

Advanced Techniques for Combinatorial Algorithms: Parallel Algorithms

Gianluca Della Vedova

Univ. Milano – Bicocca
<https://gianluca.dellavedova.org>

April 27, 2020

RAM model

- Random Access Memory

RAM model

- Random Access Memory
- One processor

RAM model

- Random Access Memory
- One processor
- sequential algorithms

RAM model

- Random Access Memory
- One processor
- sequential algorithms
- Flat memory

RAM model

- Random Access Memory
- One processor
- sequential algorithms
- Flat memory
- Infinite memory

PRAM model

- Parallel RAM

PRAM model

- Parallel RAM
- p RAMs

PRAM model

- Parallel RAM
- p RAMs
- Shared memory

PRAM model

- Parallel RAM
- p RAMs
- Shared memory
- Synchronized (running on the same clock)

PRAM model

- Parallel computation is **rapidly** becoming a **dominant** theme in all areas of computer science and its applications. It is likely that, **within a decade**, virtually all developments in computer architecture, systems programming, computer applications and the design of algorithms will be taking place within the context of parallel computation.

PRAM model

- Parallel computation is **rapidly** becoming a **dominant** theme in all areas of computer science and its applications. It is likely that, **within a decade**, virtually all developments in computer architecture, systems programming, computer applications and the design of algorithms will be taking place within the context of parallel computation.
- Karp, R M. and Ramachandran, V. Chapter 17. Parallel Algorithms for Shared-Memory Machines. Handbook of Theoretical Computer Science: Algorithms and complexity, Volume 1. **1990**.

PRAM model

- Selim Akl, Parallel Computation: Models and Methods, Prentice Hall, 1997.
- Selim Akl, Design & Analysis of Parallel Algorithms, Prentice Hall, 1989.
- Cormen, Leiserson, and Rivest, Introduction to Algorithms, 1st edition, 1990, McGraw Hill and MIT Press, Chapter 30 on parallel algorithms.
- Joseph JaJa, An Introduction to Parallel Algorithms, Addison Wesley, 1992.

PRAM model

- It's a MODEL!

PRAM model

- It's a **MODEL**!
- MIMD (Multiple Instruction Multiple Data)

PRAM model

- It's a **MODEL**!
- MIMD (Multiple Instruction Multiple Data)
- Processor ID

PRAM model

- It's a **MODEL!**
- MIMD (Multiple Instruction Multiple Data)
- Processor ID
- No communication cost

PRAM model

- It's a **MODEL!**
- MIMD (Multiple Instruction Multiple Data)
- Processor ID
- No communication cost
- Shared memory, same access time

PRAM model

	Read	Write
Exclusive	ER	EW
Concurrent	CR	CW

- Different accesses

PRAM model

	Read	Write
Exclusive	ER	EW
Concurrent	CR	CW

- Different accesses
- CRCW is better than EREW.

PRAM model

	Read	Write
Exclusive	ER	EW
Concurrent	CR	CW

- Different accesses
- CRCW is better than EREW.
- But how much?

PRAM model

	Read	Write
Exclusive	ER	EW
Concurrent	CR	CW

- Different accesses
- CRCW is better than EREW.
- But how much?
- There are different CRCW models:

PRAM model

	Read	Write
Exclusive	ER	EW
Concurrent	CR	CW

- Different accesses
- CRCW is better than EREW.
- But how much?
- There are different CRCW models:
 - Common CRCW: concurrent writes if same value from all processors

PRAM model

	Read	Write
Exclusive	ER	EW
Concurrent	CR	CW

- Different accesses
- CRCW is better than EREW.
- But how much?
- There are different CRCW models:
 - Common CRCW: concurrent writes if same value from all processors
 - Priority CRCW: highest priority processor wins

