This same argument shows $a_{i+1} \geq z_{2i+1}$ and $b_{i+1} \geq z_{2i+1}$. So z_{2i-1} and z_{2i} must be a_i and b_i .

8 Odd-Even Merge Sorting

See section 10.2.1. We give the following divide and conquer algorithm

Sort($x_1, \dots x_n,$)

Merge(Sort($x_1, \dots x_{n/2}$), Sort($x_{n/2}, \dots x_n$))

This can be implemented on an EREW PRAM to run in time $T(n, n) = \log^2 n$ thus giving efficiency $\frac{n \log n}{n \log^2 n} = \Theta(1/\log n)$.

9 Pointer Doubling Problem

In the pointer doubling problem each of the n processors is given a pointer to a unique element of a singly linked list of n items. The goal is for each processor to learn its location in the linked list, e.g. the the processor with the 17th element in the list should know that it is 17th in the list.

```
for i= 1 to n-1  in parallel do
    d[i]=1

d[n]=0

repeat log n times
    for each item i in parallel
        if next[i] != nil then
            d[i]=d[i] + d[next[i]]
            next[i]=next[next[i]]
```

The correctness of the code follows from the following loop invariant: The position of i equals $d[i] + d[\text{next}[i]] + d[\text{next}[\text{next}[i]] + \dots$

Note that this is essentially solving the parallel prefix problem, with the work done before the recursion instead of after. To solve the parallel prefix problem we would replace the initialization

```
for i= 1 to n-1  in parallel do
    d[i]=1
```

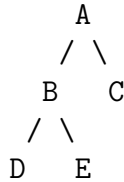
by

```
for i= 1 to n-1  in parallel do
    d[i]=x[i]
```

10 Eulerian Tour Technique to Find Tree Depths

The input is a binary tree with one processor per node. Assume that each processor knows the location of one node. The problem is to compute the

depth of each node in the tree. We show by example how to reduce this to pointer doubling. From the following tree



We create the list (the first line)

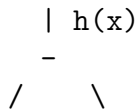
1, 1, -1, 1, -1, -1, 1, -1
A B D B E B A C A

and call pointer doubling with $d[i]$ initialized to either 1 or -1 appropriately. The depth of a node is then computed by looking at the sum up to the point shown in the second line.

11 Expression Evaluation

The input is an algebraic expression in the form of a binary tree, with the leaves being the elements, and the internal nodes being the algebraic operations. The goal is to compute the value of the expression. Some obvious approaches won't work are: 1) Evaluate nodes when both values of children are known, and 2) parallel prefix. The first approach won't give you a speed up if the tree is unbalanced. The second approach won't work if the operators are not be associative.

First assume that the only operation is subtraction. We label edges by functions. We now define the cut operation. If we have a subtree that looks like



$$\begin{array}{r}
 f(x)/ \quad \backslash g(x) \\
 - \quad \text{constant } c \\
 / \quad \backslash \\
 A \quad B
 \end{array}$$

and cut on the root of this subtree we get

$$\begin{array}{r}
 | h(f(x) - g(c)) \\
 - \\
 / \quad \backslash \\
 A \quad B
 \end{array}$$

If we have a subtree that looks like

$$\begin{array}{r}
 | h(x) \\
 - \\
 / \quad \backslash \\
 f(x)/ \quad \backslash g(x) \\
 \text{constant } c \quad - \\
 \quad \quad \quad / \quad \backslash \\
 \quad \quad \quad A \quad B
 \end{array}$$

and cut on the root of this subtree we get

$$\begin{array}{r}
 | h(f(c) - g(x)) \\
 - \\
 / \quad \backslash \\
 A \quad B
 \end{array}$$

Thus we are left with finding a class of functions, with the base elements being constants, that are closed under composition, subtraction of constants, and subtraction from constants. This class is the functions of the form $ax + b$, for real/rational a and b .

Note that in one step we can apply cuts to all nodes with an odd numbered left child that is a leaf. Note that in one step we can apply cuts to all nodes with an odd numbered right child that is a leaf. This leads to the following algorithm:

Repeat $\log n$ times

 For each internal node v in parallel

 if v has odd numbered left child that is a leaf then cut at v

 if v has odd numbered right child that is a leaf then cut at v

 renumber the leaves using pointer doubling

Note that in $\log n$ steps we will down to a constant sized tree since each iteration of the outer loop reduces the number of leaves by one half. So $T(n, n) = \log n$

12 A Problem that is Hard to Parallelize

No one knows a fast parallel algorithm for the following problem, known as the Monotone Circuit Value Problem.

INPUT: A Boolean formula (or circuit, it doesn't matter) F consisting of the connectives AND and OR, and a truth assignment A for the variables in F .

OUTPUT: 1 if A satisfies F (that is makes F true), and 0 otherwise.

More precisely, no one knows of a parallel algorithm that runs in time $O(\log^k n)$ for some k , with a polynomial number of processors. Here n is the size of the formula.