COMP 633 - Parallel Computing

Lecture 2
August 23, 2018

PRAM (1): The PRAM model and complexity measures
Topics today

• PRAM model
  – execution model
  – programming model

• Work-Time model
  – programming model
  – complexity metrics
  – Brent’s theorem: translation to PRAM programs

• Parallel prefix algorithm
  – derivation
  – applications
PRAM model of parallel computation

- PRAM = Parallel Random Access Machine
  - $p$ processors
  - shared memory
  - each processor has a unique identity $1 \leq i \leq p$
  - synchronous PRAM model
    - Single Instruction, Multiple Data
    - each processor may be active (√) or inactive (×)
    - each instruction is executed only by active processors
    - each instruction completes in unit time
PRAM program

- PRAM program
  - sequential program
  - expressions involving processor id $i$ have a unique value in each processor
    - $i$ can be used as an array index
      \[ X[i] := 10 \times i \]
    - conditionals specify active processors
      \[
      \begin{align*}
      \text{if } \text{odd}(i) \text{ then} \\
      X[i] &:= X[i] + X[i+1] \\
      \text{endif}
      \end{align*}
      \]
      \[
      \begin{align*}
      \text{if } i \leq 2 \text{ then} \\
      X[i] &:= 1 \\
      \text{else} \\
      X[i] &:= -1 \\
      \text{endif}
      \end{align*}
      \]
Concurrent memory access - Read

• Concurrent reads (CR)
  – all readers of a given location see the same value
    \[ X[i] := y \]  \[ X[i] := B\left[ \left\lfloor i/2 \right\rfloor \right] \]
    value of \( y \) read concurrently by all \( p \) processors
    the first \( p/2 \) elements of \( B \) are read concurrently by two processors

• Eliminating bounded-degree concurrent reads
  – replace \( X[i] := B\left[ \left\lfloor i/2 \right\rfloor \right] \) with

\[
\begin{align*}
\text{if} \ & \text{odd}(i) \ \text{then} \quad X[i] := B\left[ \left\lfloor i/2 \right\rfloor \right] \\
\text{endif} \\
\text{if} \ & \text{even}(i) \ \text{then} \quad X[i] := B\left[ \left\lfloor i/2 \right\rfloor \right] \\
\text{endif}
\end{align*}
\]

concert concurrent read is eliminated but number of steps is doubled

Ex. \( p = 6 \)

\begin{array}{c}
\text{X} \\
1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3
\end{array}

\begin{array}{c}
\text{B} \\
1 \quad 2 \quad 3
\end{array}
Concurrent memory access - Write

- Concurrent writes (CW)
  - Stored value depends on write arbitration policy:
    - Arbitrary CW
      - nondeterministic choice among values written
    - Common CW
      - All processors that write a value must write the same value, else error
    - Priority CW
      - value written by processor with lowest processor id
    - Combining Write
      - all values combined using a specified associative operation (e.g. “+”)

- Example (p = 6)

  \[ y := X[i] \]
  \[ B\lceil i/2 \rceil := X[i] \]
Concurrent writes:

- Let \( B[1:p] \) be an array of boolean values and define \( c = B_1 \lor B_2 \lor \ldots \lor B_p \)

  - use \( p \) processors and concurrent writes to compute \( c \) in a constant number of steps
    a) with combining CW
    b) with a CW policy other than combining CW (which?)
Concurrent memory access

- **PRAM variants**
  - EREW, CREW, ERCW, CRCW
  - differ in performance, not expressive power
    - EREW < CREW < CRCW
  - loosely reflect difficulty of model implementation

- **The following are considered EREW**
  - references to
    - processor id \(i\)
    - number of processors \(p\)
    - problem size \(n\)
  
  - references to local variables
    \[
    \text{local } h; \quad h := 2*i + 1; \quad X[h] := X[i]
    \]

  - expression evaluation is synchronous, e.g.
    \[
    X[i] := X[i] + X[i+1]
    \]
    is EREW
A PRAM program

- Simple problem: vector addition
  - given \( V, W \) vectors of length \( n \)
  - compute \( Z = V + W \)