Efficient Algorithm

- $t(n) = \text{polylogarithmic time}$

Efficient Algorithm

- $t(n)$ = polylogarithmic time
- $p(n)$ = polynomial number of processors

Efficient Algorithm

- $t(n)$ = polylogarithmic time
- $p(n)$ = polynomial number of processors
- NC

Efficient Algorithm

- $t(n)$ = polylogarithmic time
- $p(n)$ = polynomial number of processors
- NC
- $\text{NC} \subseteq \text{P}$

Efficient Algorithm

- $t(n)$ = polylogarithmic time
- $p(n)$ = polynomial number of processors
- NC
- $\text{NC} \subseteq \text{P}$
- Hardness = P-complete problems

Simulations

- EREW PRAM can simulate CRCW PRAM

Simulations

- EREW PRAM can simulate CRCW PRAM
- Time multiplied by $O(\log p(n))$

Optimal Algorithm

- **work** $w(n) \leq t(n)p(n)$

Optimal Algorithm

- **work** $w(n) \leq t(n)p(n)$
- $t(n)$ = polylogarithmic time

Optimal Algorithm

- **work** $w(n) \leq t(n)p(n)$
- $t(n)$ = polylogarithmic time
- $w(n) = O(T(n))$, where $T(n)$ = time complexity of **best known** sequential algorithm

Algorithms

Sum of elements of an array

Algorithm 1: Sum

```
1 if  $n = 1$  then  
2   | return  $x[1]$   
3 else  
4   | return  $\text{Sum}(\{x[2i - 1] + x[2i] : 1 \leq i \leq n/2\})$ 
```

Algorithm 2: Iterative Sum

```
1 for  $i \leftarrow 1$  to  $n$  in parallel do  
2   |  $B[i] \leftarrow x[i]$   
3 for  $k \leftarrow 1$  to  $(\log_2 n) - 1$  do  
4   | for  $i \leftarrow 1$  to  $2^{k-1}$  in parallel do  
5     |  $B[i] \leftarrow B[i] + B[i + 1]$   
6 return  $\text{Sum}(\{x[2i - 1] + x[2i] : 1 \leq i \leq n/2\})$ 
```

Prefix sum problem (PRAM)

- Input

Prefix sum problem (PRAM)

- Input
- Sequence $\langle x_1, \dots, x_n \rangle$ of elements

Prefix sum problem (PRAM)

- Input
- Sequence $\langle x_1, \dots, x_n \rangle$ of elements
- Associative operation $+$

Prefix sum problem (PRAM)

- Input
- Sequence $\langle x_1, \dots, x_n \rangle$ of elements
- Associative operation $+$
- Output

Prefix sum problem (PRAM)

- Input
- Sequence $\langle x_1, \dots, x_n \rangle$ of elements
- Associative operation $+$
- Output
- $S = \langle S_1, \dots, S_n \rangle$, with $S_i = x_1 + \dots + x_i$

Prefix sum problem (PRAM)

- Input
- Sequence $\langle x_1, \dots, x_n \rangle$ of elements
- Associative operation $+$
- Output
- $S = \langle S_1, \dots, S_n \rangle$, with $S_i = x_1 + \dots + x_i$
- trivial sequential algorithm

Prefix sum

Algorithm 3: PrefixSum

```
1 if  $n = 1$  then
2   return  $(x_1)$ 
3 for  $i \leftarrow 1$  to  $n/2$  do
4    $y_i \leftarrow x_{2i-1} + x_{2i};$ 
5  $S^* = \text{PrefixSum}([y_1, \dots, y_{n/2}]);$ 
    $/* S_j^* = x_1 + \dots + x_{2j} */$ 
6 for  $i \leftarrow 1$  to  $n$  do
7   if  $i$  is even then
8      $S_i \leftarrow S_{i/2}^*;$ 
9   else
10     $S_i \leftarrow S_{i/2}^* + x_i$ 
11 return  $(S_1, \dots, S_n)$ 
```

