Complexity and Advanced Algorithms

Introduction to Parallel Algorithms

Why Parallel Computing?

- Save time, resources, memory, ...
- Who is using it?
  - Academia
  - Industry
  - Government
  - Individuals?
- Two practical motivations:
  - Application requirements
  - Architectural concerns.
- Why now?
  - Soon may not be able to buy a good single core computer!
  - Need to therefore study how to use parallel computers.

1. Application Requirements

- A computational fluid dynamics(CFD) calculation on an air plane wing 512 X 64 X 256 grid
  - 5000 fl-pt operations per grid point
  - 5000 steps 2.1x10^{14} ft-ops.
  - 3.5 minutes on a machine sustaining 1 trillion flops
  - simulation of a full aircraft 3.5 x 10^{17} grid points total of 8.7 x 10^{24} ft-pt operations on the same machine requires more than 275,000 years to complete.

1. Application Requirements

- Digital movies and special effects
  - 10^{14} fl-pt operations per frame and
  - 50 frames per second
  - 90-minute movie represents 2.7×10^{19} fl-pt operations.
  - It would take 2,000 1-Gflops CPUs approximately 150 days to complete the computation.

1. Application Requirements

- Simulation of magnetic materials at the level of 2000-atom systems require 2.64 Tflops of computational power and 512 GB of storage.
  - Full hard disk simulation 30 Tflops and 2 TB
  - Current investigations limited about 1000 atoms 0.5 Tflops 250 GB
  - Future investigations involving 10,000 atoms 100 Tflops 2.5TB
- Inventory planning, risk analysis, workforce scheduling and chip design.

2. Architectural Advances

- Moore's Law:
  - The number of transistors that can be inexpensively placed on an integrated circuit is increasing exponentially, doubling approximately every two years.
- Present Difficulties
  - Memory Wall
  - Power Wall
  - ILP Wall

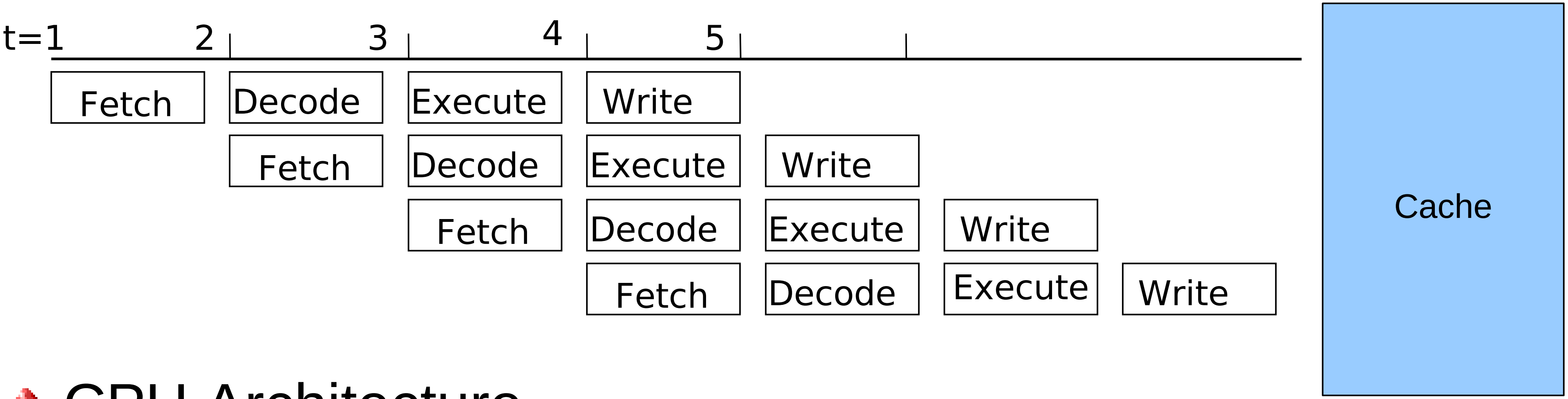
The Brick Wall - 1

- Memory Wall
  - Memory latency up to 200 cycles per load/store.
  - Floating point operations take no more than 4 cycles.
  - Earlier, it was thought that “multiply is slow but load and store is fast”.

The Brick Wall - 2

- Power Wall
  - Enormous increase in power consumption.
  - Power leakage.
  - However, presently “Power is expensive but transistors are free’’.

Basic Architecture Concepts

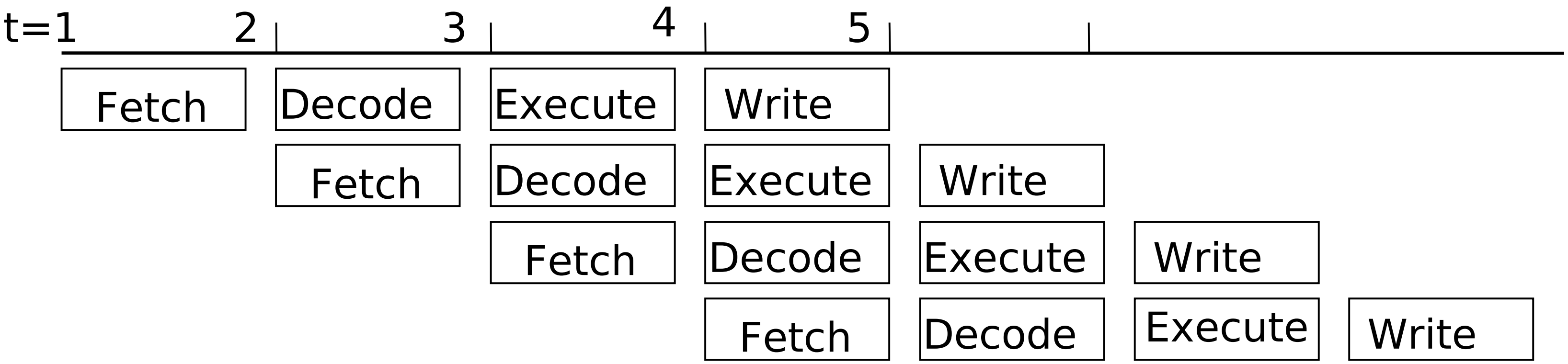


t=1	2	3	4	5		
Fetch	Decode	Execute	Write			
	Fetch	Decode	Execute	Write		
		Fetch	Decode	Execute	Write	
			Fetch	Decode	Execute	Write

Cache

- CPU Architecture
  - 4 stages of instruction execution
    - Too many cycles per instruction (CPI)
  - To reduce the CPI, introduce pipelined execution
    - Needs buffers to store results across stages.
    - A cache to handle slow memory access times

Basic Architecture Concepts



Cache

- CPU Architecture
  - 4 stages of instruction execution
    - Too many cycles per instruction (CPI)
  - To reduce the CPI, introduce pipelined execution
    - Needs buffers to store results across stages.
    - A cache to handle slow memory access times
    - Caches, out-of-order execution, branch prediction, ...

The Brick Wall - 3

- ILP Wall
  - ILP via branch prediction, out-of-order and speculative execution
  - Diminishing returns from instruction level parallelism.

Conventional Wisdom in Computer Architecture

- Power Wall + Memory Wall + ILP Wall = Brick Wall
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Uniprocessor performance only 2X / 5 yrs?

