PARALLEL PROCESSING SYSTEMS

Chapter 1: Introduction to Parallelism

References

- Introduction to Parallel Processing- Algorithms and Architectures, Behrooz Parhami, 2002, Kluwer Academic Publishers

Why parallel processing?

- The quest for higher-performance digital computers seems unending
- Moore’s law :The growth of microprocessor speed/performance by a factor of 2 every 18 months (or about 60% per year)
- growth is the result of a combination of two factors:
  - Increase in complexity of VLSI chips
  - Introduction of, and improvements in architectural features such as
    - on-chip cache memories
    - large instruction buffers
    - multiple instruction issue per cycle
    - multi-threading
    - deep pipelines
    - out-of-order instruction execution
    - branch prediction

Why parallel processing?

- Moore’s law
  - was originally formulated in 1965
  - seems to hold regardless of how one measures processor performance
    - counting the number of executed instructions per second (IPS)
    - counting the number of floating-point operations per second (FLOPS)
    - sophisticated benchmark suites that attempt to measure the processor's performance on real applications

Why parallel processing?

- Moore’s law
  - it is expected that Moore's law will continue to hold for the near future
  - But there is a limit that will eventually be reached.
    - dictated by physical laws
      - speed-of-light argument
        - The speed of light is about 30 cm/ns.
        - Signals travel on a wire at a fraction of the speed of light.
        - If the chip diameter is 3 cm
          - any computation that involves signal transmission from one end of the chip to another cannot be executed faster than 10^{10} times per second.
        - we are in fact not very far from limits imposed by the speed of signal propagation and several other physical laws

Why parallel processing?

- once the physical limit has been reached
  - the only path to improved performance is the use of multiple processors.
    - any parallel processor will also be limited by the speed at which the various processors can communicate
      - the limit is less serious here
        - communication does not have to occur for every low-level computation
        - large number of computation steps can be performed between two successive communication steps (for many applications)

Why parallel processing?

- another way to show the need for parallel processing
  - Applications that need TFLOPS or PFLOPS performance
    - space research
    - climate modeling
    - auto crash or engine combustion simulation
    - design of pharmaceuticals
    - design and evaluation of complex ICs
    - scientific visualization, and multimedia

Why parallel processing?

- another way to show the need for parallel processing
  - E.g., The model for heat transfer from southern oceans to the South Pole
    - ocean is divided into
      - 4096 regions E–W
      - 1024 regions N–S
      - 12 layers in depth
    - A single iteration of the model
      - simulates ocean circulation for 10 minutes
      - involves about 30B floating-point operations
    - To carry out the simulation for 1 year
      - about 50,000 iterations are required
    - Simulation for 6 years would involve 10^{16} floating-point operations

Why parallel processing?

- The motivations can be summarized as follows:
  1. Higher speed, or solving problems faster.
     - important when applications have “hard” or “soft” deadlines
       - E.g., at most a few hours of computation time to do 24-hour weather forecasting
  2. Higher throughput, or solving more instances of given problems
     - important when many similar tasks must be performed
       - E.g., banks and airlines use transaction processing systems that handle large volumes of data.
  3. Higher computational power, or solving larger problems.
     - allow us to
       - use more accurate models
       - or to carry out simulation runs for longer periods of time
         - (e.g., 5-day weather forecasting).

Why parallel processing?

