Theoretical Basics

Week 1 – Part 2

Amittai Aviram – 2020-09-08 – Boston University

Speedup

Let p be the number of processing units, $T(p)$ be the amount of time taken by p processors to accomplish a constant unit of work, and $S(p)$ be the *speedup* provided by parallel processing on p processors as compared with use of a single processor. Then the speedup is given by

$$S = \frac{T(1)}{T(p)}$$

The maximal speedup is usually *linear*.

Efficiency and Cost

The *efficiency* of a parallel computing system is how much speedup you get per processing unit—given as a percent.

$$E = \frac{S}{p} = \frac{T(1)}{T(p) \times p}$$

The ideal is 100%, which is linear speedup.

The *cost* of the system is measured in *time*—the total time taken by all processors.

$$C = T(p) \times p$$

Computation and Communication

- The cost of communicating data can significantly affect the efficiency of the system.
- The higher the raio of communication to computation time, the less efficient the system.

Scalability

The Maintenance of Efficiency as p Increases

- Strong Scalability
  The algorithm or system remains efficient as we add processors while keeping the data input size constant.

- Weak Scalability
  The algorithm or system remains efficient as both the number of processors *and* the size of the input grow.

- An algorithm or system is *not scalable* if its performance degrades significantly as the size of the system (and input) grows.

