Parallel Computing
Chapter 7
Performance and Scalability

Jun Zhang

Department of Computer Science

University of Kentucky

7.1 Parallel Systems

- Definition: A parallel system consists of an algorithm and the parallel architecture that the algorithm is implemented.
- Note that an algorithm may have different performance on different parallel architecture.
- For example, an algorithm may perform differently on a linear array of processors and on a hypercube of processors.

7.2 Performance Metrices for Parallel Systems

- **Run Time**: The **parallel run time** is defined as the time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes execution.

- Notation: Serial run time T_S , parallel run time T_P .

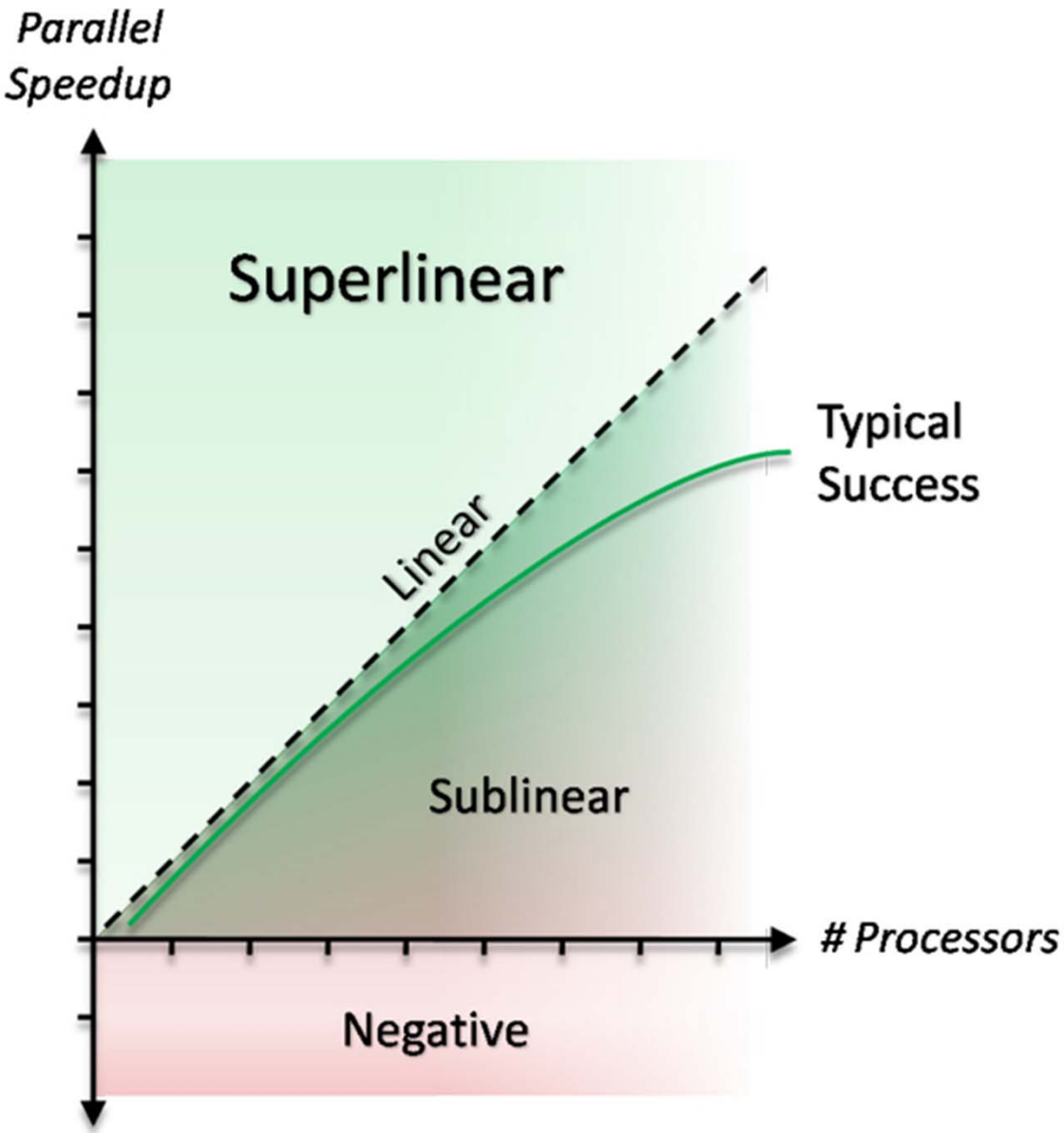
7.3 Speedup

- The **speedup** is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on *p* processors.

$$S = \frac{T_S}{T_P}$$

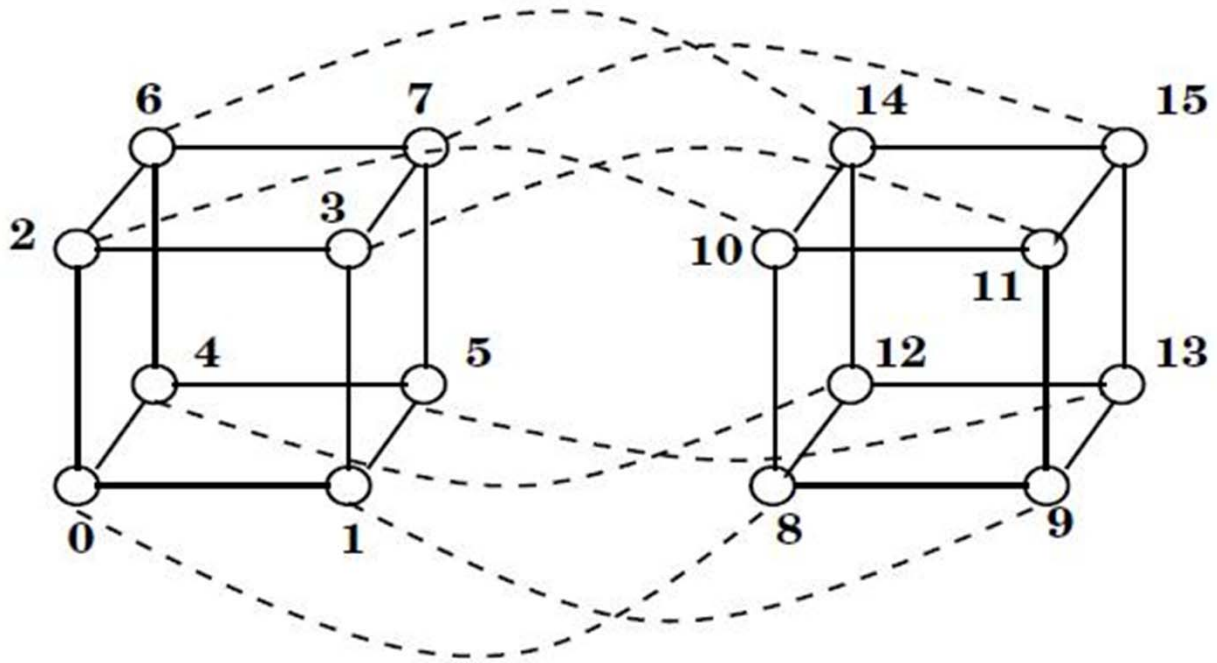
- **Example** Adding *n* numbers on an *n* processor hypercube

$$T_S = \Theta(n), \quad T_P = \Theta(\log n), \qquad S = \Theta(\frac{n}{\log n}),$$


Parallel
Speedup
Superlinear
Linear
Typical
Success
Sublinear
Processors
Negative

Using Reduction Algorithm

Example 4.1. Computing the sum of 16 numbers on a 16-processor hypercube



7.4 Efficiency

- The efficiency is defined as the ratio of speedup to the number of processors. Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E = \frac{S}{p} = \frac{T_S}{pT_p}$$

- **Example** Efficiency of adding *n* numbers on an *n*-processor hypercube

$$E = \Theta\left(\frac{n}{\log n}.\frac{1}{n}\right) = \Theta\left(\frac{1}{\log n}\right)$$

7.5 Cost

- The cost of solving a problem on a parallel system is defined as the product of run time and the number of processors.
- A **cost-optimal** parallel system solves a problem with a cost proportional to the execution time of the fastest known sequential algorithm on a single processor.