Multicores to the Rescue

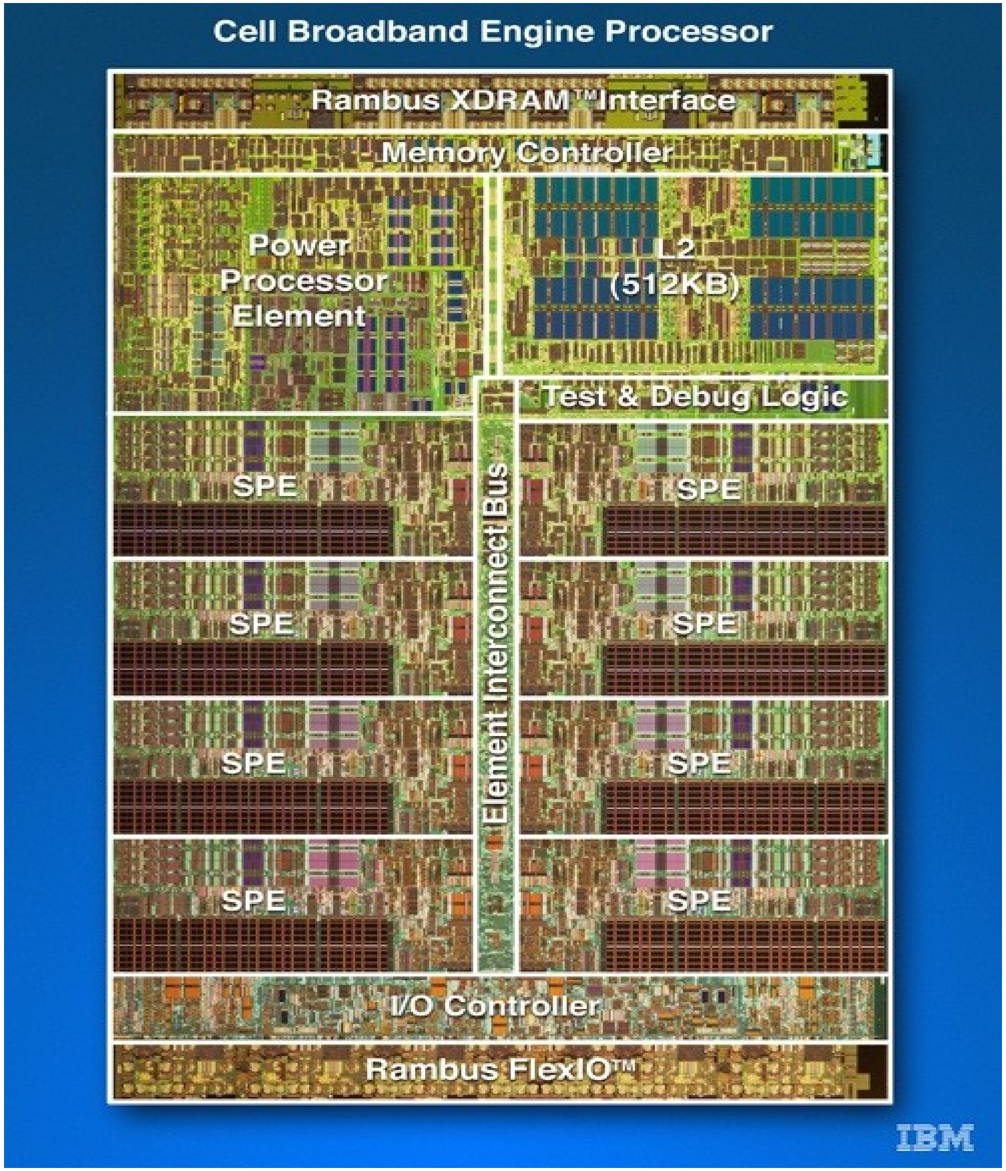
- Predicted that 100+ core computers would be a reality soon.
- Increased number of cores without significant improvement in clock rates.
  - Due to silicon technology improvements
- Big questions
  - How to exploit these cores in parallel?
  - What are the killer applications that can democratize these new models?
    - Search, web, ???

Multicore and Manycore Processors

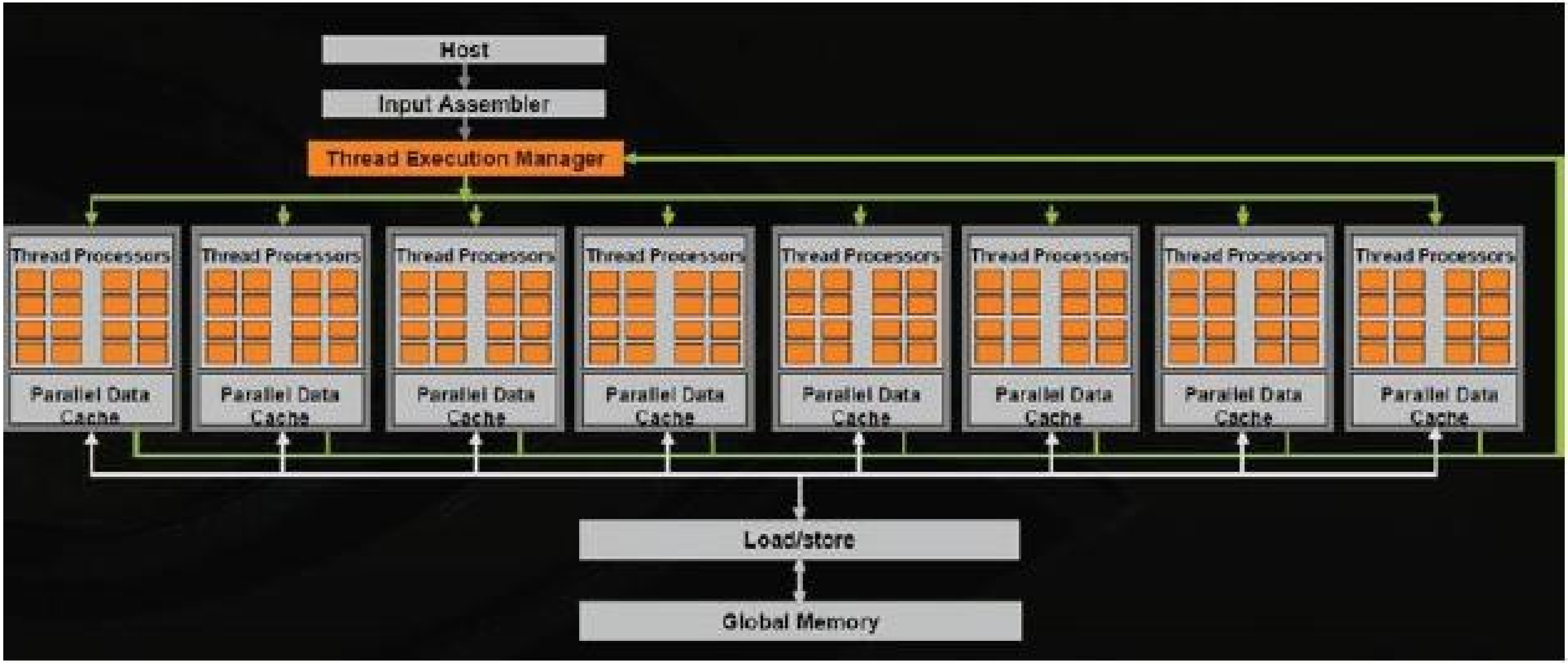
- IBM Cell
- NVidia GeForce 8800 includes 128 scalar processors and Tesla
- Sun T1 and T2
- Tilera Tile64
- Picochip combines 430 simple RISC cores
- Cisco 188
- TRIPS

Cell Broadband Engine

- Cell processing elements
  - A standard PowerPC core
  - 8 SIMD Cores (SPEs)
- Dual XDR memory controller and two I/O controllers
- Game controllers, HPC
- Power 100 W
- SPE local store of 256 KB



Nvidia G8800 Graphics Processor



- Each of 16 cores similar to a vector processor with 8 lanes (128 stream processors total)
  - Processes threads in SIMD groups of 32 (a “warp”)
  - Some stripmining done in hardware
- Threads can branch, but loses performance compared to when all threads are running same code
- Only attains high efficiency on very data-parallel code (10,000s operations)

The NVidia Telsa

- The Tesla C870 GPU computing processor transforms a standard workstation into a personal supercomputer. With 128 streaming processor cores, the CUDA C-language development environment and developer tools, and a range of applications.

