The PRAM Model
and Algorithms

Advanced Topics Spring 2008
Prof. Robert van Engelen
Overview

- The PRAM model of parallel computation
- Simulations between PRAM models
- Work-time presentation framework of parallel algorithms
- Example algorithms
The PRAM Model of Parallel Computation

- Parallel Random Access Machine (PRAM)
- Natural extension of RAM: each processor is a RAM
- Processors operate synchronously
- Earliest and best-known model of parallel computation

*Shared Memory*  

Shared memory with $m$ locations

$p$ processors, each with private memory

All processors operate synchronously, by executing load, store, and operations on data
Synchronous PRAM

- Synchronous PRAM is a SIMD-style model
  - All processors execute the same program
  - All processors execute the same PRAM step instruction stream in “lock-step”
  - Effect of operation depends on local data
  - Instructions can be selectively disabled (if-then-else flow)

- Asynchronous PRAM
  - Several competing models
  - No lock-step
Classification of PRAM Model

- A PRAM step ("clock cycle") consists of three phases
  1. *Read*: each processor may read a value from shared memory
  2. *Compute*: each processor may perform operations on local data
  3. *Write*: each processor may write a value to shared memory

- Model is refined for concurrent read/write capability
  - Exclusive Read Exclusive Write (EREW)
  - Concurrent Read Exclusive Write (CREW)
  - Concurrent Read Concurrent Write (CRCW)

- CRCW PRAM
  - Common CRCW: all processors must write the same value
  - Arbitrary CRCW: one of the processors succeeds in writing
  - Priority CRCW: processor with highest priority succeeds in writing
Comparison of PRAM Models

- A model $A$ is less powerful compared to model $B$ if either
  - The time complexity is asymptotically less in model $B$ for solving a problem compared to $A$
  - Or the time complexity is the same and the work complexity is asymptotically less in model $B$ compared to $A$

- From weakest to strongest:
  - EREW
  - CREW
  - Common CRCW
  - Arbitrary CRCW
  - Priority CRCW
Simulations Between PRAM Models

- An algorithm designed for a weaker model can be executed within the same time complexity and work complexity on a stronger model.
- An algorithm designed for a stronger model can be simulated on a weaker model, either with:
  - Asymptotically more processors (more work)
  - Or asymptotically more time
Simulating a Priority CRCW on an EREW PRAM

- Theorem: An algorithm that runs in $T$ time on the $p$-processor priority CRCW PRAM can be simulated by EREW PRAM to run in $O(T \log p)$ time
  - A concurrent read or write of an $p$-processor CRCW PRAM can be implemented on a $p$-processor EREW PRAM to execute in $O(\log p)$ time
  - $Q_1, \ldots, Q_p$ CRCW processors, such that $Q_i$ has to read (write) $M[j_i]$
  - $P_1, \ldots, P_p$ EREW processors
  - $M_1, \ldots, M_p$ denote shared memory locations for special use
  - $P_i$ stores $<j_i, i>$ in $M_i$
  - Sort pairs in lexicographically non-decreasing order in $O(\log p)$ time using EREW merge sort algorithm
  - Pick representative from each block of pairs that have same first component in $O(1)$ time
  - Representative $P_i$ reads (writes) from $M[k]$ with $<k, _>$ in $M_i$ and copies data to each $M$ in the block in $O(\log p)$ time using EREW segmented parallel prefix algorithm
  - $P_i$ reads data from $M_i$
Reduction on the EREW PRAM

- Reduce $p$ values on the $p$-processor EREW PRAM in $O(\log p)$ time
- Reduction algorithm uses exclusive reads and writes
- Algorithm is the basis of other EREW algorithms
Sum on the EREW PRAM

Sum of $n$ values using $n$ processors ($i$)

Input: $A[1,\ldots,n]$, $n = 2^k$

Output: $S$

begin
    $B[i] := A[i]$
    for $h = 1$ to $\log n$ do
        if $i \leq n/2^h$ then
        end
    if $i = 1$ then
        $S := B[i]$
    end
Matrix Multiplication

- Consider $n \times n$ matrix multiplication with $n^3$ processors
- Each $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ can be computed on the CREW PRAM in parallel using $n$ processors in $O(\log n)$ time

- On the EREW PRAM exclusive reads of $a_{ij}$ and $b_{ij}$ values can be satisfied by making $n$ copies of $a$ and $b$, which takes $O(\log n)$ time with $n$ processors (broadcast tree)
- Total time is still $O(\log n)$
- Memory requirement is huge
Matrix Multiplication on the CREW PRAM

Matrix multiply with $n^3$ processors $(i,j,l)$

Input: $n \times n$ matrices $A$ and $B$, $n = 2^k$

Output: $C = AB$

begin


for $h = 1$ to $\log n$ do

if $i \leq n/2^h$ then

$C'[i,j,l] := C'[i,j,2l-1] + C'[i,j,2l]$

if $l = 1$ then

$C[i,j] := C'[i,j,1]$

end
The WT Scheduling Principle

- The work-time (WT) scheduling principle schedules $p$ processors to execute an algorithm
  - Algorithm has $T(n)$ time steps
  - A time step can be parallel, i.e. pardo
- Let $W_i(n)$ be the number of operations (work) performed in time unit $i$, $1 \leq i \leq T(n)$
- Simulate each set of $W_i(n)$ operations in $\left\lceil \frac{W_i(n)}{p} \right\rceil$ parallel steps, for each $1 \leq i \leq T(n)$
- The $p$-processor PRAM takes
  $$\sum_i \left\lceil \frac{W_i(n)}{p} \right\rceil \leq \sum_i (\left\lfloor \frac{W_i(n)}{p} \right\rfloor + 1) \leq \left\lfloor \frac{W(n)}{p} \right\rfloor + T(n)$$
  steps, where $W(n)$ is the total number of operations
Work-Time Presentation