- **Example** Adding *n* numbers on an *n*-processor hypercube.

  Cost is $\Theta(n \log n)$ for the parallel system and $\Theta(n)$ for sequential algorithm. The system is not cost-optimal.

7.6 Granularity and Performance

- Use less than the maximum number of processors.
- Increase performance by increasing granularity of computation in each processor.
- **Example** Adding *n* numbers cost-optimally on a hypercube.

  Use *p* processors, each holds *n/p* numbers. First add the *n/p* numbers locally. Then the situation becomes adding *p* numbers on a *p* processor hypercube. Parallel run time and cost:

$$\Theta(n/p + \log p), \qquad \Theta(n + p \log p)$$

7.7 Scalability

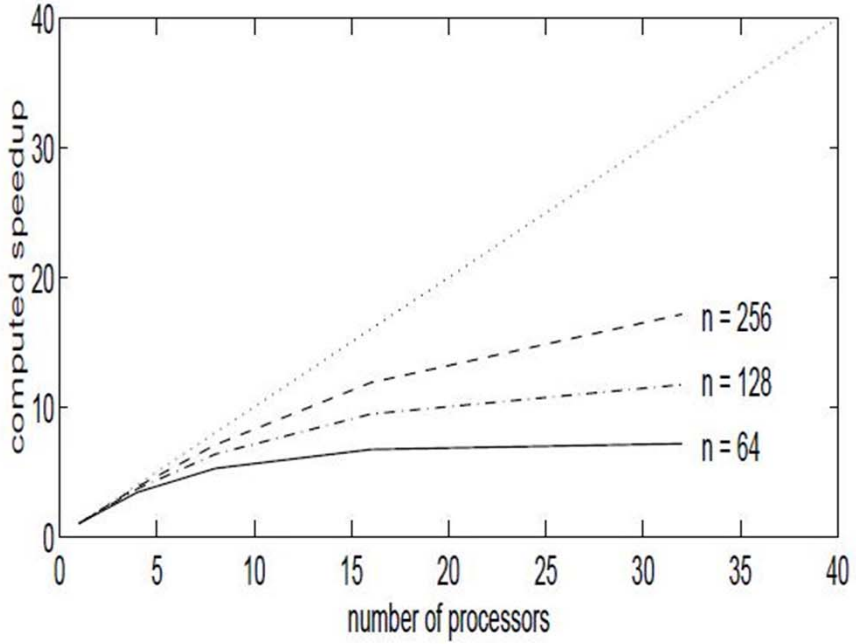
- Scalability is a measure of a parallel system's capacity to increase speedup in proportion to the number of processors.
- **Example** Adding *n* numbers cost-optimally