- PRAM program
  - constructed to operate with arbitrary
    - problem size \( n \)
    - number of processors \( p \)
  - work to be performed must explicitly be “scheduled” across processors
  - time complexity with \( p \) procs
    - \( T_c(n, p) = \)
  - PRAM model?

\[\begin{align*}
\text{Input: } V[1:n], W[1:n] \text{ in shared memory} \\
\text{Output: } Z[1:n] \text{ in shared memory}
\end{align*}\]

\[
\text{local integer } h, k \\
\text{for } h := 1 \text{ to } \left\lfloor \frac{n}{p} \right\rfloor \text{ do} \\
\quad k := (h-1) \cdot p + i \\
\quad \text{if } k \leq n \text{ then} \\
\quad \quad Z[k] := V[k] + W[k] \\
\quad \text{endif} \\
\text{enddo}
\]
Work-Time paradigm

• W-T parallel programming model
  – high-level PRAM programming model
    • specifies available parallelism
    • no explicit scheduling of parallelism over processors
  – simplifies algorithm presentation and analysis
  – W-T programs can be mechanically translated to PRAM programs

• W-T program
  – sequential program
  – forall construct
    • specification of available parallelism
    • number of processors is not a parameter of the model!

WT program for vector addition

\[
\text{Input: } V[1:n], W[1:n] \\
\text{Output: } Z[1:n] \\
\text{forall } i \text{ in } 1:n \text{ do} \\
\quad Z[i] := V[i] + W[i] \\
\text{enddo}
\]
Programming notation for the W-T framework

• standard sequential programming notation
  – statements
    • assignment
    • statement composition
    • alternative construct (if ... then ... else ..endif)
    • repetitive construct (for, while)
  – expressions
    • arithmetic and logical functions
    • variable reference
    • (recursive) function and procedure invocation

• forall statement
  – specifies T may be executed simultaneously for each value of i in D
  – no restriction on T
    • can be a sequence of statements
    • can invoke (recursive) functions
    • can be another (nested) forall statement

forall i in D do
  statement T depending on i
enddo
W-T complexity metrics

• Work complexity $W(n)$
  – total number of operations performed (as a function of input size $n$)

• Step complexity $S(n)$
  – number of parallel steps required (as a function of input size $n$)
  – assuming unbounded parallelism

• Inductively defined over constructs of W-T programming notation
W-T complexity measures: simple example

\[
\text{forall } i \text{ in } 2:n-1 \text{ do} \\
\hspace{1cm} R[i] := (R[i-1] + R[i] + R[i+1])/3 \\
\text{enddo}
\]

\[
\text{for } h := 1 \text{ to } k \text{ do} \\
\hspace{1cm} \text{forall } i \text{ in } 2:n-1 \text{ do} \\
\hspace{2cm} R[i] := (R[i-1] + R[i] + R[i+1])/3 \\
\hspace{1cm}\text{enddo} \\
\text{enddo}
\]
Work and Step Complexity of the forall construct

• How to define work and time complexity of the `forall` construct?

\[
\begin{align*}
\text{P: } & \text{ forall } i \text{ in } D \text{ do} \\
& \text{ body } T \text{ depending on } i \\
& \text{ enddo}
\end{align*}
\]

– assume we can determine $W(T_i)$ and $S(T_i)$ for each $i$ in $D$

• $W(P) =$

• $S(P) =$
W-T complexity measures: vector summation

- let $n = 2^k$

```plaintext
forall i in 1:n/2 do
    S[i] := S[2i - 1] + S[2i]
enddo
```

```plaintext
for h := 1 to k do
    forall i in 1:n/2^h do
        S[i] := S[2i - 1] + S[2i]
    enddo
enddo
```

$n = 4$, $k = 2$
W-T complexity measures: vector summation

- Vector summation (sum - reduction)
  - given \( V[1..n] \), \( n = 2^k \)
  - compute \( s = \text{sum}(V[1:n]) \)
  - optimal sequential time \( T_s(n) = \Theta(n) \)

- Complexity
  \( W(n) = \) 
  \( S(n) = \) 

PRAM model needed?