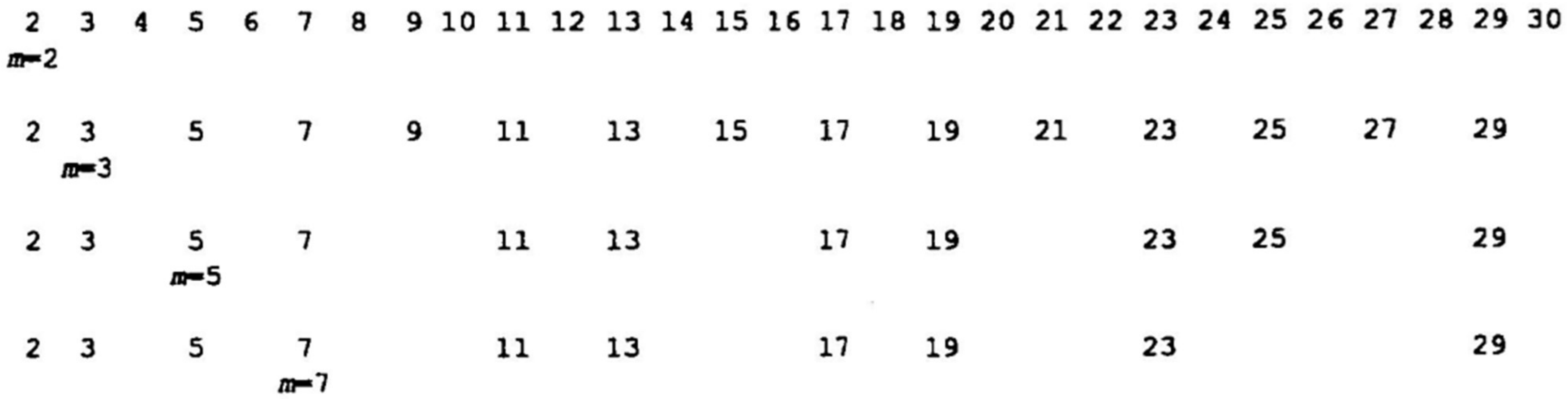
- figure-of-merit in parallel processors
  - the computation speed-up factor with respect to a uniprocessor
  - Captures all three aspects above
  - The ideal efficiency in parallel systems is to achieve a computation speed-up factor of p with p processors
    - in many cases cannot be achieved
    - some speed-up is generally possible
    - actual gain depends on
      - the architecture used for the system
      - the algorithm run on it

Why parallel processing?

- A motivating example
  - Problem: constructing the list of all prime numbers in [1, n] for a given integer n>0
    - Sieve of Eratosthenes (a simple algorithm):
      - Start with the list of numbers 1, 2, 3, 4, . . . , n
        - represented as a “mark” bit-vector
        - initialized to 1000 . . . 00.
      - In each step
        - the next unmarked number m is a prime.
          - associated with a 0 in element m of the mark bit-vector
        - Find this element m and mark all multiples of m beginning with m^2.
        - When $m^2 > n$, the computation stops and all unmarked elements are prime numbers.

Why parallel processing?

- A motivating example



2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
m=2

2 3 5 7 9 11 13 15 17 19 21 23 25 27 29
m=3

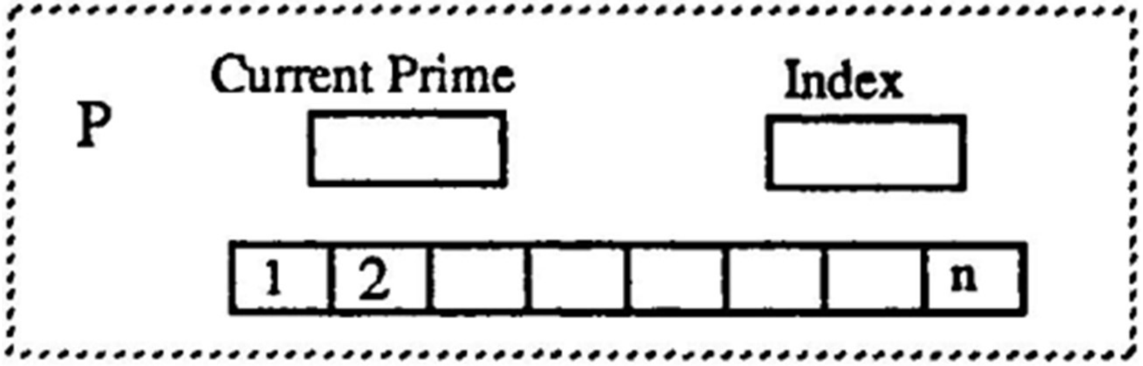
2 3 5 7 11 13 17 19 23 25 29
m=5

2 3 5 7 11 13 17 19 23 29
m=7

Figure 1.3. The sieve of Eratosthenes yielding a list of 10 primes for $n = 30$. Marked elements have been distinguished by erasure from the list.