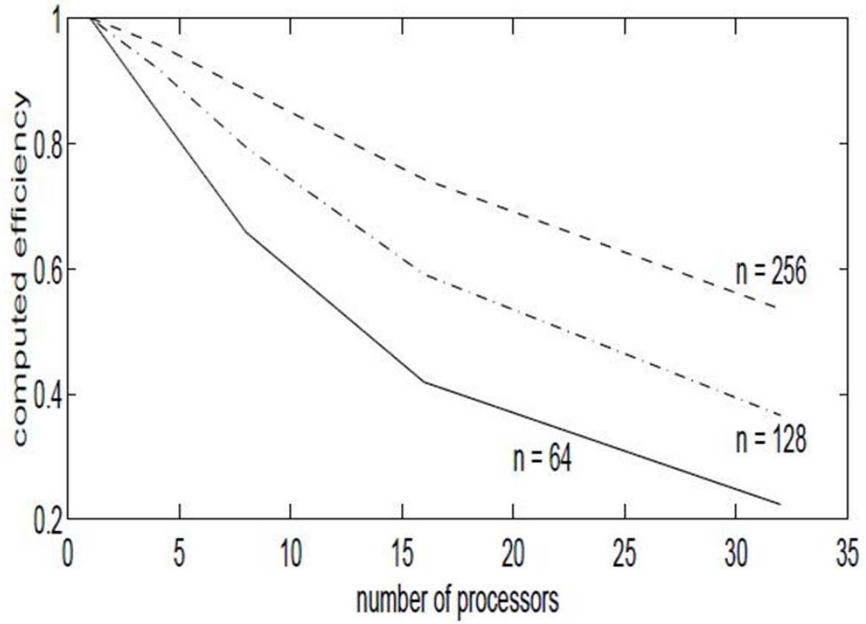
$$T_p = \frac{n}{p} + 2 \log p$$

$$S = \frac{np}{n + 2p \log p}$$

$$E = \frac{S}{p} = \frac{n}{n + 2p \log p}$$

Well-known **Amdahl's law** dictates the achievable speedup and efficiency.


computed speedup
number of processors
n = 256
n = 128
n = 64
0
10
20
30
40
0
5
10
15
20
25
30
35
40


computed efficiency
number of processors
n = 256
n = 128
n = 64
0.2
0.4
0.6
0.8
1
0
5
10
15
20
25
30
35

7.8 Amdahl's Law (1967)

- The speedup of a program using multiple processors in parallel computing is limited by the time needed for the serial fraction of the problem.
- If a problem of size *W* has a serial component *Ws*, the speedup of the program is

$$T_p = \frac{W - W_s}{p} + W_s$$

$$S = \frac{W}{(W - W_s)/p + W_s}$$

$$S \leq \frac{W}{W_s} \quad \text{as} \quad p \to \infty$$

7.9 Amdahl's Law

- If W_s=20%, W-W_s=80%, then

$$S = \frac{1}{0.8 / p + 0.2}$$

$$S \leq \frac{1}{0.2} = 5 \quad \text{as} \quad p \to \infty$$

- So no matter how many processors are used, the speedup cannot be greater than 5
- Amdahl's Law implies that parallel computing is only useful when the number of processors is small, or when the problem is perfectly parallel, i.e., embarrassingly parallel

