Fundamental Algorithms 8 - Solution Examples

Exercise 1 (Parallel Scalar)

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:

Algorithm 1: SCALARSEQ

\textbf{Input:} \( A \) : Array[1..n], \( B \) : Array[1..n]

\textbf{Result:} Scalar product of \( A \) and \( B \)

\( res \leftarrow 0 \);

\textbf{for} \( i = 1 \) \textbf{to} \( n \) \textbf{do} \( res \leftarrow res + A[i] \cdot B[i] \);

\textbf{return} \( res \);

Algorithm 2: SCALARPRAM

\textbf{Input:} \( A \) : Array[1..2^k], \( B \) : Array[1..2^k]

\textbf{Result:} Scalar product of \( A \) and \( B \)

\( C \leftarrow \text{Array}[1..2^k] \);

\textbf{for} \( i = 1 \) \textbf{to} \( n \) \textbf{in parallel} \textbf{do} \( C[i] \leftarrow A[i] \cdot B[i] \);

\textbf{for} \( l = 1 \) \textbf{to} \( k \) \textbf{do}

\textbf{for} \( j = 1 \) \textbf{to} \( 2^{k-l} \) \textbf{in parallel} \textbf{do} \( C[2^l j] \leftarrow C[2^l j] + C[2^l j + 2^{l-1}] \);

\textbf{end}

\textbf{return} \( C[1] \);

In the first loop, \( n \) processors can be used, in the second one only at most \( \frac{1}{2}n \). The time complexity thus is \( \Theta(\log n) \), as \( k = \log n \) on \( n \) processors. The complexity remains \( \Theta(\log n) \) on \( \frac{1}{2}n \) processors, since the first loop could also be executed on \( \frac{1}{2}n \) processors in \( \Theta(1) \) runtime (with each processor executing two multiplications).

Exercise 2 (Parallel Vector)

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

Solution:

Using \( n^2 \) processors, the complexity of MATVECPRAM is \( \Theta(\log n) \) due to the complexity of SCALARPRAM. Unfortunately, this implementation causes concurrent reads to \( X \) in SCALARPRAM, which works only on CREW PRAM, not on EREW PRAM. Instead, one has to replicate \( X \) for each of the \( n \) calls to SCALARPRAM, and then call SCALARPRAM for each copy.

For the first loop, MATVECEREW uses \( n \) processors in parallel to achieve \( \Theta(1) \) runtime. The second one is \( \Theta(\log n) \), using up to \( \frac{1}{2}n^2 \) processors and \( n \) parallel calls to SCALARPRAM (\( \Theta(\log n) \) each). Together, we obtain an overall time complexity of \( \Theta(\log n) \) using at most \( n^2 \) processors.
Algorithm 3: MATVecSEQ

Input: $M$: Array[1..n,1..n]
       $X$: Array[1..n]

Result: Matrix-Vector-product of $M$ and $X$

$C \leftarrow$ Array[1..n];
for $i = 1$ to $n$ do
   $C[i] \leftarrow 0$;
   for $j = 1$ to $n$ do
      $C[i] \leftarrow C[i] + M[i,j] \cdot X[i]$;
   end
end
return $C$;

Algorithm 4: MATVecPRAM

Input: $M$: Array[1..2^k,1..2^k]
       $X$: Array[1..2^k]

Result: Matrix-Vector-product of $M$ and $X$

$C \leftarrow$ Array[1..2^k];
for $i = 1$ to $n$ in parallel do
   $C[i] \leftarrow$ ScalarPRAM($M[i,1..2^k]$, $X[1..2^k]$);
end
return $C$;

Algorithm 5: MATVecEREW

Input: $M$: Array[1..2^k,1..2^k]
       $X$: Array[1..2^k]

Result: Matrix-Vector-product of $M$ and $X$

$C \leftarrow$ Array[1..2^k];
$X' \leftarrow$ Array[1..2^k][1..2^k];
for $i = 1$ to $n$ in parallel do
   $X'[1,i] \leftarrow X[i]$;
for $l = 1$ to $k$ do
   for $j = 1$ to $2^{k-l}$ in parallel do
      for $i = 1$ to $n$ in parallel do
         $X'[2^lj, i] \leftarrow X'[2^lj-2^{l-1}, i]$;
      end
   end
for $i = 1$ to $n$ in parallel do
   $C[i] \leftarrow$ ScalarPRAM($M[i,1..2^k]$, $X[1..n]$);
end
return $C$;

Exercise 3 (Parallel Optimization)

Given the following parallel algorithm PREFIXPRAM for prefix multiplication (with EREW-PRAM).

Assume that the $j$-loop of the above program is 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:

When the $j$-loop of the program is changed to a sequential loop, then $A[j - 2^l]$ 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 $A[j]$.
Algorithm 6: PrefixPRAM

Input: $A$: Array[1..$2^k$]

$\text{tmp} \leftarrow \text{Array}[1..2^k]$;

\textbf{for} $l = 0 \text{ to } k−1 \textbf{ do}

\hspace{1em} \textbf{for } j = 2^l + 1 \text{ to } n \text{ in parallel do}

\hspace{2em} \text{tmp}[j] \leftarrow A[j - 2^l];

\hspace{2em} A[j] \leftarrow \text{tmp}[j] \cdot A[j];

\hspace{1em} \textbf{end}

\textbf{end}