\[ \text{Input: } V[1:n] \text{ vector of integers, } n = 2^k \]
\[ \text{Output: } s = \text{sum}(V[1:n]) \]

\[ \text{P1: } \forall i \in 1:n \text{ do} \]
\[ \quad B[i] := V[i] \]
\[ \quad \text{enddo} \]

\[ \text{P2: } \text{for } h := 1 \text{ to } k \text{ do} \]
\[ \quad \forall i \in 1:n/2^h \text{ do} \]
\[ \quad \quad B[i] := B[2i-1]+B[2i] \]
\[ \quad \text{enddo} \]
\[ \quad \text{enddo} \]

\[ \text{P3: } s := B[1] \]
Brent’s theorem and $T_c(n,p)$

- Brent’s theorem schedules a W-T program for a $p$-processor PRAM
  - idea
    - simulate each parallel step in W-T program using $p$ processors
    - the work $W_i(n)$ to be performed in step $i$ can be completed using $p$ processors in time
      \[
      \left\lceil \frac{W_i(n)}{p} \right\rceil
      \]
  - bound concurrent runtime $T_C(n,p)$ of resultant PRAM program
    - by summing over all $S(n)$ steps

\[
T_c(n,p) = \sum_{i=1}^{S(n)} \left\lceil \frac{W_i(n)}{p} \right\rceil \leq \sum_{i=1}^{S(n)} \left( \left\lceil \frac{W_i(n)}{p} \right\rceil + 1 \right) \leq \left\lceil \sum_{i=1}^{S(n)} \frac{W_i(n)}{p} \right\rceil + S(n) = \left\lceil \frac{W(n)}{p} \right\rceil + S(n)
\]

\[
\left\lceil \frac{W(n)}{p} \right\rceil = \left\lceil \sum_{i=1}^{S(n)} \frac{W_i(n)}{p} \right\rceil \leq \sum_{i=1}^{S(n)} \left\lceil \frac{W_i(n)}{p} \right\rceil = T_c(n,p)
\]
**Scheduling W-T vector summation algorithm**

<table>
<thead>
<tr>
<th>W-T vector summation algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: V[1:n] vector of integers, n = 2^k</td>
</tr>
<tr>
<td><strong>Output</strong>: s = sum(V[1:n])</td>
</tr>
</tbody>
</table>

**P1:**
forall i in 1:n do
    B[i] := V[i]
enddo

**P2:**
for h := 1 to k do
    forall i in 1:n/2^h do
    enddo
enddo

**P3:** s := B[1]

---

<table>
<thead>
<tr>
<th>PRAM vector summation algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: V[1:n] vector of integers, n = 2^k</td>
</tr>
<tr>
<td><strong>Output</strong>: s = sum(V[1:n])</td>
</tr>
<tr>
<td>p &gt; 0 processor PRAM; processor index i</td>
</tr>
</tbody>
</table>

**local integer** j, r;

**P1:**
for j := 1 to ⌈n/p⌉ do
    r := (j-1)•p + i
    if r ≤ n then B[r] := V[r] endif
enddo

**P2:**
for h := 1 to k do
    for j := 1 to ⌈(n/2^h)/p⌉ do
        r := (j-1)•p + i
        if r ≤ n/2^h then
        endif
    enddo
enddo

**P3:** if i ≤ 1 then s := B[1] endif
Performance of translated W-T program

- Count steps needed to perform the additions
  - Brent’s theorem predicts
    \[ T_c(n, p) = O\left(\frac{n-1}{p} + \log n\right) \]
  - counts for various \( p \)
    \[
    \begin{array}{ll}
    p & T_c(n, p) \\
    1 & (n-1)/p \\
    > n & \log n \\
    3, n = 2^k, k \text{ even} & \approx \left\lfloor (n-1)/p \right\rfloor + \frac{1}{2} \log n
    \end{array}
    \]

- Upper bound is tight (for this program)
- translation retains EREW model

PRAM vector summation algorithm

**Input:** \( V[1:n] \) vector of integers, \( n = 2^k \)
**Output:** \( s = \text{sum}(V[1:n]) \)
\( p > 0 \) processor \( PRAM \); processor index \( i \)

**local integer** \( j, r; \)

**P1:**
\[
\text{for } j := 1 \text{ to } \left\lceil n/p \right\rceil \text{ do} \\
\quad r := (j-1) \cdot p + i \\
\quad \text{if } r \leq n \text{ then } B[r] := V[r] \text{ endif} \\
\text{enddo}
\]