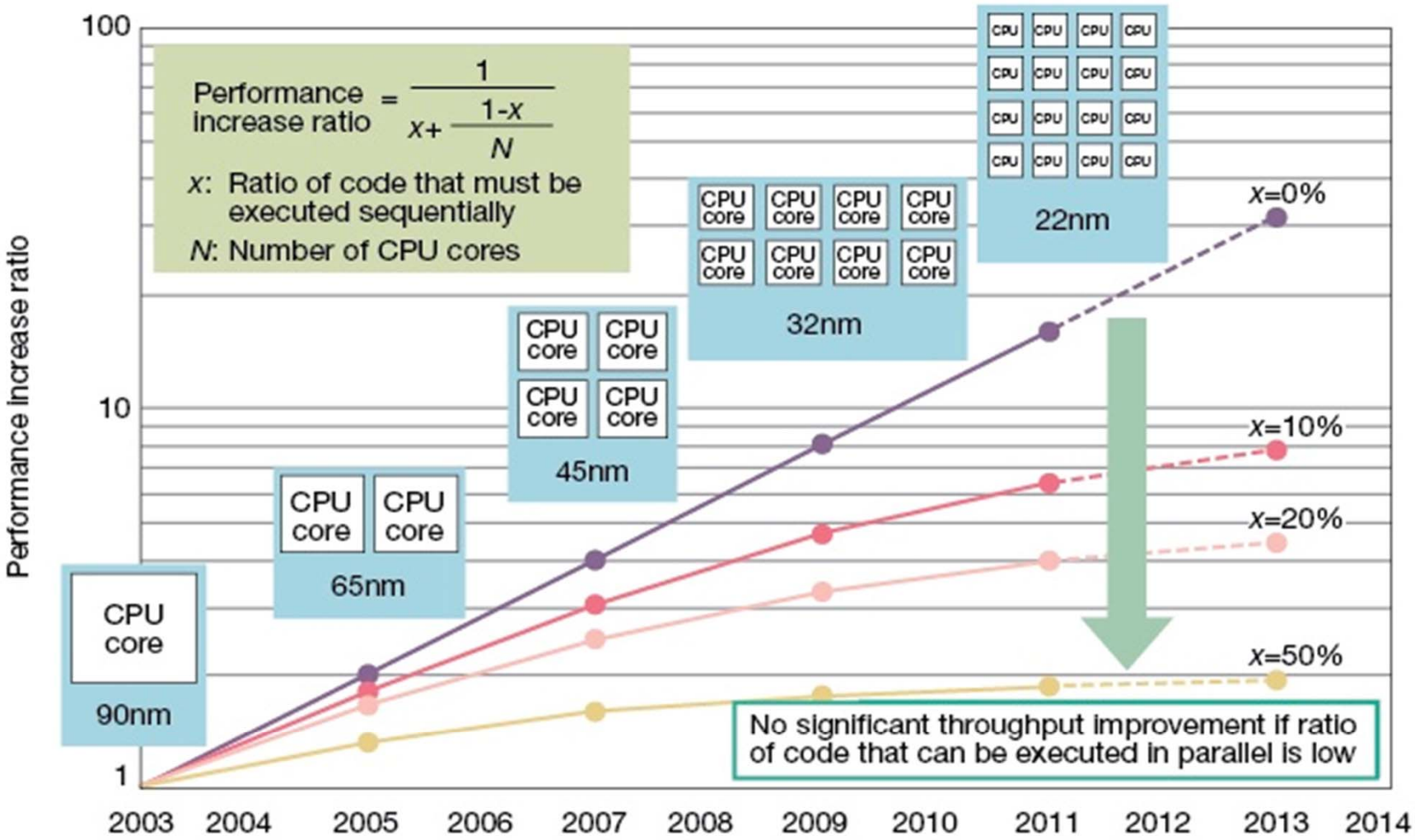


Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor performance. Diagram assumes no overhead in parallel processing. Years shown for design rules based on Intel planned and actual technology. Core count assumed to double for each rule generation.

7.10 Gustafson's Law (1988)

- Also known as the Gustafson-Barsis's Law
- Any sufficiently large problem can be efficiently parallelized with a speedup