Example

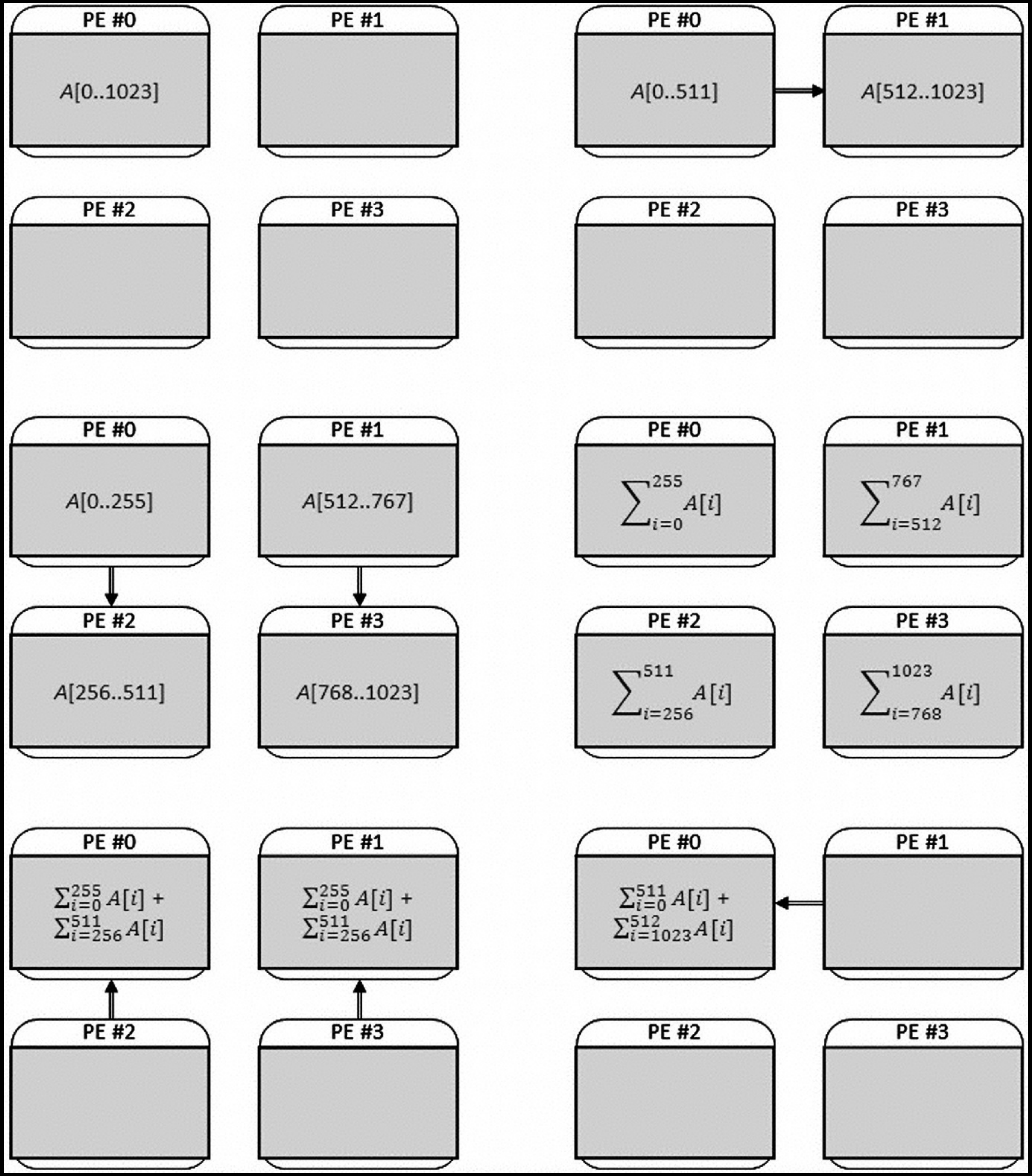
Summing 1024 Numbers

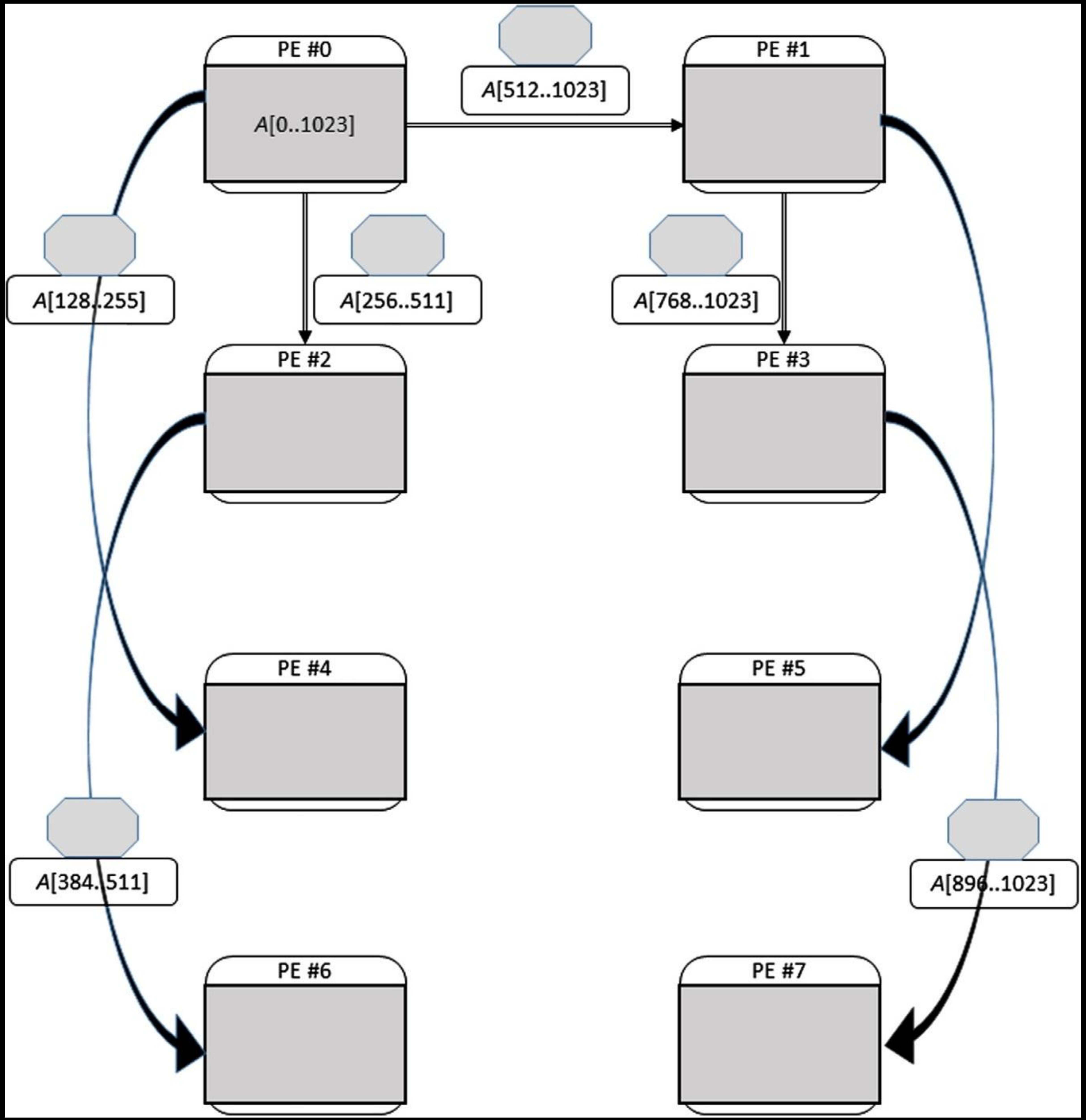
- Given: a distributed system of interconnected processors.
- We hold the input size constant so as to measure *strong scalability*.
- Assumptions:
  - Each processor takes 1 time unit to add two numbers.
  - Each communication between processors takes 3 time units.
  - We have $p = 2^k$ processors (for some small integer *k*).