Tesla C870 GPU specifications
- One GPU (128 thread processors)
- 518 gigaflops (peak)
- 1.5 GB dedicated memory
- Fits in one full-length, dual slot with one open PCI Express x16 slot

Massively Multi-threaded Processor Architecture
Solve compute problems at your workstation that previously required a large cluster

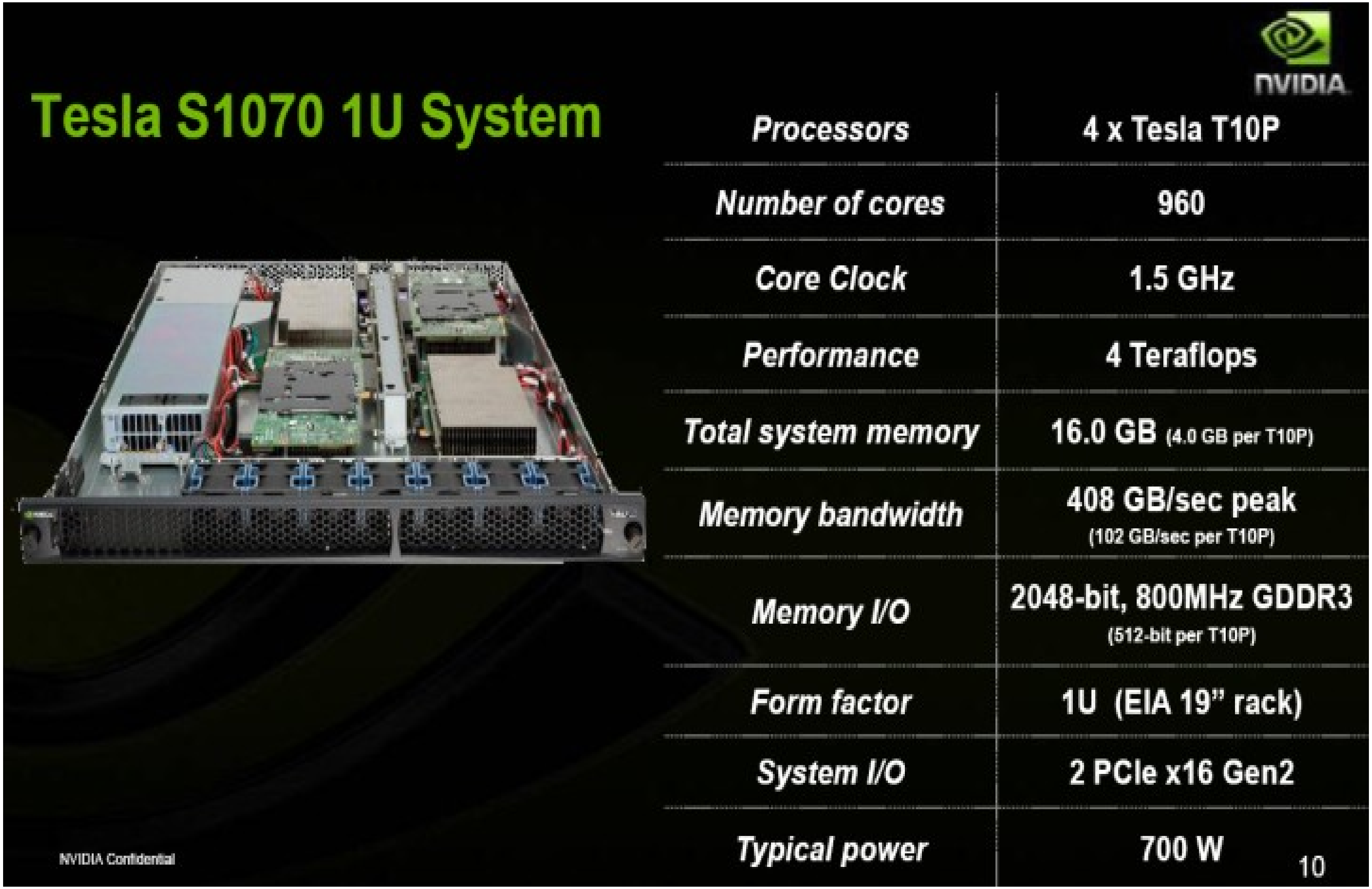
128 Floating Point Processor Cores
Achieve up to 350 GFLOPS of performance (512 GFLOPS peak) with one C870 GPU

Multi-GPU Computing
Solve large-scale problems by dividing it across multiple GPUs

- **Shared Data Memory**
  Groups of processor cores can collaborate using shared data

- **High Speed, PCI-Express Data Transfer**
  Fast and high-bandwidth communication between CPU and GPU

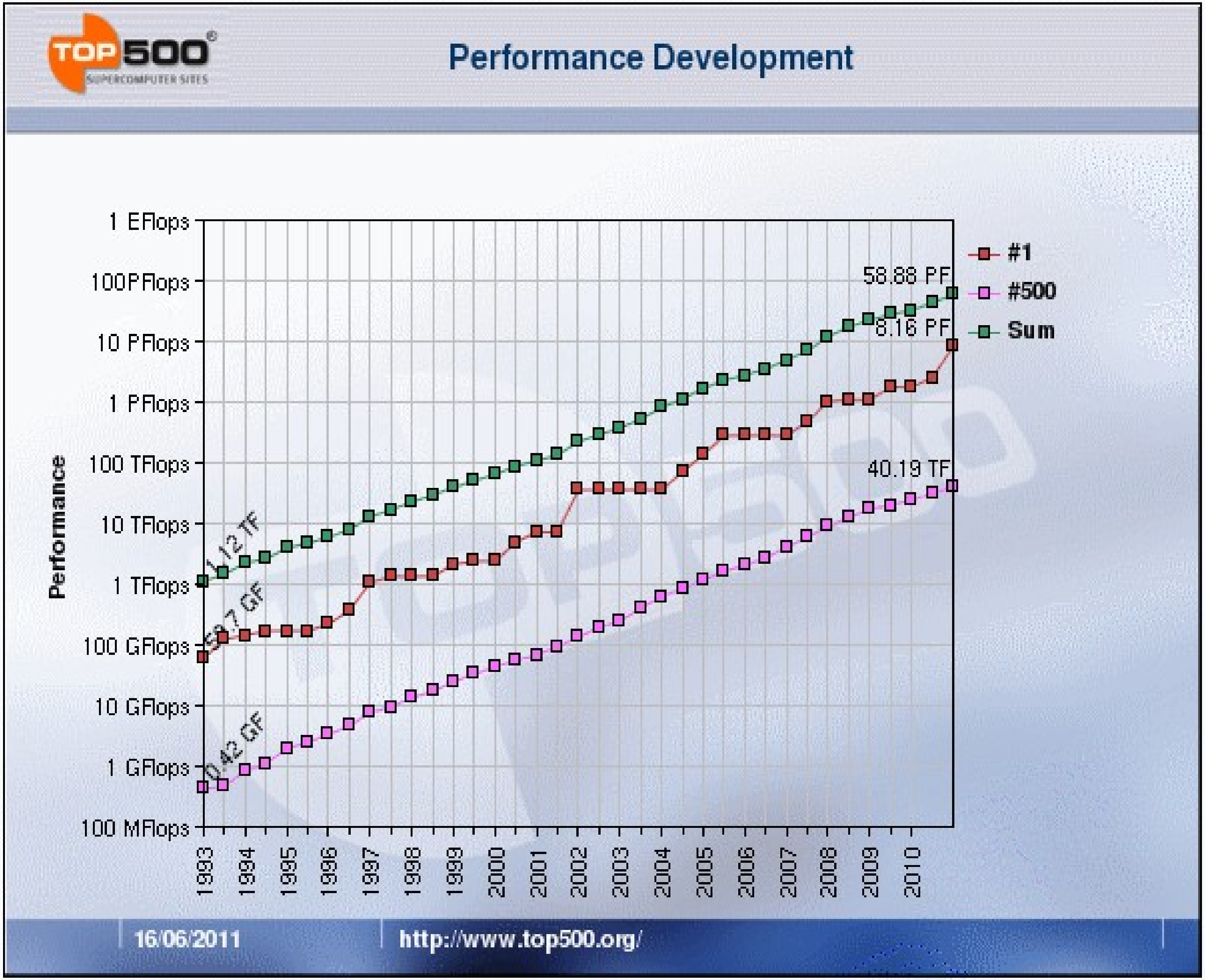
The Nvidia Tesla



Tesla S1070 1U System

Processors	4 x Tesla T10P
Number of cores	960
Core Clock	1.5 GHz
Performance	4 Teraflops
Total system memory	16.0 GB (4.0 GB per T10P)
Memory bandwidth	408 GB/sec peak (102 GB/sec per T10P)
Memory I/O	2048-bit, 800MHz GDDR3 (512-bit per T10P)
Form factor	1U (EIA 19" rack)
System I/O	2 PCIe x16 Gen2
Typical power	700 W

