Teaching Parallel Computing through Parallel Prefix

Srinivas Aluru
Iowa State University

Prefix sum

Input: a binary associative operator $\otimes$,
and n elements: $x_0, x_1, x_2, \ldots x_{n-1}$.

Output: n elements: $s_0, s_1, s_2, \ldots s_i \ldots s_{n-1}$;
where $s_i = x_0 \otimes x_1 \otimes \ldots \otimes x_i$.

Example (operator: $+$)

elements	16	23	7	31	9
	16	16 + 23	16 + 23 + 7	16 + 23 + 7 + 31	16 + 23 + 7 + 31 + 9
prefix sums	16	39	46	77	86

Serial algorithm

```
PREFIX_SUM(X, n)
s_0 ← x_0
for i ← 1 to n − 1 do
    s_i ← s_{i−1} ⊕ x_i
end for
return S
```

Note: (1) Run-time $O(n)$.

(2) There is a serial dependency for calculating s_i on s_{i-1}.

How do we parallelize this?

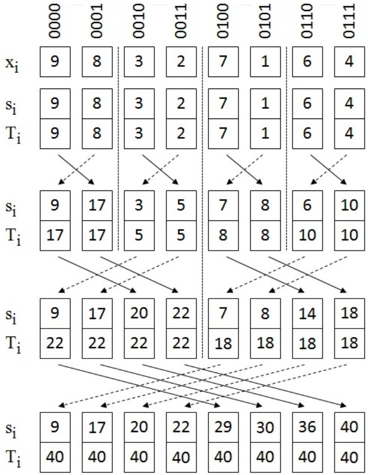
Parallel prefix algorithm

Number of elements $= n$
Number of processors $= p$

Consider the case:

$n = p = 2^d$
Element on P_i : x_i
Prefix sum on P_i : s_i
Total sum on P_i : T_i

Computation time$= O(log\ p)$
Communication time$= O(log\ p)$



Pseudocode

PARALLEL_PREFIX_SUM(id, X_{id}, p)

$prefix\_sum \leftarrow X_{id}$
$total\_sum \leftarrow prefix\_sum$
$d \leftarrow log_2\, p$
for $i \leftarrow 0$ **to** $d - 1$ **do**
    Send $total\_sum$ to the processor with id' where $id' = id \otimes 2^i$
    $total\_sum \leftarrow total\_sum + received\ total\_sum$
    if $id' < id$ **then**
        $prefix\_sum \leftarrow total\_sum + received\ total\_sum$
    end if
end for
return $prefix\_sum$

Note. Run-time $= O(log\ p) \neq \frac{sequential\ runtime}{p} = O(1)$

General solution

Realistic case:

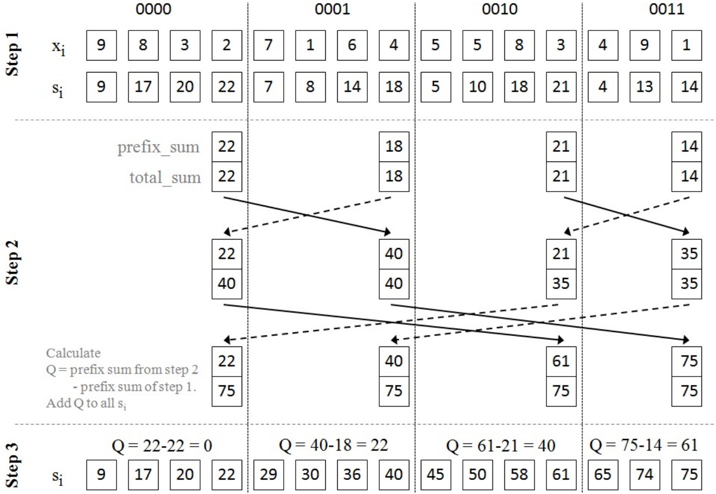
1. $n > p$
2. n is not a multiple of p
   - Each processor has either $\lceil \frac{n}{p} \rceil$ or $\lfloor \frac{n}{p} \rfloor$ elements.
3. p is not a power of 2.
   - $d = \lceil log_2 p \rceil$
   - In any communication phase, do nothing if the computed *id* of the processor to communicate with is $\geq p$.