Algorithm

Distributed Divide-and-Conquer

- The first node divides the input list in half and sends one half to its neighbor.
- This division is repeated recursively until each of the prorcessors receives an even share of the input.
  - E.g., for $p = 8 = 2^3$ processors, node 0 divides its input in half three times, node 1 divides its input (from node 0) twice, etc.
- After dividing up the input, the nodes must gather the results recursively back from their neighbors until the total winds up in node 0.


PE #0
A[0..1023]
PE #1
PE #2
PE #3
PE #0
A[0..511]
PE #1
A[512..1023]
PE #2
PE #3
PE #0
A[0..255]
PE #1
A[512..767]
PE #2
A[256..511]
PE #3
A[768..1023]
PE #0
$\sum_{i=0}^{255} A[i]$
PE #1
$\sum_{i=512}^{767} A[i]$
PE #2
$\sum_{i=256}^{511} A[i]$
PE #3
$\sum_{i=768}^{1023} A[i]$
PE #0
$\sum_{i=0}^{255} A[i] +$
$\sum_{i=256}^{511} A[i]$
PE #1
$\sum_{i=0}^{255} A[i] +$
$\sum_{i=256}^{511} A[i]$
PE #2
PE #3
PE #0
$\sum_{i=0}^{511} A[i] +$
$\sum_{i=1023}^{512} A[i]$
PE #1
PE #2
PE #3


PE #0
A[0..1023]
PE #1
A[512..1023]
A[128..255]
A[256..511]
A[768..1023]
PE #2
PE #3
PE #4
PE #5
A[384..511]
A[896..1023]
PE #6
PE #7

Cost Comparisons

Example with $n = 1024$

- $p = 1$: $T(1,n) = n - 1$
  $T = 1023$

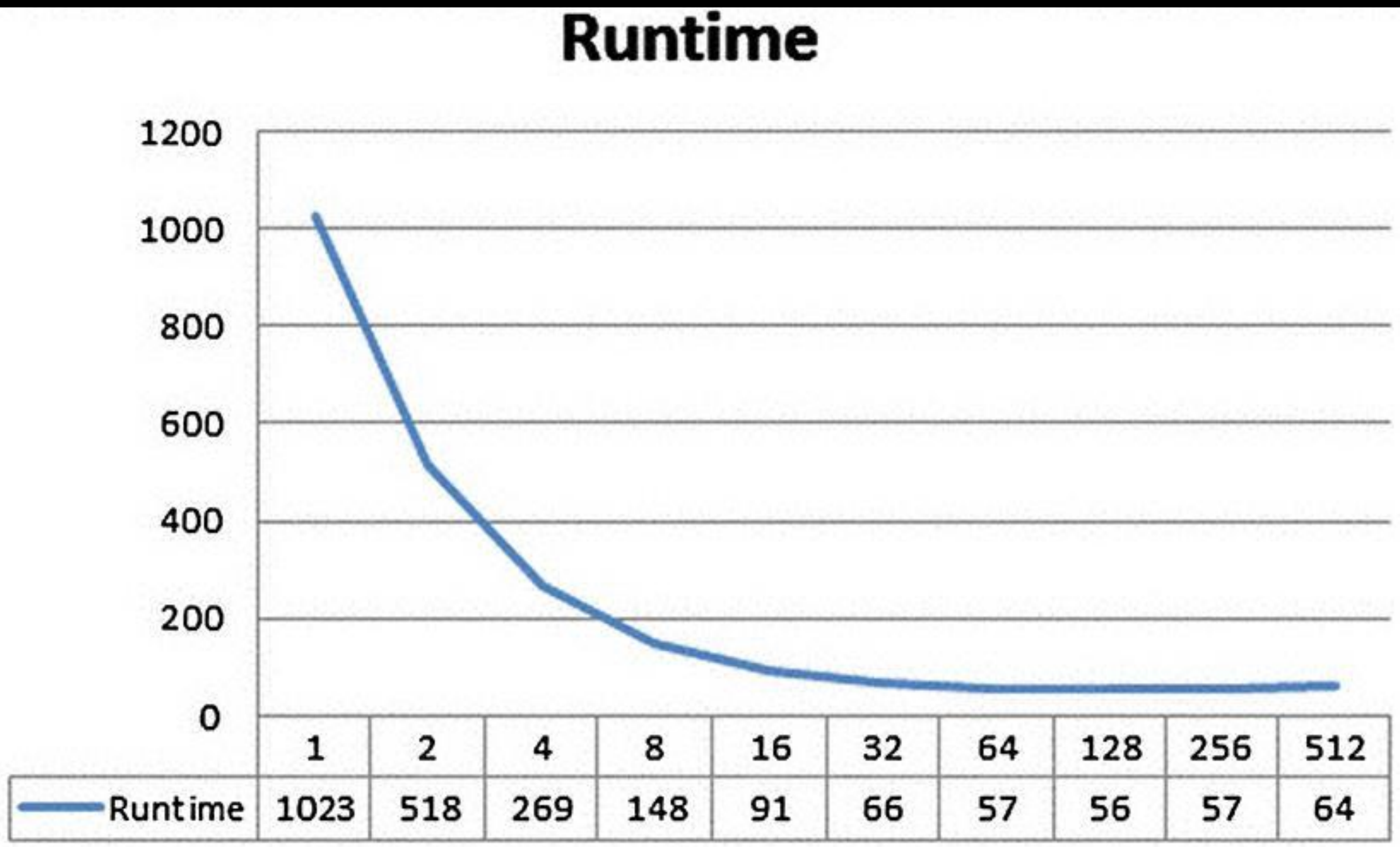
- $p = 2$: $T(2,n) = 3 + n/2 - 1 + 3 + 1$
  $T = 517$ $S = 1023/517 = 1.98$ $E = 1023/(517 \times 2) = 99\,\%$

