

<http://tadsessions.espace.com.eg>

Parallel Algorithms

Theory and Practices

By: Ali Maher



- Improving the performance of individual processors by reducing the instruction-cycle time has become increasingly difficult, since the speed of electronic circuits is limited by the *speed of light*.
- For those interested in high-speed computing, studying parallel algorithms is no longer an academic exercise; it is a necessity. Many problems can be solved by massive parallelism.

- **Parallel Processing:** is information processing that emphasizes the concurrent manipulation of data elements belonging to one or more processes solving a *single problem*.
- **Parallel Computer:** is a multiple-processor computer capable of parallel processing.

- **Parallel computers:**

- work in tight synchrony, share memory to a large extent and have a very fast and reliable communication mechanism between them.
- cooperate to solve more efficiently difficult problems.
- cooperation in a *positive* sense

- **Distributed computers:**

- more independent, communication is less frequent and less synchronous, and the cooperation is limited.
- have individual goals and private activities.
- Cooperation in a negative sense.

- **Throughput:** The number of results produced per unit time.
- **Speedup:** the ratio between the time needed for the most efficient sequential algorithm to perform a computation and the time needed to perform it on a parallel computer.
- **Parallel / concurrent / Pipelined** operations.

- **Data Parallelism:** The use of multiple functional units to apply the same operation simultaneously to elements of data set.
- **Control Parallelism:** Applying different operations to different data elements simultaneously.

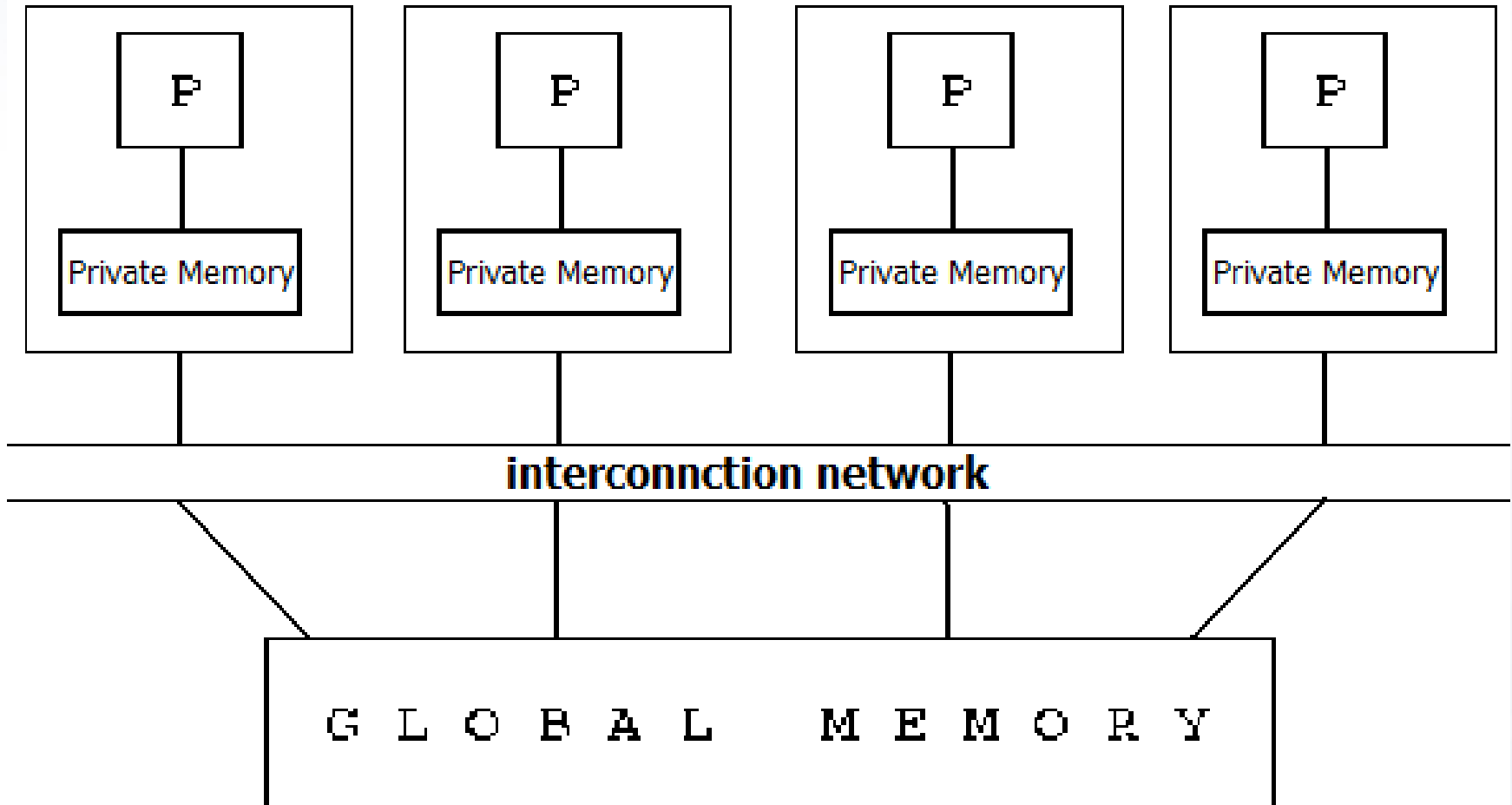
- An algorithm is **scalable** if the level of parallelism increases at least linearly with the problem size. An Architecture is scalable if it continues to yield the same performance per processor, albeit used on a larger problem size, as the number of processors increases.
- Data-parallel algorithms are more scalable than control-parallel algorithms.

