Lindsey Bieda and Joe Frambach
Reduction and Parallel Problems
11.07.2011

6. We consider the problem of multiplying two n by n matrices.
Given input matrices A and B, calculate $AB = C$.

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix}$$

1. Design a parallel algorithm that runs in time n on a CREW PRAM with n^2 processors. What is the efficiency of this algorithm?

   parallel-for i=1 to n:
   parallel-for j=1 to n:
   sum = 0
   for k = 1 to n:
   sum += $a_{i,k} \cdot b_{k,j}$
   $c_{i,j}$ = sum

2. Design a parallel algorithm that runs in time O($log\ n$) time on a CREW PRAM with n^3 processors. What is the efficiency of this algorithm?

   parallel-for i=1 to n:
   parallel-for j=1 to n:
   parallel-for k = 1 to n:
   $product_{i,j,k} = a_{i,k} \cdot b_{k,j}$
   Each element of C is the sum of $product_{i,j,k}$ *for k from 1 to n.*
   The summation of n numbers with n processors on a CREW PRAM is done in O(log n) time.
   A call tree of height log n and n leaves is created:
   At each level of the tree, half of the processors are responsible for summing the lower level.
   Thus, in O(log n) time, $C_{i,j}$ *is computed.*
   $c_{i,j} = \sum_{k=1}^{n} product_{i,j,k}$.

3. Design a parallel algorithm that runs in time O($log\ n$) time on a CREW PRAM with $n^3/log\ n$ processors. What is the efficiency of this algorithm?

   This follows exactly as described for n^3 processors, except each processor is now responsible for $log\ n$ multiplications rather than just one. Something like this should approximately suffice:
   parallel-for i = 1 to $n^3/log\ n$: *Each processor...*
   for j = 1 to $log\ n$: *...is responsible for log n computations*
   $product_{i/n^2,i/n,i\%n+j} = a_{i/n^2,i\%n+j} \cdot b_{i\%n+j,i/n}$
   parallel-for i = 1 to n:
   parallel-for j = 1 to n:
   Now we have n/log n processors to compute the sum of $product_{i,j,k}$ *in O(log n) time.*
   This is done just as the multiplications are done.

A call tree of height log $(n/\log n)$ with $n/\log n$ leaves is created.
Each leaf processor is responsible for summing log n numbers in O(log n) time.
The summations backtrace in O(log $(n/\log n)$) time.

4. Design a parallel algorithm that runs in time O($log\ n$) time on a EREW PRAM with $n^3/log\ n$ processors. What is the efficiency of this algorithm? HINT: Recall problem 3.

   Problem 3 provides an O($log\ n$) algorithm to copy a number n times with $n/log\ n$ processors on an EREW PRAM. In O($log\ n$) time, we can create n copies of A and B.
   parallel-for i = 1 to n:
   parallel-for j = 1 to n:
   Call Problem3 on $a_{i,j}$.
   Call Problem3 on $b_{i,j}$.

   Now, the matrix multiplication can be done exactly as in problem 1.3, except each set of n processors have their own "slice" of A, and their own "slice" of B, granting exclusive read. TODO: actually work out the details and efficiency.

7. Design a parallel algorithm that given a polynomial p(x) of degree n and an integer k computes the value of p(k). You algorithm should run in time O($log\ n$) on a EREW PRAM with O($n/log\ n$) processors. Assume that the polynomial is represented by its coefficients.

Refer to Problem 3:

> Consider the problem of taking as input an integer n and an integer x,
> and creating an array A of n integers, where each entry of A is equal to x.
> Give an algorithm that runs in time O($log\ n$) on a EREW PRAM using
> $n/logn$ processors.
> What is the efficiency of this algorithm?

This is used to generate n copies of k.

Next, refer to Problem 4:

> Design a parallel algorithm for the parallel prefix problem that runs in
> time O($log\ n$) with $n/log\ n$ processors on a EREW PRAM.

This can be modified to use multiplication rather than addition since both operations are associative.
This is used to generate $k,\ k^2,\ \ldots,\ k^n$.

Finally, we need to sum $p_n k^n\ +\ p_{n-1}k^{n-1}\ +\ \ldots\ +\ p_1 k\ +\ p_0$.
A divide-and-conquer approach is used. The terms are split in half, for two processors, split in half again for four processors, until all $n/log\ n$ processors are in use, each calculating the sum of $log\ n$ products. The tree has a height of $log\ log\ n$. This summation runs in O($log\ n$) time.

8. We consider the problem of computing F_n, the nth Fibonacci number, given an integer n as input. Show how to solve this problem in time O($log\ n$) on a EREW PRAM with n processors. Make the unrealistic assumption that F_n will fit within one word of memory for all n, that is, assume that all arithmetic operations take constant time. Recall that F_n is defined by the following recurrence: $F_0 = F_1 = 1$, and $F_n = F_{n-1}+F_{n-2}$ for $n > 1$. HINT: Note that for $j > 0$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_j \\ F_{j-1} \end{bmatrix} = \begin{bmatrix} F_{j+1} \\ F_j \end{bmatrix}$$

First we explored the process of repeatedly squaring the matrix described above:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}^2 = \begin{bmatrix} 5 & 3 \\ 3 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 3 \\ 3 & 2 \end{bmatrix}^2 = \begin{bmatrix} 34 & 21 \\ 21 & 13 \end{bmatrix}$$

So we begin with Fib(n) at the root of the call tree. It splits into two parallel calls to Fib(n/2), and so forth, until there are n processors at the leaves all calculating:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2$$

.
The tree evaluates the matrix multiplications in parallel. The tree has a height of $log\ n$ and the matrix multiplication takes O(1) time since the dimensions are static (2).
The top-left cell of the resulting matrix at the root contains Fib(n).

Extra Credit: Reduction 24. The input to the triangle problem is a subset W of the Cartesian product $X \times Y \times Z$ of sets X, Y and Z, each of cardinality n. The problem is to determine if there is a subset U of W such that 1) every element of X is in exactly one element of U, 2) every element of Y is in exactly one element of U, and 3) every element of Z is in exactly one element of U. Here's a story version of the same problem. You have disjoint collections of n pilots, n copilots, and n flight engineers. For each possible triple of pilot, copilot, and flight engineer, you know if these three people are compatible or not. You goal is to determine if you can assign these $3n$ people to n flights so that every flight has one pilot, one copilot, and one flight engineer that are compatible. Show that this problem is NP-hard using a reduction from 3SAT.