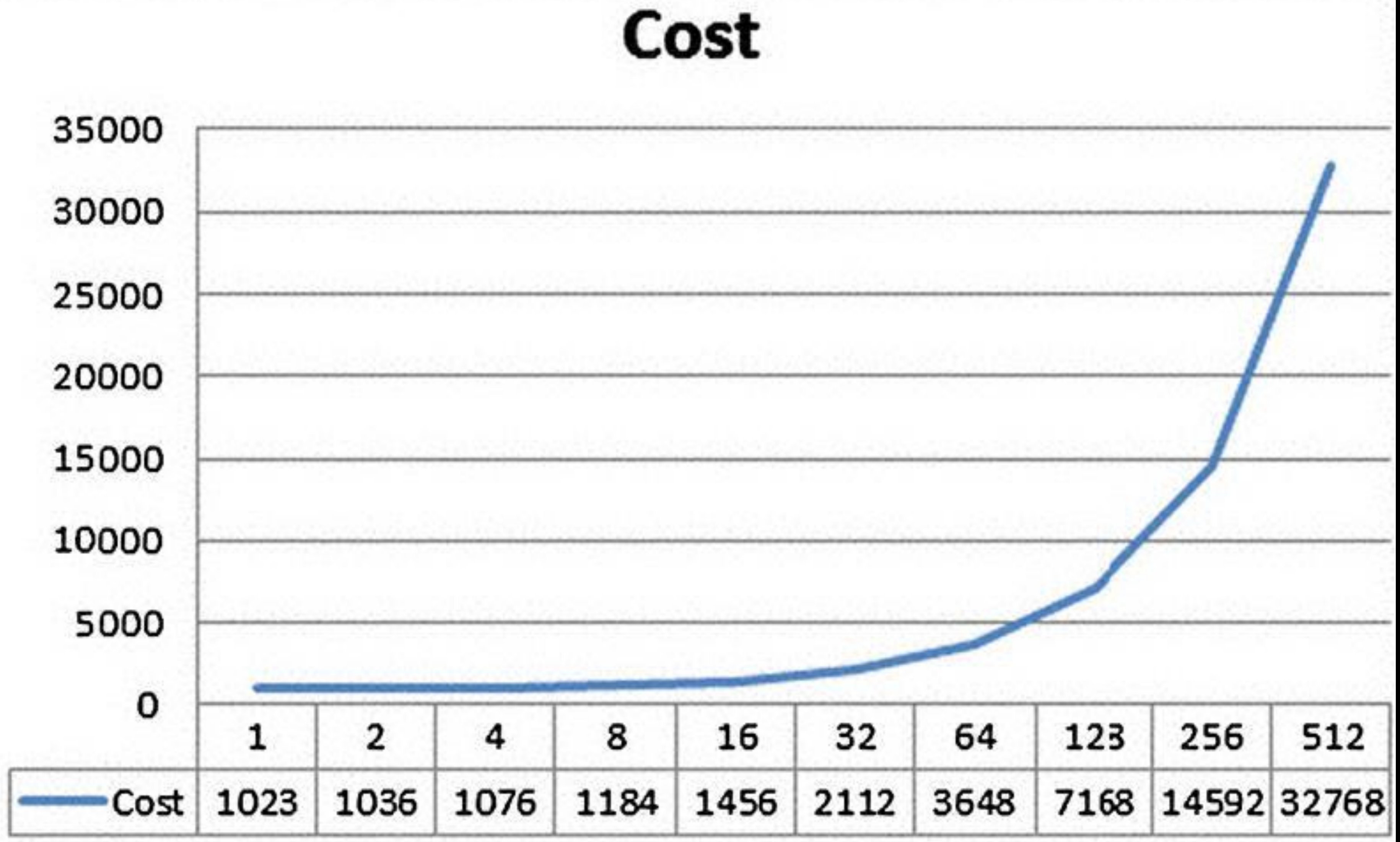
- $p = 4$: $T(4,n) = 3 + 3 + n/4 - 1 + 3 + 1 + 3 + 1$
  $T = 269$ $S = 1023/269 = 3.80$ $E = 1023/(269 \times 4) = 95\,\%$