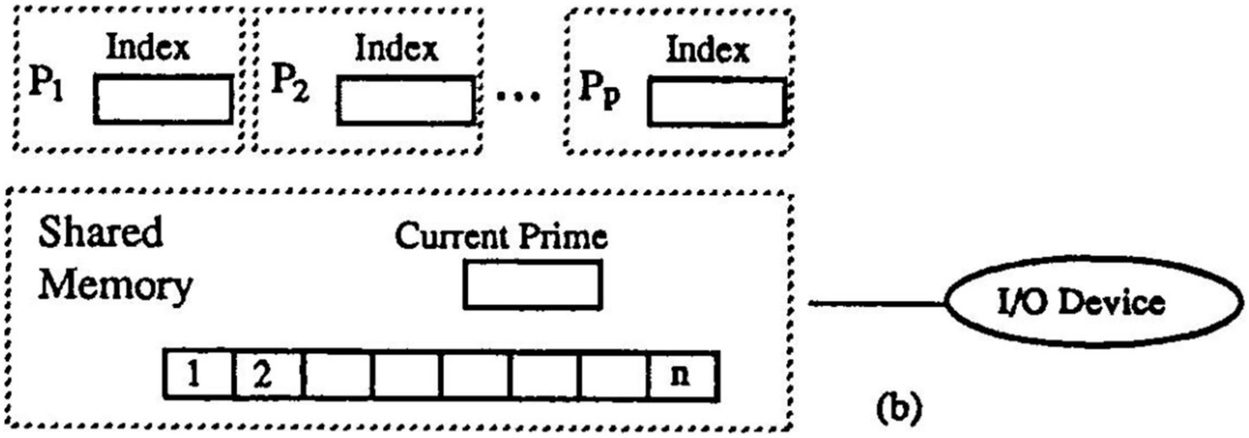
Why parallel processing?

- A motivating example
  - a single-processor implementation
    - The variable “current prime”
      - is initialized to 2
      - in later stages, holds the latest prime number found.
    - For each prime found, variable “index”
      - is initialized to the square of this prime
      - is then incremented by the current prime in order to mark all its multiples



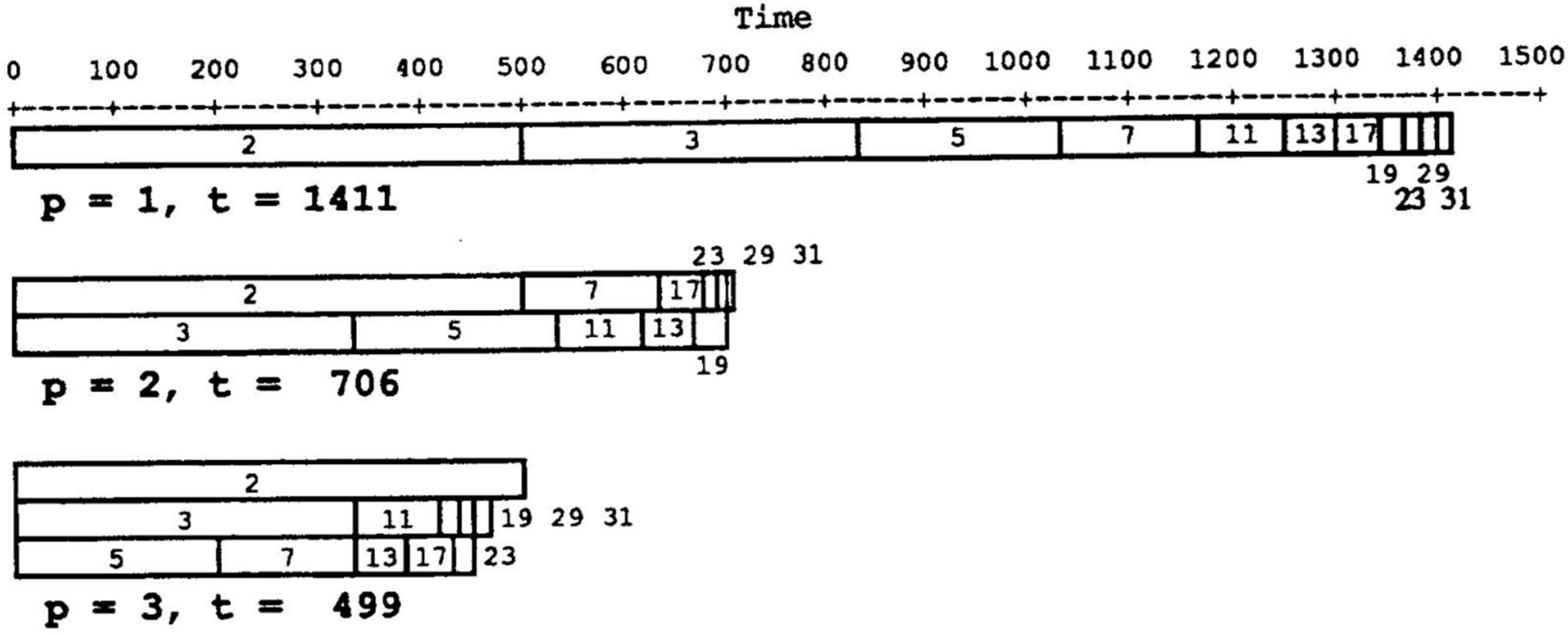
Why parallel processing?