General solution

Steps (for simplicity think that each processor has $\frac{n}{p}$ elements):

1. Each processor computes the prefix sums of the $\frac{n}{p}$ elements it has locally
2. Using the last prefix sum on each processor, run a p-element parallel prefix algorithm
3. On each processor, combine the result from the parallel prefix algorithm with each local prefix sum computed in Step 1.

Example ($n = 15, p = 4$)



Run-time complexity

- Step 1: Computation of prefix sum locally of $\frac{n}{p}$ elements.
  - Computation time $= O(\frac{n}{p})$
  - Communication time $= 0$
- Step 2: Parallel prefix using last prefix sum on each processor
  - Computation time $= O(log\ p)$
  - Communication time $= O(log\ p)$
- Step 3: Updating $\frac{n}{p}$ prefix sums from step 1 with results from step 2.
  - Computation time $= O(\frac{n}{p})$
  - Communication time $= 0$
- **Overall**
  - Computation time $= O(\frac{n}{p} + log\ p)$
  - Communication time $= O(log\ p)$

Applications

1. Evaluation of a polynomial
2. Solving linear recurrences
3. Random number generation
4. Sequence alignment
5. N-body problem

Evaluation of Polynomial

Input: (1) A real number x_0,
(2) n integer coefficiencts $\{a_0, a_1, a_2 \ldots a_{n-1}\}$.
Output: $P(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_{n-1} x_0^{n-1}$.

- Sequential run-time: $O(n)$.

Solution using parallel prefix

- Let $a_i's$ be distributed evenly on p processors.
- Hence processor P_i has $a_{i\frac{n}{p}}$ to $a_{(i+1)\frac{n}{p}-1}$.
- Local sum required on processor P_i,

$$sum(i) = \sum_{j=0}^{\frac{n}{p}-1} a_{i\frac{n}{p}+j}\, x_0^{i\frac{n}{p}+j}$$

- To get required powers of x_0, we use parallel prefix.

Solution using parallel prefix

- P_0 reads x_0 and broadcasts to all processors.
- Run n-element parallel prefix using x_0 and operator X.
- Processor P_i has $x_0^{i\frac{n}{p}}$.
- Each processor computes sum of $\frac{n}{p}$ terms in $O(n/p)$ time.

Run-time: $O(\frac{n}{p} + \log p)$.

Linear Recurrences

Input: (1) Real numbers x_0, x_1.
(2) Integer coefficients a, b.

Output: Sequence $\{x_2, x_3, ..., x_n\}$ such that $x_i = ax_{i-1} + bx_{i-2}$

- Relation can be rewritten as $\begin{bmatrix} x_i & x_{i-1} \end{bmatrix} = \begin{bmatrix} x_{i-1} & x_{i-2} \end{bmatrix} \begin{bmatrix} a & 1 \\ b & 0 \end{bmatrix}$
- Hence $\begin{bmatrix} x_i & x_{i-1} \end{bmatrix} = \begin{bmatrix} x_1 & x_0 \end{bmatrix} \begin{bmatrix} a & 1 \\ b & 0 \end{bmatrix}^{i-1}$
- Can be extended to dependency on previous k terms.

Random number generation

Input: (1) Integer *multiplier a*
(2) Integer *increment b*
(3) Integer *modulus m*

Output: Pseudo random sequence $\{x_1, ..., x_n\}$ according to Linear Congruential Generator: $x_{i+1} = (ax_i + b) \bmod m$.

