Technische Universität München
Institut für Informatik
Theoretische Informatik

WS 2015/16
Worksheet 8
7.12.2015

Fundamental Algorithms 8

Exercise 1

Write a parallel program that computes the scalar product of two vectors (stored in two arrays). Discuss the runtime complexity on the EREW PRAM model. How many processors can be used?

Solution

Sequential algorithm

```
ScalarProduct(A:Array[1..n], B:Array[1..n]) : Integer {
  res := 0;
  for i from 1 to n do
    res = res + A[i]*B[i];
  return res;
}
```

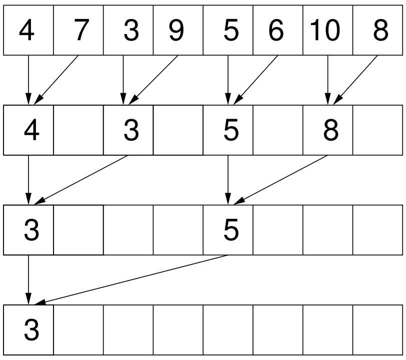
Parallel version: first compute vector product in parallel, then use fan-in to compute sum:

```
ScalarProductPRAM(A:Array[1..n], B:Array[1..n]) : Integer {
  // n assumed to be 2^k
  // Model: EREW PRAM
  Create Array C[1..n];

  for i from 1 to n do in parallel
    C[i] = A[i]*B[i];

  for L from 0 to k-1 do
    for j from 1 by 2^(L+1) to n do in parallel
      C[j] = C[j]+C[j+2^L];

  return C[1];
}
```

- First loop: n processors, second one $n/2$.
- Time complexity thus: $\Theta(\log n)$, as $k = \log n$, on n processors (due to first loop)

- Time complexity of $\Theta(\log n)$ on $n/2$ processors would also be possible, because the first loop could also be executed on $n/2$ processors in $\Theta(1)$ runtime.

For the binary fan-in, the given implementation corresponds to the following scheme:



Exercise 2

Extend the program of exercise 1 to compute a matrix-vector or matrix-matrix product. Again, discuss the runtime complexity on the EREW PRAM and state the number of processors that are used.

Solution for matrix-vector product

Sequential algorithm

```
MatrixVectorProduct(M: Array[1..n,1..n], X: Array[1..n]) : Array[1..n] {
    for i from 1 to n do
      C[i] = 0
       for j from 1 to n do
         C[i] = C[i] + M[i,j]*X[i];
    return C;
}
```

Parallel version

```
MatrixVectorProductPRAM(M: Array[1..n,1..n], X: Array[1..n]): Array[1..n]{
    // n assumed to be 2^k

    for i from 1 to n do in parallel
      C[i] = ScalarProductPRAM(M[i,1..n], X[1..n]);
    return C;
}
```

in $\Theta(\log n)$ due to complexity of ScalarProductPRAM for n^2 processors (also possible with $n^2/2$ processors), using n parallel function calls to ScalarProductPRAM. Problem: concurrent reads to X in ScalarProductPRAM, works only on CREW PRAM, not on EREW PRAM.

Thus, replicate X for each of the n calls to ScalarProductPRAM, and then call ScalarProductPRAM for each copy:

```
MatrixVectorProductEREW(M: Array[1..n,1..n], X: Array[1..n]): Array[1..n]{
  // n assumed to be 2^k
  // Model: EREW PRAM

  for i from 1 to n do in parallel
    XX[1,i] = X[i];

  for l from 1 to k do
    for j from 2^(l-1)+1 to 2^l do in parallel
      for i from 1 to n do in parallel
        XX[j,i] = XX[j-2^(l-1),i];

  for i from 1 to n do in parallel
    C[i] = ScalarProductPRAM(M[i,1..n], XX[i,1..n]);
  return C;
}
```

- The first loop is in $\Theta(1)$ using n processors in parallel,
- the second one in $\Theta(\log n)$, using up to $n^2/2$ processors, and
- the n parallel calls to ScalarProductPRAM as before in $\Theta(\log n)$ each,
- leading to an overall time complexity of $\Theta(\log n)$ using at most n^2 processors at the same time.

Solution for matrix-matrix product

Similar, but one level more to think about.

Exercise 3

Given is the following parallel algorithm for prefix multiplication (for an EREW-PRAM).

```
PrefixPRAM(A: Array[1..n]) {
  // n assumed to be 2^k
  // Model: EREW PRAM (n-1 processors)

  for l from 0 to k-1 do
     for j from 2^l+1 to n do in parallel {
        tmp[j] := A[j-2^l];
        A[j] := tmp[j]*A[j];
     }
}
```

Assume that the j-loop of the above program is changed to

```
     for j from 2^l+1 to n do { ... }
```

(i.e., changed to a sequential loop). State why the resulting algorithm is no longer correct, and suggest how to change the j-loop to obtain a correct sequential implementation. Also, state why the parallel loop works correctly.

Solution:

If the j-loop of the program is changed to

```
for j from 2^l+1 to n do { ... }
```

then A[j−2ˆl] is already changed to its new value, when A[j] is updated. We obtain a correct implementation, if the j-loop is executed in reverse order, or if the j-loop is split into two loops: the first loop to compute all tmp[j], and the second loop to update the A[j]. The parallel loop works correctly, because all tmp[j] are assigned their value at the same time, i.e., before these values are copied to the A[j].