Prefix sum

Algorithm 4: PrefixSum

```
1 if  $n = 1$  then
2   return  $(x_1)$ 
3 for  $i \leftarrow 1$  to  $n/2$  in parallel do
4    $y_i \leftarrow x_{2i-1} + x_{2i};$ 
5  $S^* = \text{PrefixSum}([y_1, \dots, y_{n/2}]);$ 
    $/* S_j^* = x_1 + \dots + x_{2j} */$ 
6 for  $i \leftarrow 1$  to  $n$  in parallel do
7   if  $i$  is even then
8      $S_i \leftarrow S_{i/2}^*;$ 
9   else
10     $S_i \leftarrow S_{i/2}^* + x_i$ 
11 return  $(S_1, \dots, S_n)$ 
```

Prefix sum

- EREW

Prefix sum

- EREW
- $O(\log n)$ time, $O(n)$ processors

Prefix sum

- EREW
- $O(\log n)$ time, $O(n)$ processors
- $w(n) = O(n)$

Prefix sum

- EREW
- $O(\log n)$ time, $O(n)$ processors
- $w(n) = O(n)$
- $O(n/\log n)$ processors are enough

Prefix sum

- EREW
- $O(\log n)$ time, $O(n)$ processors
- $w(n) = O(n)$
- $O(n/\log n)$ processors are enough

Brent's scheduling principle

Prefix sum

- EREW
- $O(\log n)$ time, $O(n)$ processors
- $w(n) = O(n)$
- $O(n/\log n)$ processors are enough

Brent's scheduling principle

- w work on p processors in time t

Prefix sum

- EREW
- $O(\log n)$ time, $O(n)$ processors
- $w(n) = O(n)$
- $O(n/\log n)$ processors are enough

Brent's scheduling principle

- w work on p processors in time t
- $p_1 < p$ (use fewer processors)

Prefix sum

- EREW
- $O(\log n)$ time, $O(n)$ processors
- $w(n) = O(n)$
- $O(n/\log n)$ processors are enough

Brent's scheduling principle

- w work on p processors in time t
- $p_1 < p$ (use fewer processors)
- time $\lfloor w/p_1 \rfloor + t$, work w (more time, same work)

Find Maximum

Instance

An array A of n integers

Find Maximum

Instance

An array A of n integers

Question

Find the largest element in A .

Find Maximum

Instance

An array A of n integers

Question

Find the largest element in A .

Goal

Fastest algorithm

Find Maximum

Algorithm 5: Find1. Find Maximum in an Array A

```
1 for  $i \leftarrow 1$  to  $n$  in parallel do
2    $B[i] \leftarrow \text{true};$ 
3 for  $i \leftarrow 1$  to  $n$  in parallel do
4   for  $j \leftarrow 1$  to  $n$  in parallel do
5     if  $A[i] < A[j]$  or  $A[i] = A[j]$  and  $i < j$  then
6        $B[i] \leftarrow \text{false};$ 
7 for  $i \leftarrow 1$  to  $n$  in parallel do
8   if  $B[i]$  then
9     Return  $A[i]$ 
```

Time? Work?

Find Maximum

Algorithm 6: Find1. Find Maximum in an Array A

```
1 for  $i \leftarrow 1$  to  $n$  in parallel do
2    $B[i] \leftarrow \text{true};$ 
3 for  $i \leftarrow 1$  to  $n$  in parallel do
4   for  $j \leftarrow 1$  to  $n$  in parallel do
5     if  $A[i] < A[j]$  or  $A[i] = A[j]$  and  $i < j$  then
6        $B[i] \leftarrow \text{false};$ 
7 for  $i \leftarrow 1$  to  $n$  in parallel do
8   if  $B[i]$  then
9     Return  $A[i]$ 
```