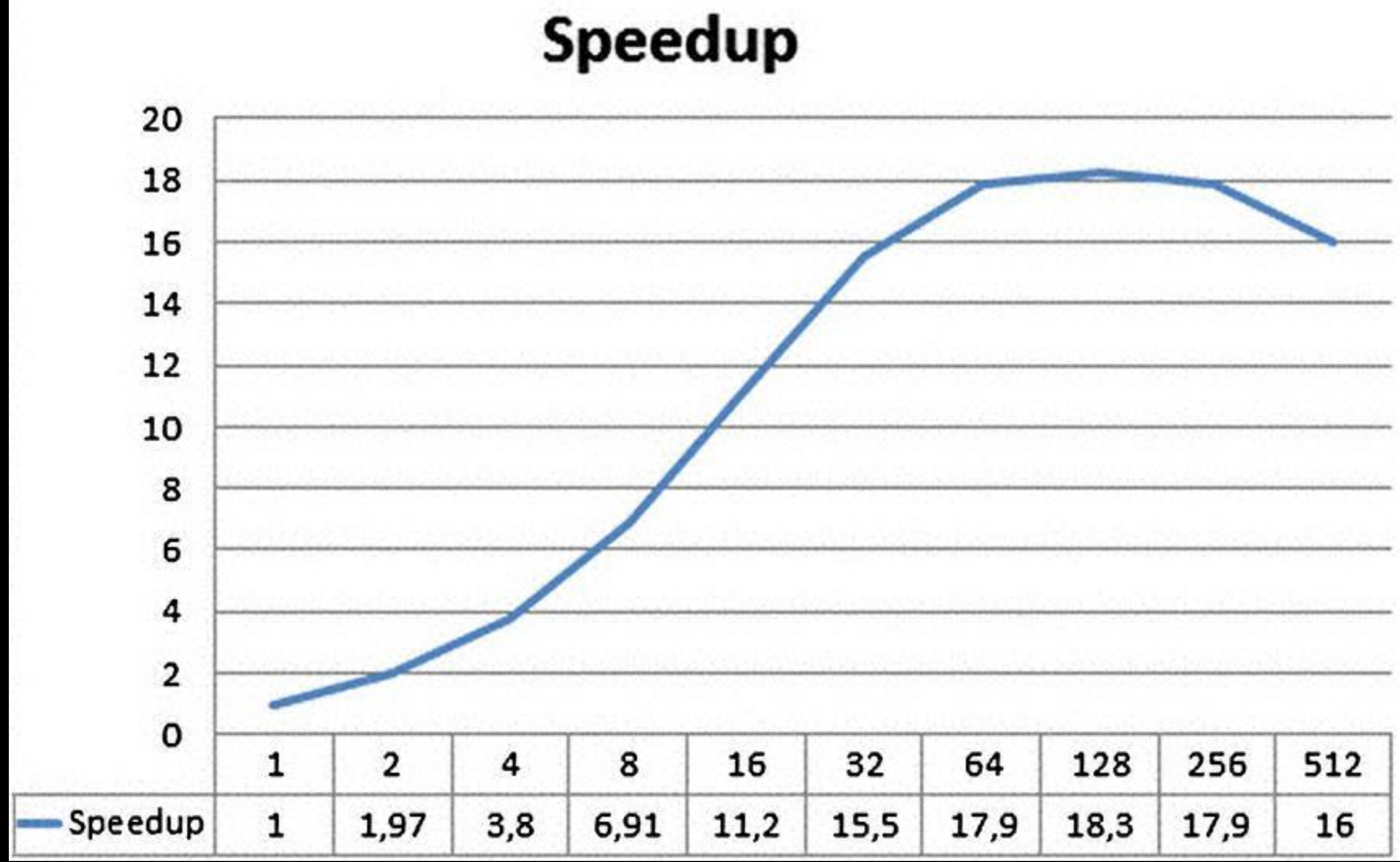
- $p = 8$: $T(8,n) = 3 + 3 + 3 + n/8 - 1 + 3 + 1 + 3 + 1 + 3 + 1$
  $T = 148$ $S = 1023/148 = 6.91$ $E = 1023/(148 \times 8) = 86\,\%$