**P2:**
\[
\text{for } h := 1 \text{ to } k \text{ do} \\
\quad \text{for } j := 1 \text{ to } \left\lceil (n/2^h)/p \right\rceil \text{ do} \\
\quad\quad r := (j-1) \cdot p + i \\
\quad\quad \text{if } r \leq n/2^h \text{ then} \\
\quad\quad\quad B[r] := B[2r-1] + B[2r] \\
\quad\quad\text{endif} \\
\quad\text{enddo} \\
\text{enddo}
\]

**P3:**
\[
\text{if } i \leq 1 \text{ then } s := B[1] \text{ endif}
\]
Parallel prefix-sum

- **Inclusive prefix sum**
  - **Input**
    - Sequence $X$ of $n = 2^k$ elements, binary associative operator $+$
  - **Output**
    - Sequence $S$ of $n = 2^k$ elements, with $S_i = x_1 + \ldots + x_i$
  - **Example**:
    - $X = [1, 4, 3, 5, 6, 7, 0, 1]$
    - $S = [1, 5, 8, 13, 19, 26, 26, 27]$
    - $T_S(n) = \Theta(n)$

- **Uses of prefix sum**
  - efficient parallel implementation of sequential “scan” through consecutive actions
    - ex: Given series of bank transactions $T[1:n]$, with $T[i]$ positive or negative, and $T[1]$ the opening deposit $> 0$
      - Was the account ever overdrawn?
    - explicit or implicit component of many parallel algorithms
Prefix sum algorithm

- Recursive solution
  - $X_i$ stands for $X[i]$ and $X_{ij}$ stands for $X[i]+X[i+1]+...+X[j]$
Parallel prefix sum algorithm – WT model

**Input:** \( X[1..n] \) vector of integers

**Output:** \( S[1..n] \)

\[
\text{par\_prefix\_sum}( X[1..n] ) = \\
\text{var } Y[1..n/2], Z[1..n/2], S[1..n]; \\
S[1] := X[1]; \\
\text{if } n > 1 \text{ then} \\
\quad \text{forall } 1 \leq i \leq n/2 \text{ do} \\
\quad \quad Y[i] := X[2i-1] + X[2i] \\
\quad \text{endo} \\
\quad Z[1..n/2] := \text{par\_prefix\_sum}(Y[1..n/2]); \\
\text{forall } 2 \leq i \leq n \text{ do} \\
\quad \text{if even}(i) \text{ then} \\
\quad \quad S[i] := Z[i/2] \\
\quad \text{else} \\
\quad \quad S[i] := Z[(i-1)/2] + X[i] \\
\quad \text{endif} \\
\text{endo} \\
\text{return } S[1..n]
\]
Balanced trees in arrays

• Balanced Tree Ascend / Descend
  – Key idea
    • view input data as balanced binary tree
    • sweep tree up and/or down
  – “Tree” not a data structure but a control structure (e.g., recursion)

• Example
  – vector summation

```plaintext
    +
   /|
  +  +
 /|
+ + +
/|
+ + +
/|
1 2 3 4 5 6 7 8
```

```plaintext
  1 2 3 4 5 6 7 8
  1 3 3 10 5 11 7 36
  1 3 3 10 5 11 7 26
  1 3 3 7 5 11 7 15
  1 2 3 4 5 6 7 8
```
In-place prefix sum

- In-place prefix sum
- + ascend phase
- + descend phase
- retained value
- S(n)
- W(n)
- Space
- PRAM model
In-place prefix-sum algorithm – WT model

Input: X[1..n] vector of values, n = 2^k
Output: S[1..n] vector of prefix sums

parallel_prefix_sum( X[1..n] ) =
forall i in 1:n do
    S[i] := X[i]
enddo

for h = 1 to k do
    forall i in 1:n/2^h do
    enddo
enddo

for h = k downto 1
    forall i in 2:n/2^{h-1} do
        if odd(i) then
            S[2^{h-1}i] := S[2^{h-1}i - 2^{h-1}] + S[2^{h-1}i]
        endif
    enddo
enddo
Scan-based primitives