Some Performance Numbers



Some Performance Numbers

The K Supercomputer
Country: Japan
Rated at 8 PFlops

Tianhe 1 Supercomputer
Country: China
Rated at 2 PFlops

The Jaguar Cray XT5
Country: USA
Rated at 1.7 PFlops

The Nebulae
Country: China
Rated at 1.27 PFlops

Tsubame Supercomputer
Country: Japan
Rated at 1.19 PFlops

The Eka Supercomputer
Country: India
Rated at 170 TFlops

The Academic Interest

- Algorithmics and compelxity
  - How to design parallel algorithms?
  - What are good theoretical models for parallel computing?
  - How to analyze parallel algorithms?
  - Can every sequential algorithm be parallelized?
  - What are some complexity classes wrt parallel computing?

The Academic Interest

- Systems and Programming
    - How to write parallel programs?
    - What are some tools and environments.
    - How to convert algorithms to efficient implementations.
    - What are the differences to sequential programming?
    - What are the performance measures?
    - Can sequential programs be automatically converted to parallel programs?

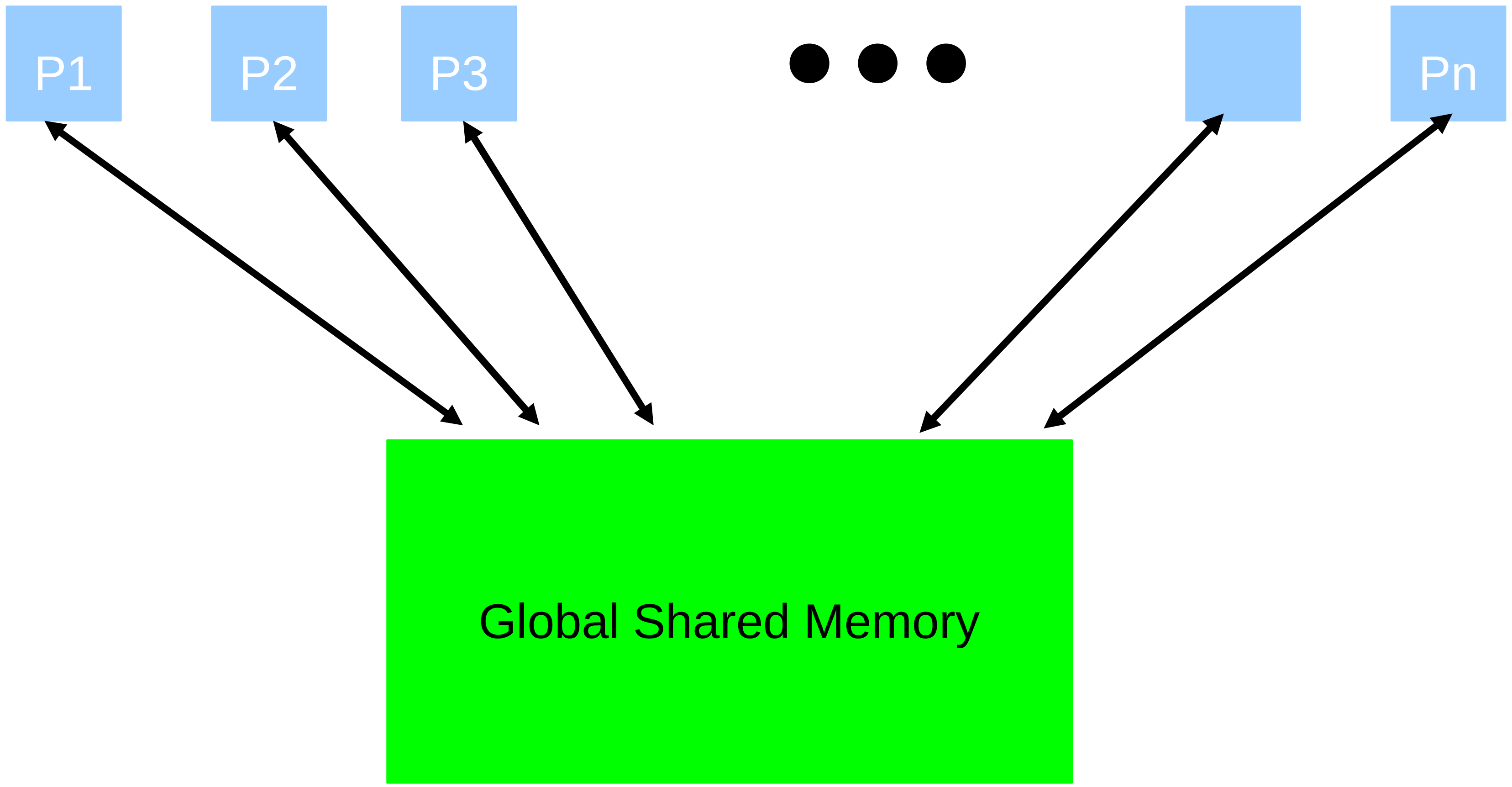
The Academic Interest

- Architectures
    - What are standard architectural designs?
    - What new issues are raised due to multiple cores?
    - Downstream concerns
        - Does a programmer have to worry about this?
        - How to support the systems software as architecture changes?

The Course Coverage

- Focus on algorithms and complexity
- Models for parallel algorithms
- Algorithm design methodologies with application
    - Semi-numerical
    - Lists
    - Trees and graphs
- Some parallel programming practice
- Complexity, characterization, and connection to sequential complexity classes.

The PRAM Model



- An extension of the von Neumann model.

The PRAM Model

- A set of n identical processors
- A common access shared memory
- Synchronous time steps
- Access to the shared memory costs the same as a unit of computation.
- Different models to provide semantics for concurrent access to the shared memory
  - EREW, CREW, CRCW(Common, Aribitrary, Priority, ...)

The Semantics

- In all cases, it is the programmer to ensure that his program meets the required semantics.
- EREW : Exclusive Read, Exclusive Write
  - No scope for memory contention.
  - Usually the weakest model, and hence algorithm design is tough.
- CREW : Concurrent Read, Exclusive Write
  - Allow processors to read simultaneously from the same memory location at the same instant.
  - Can be made practically feasible with additional hardware

The Semantics

- CRCW : Concurrent Read, Concurrent Write
  - Allow processors to read/write simultaneously from/to the same memory location at the same instant.
  - Requires further specification of semantics for concurrent write. Popular variants include
    - COMMON : Concurrent write is allowed so long as the all the values being attempted are equal. Example: Consider finding the Boolean OR of n bits.
    - ARBITRARY : In case of a concurrent write, it is guaranteed that some processor succeeds and its write takes effect.
    - PRIORITY : Assumes that processors have numbers that can be used to decide which write succeeds.