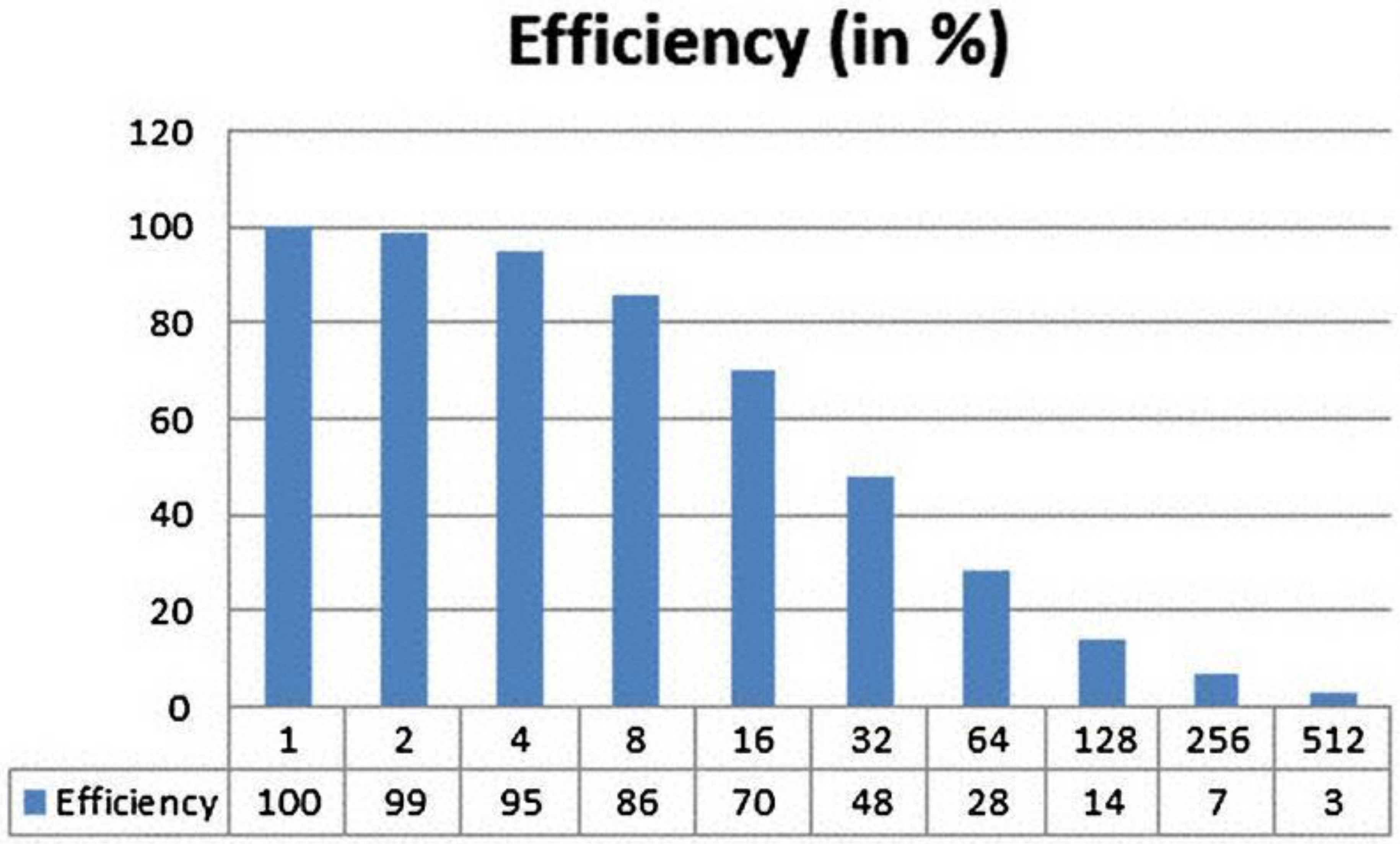
Generalizing

- For $p = 2^q$ processors and $n = 2^k$ input numbers
- Components
  - Data distribution time: $3 \times q$
  - Local sum computation time: $n/p - 1 = 2^{k-q} - 1$
  - Time to collect partial results: $3 \times q$
  - Time to add partial results locally: q
- $T(p, n) = T(2^q, 2^k) = 3q + 2^{k-q} - 1 + 3q + q = 2^{q-k} - 1 + 7q$


