Introduction to Parallel Computing

George Karypis

Analytical Modeling of Parallel Algorithms

Sources of Overhead in Parallel Programs

- The total time spent by a parallel system is usually higher than that spent by a serial system to solve the same problem.
  - Overheads!
    - Interprocessor Communication & Interactions
    - Idling
      - Load imbalance, Synchronization, Serial components
    - Excess Computation
      - Sub-optimal serial algorithm
      - More aggregate computations
- Goal is to minimize these overheads!

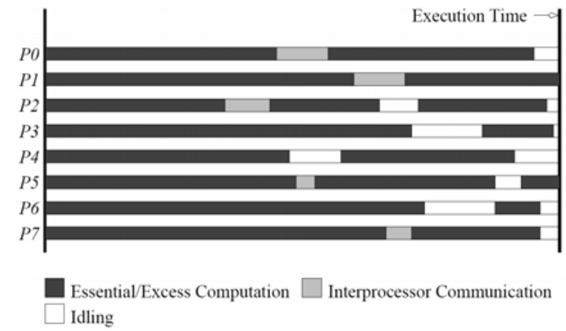

Figure 5.1 The execution profile of a hypothetical parallel program executing on eight processing elements. Profile indicates times spent performing computation (both essential and excess), communication, and idling.

Performance Metrics

- Parallel Execution Time
  - Time spent to solve a problem on *p* processors.
    - T_p
- Total Overhead Function
  - $T_o = pT_p - T_s$
- Speedup
  - $S = T_s/T_p$
  - Can we have superlinear speedup?
    - exploratory computations, hardware features
- Efficiency
  - $E = S/p$
- Cost
  - $p\,T_p$ (processor-time product)
  - Cost-optimal formulation
- Working example: Adding *n* elements on *n* processors.

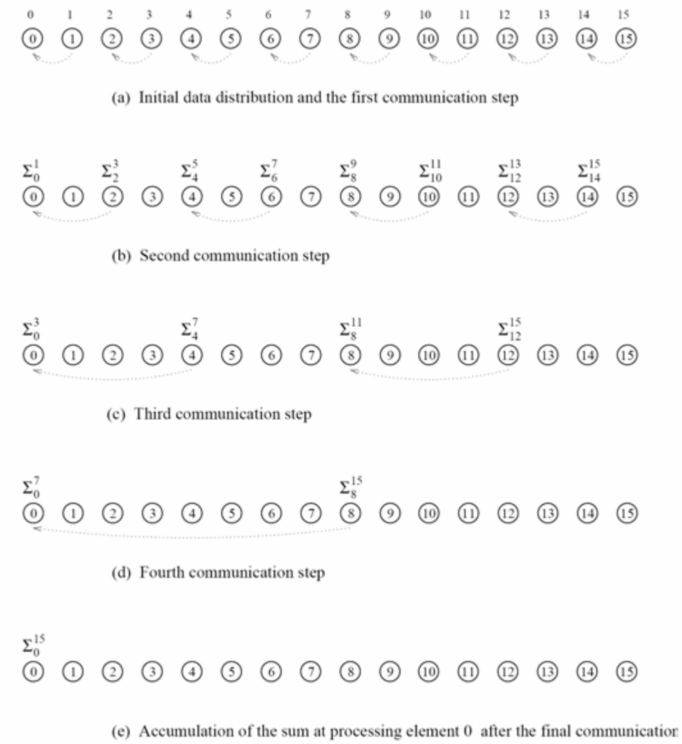

Figure 5.2 Computing the globalsum of 16 partial sums using 16 processing elements. Σ_i^j denotes the sum of numbers with consecutive labels from i to j.