PRAM Model – Advantages and Drawbacks

Advantages

- A simple model for algorithm design
- Hides architectural details for the designer.
- A good starting point

Disadvantages

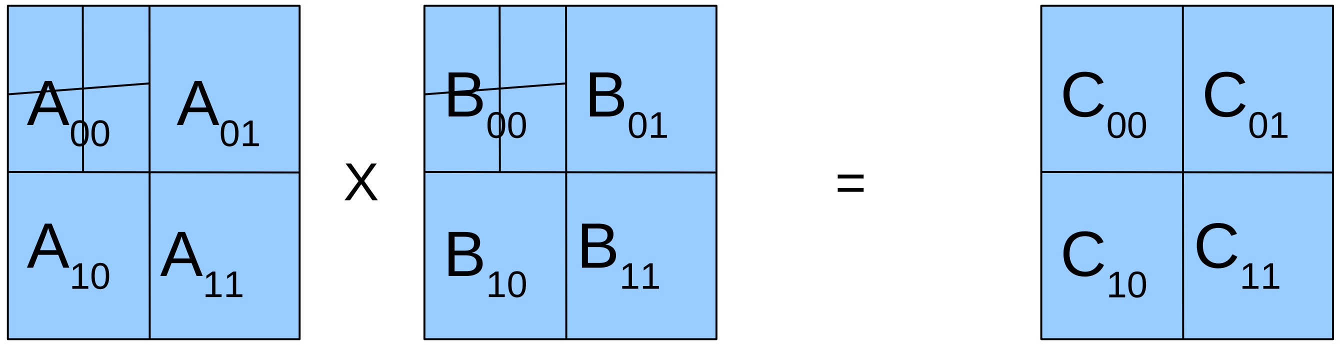
- Ignores architectural features such as:
  - memory bandwidth,
  - communication cost and latency,
  - scheduling, ...
- Hardware may be difficult to realize

Example 1 – Matrix Multiplication

- One of the fundamental parallel processing tasks.
- Applications to several important problems in linear algebra, signal processing and optimization.
- Several techniques that work in parallel also.

Example I – Matrix Multiplication

- Recall that in C = A x B, $C[i,j] = \Sigma A[i,k].B[k,j]$.
- Consider the following recursive approach:
  - –Works well in practice.



$$C_{00} = A_{00} . B_{00} + A_{01} . B_{10}$$

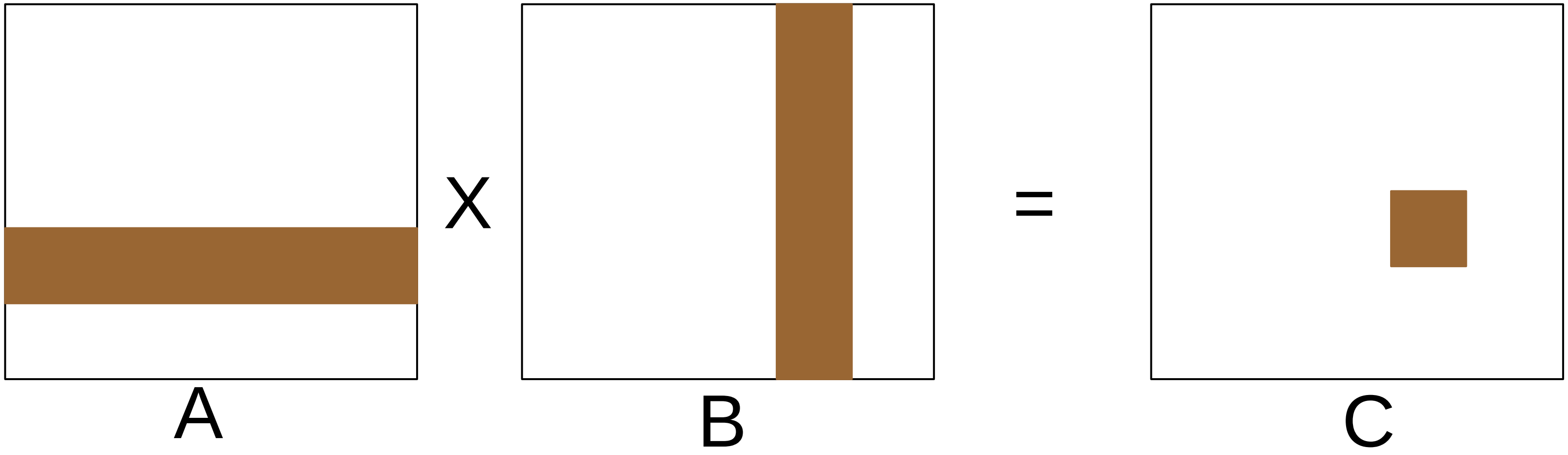
$$C_{01} = A_{00} . B_{01} + A_{01} . B_{11}$$

$$C_{10} = A_{10} . B_{00} + A_{11} . B_{10}$$

$$C_{11} = A_{10} . B_{01} + A_{11} . B_{11}$$

Example I – Matrix Multiplication

- Other approaches include Cannon's algorithm



- Can overlap computation with communication.
- Works well when the number of processors is more.

Example 2 – New Parallel Algorithm

```
Listing 1:
S(1) = A(1)
for i = 2 to n do
   S(i) = S(i-1) o A(i)
```

- Prefix Computations: Given an array A of n elements and an associative operation o, compute A(1) o A(2) o ... A(i) for each i.
- A very simple sequential algorithm exists for this problem.

Parallel Prefix Computation

- The sequential algorithm in Listing 1 is not efficient in parallel.
- Need a new algorithm approach.
    - Balanced Binary Tree