- A motivating example
  - first parallel solution using p processors
    - The list of numbers and the current prime are stored in a shared memory.
    - An idle processor
      - refers to the shared memory
      - updates the current prime
      - uses its private index to
        - step through the list
        - mark the multiples of that prime
    - Division of work is thus self-regulated.



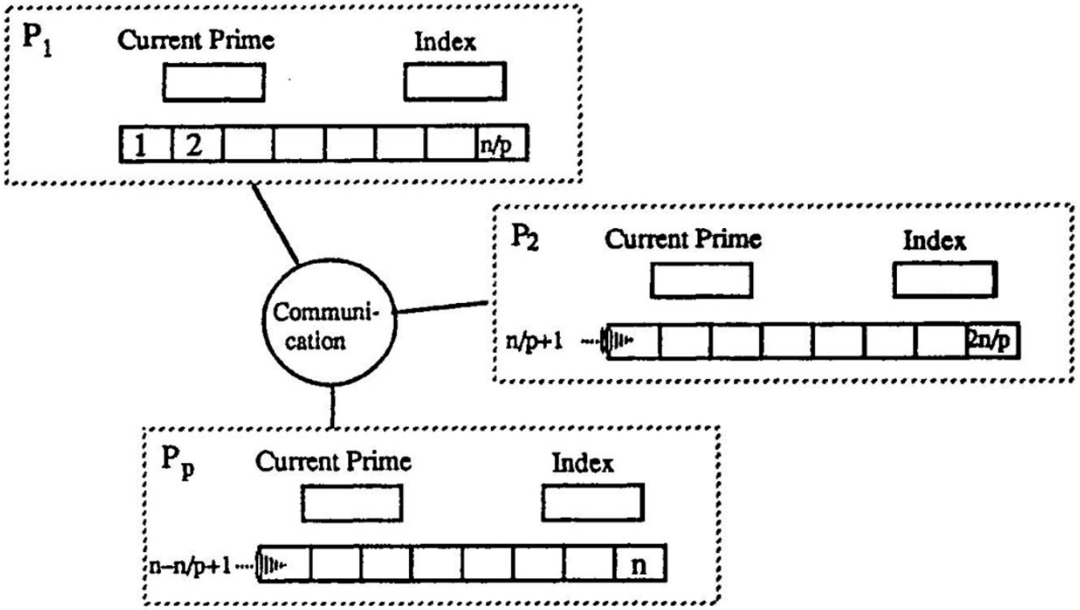
Why parallel processing?

- A motivating example
  - first parallel solution using p processors
    - activities of the processors and the termination time for n = 1000 and 1 ≤ p ≤ 3
    - using more than three processors would not reduce the computation time



Why parallel processing?

- A motivating example
  - a data-parallel approach
    - the bit-vector representing the n integers is divided into p equal-length segments
      - each segment stored in the private memory of one processor
    - Assume that $p < \sqrt{n}$,
      - Processor 1, acts as a coordinator
        - all the primes whose multiples must be marked reside in
        - It finds the next prime and broadcasts it to all other processors
          - they then proceed to mark the numbers in their sublists



Why parallel processing?