Time? Work? $T(n) = O(1)$, $W(n) = O(n^2)$

Find Maximum

Algorithm 7: Find2. Find Maximum in an Array A

```
1 if  $n > 16$  then
2   for  $i \leftarrow 1$  to  $\sqrt{n}$  in parallel do
3      $B[i] \leftarrow \text{Find2}(A[1 + \lfloor (i-1)/\sqrt{n} \rfloor : \lfloor i/\sqrt{n} \rfloor]);$ 
4    $\text{Find1}(B);$ 
5 else
6    $\text{Find1}(A);$ 
```

$T(n)$

$W(n)$

Find Maximum

Algorithm 8: Find2. Find Maximum in an Array A

```
1 if  $n > 16$  then
2   for  $i \leftarrow 1$  to  $\sqrt{n}$  in parallel do
3      $B[i] \leftarrow \text{Find2}(A[1 + \lfloor (i-1)/\sqrt{n} \rfloor : \lfloor i/\sqrt{n} \rfloor]);$ 
4    $\text{Find1}(B);$ 
5 else
6    $\text{Find1}(A);$ 
```

$$T(n) \leq T(\sqrt{n}) + c_1 \Rightarrow T(n) = O(\log \log n)$$

$$W(n) \leq \sqrt{n}W(\sqrt{n}) + c_2 n \Rightarrow W(n) = O(n \log \log n)$$

Find Maximum

Algorithm 9: Find3. Find Maximum in an Array A

```
1 for  $i \leftarrow 1$  to  $n / \log \log n$  in parallel do  
2    $B[i] \leftarrow \min(A[1 + \lfloor (i - 1) \log \log n \rfloor : \lfloor i / \log \log n \rfloor]);$   
3 Find2( $B$ );
```

$T(n) =$

$W(n) =$

Find Maximum

Algorithm 10: Find3. Find Maximum in an Array A

```
1 for  $i \leftarrow 1$  to  $n / \log \log n$  in parallel do  
2    $B[i] \leftarrow \min(A[1 + \lfloor (i - 1) \log \log n \rfloor : \lfloor i / \log \log n \rfloor]);$   
3 Find2( $B$ );
```

$$T(n) = O(\log \log n)$$

$$W(n) = O(n)$$

Pointer Jumping

- Problem: given a single-link list L , propagate the value of the last element to the entire list

```
1 foreach  $L[i]$  in parallel do
2   for  $k \leftarrow 1$  to  $\log_2 n$  do
3     if  $\text{next}(i) \neq \text{NIL}$  then
4        $\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$ 
5    $\text{value}[i] \leftarrow \text{value}$ 
```

List Ranking

- Problem: given a list L , find the position of each element in L

Algorithm 11: List Ranking via pointer jumping

```
1 foreach  $L[i]$  in parallel do
2   if  $next(i) = NIL$  then
3      $rank[i] \leftarrow 0$ 
4   else
5      $rank[i] \leftarrow 1$ 
6   for  $k \leftarrow 1$  to  $\log_2 n$  do
7      $rank[i] \leftarrow rank[i] + rank[next[i]]$ ;
8      $next[i] \leftarrow next[next[i]]$ 
```

List Ranking

Proof

List Ranking

Proof

- At iteration k :

List Ranking

Proof

- At iteration k :
- if $next[i] \neq NIL$ then $rank[i] = 2^k$

List Ranking

Proof

- At iteration k :
- if $next[i] \neq NIL$ then $rank[i] = 2^k$
- if $next[i] = NIL$ then $rank[i]$ is the distance between $L[i]$ and the end of the list

List Ranking

Proof

- At iteration k :
- if $next[i] \neq NIL$ then $rank[i] = 2^k$
- if $next[i] = NIL$ then $rank[i]$ is the distance between $L[i]$ and the end of the list
- $next[i] = NIL$ for the last 2^k elements of L

Binary trees

- Problem: to determine depth of each node

Binary trees

- Problem: to determine depth of each node
- parent, left child, right child

Binary trees

- Problem: to determine depth of each node
- parent, left child, right child
- 3 processors for each node