Effect of Granularity on Performance

- Scaling down the number of processors
- Achieving cost optimality
- Naïve emulations vs Intelligent scaling down
  - adding n elements on p processors

Scaling Down by Emulation

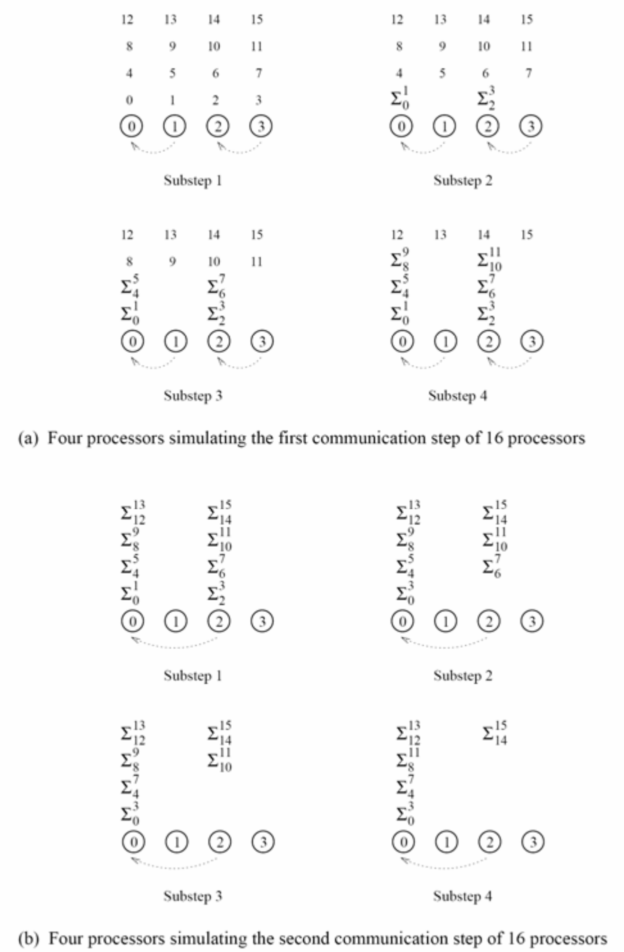

Figure 5.5 Four processing elements simulating 16 processing elements to compute the sum of 16 numbers (first two steps). Σ_i^j denotes the sum of numbers with consecutive labels from i to j.

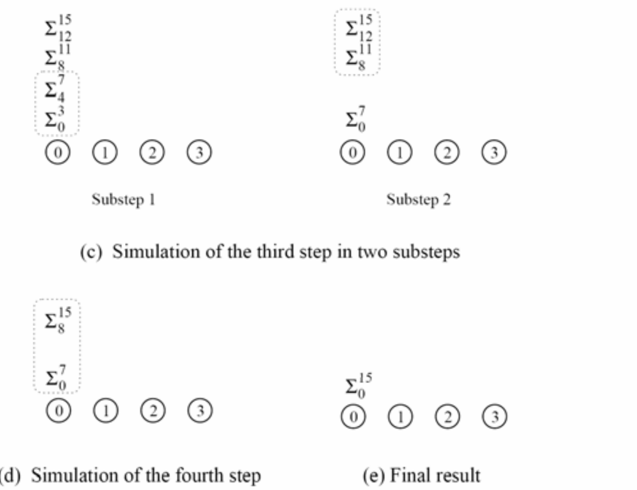

Figure 5.5 (continued) Four processing elements simulating 16 processing elements to compute the sum of 16 numbers (last three steps).

Intelligent Scaling Down

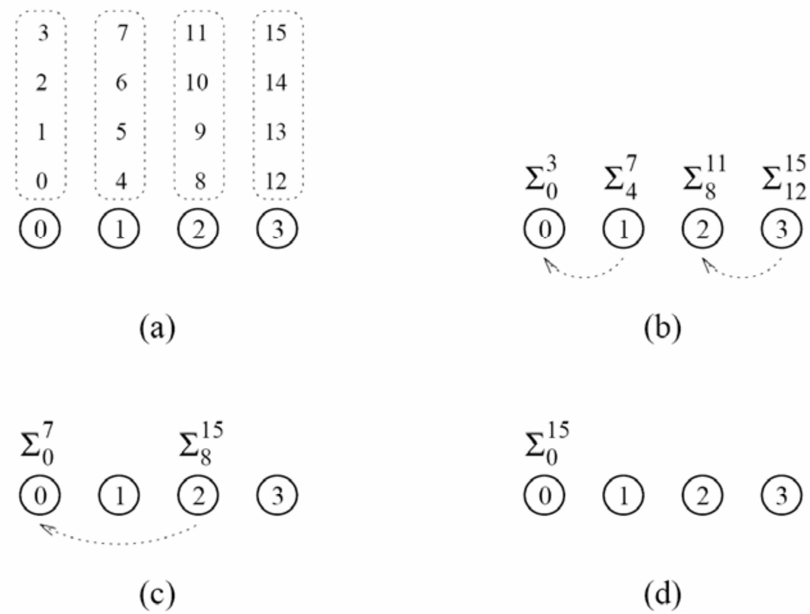

Figure 5.6 A cost-optimal way of computing the sum of 16 numbers using four processing elements.

