

Advanced Techniques for Combinatorial Algorithms: Introduction to Parallel Algorithms

Gianluca Della Vedova

Univ. Milano–Bicocca

<http://gianluca.dellavedova.org>

2016-01-29T18:16:39+0000 71e9e9f

Gianluca Della Vedova

- Advanced Techniques for Combinatorial Algorithms

Gianluca Della Vedova

- Advanced Techniques for Combinatorial Algorithms
- <https://gitlab.com/dellavg/advanced-algorithms>

Gianluca Della Vedova

- Advanced Techniques for Combinatorial Algorithms
- <https://gitlab.com/dellavg/advanced-algorithms>
- <http://gianluca.dellavedova.org>

Gianluca Della Vedova

- Advanced Techniques for Combinatorial Algorithms
- <https://gitlab.com/dellavg/advanced-algorithms>
- <http://gianluca.dellavedova.org>
- gianluca.dellavedova@unimib.it

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

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

Reference: Selim Akl, Parallel Computation: Models and Methods, Prentice Hall, 1997.

Reference: Selim Akl, Design & Analysis of Parallel Algorithms, Prentice Hall, 1989.

Reference: Cormen, Leiserson, and Rivest, Introduction to Algorithms, 1st edition, 1990, McGraw Hill and MIT Press, Chapter 30 on parallel algorithms.

Reference: 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

	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?
- 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?
- 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**
- **NC** \subseteq **P**

Efficient Algorithm

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

Optimal Algorithm

- **work** $w(n) = t(n)p(n)$

Optimal Algorithm

- **work** $w(n) = t(n)p(n)$
- $t(n) = \text{polylogarithmic time}$

Optimal Algorithm

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

Simulations

- EREW PRAM can simulate CRCW PRAM

Simulations

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

Algorithms

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 1: PrefixSum

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

Prefix sum

Algorithm 2: PrefixSum

1 **for** $i \leftarrow 1$ **to** $n/2$ *in parallel* **do**

2 $y_i \leftarrow x_{2i-1} + x_{2i}$;

3 $S_i^* = \text{PrefixSum}([y_1, \dots, y_{n/2}]);$

$/* y_i = x_1 + \dots + x_{2i}$

$*/$

4 **for** $i \leftarrow 1$ **to** n *in parallel* **do**

5 **if** i *is even* **then**

6 $S_i \leftarrow S_{i/2}^*$;

7 **else**

8 $S_i \leftarrow S_{i/2}^* + x_i$

Prefix sum

- EREW

Prefix sum

- EREW
- $O(n)$ processors

Prefix sum

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

Prefix sum

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

Prefix sum

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

Brent's scheduling principle

Prefix sum

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

Brent's scheduling principle

- t parallel steps on p processors

Prefix sum

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

Brent's scheduling principle

- t parallel steps on p processors
- $p_1 < p$

Prefix sum

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

Brent's scheduling principle

- t parallel steps on p processors
- $p_1 < p$
- $\lfloor p/p_i \rfloor + t$ parallel steps

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 3: 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 4: 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 5: Find2. Find Maximum in an Array A

```
1 if  $n > 16$  then
2   for  $i \leftarrow 1$  to  $n$  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    $\text{Find1}(A);$ 
```

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

$$W(n) \leq \sqrt{n}W(\sqrt{n}) + c_2n \Rightarrow W(n) = O(n \log \log n)$$

Find Maximum

Algorithm 6: 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);

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

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

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

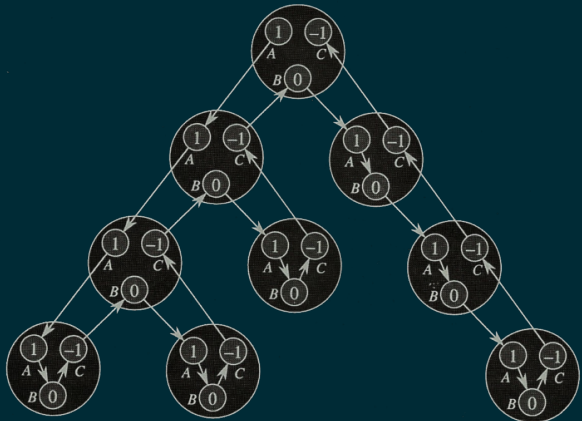
Binary trees

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

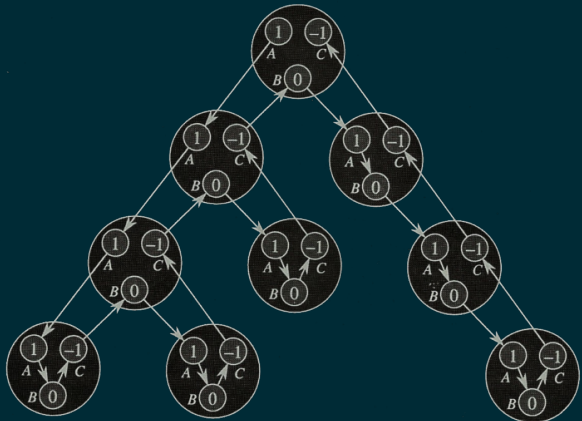
Binary trees

- Problem: to determine depth of each node
- parent, left child, right child
- 3 processors for each node
- $O(n)$ -time sequential algorithm
- Level-wise visit, each node in parallel
- $t(n) = \text{height}$

Euler tour



Euler tour



Depth = prefix sum

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 7: ConnectedComponents

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

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

Reference: A Survey of Parallel Algorithms for Shared-Memory Machines

<http://techreports.lib.berkeley.edu/accessPages/CSD-88-408.html>

Reference: Vishkin, Uzi (2009), Thinking in Parallel: Some Basic Data-Parallel Algorithms and Techniques. <http://www.umiacs.umd.edu/users/vishkin/PUBLICATIONS/classnotes.pdf>