- The WT presentation can be used to determine computation and communication requirements of an algorithm.
- The upper-level WT presentation framework describes the algorithm in terms of a sequence of time units.
- The lower-level follows the WT scheduling principle.
Matrix Multiplication on the CREW PRAM WT-Presentation

**Input:** $n \times n$ matrices $A$ and $B$, $n = 2^k$

**Output:** $C = AB$

begin

for $1 \leq i, j, l \leq n$ pardo


for $h = 1$ to $\log n$ do

for $1 \leq i, j \leq n$, $1 \leq l \leq n/2^h$ pardo

$C'[i,j,l] := C'[i,j,2l-1] + C'[i,j,2l]$

for $1 \leq i, j \leq n$ pardo

$C[i,j] := C'[i,j,1]$

end

WT scheduling principle: $O(n^3/p + \log n)$ time
PRAM Recursive Prefix Sum Algorithm

Input: Array of \((x_1, x_2, \ldots, x_n)\) elements, \(n = 2^k\)
Output: Prefix sums \(s_i, 1 \leq i \leq n\)
begin
  if \(n = 1\) then \(s_1 = x_1\); exit
  for \(1 \leq i \leq n/2\) pardo
    \(y_i := x_{2i-1} + x_{2i}\)
    Recursively compute prefix sums of \(y\) and store in \(z\)
  for \(1 \leq i \leq n\) pardo
    if \(i\) is even then \(s_i := z_{i/2}\)
    else if \(i = 1\) then \(s_1 := x_1\)
    else \(s_i := z_{(i-1)/2} + x_i\)
end
Proof of Work Optimality

Theorem: The PRAM prefix sum algorithm correctly computes the prefix sum and takes $T(n) = O(\log n)$ time using a total of $W(n) = O(n)$ operations.

Proof by induction on $k$, where input size $n = 2^k$

- Base case $k = 0$: $s_1 = x_1$
- Assume correct for $n = 2^k$
- For $n = 2^{k+1}$
  - For all $1 \leq j \leq n/2$ we have
    $$z_j = y_1 + y_2 + \ldots + y_j = (x_1 + x_2) + (x_3 + x_4) \ldots + (x_{2j-1} + x_{2j})$$
  - Hence, for $i = 2j \leq n$ we have $s_i = s_{2j} = z_j = z_{i/2}$
  - And $i = 2j+1 \leq n$ we have $s_i = s_{2j+1} = s_{2j} + x_{2j+1} = z_j + x_{2j+1} = z_{(i-1)/2} + x_i$
- $T(n) = T(n/2) + a$ $\Rightarrow T(n) = O(\log n)$
- $W(n) = W(n/2) + bn$ $\Rightarrow W(n) = O(n)$
PRAM Nonrecursive Prefix Sum

**Input:** Array $A$ of size $n = 2^k$

**Output:** Prefix sums in $C[0,j]$, $1 \leq j \leq n$

begin

for $1 \leq j \leq n$ pardo

$B[0,j] := A[j]$

for $h = 1$ to $\log n$ do

for $1 \leq j \leq n/2^h$ pardo

$B[h,j] := B[h-1,2j-1] + B[h-1,2j]$

for $h = \log n$ to 0 do

for $1 \leq j \leq n/2^h$ pardo

if $j$ is even then $C[h,j] := C[h+1,j/2]$

else if $i = 1$ then $C[h,1] := B[h,1]$

else $C[h,j] := C[h+1,(j-1)/2] + B[h,j]$

end
First Pass: Bottom-Up

\[ B[3,j] = \]

\[ B[2,j] = \]

\[ B[1,j] = \]

\[ B[0,j] = \]

\[ A[j] = \]
Second Pass: Top-Down

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[0,j]</td>
<td>5</td>
<td>3</td>
<td>-6</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>-2</td>
</tr>
<tr>
<td>C[0,j]</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>11</td>
<td>21</td>
<td>19</td>
</tr>
<tr>
<td>A[j]</td>
<td>5</td>
<td>3</td>
<td>-6</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>-2</td>
</tr>
<tr>
<td>B[1,j]</td>
<td>8</td>
<td>-4</td>
<td>17</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C[1,j]</td>
<td>8</td>
<td>4</td>
<td>21</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[2,j]</td>
<td>4</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C[2,j]</td>
<td>4</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[3,j]</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C[3,j]</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pointer Jumping

- Finding the roots of a forest using pointer-jumping
**Input:** A forest of trees, each with a self-loop at its root, consisting of arcs \((i, P(i))\) and nodes \(i\), where \(1 \leq i \leq n\)

**Output:** For each node \(i\), the root \(S[i]\)

begin
  for \(1 \leq i \leq n\) pardo
    \(S[i] := P[i]\)
    while \(S[i] \neq S[S[i]]\) do
      \(S[i] := S[S[i]]\)
  end

\(T(n) = O(\log h)\) with \(h\) the maximum height of trees

\(W(n) = O(n \log h)\)
PRAM Model Summary

- PRAM removes algorithmic details concerning synchronization and communication, allowing the algorithm designer to focus on problem properties.
- A PRAM algorithm includes an explicit understanding of the operations performed at each time unit and an explicit allocation of processors to jobs at each time unit.
- PRAM design paradigms have turned out to be robust and have been mapped efficiently onto many other parallel models and even network models.
  - A SIMD network model considers communication diameter, bisection width, and scalability properties of the network topology of a parallel machine such as a mesh or hypercube.
Further Reading

- An Introduction to Parallel Algorithms, by J. JaJa, 1992