- A motivating example
  - a data-parallel approach
    - The overall solution time consists of two components
      - the time spent on transmitting the selected primes to all processors (communication time)
        - Typically, grows with the number of processors
          - though not necessarily in a linear fashion
      - the time spent by individual processors marking their sub lists (computation time)

Why parallel processing?

- A motivating example
  - a data-parallel approach
    - adding more processors beyond a certain optimal number
      - does not lead to any improvement in the total solution time or in attainable speed-up
      - because of the communication overhead

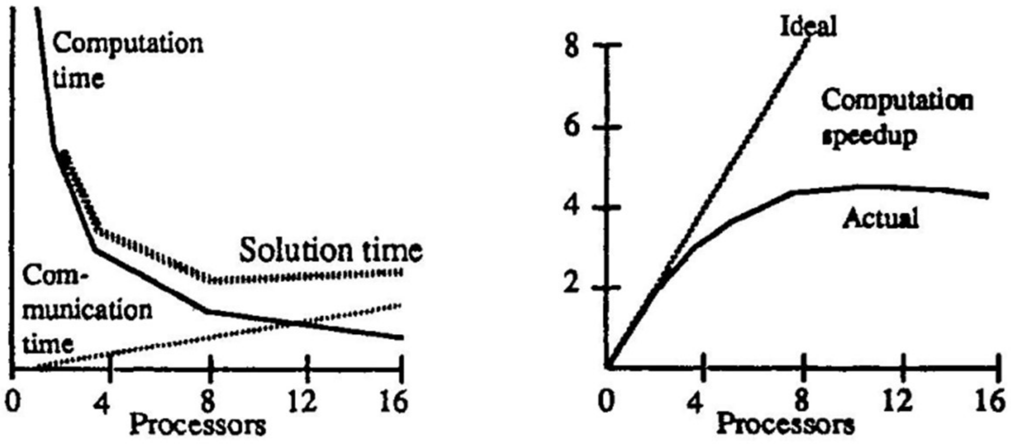


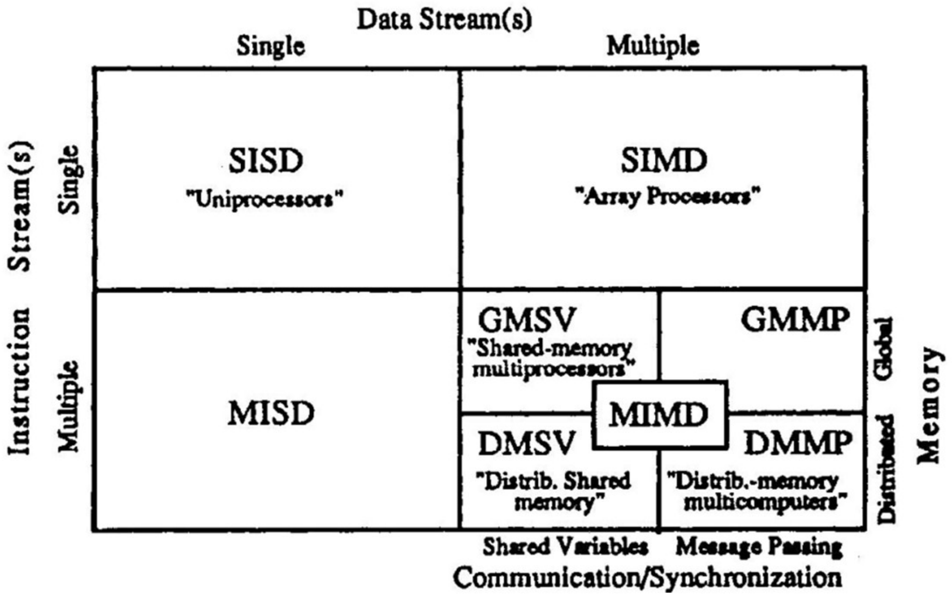
Figure 1.8. Trade-off between communication time and computation time in the data-parallel realization of the sieve of Eratosthenes.