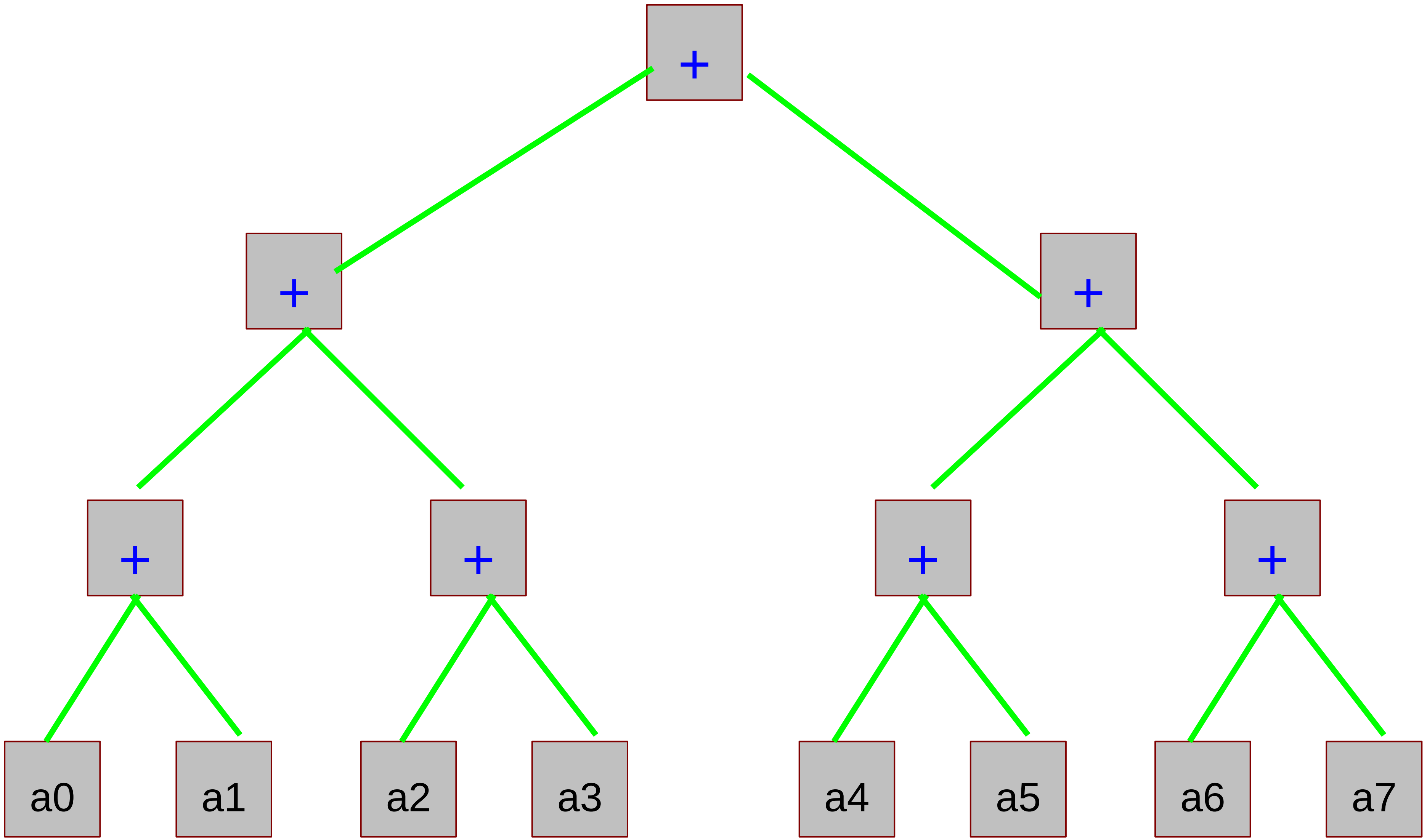
Balanced Binary Tree

- An algorithm design approach for parallel algorithms
- Many problems can be solved with this design technique.
- Easily amenable to parallelization and analysis.

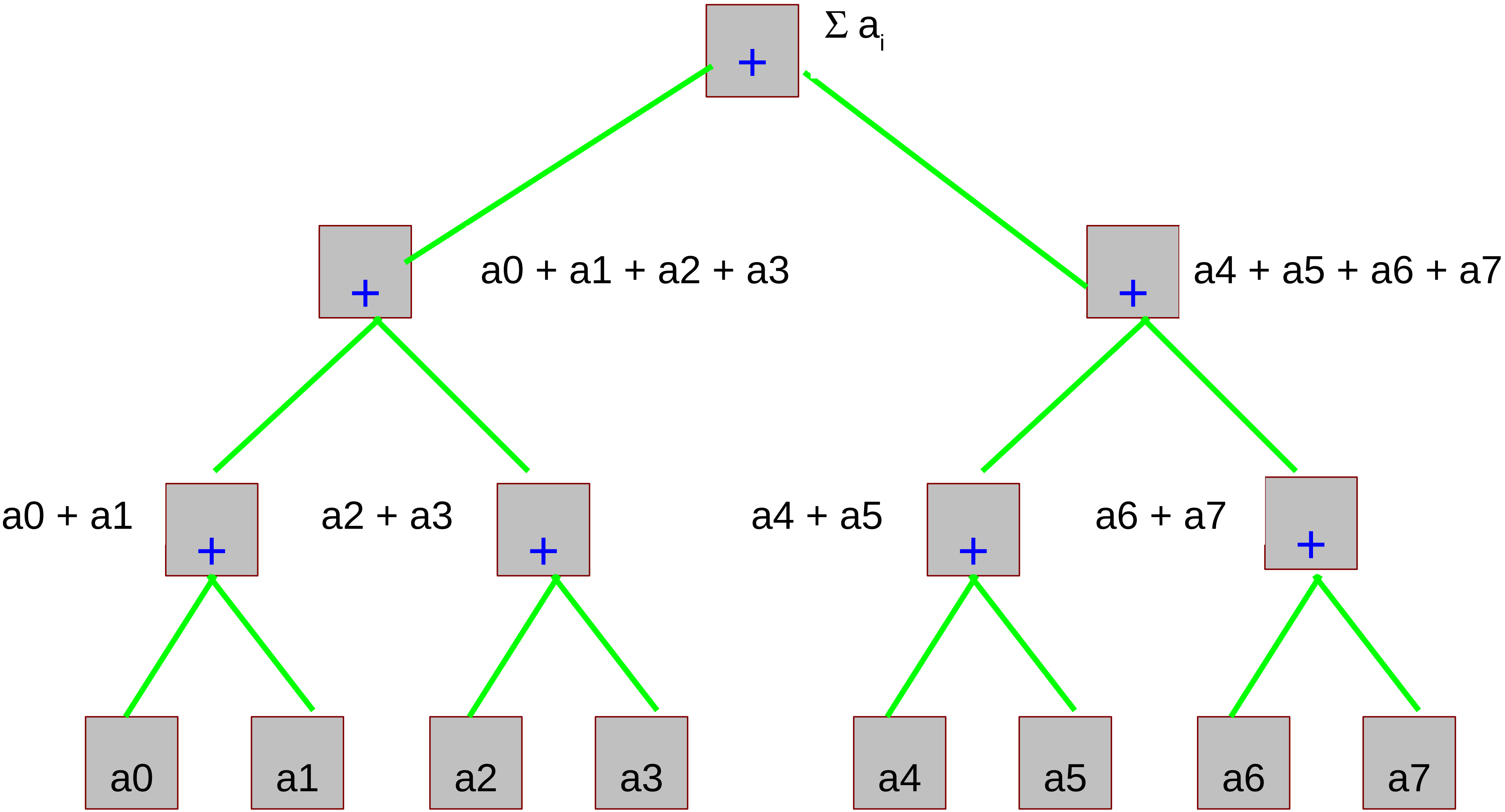
Balanced Binary Tree

- A complete binary tree with processors at each internal node.
- Input is at the leaf nodes
- Define operations to be executed at the internal nodes.
  - Inputs for this operation at a node are the values at the children of this node.
- Computation as a tree traversal from leaf to root.