$$S = p - \alpha\,(p - 1)$$

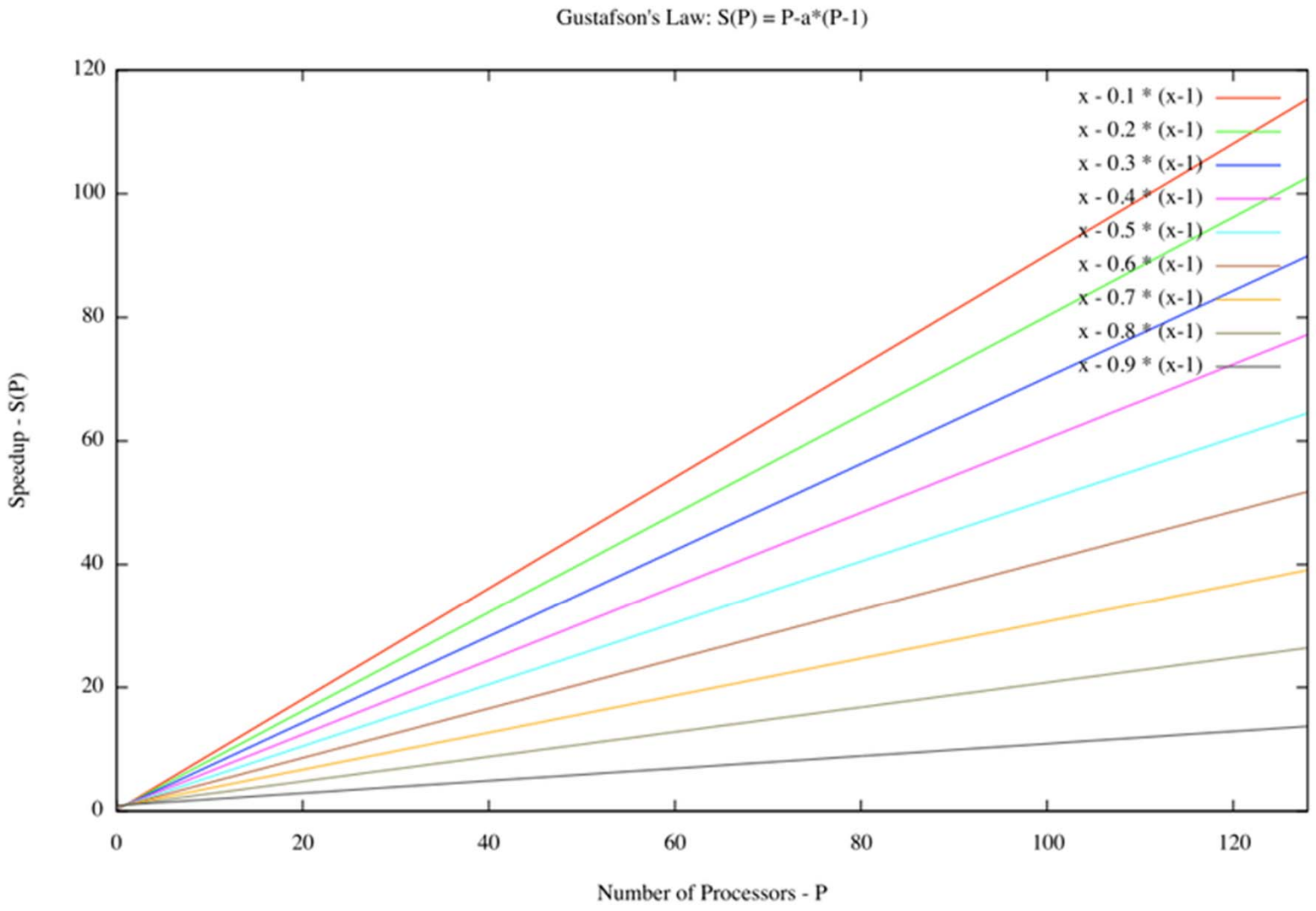
- where *p* is the number of processors, and α is the serial portion of the problem
- Gustafson proposed a fixed time concept which leads to scaled speedup for larger problem sizes.
- Basically, we use larger systems with more processors to solve larger problems

7.10 Gustafson's Law (Cont)

- Execution time of program on a parallel computer is (*a*+*b*)
- *a* is the sequential time and *b* is the parallel time
- Total amount of work to be done in parallel varies linearly with the number of processors. So *b* is fixed as *p* is varied. The total run time is (*a* + *p***b*)
- The speedup is (*a*+*p***b*)/(*a*+*b*)
- Define α = *a*/(*a*+*b*) , the sequential fraction of the execution time, then

$$S = p - \alpha (p - 1)$$

Gustafson's Law



7.11 Scalability (cont.)

- Increase number of processors --> decrease efficiency
- Increase problem size --> increase efficiency

- Can a parallel system keep efficiency by increasing the number of processors and the problem size simultaneously???

  Yes: --> **scalable** parallel system

  No: --> **non-scalable** parallel system

  A scalable parallel system can always be made cost-optimal by adjusting the number of processors and the problem size.

