

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 +=  $a_{i,k} \cdot b_{k,j}$ 
     $c_{i,j} = \text{sum}$ 
```

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:
       $\text{product}_{i,j,k} = a_{i,k} \cdot b_{k,j}$ 
    Each element of C is the sum of producti,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, Ci,j is computed.
     $c_{i,j} = \sum_{k=1}^n \text{product}_{i,j,k}$ .
```

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
     $\text{product}_{i/n^2, i/n, i\%n+j} = a_{i/n^2, i\%n+j} \cdot b_{i\%n+j, i/n}$ 
parallel-for i = 1 to n:
  parallel-for j = 1 to n:
    Now we have n/log n processors to compute the sum of producti,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 $p(k)$. Your 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.

8. We consider the problem of computing F_n , the n th 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 $j > 0$

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

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. Your 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.