- **Amdahl's law** (Amdahl 1967): Let f be the fraction of operations in a computation that must be performed sequentially, where $0 \leq f \leq 1$. The maximum speedup S achievable by a parallel computer with p processors performing the computation is $S \leq 1 / [f + (1-f)/p]$
- **Amdahl effect:** As the size of the problem increase, the fraction f of inherently sequential operations decreases, making the problem more amenable to parallelization.

- Speedup: $S(p) = T_1 / T_p$
- Efficiency: $E = S(p) / p$
- Cost = $p * T_p$
- $T_p = 1 / [f + (1-f)/p]$, where f is the fraction of the problem that is inherently sequential
- Upper bound of the speedup = $1/f$ (using infinite number of processors)

- PRAM (Parallel random access machine) model is a model for parallel computations assumes:
 - Unlimited shared memory
 - Unlimited number of processors, each
 - Has unlimited local memory
 - Knows its ID
 - Able to access the shared memory
 - Ignoring the complexity of interprocess communications.
 - I/Os are placed in the shared memory
 - Instructions are synchronized across the processors

PRAM Model (cont.)



- PRAM is a theoretical unrealistic model to help algorithm designers to focus on parallel computation only.

- At any time unit each Processor can:
 - Read a memory cell
 - Run an internal instruction
 - Write another memory cell
- As a consequence any two processors can communicate in constant time!
 - P1 writes the message in cell x at time t
 - P2 reads the message in cell x at time $t+1$

- for each individual processor
 - ***time***: number of instructions executed
 - ***space***: number of memory cells accessed
- PRAM machine
 - ***time***: time taken by the longest running processor
 - ***hardware***: maximum number of active processors

- Algorithm's designers can forget the communication problems and focus their attention on the parallel computation only.
- There exists algorithms simulating any PRAM algorithm on bounded degree networks.

- EREW (Exclusive Read Exclusive Write)
- CREW (Concurrent Read Exclusive Write)
- ERCW (Exclusive Read Concurrent Write)
- CRCW (Concurrent Read Concurrent Write)

Procedure Sort(modifies A: array 1..n of integer)

for i in 1..n pardo K[i]:=0

for i in 1..n pardo

for j in 1..n pardo

if $A[i] \leq A[j]$ then K[j]:=K[j]+1

for i in 1..n pardo A[K[i]]:=A[i]

end

PRAM Sorting Illustration [1]



A[i]	3	1	4	2	5
3					
1					
4					
2					
5					

Input vector to be sorted

PRAM Sorting Illustration [2]