7.12 Scalability (cont.)

- **E.g. 7.7** Adding *n* numbers on an *n*-processor hypercube, the efficiency is

$$E = \Theta(\frac{1}{\log n})$$

- No way to fix *E* (make the system scalable)
- Adding *n* numbers on a *p* processor hypercube optimally, the efficiency is

$$E = \frac{n}{n + 2p \log p}$$

Choosing $n = \Omega(p \log p)$ makes *E* a constant.

7.13 Isoefficiency Metric of Scalability

- **Degree of scalability:** The rate to increase the size of the problem to maintain efficiency as the number of processors changes.
- **Problem size:** Number of basic computation steps in best sequential algorithm to solve the problem on a single processor ($W = T_S$).

- **Overhead function:** Part of the parallel system cost (processor-time product) that is not incurred by the fastest known serial algorithm on a serial computer

$$T_O = pT_p - W$$

7.14 Isoefficiency Function

$$T_P = \frac{W + T_O(W, p)}{p}$$

$$E = \frac{1}{1 + T_O(W, p)/W}$$

- Solve the above equation for *W*

$$W = \frac{E}{1-E} T_O(W, p) = K T_O(W, p)$$

- The isoefficiency function determines the growth rate of *W* required to keep the efficiency fixed as *p* increases.
- Highly scalable systems have small isoefficiency function.

7.15 Sources of Parallel Overhead

- Interprocessor communication: increase data locality to minimize communication.
- Load imbalance: distribution of work (load) is not uniform. Inherent parallelism of the algorithm is not sufficient.
- Extra computation: modify the best sequential algorithm may result in extra computation. Extra computation may be done to avoid communication.

7.6 Minimum Execution Time

$$T_p = \frac{n}{p} + 2 \log\ p$$

A Few Examples

- **Example** Overhead Function for adding *n* numbers on a *p* processor hypercube.

  Parallel run time is: $T_p = \frac{n}{p} + 2\log\ p$

  Parallel system cost is: $pT_p = n + 2p\log\ p$

  Serial cost (and problem size) is: $T_S = n$

  The overhead function is:

$$\begin{aligned} T_O &= pT_p - W \\ &= (n + 2p\log\ p) - n \\ &= 2p\log\ p \end{aligned}$$

  What can we know here?

Example

- **Example** Isoefficiency Function for adding *n* numbers on a *p* processor Hypercube.

  The overhead function is:

  $$T_O = 2p \log p$$

  Hence, isoefficiency function is

  $$W = 2Kp \log p$$

  Increasing the # of processors from p_0 to p_1 , the problem size has to be increased by a factor of

  $$p_1 \log p_1 / (p_0 \log p_0)$$

  If $p = 4, n = 64, E = 0.8.$ If $p = 16$, for $E = 0.8$, we have to have $n = 512$.

Example

- **Example** Isoefficiency Function with a Complex Overhead Function.

  Suppose overhead function is:

$$T_O = p^{3/2} + p^{3/4} W^{3/4}$$

  For the first term:

$$W = K p^{3/2}$$

  For the second term:

$$W = K p^{3/4} W^{3/4}$$

  Solving it yields

$$W = K^4 p^3$$

  The second term dominates, so the overall asymptotic isoefficiency function is $\Theta(p^3)$.

Example

- **Example** Minimum Cost-Optimal Execution time for Adding *n* Numbers

  Minimum execution time is:

$$p = \frac{n}{2}, \qquad T_P^{\min} = 2 \log n$$

  If, for cost-optimal

$$W = n = f(p) = p \log p \qquad (1)$$

  then

$$\log n = \log p - \log \log p \approx \log p$$

  Now solve (1) for *p*, we have

$$p = f^{-1}(n) = n / \log p \approx n / \log n$$

$$T_P^{Cost-Optimal} = 3 \log n - 2 \log \log n$$