Types of parallelism: a taxonomy

- Two main categories for parallel computers
    - Control-flow parallel computers
        - are essentially based on the same principles as the sequential or von Neumann computer
            - except that multiple instructions can be executed at any given time
    - Data-flow parallel computers
        - sometimes referred to as “non-von Neumann,”
        - are completely different
        - they have no pointer to active instruction(s) or a locus of control.
        - The control is totally distributed
- we will focus exclusively on control-flow parallel computers

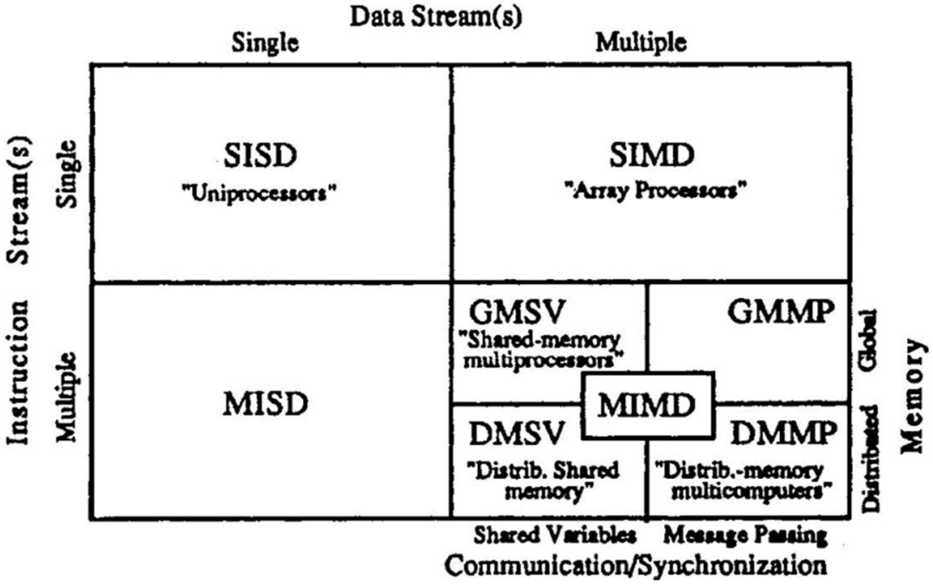
Types of parallelism: a taxonomy

- Flynn proposed a four-way classification of computer systems
  - SISD
    - represents ordinary “uniprocessor” machines
  - SIMD
    - several processors directed by instructions issued from a central control unit
    - sometimes characterized as “array processors.”
  - MISD
    - have not found widespread application
  - MIMD
    - Further classified based on
      - their memory structure (global or distributed)
      - mechanism used for communication/synchronization (shared variables or message passing).



Types of parallelism: a taxonomy

- Flynn proposed a four-way classification of computer systems
  - MIMD
    - GMSV
      - loosely referred to as (shared-memory) multiprocessors
    - GMMP
      - is not widely used
    - DMMP
      - known as (distributed-memory) multicomputers
    - DMSV
      - is becoming popular combining
        - the implementation ease of distributed memory
        - the programming ease of the shared-variable scheme
      - is some-times called distributed shared memory



Roadblocks to parallel processing

- The software inertia
  - billions of dollars worth of existing software makes it hard to switch to parallel systems
  - This objection is valid in the short term
  - In the long term
    - New applications will be developed
    - many new problems will become solvable with increased performance.
    - Students are already being trained to think parallel.
    - tools are being developed to transform sequential code into parallel code automatically

Roadblocks to parallel processing

- Amdahl’s law
  - a small fraction f of inherently sequential or unparallelizable computation severely limits the speed-up that can be achieved with p processors

$$\text{speed-up} \le 1/[f + (1 - f)/p] = p/[1 + f(p - 1)]$$

Roadblocks to parallel processing

- Amdahl’s law
  - The speed-up can never exceed 1/*f*
    - no matter how many processors are used
    - for *f* = 0.1, speed-up has an upper bound of 10
  - Fortunately
    - there exist applications with very small sequential overhead.
    - the sequential overhead need not be a constant fraction of the job independent of problem size.