Balanced Binary Tree – Prefix Sums



Balanced Binary Tree – Sum



$\Sigma\ a_i$

a0 + a1 + a2 + a3

a4 + a5 + a6 + a7

a0 + a1

a2 + a3

a4 + a5

a6 + a7

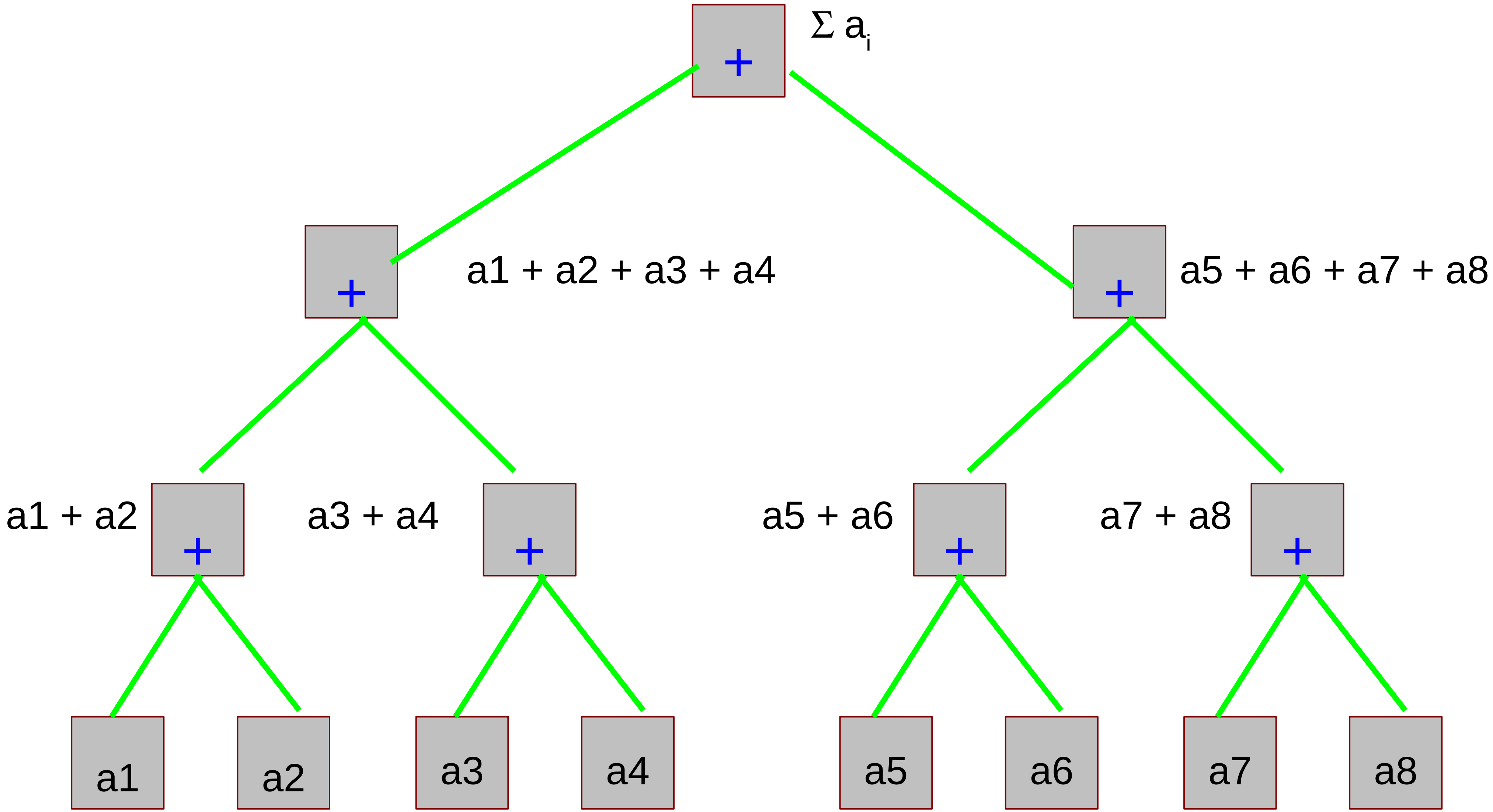
Balanced Binary Tree – Sum

- The above approach called as an ``upward traversal''
  - Data flow from the children to the root.
  - Helpful in other situations also such as computing the max, expression evaluation.
- Analogously, can define a downward traversal
  - Data flows from root to leaf
  - Helps in settings such as element broadcast

Balanced Binary Tree

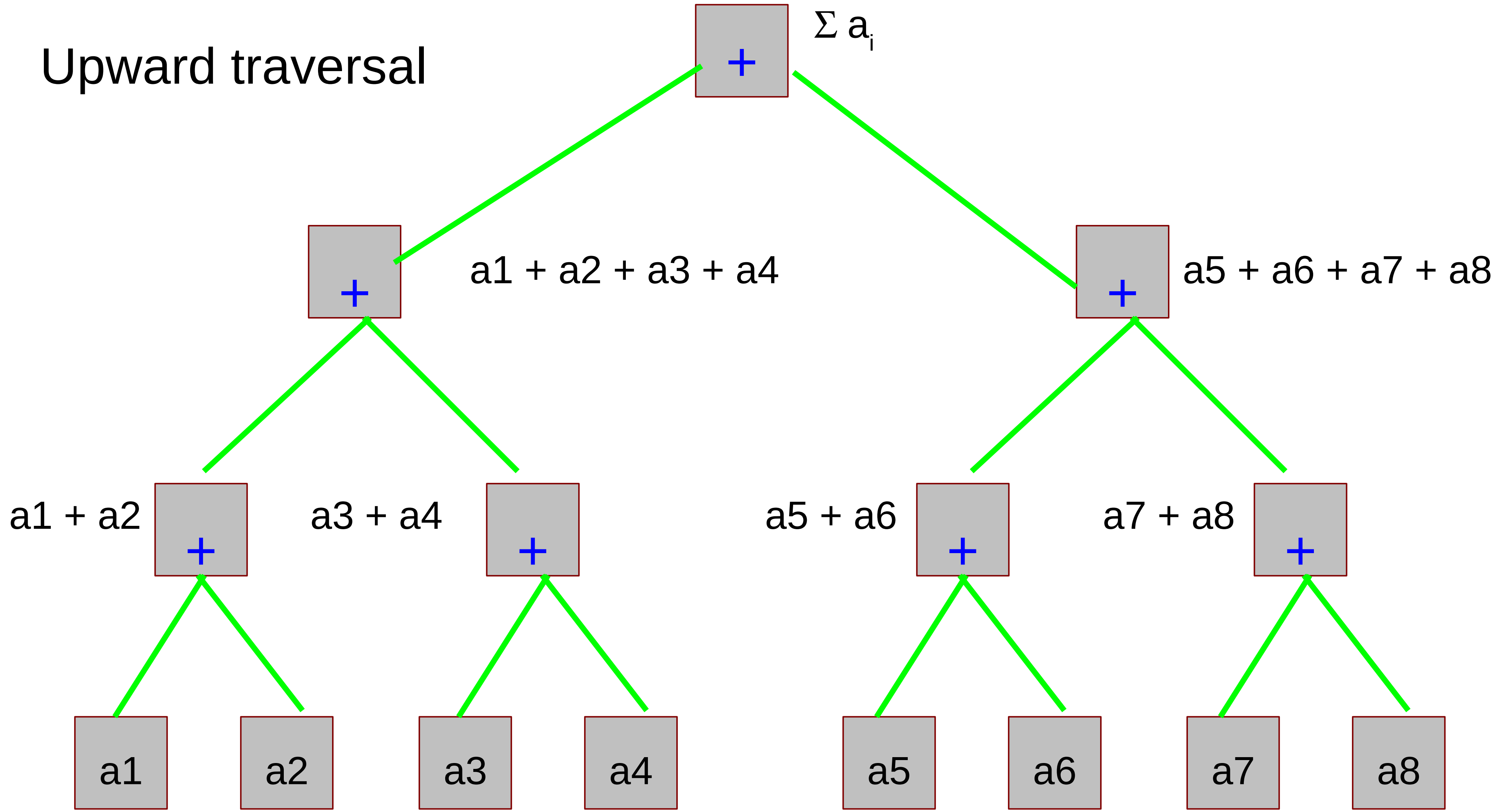
- Can use a combination of both upward and downward traversal.
- Prefix computation requires that.
- Illustration in the next slide.