Runtime
1200
1000
800
600
400
200
0
1 2 4 8 16 32 64 128 256 512
Runtime 1023 518 269 148 91 66 57 56 57 64

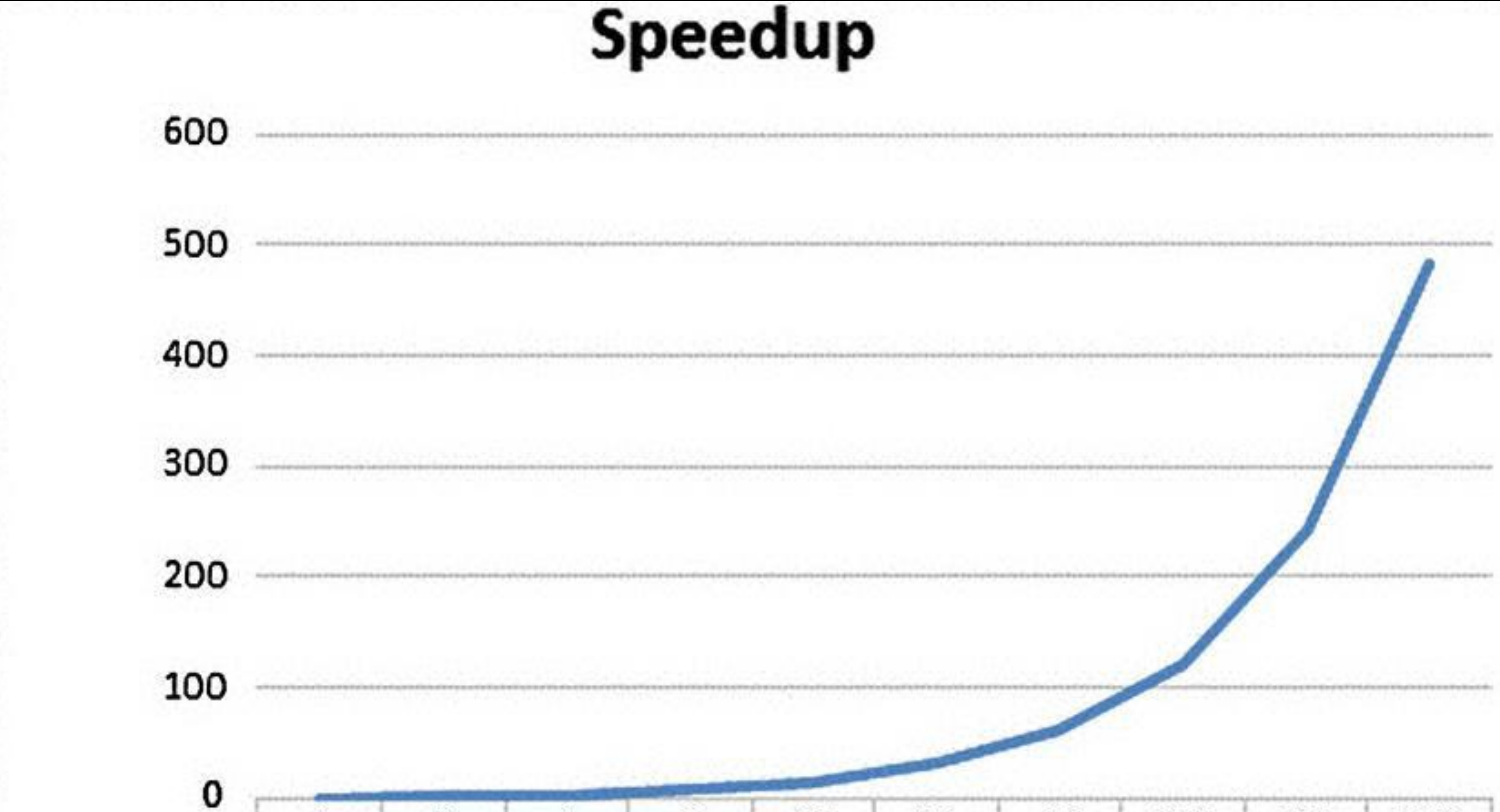

Cost
35000
30000
25000
20000
15000
10000
5000
0
1 2 4 8 16 32 64 123 256 512
Cost 1023 1036 1076 1184 1456 2112 3648 7168 14592 32768


Speedup
20
18
16
14
12
10
8
6
4
2
0
1 2 4 8 16 32 64 128 256 512
Speedup 1 1,97 3,8 6,91 11,2 15,5 17,9 18,3 17,9 16