Binary trees

- Problem: to determine depth of each node
- parent, left child, right child
- 3 processors for each node
- $O(n)$ -time sequential algorithm

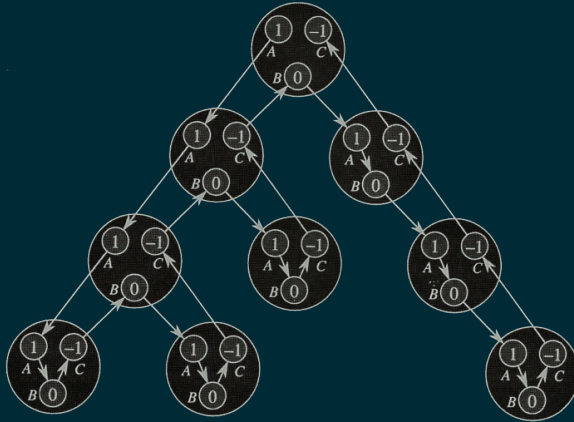
Binary trees

- Problem: to determine depth of each node
- parent, left child, right child
- 3 processors for each node
- $O(n)$ -time sequential algorithm
- Algorithm 1: Level-wise visit, each node in parallel

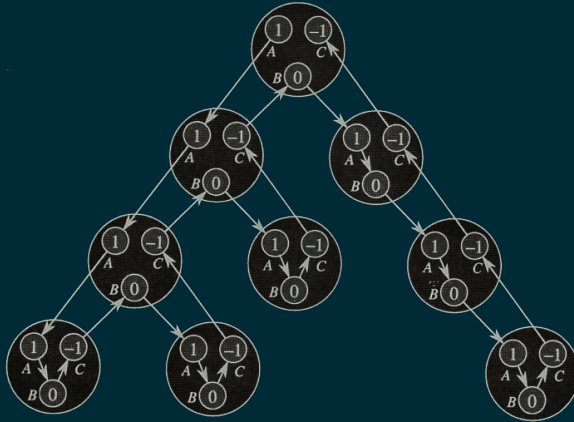
Binary trees

- Problem: to determine depth of each node
- parent, left child, right child
- 3 processors for each node
- $O(n)$ -time sequential algorithm
- Algorithm 1: Level-wise visit, each node in parallel
- $t(n) = \text{height}$ (not good)