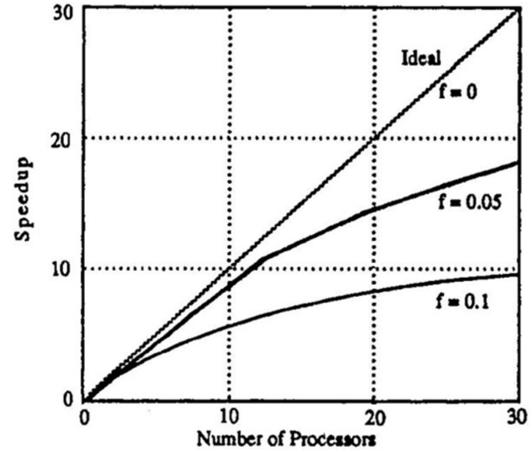
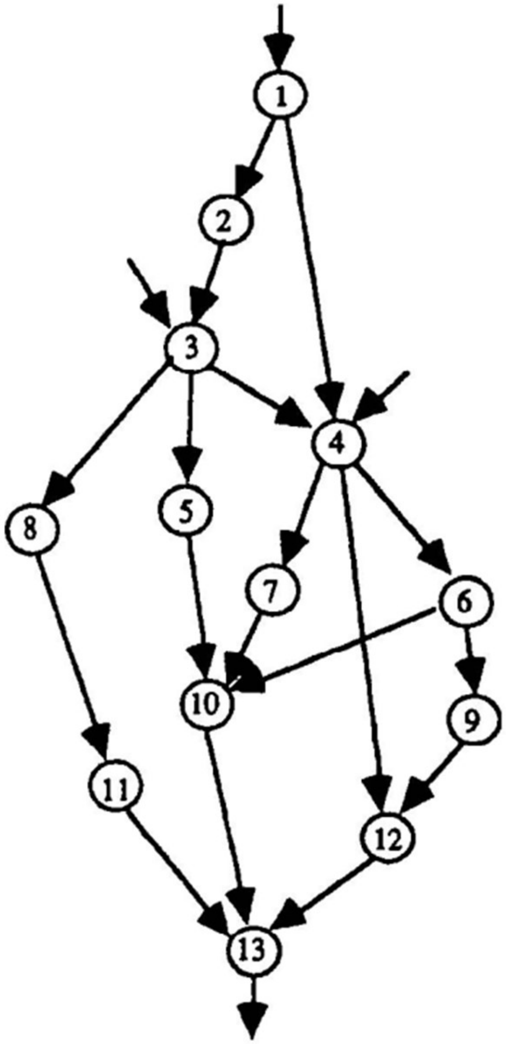