- $\begin{bmatrix} ax_i + b & 1 \end{bmatrix} = \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} a & 1 \\ b & 0 \end{bmatrix} \bmod m$
- If all additions are mod$m$, then

$$\begin{bmatrix} x_{i+1} & 1 \end{bmatrix} = \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} a & 1 \\ b & 0 \end{bmatrix}$$

- Hence $\begin{bmatrix} x_i & 1 \end{bmatrix} = \begin{bmatrix} x_0 & 1 \end{bmatrix} \begin{bmatrix} a & 1 \\ b & 0 \end{bmatrix}^i$

Sequence alignment

- An important problem in computational biology.
- DNA seqs: Strings over $\{A, C, G, T\}$.
- Goal: To find out how "well" the sequences align.
- Alignment: Stacking chars of each sequence into columns.
- Gaps (-) may be inserted for missing characters.

Example alignment and score computation

- Alignment has a score that shows quality.
- Every column of an alignment is a match, mismatch or a gap.
- Matches are preferred and hence have a positive score, others have a negative score.
- Example: If $match = 1, mismatch = 0$ and $gap = -1$, then for the following alignment $Score(\text{ATGACC, AGAATC}) = 2$

A	T	G	A	-	C	C
A	–	G	A	A	T	C
1	-1	1	1	-1	0	1

Problem definition

Input: (1) Sequences $A = a_1, a_2, ..., a_m$ and $B = b_1, b_2, ..., b_n$.
(2) Scores for match (M), mismatch (M') and gap (g).

Output: Alignment with maximum score.

Dynamic programming solution:

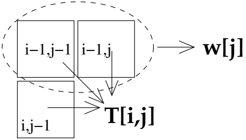
- T = Table of size $(m+1)x(n+1)$.
- $T[i,j]$ = best score between $a_1...a_i$ and $b_1...b_j$.

$$T[i,j] = max \begin{cases} T[i-1,j] - g \\ T[i,j-1] - g \\ T[i-1,j-1] + f(a_i, b_j) \end{cases}$$

- Sequential time = $O(mn)$.

Solution using parallel prefix

We compute each row of T using parallel prefix



$$w[j] = max \begin{cases} T[i-1,j] - g \\ T[i-1,j-1] + f(a_i, b_j) \end{cases}$$

Hence

$$T[i,j] = max \begin{cases} w[j] \\ T[i,j-1] - g \end{cases}$$

Solution using parallel prefix

- Let $x[j] = T[i,j] + jg$. $T[i,j]$ can be computed from $x[j]$.
- Hence

$$x[j] = max \begin{cases} w[j] + jg \\ x[j-1] \end{cases}$$

- Compute $x[j]$ using parallel prefix.

$$\text{Parallel run time: } O(\frac{mn}{p} + m\log p)$$

$$= O(\frac{mn}{p}) \text{ (if } p\log p = O(n))$$

Upward/Downward accumulation

Input: (1) Tree with nodes $\{v_1 ... v_n\}$.
(2) Number x_i at node v_i.

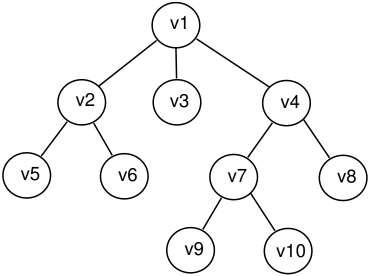
Output(UA): At each node v_i, sum of no.s at all descendant of v_i.

Output(DA): At each node v_i, sum of no.s at all ancestors of v_i.

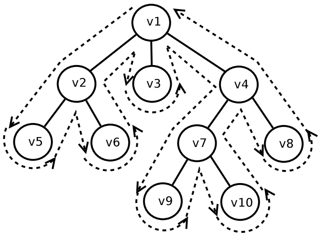
$$\text{Sequential runtime} = O(n)$$

Example

-$UA(v_4) = x_4 + x_7 + x_8 + x_9 + x_{10}$
-$DA(v_7) = x_1 + x_4 + x_7$



Euler tour



- E: v_1 v_2 v_5 v_2 v_6 v_2 v_1 v_3 v_1 v_4 v_7 v_9 v_7 v_{10} v_7 v_4 v_8 v_4 v_1
- Tour Length $= 1 + 2(|V| - 1)$

Solution using Euler tour

- Assume we have the Euler tour.
- For UA, Create array A, $|A| = |E|$, with the following rule:
  - If $E[j]$ is the first occurance of v_i, then $A[j] = x_i$.
  - Else $A[j] = 0$
- $A[j]$ can be computed using $E[j-1]$, $E[j]$, $E[j+1]$.