Euler tour

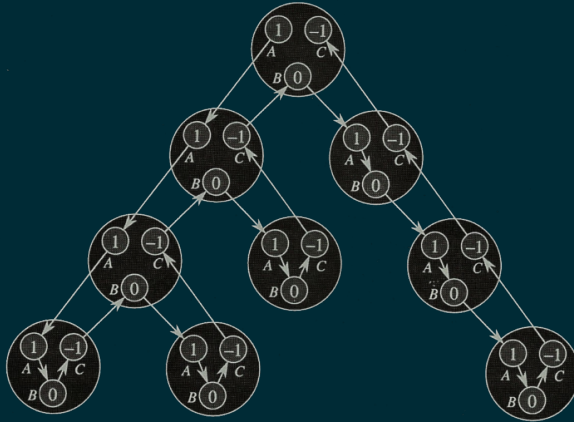


Euler tour

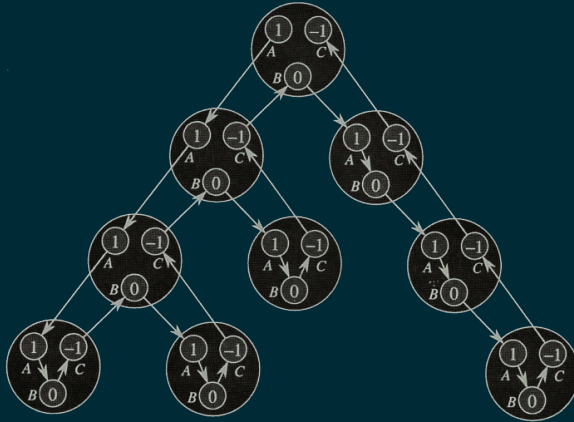


Depth = prefix sum

Size of all subtrees



Size of all subtrees



replace -1 with 0 , difference between third and first prefix sums

Matrix multiplication

- $C = AB$, simpler case A, B square matrices

Matrix multiplication

- $C = AB$, simpler case A, B square matrices
- embarrassingly parallel

Matrix multiplication

- $C = AB$, simpler case A, B square matrices
- embarrassingly parallel
- $C[i, j] = \sum_{k \leq n} A[i, k]B[k, j]$

Matrix multiplication

- $C = AB$, simpler case A, B square matrices
- embarrassingly parallel
- $C[i, j] = \sum_{k \leq n} A[i, k]B[k, j]$
- $O(\log n)$ time, $O(n^3 / \log n)$ processors

Graph Algorithms

- depth-first visit

Graph Algorithms

- depth-first visit
- No NC algorithm

Graph Algorithms

- depth-first visit
- No NC algorithm
- breadth-first visit

Graph Algorithms

- depth-first visit
- No NC algorithm
- breadth-first visit
- $O(n^{2.37})$ processors

Graph Algorithms

- depth-first visit
- No NC algorithm
- breadth-first visit
- $O(n^{2.37})$ processors
- Euler tour

Connected components

Instance

Undirected graph $G = (V, E)$

Connected components

Instance

Undirected graph $G = (V, E)$

Data structure

Connected components

Instance

Undirected graph $G = (V, E)$

Data structure

- 1 $M(v, w)$ adjacency matrix

Connected components

Instance

Undirected graph $G = (V, E)$

Data structure

- 1 $M(v, w)$ adjacency matrix
- 2 $R(v) \leftarrow v$ representative. All vertices in the same connected components have the same representative.

Connected components

Instance

Undirected graph $G = (V, E)$

Data structure

- 1 $M(v, w)$ adjacency matrix
- 2 $R(v) \leftarrow v$ representative. All vertices in the same connected components have the same representative.
- 3 $C[v, w]$ connected components with representative v and w can be merged

Connected components

Algorithm 12: ConnectedComponents

```
1 for  $\log_2 n$  times do hookings
2   foreach edge  $(v, w)$  such that  $R[v] \neq R[w]$  do
3     if  $R[v] < R[w]$  then
4        $C[R[v], R[w]] \leftarrow \text{true};$ 
5   foreach vertex  $v$  such that  $R[v] = v$  do
6      $R[v] \leftarrow \max w : C[R[v], R[w]] \text{ is true};$ 
7   for  $i \leftarrow 1$  to  $\log_2 n$  do parallel pointer jumping
8     foreach vertex  $v$  do
9        $R[v] \leftarrow R[R[v]];$ 
```

Minimum Spanning Tree

Problem

Given an undirected edge-weighted connected graph $G = (V, E)$, find a minimum-weight subset $T \subseteq E$ such that T is a tree spanning V .

Minimum Spanning Tree

Problem

Given an undirected edge-weighted connected graph $G = (V, E)$, find a minimum-weight subset $T \subseteq E$ such that T is a tree spanning V .

Lemma

Let $G = (V, E)$ be an undirected graph, let (V_1, V_2) be a bipartition of V , let T be a minimum spanning tree of G , and let e be the lightest edge connecting V_1 and V_2 . Then $e \in T$.

Additional Bibliography on PRAM

- A Survey of Parallel Algorithms for Shared-Memory Machines
<http://techreports.lib.berkeley.edu/accessPages/CSD-88-408.html>
- Vishkin, Uzi (2009), Thinking in Parallel: Some Basic Data-Parallel Algorithms and Techniques. <http://www.umiacs.umd.edu/users/vishkin/PUBLICATIONS/classnotes.pdf>