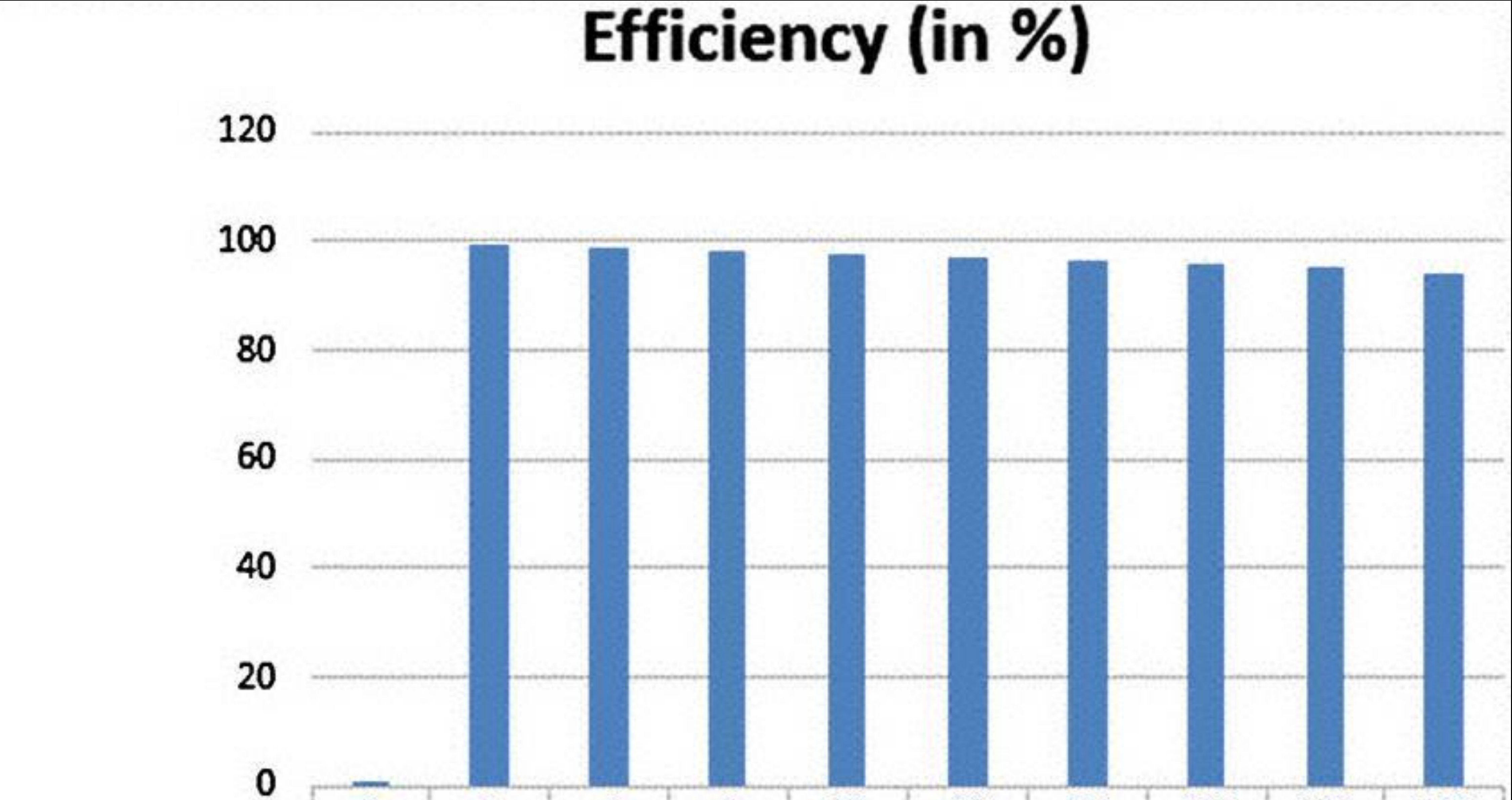

Efficiency (in %)
120
100
80
60
40
20
0
1 2 4 8 16 32 64 128 256 512
Efficiency 100 99 95 86 70 48 28 14 7 3

Weak Scalability Analysis

For $n = 1024 \times p$ and $1 \leq p \leq 512$



p	1	2	4	8	16	32	64	128	256	512
Speedup	1	1,99	3,95	7,85	15,6	31	61,5	122	243	483



p	1	2	4	8	16	32	64	128	256	512
Efficiency (in %)	1	99,5	98,7	98,1	97,4	96,8	96,1	95,5	94,9	94,3

Conclusions

- Our algorithm *is not* strongly scalable.
- BUT our algorithm *is* weakly scalable.

Looking Ahead

Options in Parallel Programming and Systems

- Data distribution
  - Distributed memory
  - Shared memory
  - Partitioned shared memory
- Partitioning the data
  - Managing data dependencies
- Balancing workloads