Balanced Binary Tree – Sum



Balanced Binary Tree – Prefix Sum

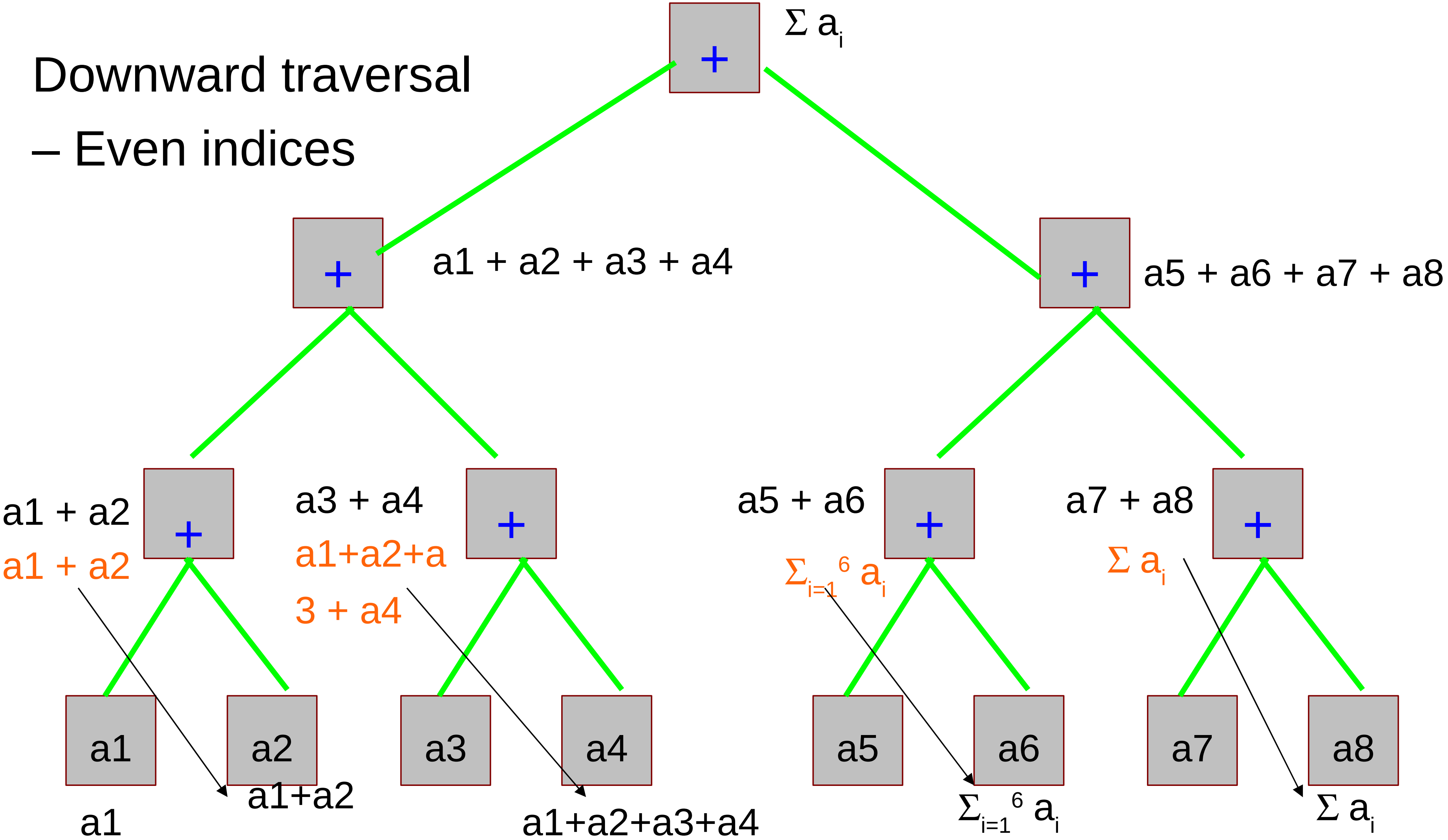
Upward traversal



Balanced Binary Tree – Prefix Sum

Downward traversal

– Even indices



$\Sigma\ a_i$

a1 + a2 + a3 + a4

a5 + a6 + a7 + a8

a1 + a2

a1 + a2

a3 + a4

a1+a2+a3 + a4

a5 + a6

$\Sigma_{i=1}^{6}\ a_i$

a7 + a8

$\Sigma\ a_i$

a1 a2 a3 a4 a5 a6 a7 a8

a1

a1+a2

a1+a2+a3+a4

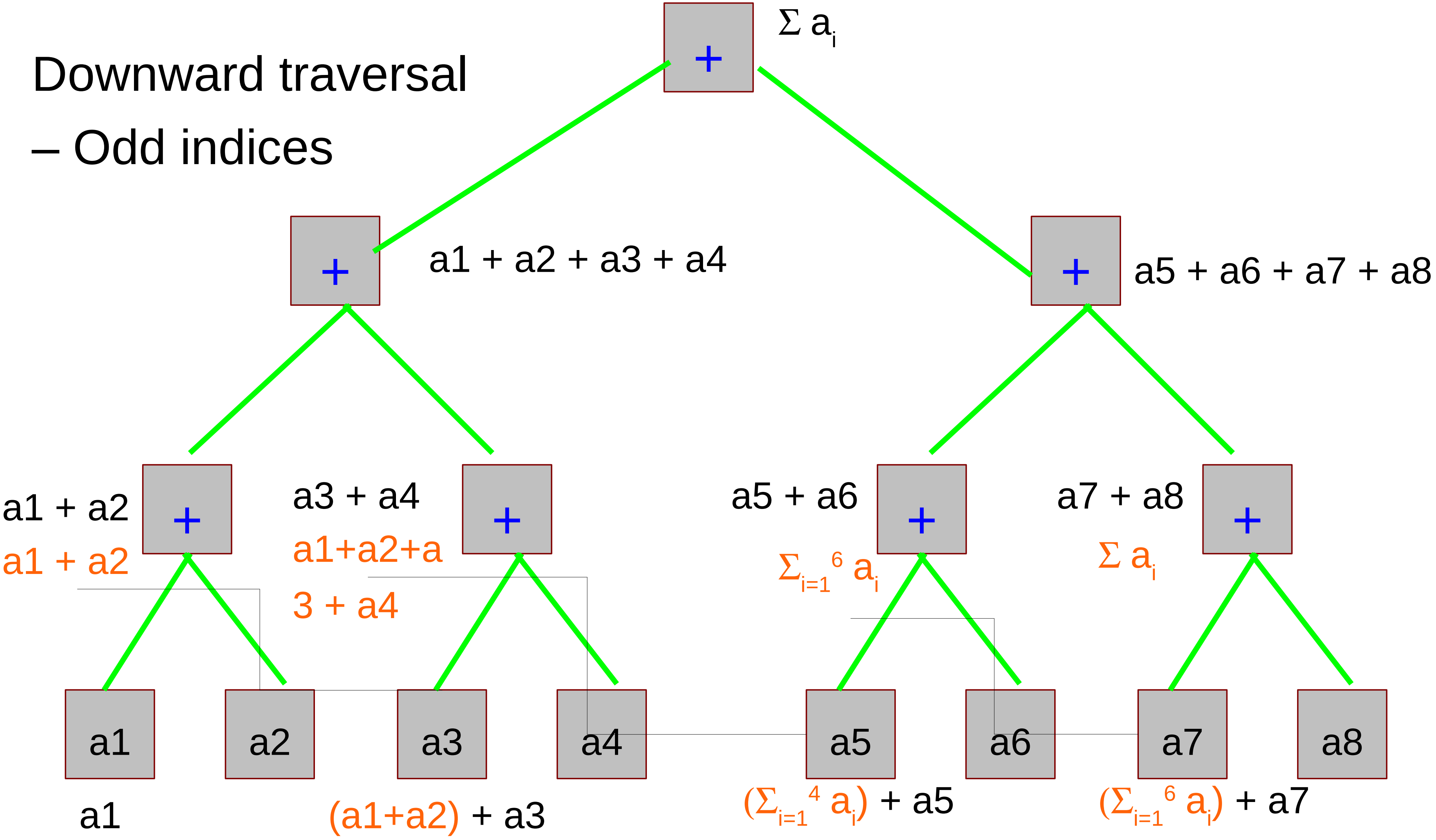
$\Sigma_{i=1}^{6}\ a_i$

$\Sigma\ a_i$

Balanced Binary Tree – Prefix Sum

Downward traversal

– Odd indices



$\Sigma\, a_i$

a1 + a2 + a3 + a4

a5 + a6 + a7 + a8

a1 + a2

a1 + a2

a3 + a4

a1+a2+a3 + a4

a5 + a6

$\Sigma_{i=1}^{6}\, a_i$

a7 + a8

$\Sigma\, a_i$

a1 a2 a3 a4 a5 a6 a7 a8

a1

(a1+a2) + a3

$(\Sigma_{i=1}^{4}\, a_i)$ + a5

$(\Sigma_{i=1}^{6}\, a_i)$ + a7

Balanced Binary Tree – Prefix Sums

- Two traversals of a complete binary tree.
- The tree is only a visual aid.
  - Map processors to locations in the tree
  - Perform equivalent computations.
  - Algorithm designed in the PRAM model.
  - Works in logarithmic time, and optimal number of operations.

```
//upward traversal
1. for i = 1 to n/2 do in parallel
     $b_i = a_{2i-2} \circ a_{2i}$
2. Recursively compute the prefix sums of $B = (b_1, b_2, ..., b_{n/2})$ and store them in $C = (c_1, c_2, ..., c_{n/2})$
//downward traversal
3. for i = 1 to n do in parallel
       i is even : $s_i = c_i$
       i= 1 : $s_1 = c_1$
       i is odd : $s_i = c_{(i-1)/2} \circ a_i$
```