Fundamental Algorithms 4

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 1 to k do
        for j from 1 to 2^{k-l} do in parallel

    return C[1];
}
• First loop: \( n \) processors, second one \( n/2 \).
• Uniform time complexity: \( \Theta(\log n) \), as \( k = \log n \).

**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

```plaintext
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[j];
    return C;
}
```

Parallel computation of the result vector elements:

```plaintext
MVProdPRAM(M: Array[1..n,1..n], X: Array[1..n]) : Array[1..n] {
    // CREW PRAM with n processors
    for i from 1 to n do in parallel
        C[i] = 0
    for j from 1 to n do
        C[i] = C[i] + M[i,j]*X[j];
    return C;
}
```

Here, all write accesses are exclusive, as a single processing unit will compute a single element \( C[i] \), such that all write accesses to \( C[i] \) are initiated by this unit. Vice versa, there is a separate processing unit dedicated to each element of \( C \). Read access to the vector \( X \) is concurrent.

Each of the \( n \) processing units will execute \( \Theta(n) \) arithmetic operations.
Parallel version based on the parallel computation of the scalar product:

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, using parallel function calls to ScalarProductPRAM. Problem: concurrent reads to X in ScalarProductPRAM, works only on CREW PRAM, not on EREW PRAM.

Parallel computation on an EREW PRAM:

We need to 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 $2^l(1−1)+1$ 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], X[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.