Scalability of a Parallel System

- The need to predict the performance of a parallel algorithm as *p* increases
- Characteristics of the T_o function
  - Linear on the number of processors
    - serial components
  - Dependence on T_s
    - usually sub-linear
- Efficiency drops as we increase the number of processors and keep the size of the problem fixed
- Efficiency increases as we increase the size of the problem and keep the number of processors fixed

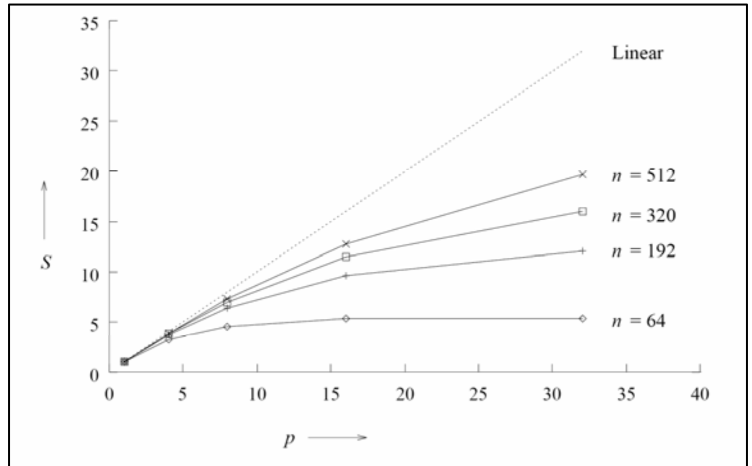

Figure 5.8 Speedup versus the number of processing elements for adding a list of numbers.

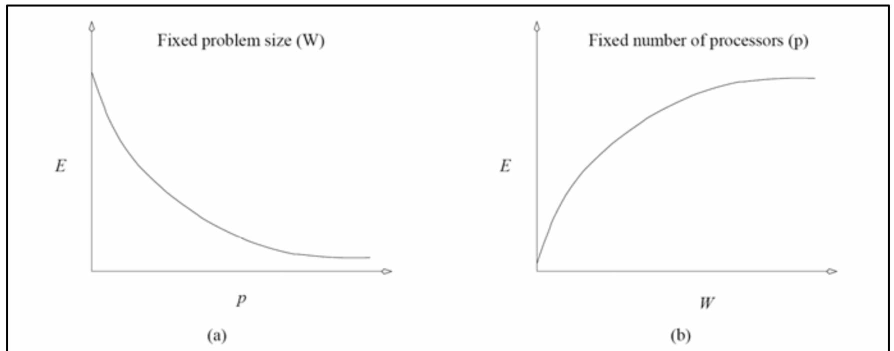

Figure 5.9 Variation of efficiency: (a) as the number of processing elements is increased for a given problem size; and (b) as the problem size is increased for a given number of processing elements. The phenomenon illustrated in graph (b) is not common to all parallel systems.

Scalable Formulations

- A parallel formulation is called *scalable* if we can maintain the efficiency constant when increasing *p* by increasing the size of the problem
- Scalability and cost-optimality are related
- Which system is more scalable?

Table 5.1 Efficiency as a function of n and p for adding n numbers on p processing elements.

n	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
64	1.0	***0.80***	0.57	0.33	0.17
192	1.0	0.92	***0.80***	0.60	0.38
320	1.0	0.95	0.87	0.71	0.50
512	1.0	0.97	0.91	***0.80***	0.62

Measuring Scalability

- What is the *problem size?*
- Isoefficiency function
  - measures the rate by which the problem size has to increase in relation to p
- Algorithms that require the problem size to grow at a lower rate are more scalable
- Isoefficiency and cost-optimality
- What is the best we can do in terms of isoefficiency?