A[i]	3	1	4	2	5
3	1	0	1	0	1
1	1	1	1	1	1
4	0	0	1	0	1
2	1	0	1	1	1
5	0	0	0	1	1

Processor P_{ij} compares $A[i]$ and $A[j]$:

$P_{ij} := "A[i] \leq A[j]"$

PRAM Sorting Illustration [3]

A[i]		3	1	4	2	5

3		1	0	1	0	1
1		1	1	1	1	1
4		0	0	1	0	1
2		1	0	1	1	1
5		0	0	0	0	1
=====						
K[i]		3	1	5	2	5

Add columnwise to K[i]

PRAM Sorting Illustration [4]

A[i]		3	1	4	2	5

3		1	0	1	0	1
1		1	1	1	1	1
4		0	0	1	0	1
2		1	0	1	1	1
5		0	0	0	0	1

=====						
K[i]		3	1	4	2	5
A[i]		1	2	3	4	5

A[K[j]] := A[j]; output

- `spawn(<processor names>)` is a meta-instruction used to activate all the processors in the argument list.
- Every PRAM algorithm starts with only one active processor.
- Given a single active processor, it is easy to see that $\lceil \log p \rceil$ activation steps are necessary and sufficient for p processors to become active.

- The design of a parallel algorithm can be viewed as consisting of four stages
 - Partitioning.
 - Communication Analysis.
 - Granularity Control.
 - Mapping.

Parallel Algorithm Design Example

Partitioning

- Consider the following scenario: n answer-scripts have to be marked, each of which contains the answers to m questions.
 - **Data:** The scripts
 - **Computation:** The marking process
- **Partitioning:** decomposition into smaller tasks which can be executed simultaneously
 - **Domain** (data) decomposition: Each script could be marked by a different marker
 - **Functional** (computational) decomposition: Each question could be marked by a different marker

Parallel Algorithm Design Example (cont.)

Communication Analysis

- Suppose one needs to compute the average mark of the n scripts.
- The marks from each of the markers would be required by the processor which will compute the average. (**communication** is needed.)
- The nature of the information flow is specified in the **communication analysis** stage of the design.

Parallel Algorithm Design Example (cont.)

Granularity control

- It may be the case that the time to communicate the marks between two markers is much greater than the time to mark a question.
- The solution is to reduce the number of markers and increase the number of scripts per marker. k marker could mark n/k scripts each.
- The **Granularity control** is to find the optimum value of k to balance between the cost of processing and communication.

Mapping

- The **mapping** stage specifies where each task is to execute.
- In this example, all tasks are of equal size and the communication is uniform, so any task can be mapped to any marker. However, in more complex situations, mapping strategies may not be obvious, requiring the use of more sophisticated techniques.

Summation of n element
 $O(\log n)$ algorithm using $\text{floor}(n/2)$ processor



Sum (EREW PRAM)

Initial cond: list of $n \geq 1$ elements $A[0..n-1]$

Final cond: Sum of elements stored in $A[0]$

Global var: $n, A[0..n-1], j$

Begin

spawn($P_0, \dots, P_{\text{floor}(n/2)-1}$)

for all P_i where $0 \leq i \leq \text{floor}(n/2) - 1$ **parado**

for $j = 0$ to $\text{ceil}(\log n) - 1$ **do**

if $i \% 2^j = 0$ and $2i + 2^j < n$ **then**

$A[2i] = A[2i] + A[2i + 2^j]$

endif

endfor

endfor

end

- NC is the class of problems solvable on PRAM in polylogarithmic time $[(\log n)^{O(1)}]$ using a number of processors that is a polynomial function of the problem size.
- Many problems in P is in NC.
- The open question is $NC = P$?

- A problem L is in **P-complete** class if every other problem in **P** can be transformed to L in **polylogarithmic** parallel time using **PRAM** with a **polynomial** number of processors.
- Example:
 - Circuit Value Problem (CVP) - Given a circuit, the inputs to the circuit, and one gate in the circuit, calculate the output of that gate.
 - Linear programming - Maximize a linear function subject to linear inequality constraints