E:	v_1	v_2	v_5	v_2	v_6	v_2	v_1	v_3	v_1	v_4	v_7	v_9	v_7	v_{10}	v_7	v_4	v_8	v_4	v_1
A:	x_1	x_2	x_5	0	x_6	0	0	x_3	0	x_4	x_7	x_9	0	x_{10}	0	0	x_8	0	0

Solution using Euler tour

- Apply parallel prefix on A.
- i_f: Index of first occurance of v_i in E.
- i_l: Index of last occurance of v_i in E.
- $UA(v_i) = A[i_l] - A[i_f - 1]$.

E:	v_1	v_2	v_5	v_2	v_6	v_2	v_1	v_3	v_1	v_4	v_7	v_9	v_7	v_{10}	v_7	v_4	v_8	v_4	v_1
A:	x_1	x_2	x_5	0	x_6	0	0	x_3	0	x_4	x_7	x_9	0	x_{10}	0	0	x_8	0	0

Solution using Euler tour

- For DA, Create array B, $|B| = |E|$, with the following rule:
  - If $E[j] = v_i$ and v_i is a leaf, then $B[j] = 0$.
  - If $E[j] = v_i$ and v_i is the first occurance of v_i, $B[j] = x_i$.
  - If $E[j] = v_i$ and v_i is the last occurance of v_i, $B[j] = -x_i$.
  - For every thing else, $A[j] = 0$

E:	v_1	v_2	v_5	v_2	v_6	v_2	v_1	v_3	v_1	v_4	v_7	v_9	v_7	v_{10}	v_7	v_4	v_8	v_4	v_1
B:	x_1	x_2	0	0	0	$-x_2$	0	0	0	x_4	x_7	0	0	0	$-x_7$	0	0	$-x_4$	$-x_1$

Solution using Euler tour

- Apply parallel prefix on B.
- i_f: Index of first occurance of v_i in E.
- If v_i is a leaf, $DA(v_i) = B[i_f] + x_i$.
- Else $DA(v_i) = B[i_f]$.

E:	v_1	v_2	v_5	v_2	v_6	v_2	v_1	v_3	v_1	v_4	v_7	v_9	v_7	v_{10}	v_7	v_4	v_8	v_4	v_1
B:	x_1	x_2	0	0	0	$-x_2$	0	0	0	x_4	x_7	0	0	0	$-x_7$	0	0	$-x_4$	$-x_1$

N-body problem

Input: - n particles $\{p_1 ... p_n\}$ at time t.
- Mass of p_i: m_i.
- Position Vector (P.V.) of p_i at time t: $\overrightarrow{r_i}$
- Velocity Vector (V.V.) of p_i at time t: $\overrightarrow{v_i}$

- Time interval t'.

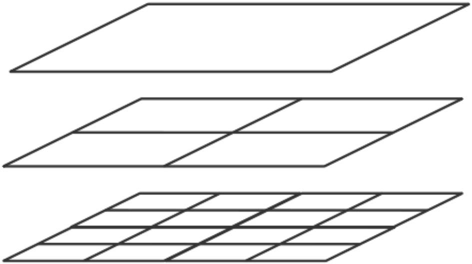
Output: P.V. of all p_i's at $t + t'$

Sequential solution

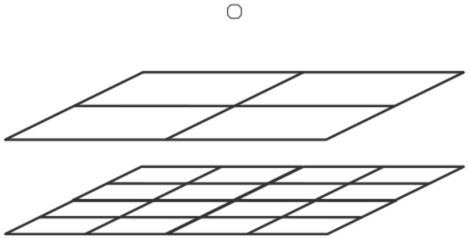
- Acceleration $\overrightarrow{a_i}$ on p_i is assumed constant for interval Δt.
- Compute P.V. after time Δt for each particle.
- Total $\binom{n}{2}$ computations.
- Repeat for $\frac{t'}{\Delta t}$ iterations.
- Run-time: $O(n^2 \frac{t'}{\Delta t})$