Figure 1.12. The limit on speed-up according to Amdahl’s law

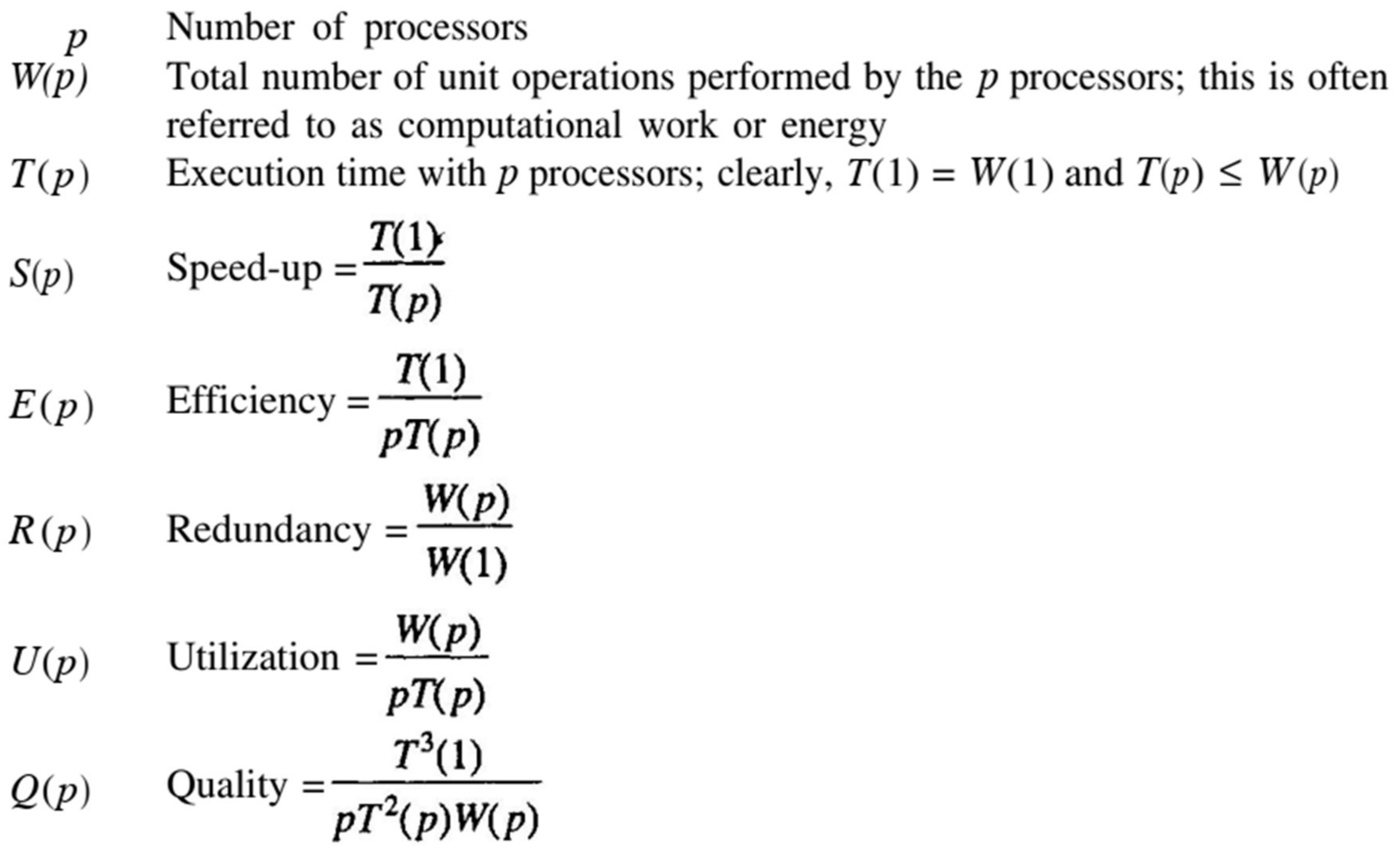
Roadblocks to parallel processing

- Closely related to Amdahl’s law
  - some applications lack inherent parallelism
    - limiting the achievable speed-up with multiple processors



Effectiveness of parallel processing

- certain measures for effectiveness of parallel algorithms



p	Number of processors
$W(p)$	Total number of unit operations performed by the p processors; this is often referred to as computational work or energy
$T(p)$	Execution time with p processors; clearly, $T(1) = W(1)$ and $T(p) \leq W(p)$
$S(p)$	Speed-up $= \dfrac{T(1)}{T(p)}$
$E(p)$	Efficiency $= \dfrac{T(1)}{pT(p)}$
$R(p)$	Redundancy $= \dfrac{W(p)}{W(1)}$
$U(p)$	Utilization $= \dfrac{W(p)}{pT(p)}$
$Q(p)$	Quality $= \dfrac{T^3(1)}{pT^2(p)W(p)}$

Effectiveness of parallel processing

- certain measures for effectiveness of parallel algorithms
    - not difficult to establish the following relationships

$$1 \le S(p) \le p$$

$$U(p) = R(p)E(p) \qquad \frac{1}{p} \le E(p) \le U(p) \le 1$$

$$E(p) = \frac{S(p)}{p} \qquad 1 \le R(p) \le \frac{1}{E(p)} \le p$$

$$Q(p) = E(p)\,\frac{S(p)}{R(p)} \qquad Q(P) \le S(p) \le p$$

Effectiveness of parallel processing

- E.g., Finding the sum of 16 numbers
  - $T(1) = W(1) = 15$
  - Assume unit-time additions and ignore all else
  - With $p = 8$ processors, we have
    - $W(8) = 15$ $T(8) = 4$
    - $E(8) = 15/(8 \times 4) = 47\%$ $S(8) = 15/4 = 3.75$
    - $R(8) = 15/15 = 1$ $Q(8) = 1.76$
    - the 8 processors perform all the additions at the same tree level
    - The relatively low efficiency is the result of limited parallelism near the root of the tree