Homework 21

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

21 October 2016

Reductions

Problem 20:

Problem 21:

Parallel

Problem 3:

To find an EREW algorithm, the key component is that no two cores can read the value of x at the same time. Our solution in $\mathcal{O}(lgn)$ is have the first core populate an array of size 1 with the value x. Then, until you have an array of sufficient size, allocate a new array of twice the size of the previous, and assign twice as many cores as previous to each copy the values at a mutually exclusive index into the new array at two mutually exclusive locations. Repeat this until you have sufficient size.

That is, on the ith step, there is an array of size 2^{i-2} from the previous iteration, and an array of size 2^{i-1} on this iteration. Have the ith core copy the values at index i in the previous array to indices $i$and $2i$ in the new array.

Let $p = n$

$E(n,p) = \frac{S}{pT(n,p)} = \frac{n}{nlg(n)} = \frac{1}{lg(n)}$

By the folding principle, your run time would be no worse than $n^{\frac{2}{3}} \log n$.

Next, an EREW PRAM algorithm can be developed in an identical manner as the above EREW algorithm except $\frac{n}{\log n}$ processors are used. However, unlike the previous problem where at the leaves of the recursive tree each processor has only 1 assignment to do, in this algorithm each processor will have to do $\log n$ assignments into the array and will have to allocate an array of size $\log n$ greater than the previous processor to provide the necessary exclusive read slots.

The efficiency of this algorithm can be calculated as follows:

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

$$E(n,p) = \frac{n}{(n/\log n)\log n}$$

$$E(n,p) = 1$$

Finally, by the folding principle, decreasing the number of processors by a factor of $n^{\frac{2}{3}}$ will result in a runtime no worse than $n^{\frac{2}{3}} \log n$

Lastly, we provide a CRCW Common algorithm. This one is much easier! Simply have one core allocate the array of size n, and for each uniquely numbered core from $i := 1 \to n$, have the ith core copy x into the ith index of the array. Return that array!

Technically, since we're only reading from shared memory and writing, our algorithm is CREW, but all CREW algorithms are also CRCW algorithms. Our algorithm runs in constant time since each core only performs a single task once (other than the initial array allocation).

$E(n,p) = \frac{S}{pT(n,p)} = \frac{n}{n} = 1$

By the folding principle, your run time would be no worse than $n^{\frac{1}{3}}$.

Problem 5:

Factorial is merely multiplication n times, and multiplication is associative. We have talked in detail about parallelizing associative operations. When $p = n$, we can give half of the cores a single multiplication to perform, resulting in $n/2$ partial solutions. Then, we use half of the cores employed in the last operation to aggregate the results into $n/4$ partial solutions. This aggregation step continues until there is only one solution, which is returned.

To state this more imperatively: in the first step, the cores are numbered from 1 to $n/2$. The ith core multiplies i and $2i$, storing the result temporarily. Then, this is repeated, however, instead of using i and $2i$, the next set of $n/4$ cores multiples together the *result* of the ith core's previous multiplication with the result of the $2i$th core's multiplication. The following code makes this approach fairly obvious. This is an EREW algorithm because all array indices are only read from or written to by the core that owns them; access is exclusive.

Function: *Factorial*
Input: *n: Integer*
Core 1: Initialize array A of size n in shared memory
for $i := 1; i \leq n; i++$ **do**
    `/* In parallel */`
    $A[i] = i$
for $i := n/2; i \leq 1; i/ = 2$ **do**
    Core 1: Initialize array B of size i in shared memory
    foreach k *to* i **do**
        `/* In parallel */`
        $B[k] = A[k] * A[2k]$
    $A \leftarrow B$
Return: *A[1]*