Octree

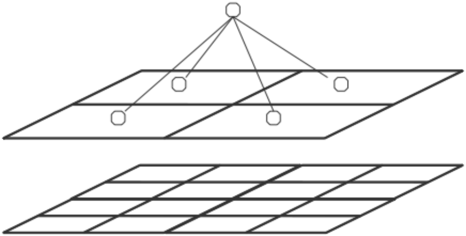
Split the space into octants (quadrants for 2-D) till each cell has one element.



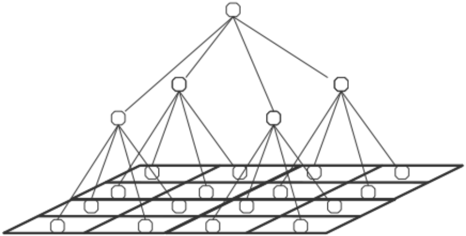
Octree



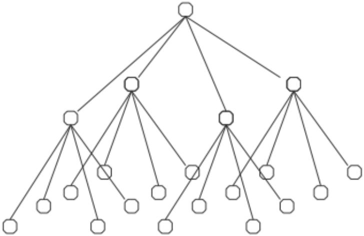
Octree



Octree



Octree



Solution using Upward/Downward accumulation

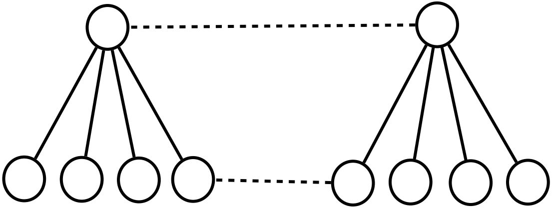
- Each leaf represents a single particle.
- Each internal node represents a cluster (cell).
- For two clusters of size s_i and s_j, acceleration can be calculated using $s_i * s_j$ computations.
- If clusters are far away, approx. acceleration can be calculated using one computation using centers of masses.
- $\overrightarrow{a_{ij}} = \frac{-GM}{||\overrightarrow{r_i} - \overrightarrow{r_j}||^3}.(\overrightarrow{r_i} - \overrightarrow{r_j})$

Solution using Upward/Downward accumulation

- We need collective mass $\sum m_i$ and center of mass $\overrightarrow{r_{cm}}$ at every cell.
- $\overrightarrow{r_{cm}} = \frac{\sum m_i \overrightarrow{r_i}}{\sum m_i}$.
- Both numerator and denominator can be evaluated using upward accumulation

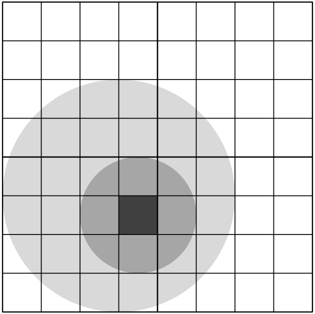
Solution using Upward/Downward accumulation

- If parents of two cells are far away, then acceleration can be calculated between parents.
- One computation instead of 16.



Solution using Upward/Downward accumulation

- If two cells are very near, acceleration has to be calculated between children.
- Acceleration between two cells is calculated if one falls in doughnut region of other.



Solution using Upward/Downward accumulation

- Compute partial acceleration due to cells in the doughnut region for each node.
- Compute total accelerations using downward accumulation.

Acknowledgements

Indranil Roy
Graduate Student, ISU
iroy@iastate.edu

Rahul Nihalani
Graduate Student, ISU
rahuln@iastate.edu