• Scan operations (parallel prefix operations) can be used to implement many useful primitives
  – Suppose we are given SCAN to compute prefix sum of integer sequences
    \[ \text{seq<int> SCAN(seq<int>)} \]
    – step complexity is \( \Theta(\lg n) \)
    – work complexity is \( \Theta(n) \)
    – PRAM model is EREW

• The next three examples have the same complexity as SCAN
COPY (or DISTRIBUTE)

```c
seq<int> COPY (int v, int n)
{
    seq<int> V[1:n];
    V[1] = v;
    forall i in 2 : n do
        V[i] := 0;
    enddo
    return SCAN (V);
}

v = 5
n = 7
V = 5 0 0 0 0 0 0
Res = 5 5 5 5 5 5 5 5
```
**ENUMERATE**

```plaintext
seq<int> ENUMERATE(seq<bool> Flag) {
    seq<int> V[1:#Flag];
    forall i in 1 : #Flag do
        V[i] := Flag[i] ? 1 : 0;
    enddo
    return SCAN(V);
}
```

Flag = T T F T F F F T
V = 1 1 0 1 0 0 0 1
Res = 1 2 2 3 3 3 3 4
seq<T> PACK(seq<T> A, seq<bool> Flag) {

seq<T> R[1:#A];
P := ENUMERATE(Flag);
forall i in 1 : #Flag do
  if Flag[i] then R[P[i]] := A[i] endif;
enddo
return R[1:P[#Flag]];
}

A = ! @ # $ % ^ &
Flag= T T F T F F F T
P = 1 2 2 3 3 3 4
R = ! @ $ &
Radix Sort

Auxiliary: FL[1:n], FH[1:n], BL[1:n], BH[1:n]

for h := 0 to b-1 do
    forall i in 1:n do
        FL[i] := (A[i] bit h) == 0
        FH[i] := (A[i] bit h) != 0
    enddo
    BL := PACK(A,FL)
    BH := PACK(A,FH)
    m := #BL
    forall i in 1:n do
        A[i] := if (i ≤ m) then BL[i] else BH[i-m]endif
    enddo
enddo

S(n) =
W(n) =
Complexity measures for W-T algorithms

- Asymptotic time complexity measures
  - (optimal) sequential time complexity $T_s(n)$
  - parallel time complexity $T_c(n,p)$

- Speedup
  - definition
  $$SP(n, p) = \frac{T_s(n)}{T_c(n, p)}$$

  - limitation
  $$SP(n, p) = \frac{T_s(n)}{T_c(n, p)} \leq \frac{T_s(n)}{W(n)/p} = \frac{pT_s(n)}{W(n)} = O(p)$$

- Average available parallelism
  - definition
  $$AAP(n) = \frac{W(n)}{S(n)}$$
Objectives in the design of W-T algorithms

- **Goal 1**: construct work efficient algorithms
  - a W-T algorithm is work efficient if \( W(n) = \Theta(T_s(n)) \)
  - work-inefficient parallel algorithms have limited appeal on a PRAM with a fixed number of processors \( p \)

\[
\lim_{n \to \infty} SP(n, p) \leq \lim_{n \to \infty} \frac{pT_s(n)}{W(n)} = p \lim_{n \to \infty} \frac{T_s(n)}{W(n)} = 0
\]
Objectives in the design of W-T algorithms

- **Goal 2:** minimize step complexity
  - get optimal speedup using \( AAP(n) = T_s(n) / S(n) \) processors

\[
SP(n, AAP(n)) = \Theta\left(\frac{T_s(n)}{T_c(n, AAP(n))}\right) = \Omega\left(\frac{T_s(n)}{AAP(n) + S(n)}\right)
\]

- when \( S(n) \) is decreased, \( AAP(n) \) is increased
  - with fixed problem size
    - can use more processors to get greater speedup
  - with fixed number of processors
    - reach optimal speedup at smaller problem size
W-T model advantages

- Widely developed body of techniques

- Ignores scheduling, communication and synchronization
  - “easiest” parallel programming

- Source-level complexity metrics
  - Work and step complexity
  - related to running time via Brent’s theorem

- Good place to start
  - many “real-world” algorithms can be derived starting from W-T algorithms