CS5351.0501/0502, Parallel Processing, Summer I, 2009

Lecture 1: Introduction and Fundamentals

(June 08, 2009)

Lecture Outline

(Reading: Chapter 1 of textbook)

```
1. Why parallel computers and parallel algorithm
2. Methods used to achieve higher performance computing
3. Terminologies
4. Pipelining vs. parallelism
5. Control parallelism
6. The Sieve of Eratosthenes
7. Amdahl's Law (Amdahl, 1967)
```

1. Why parallel computers and parallel algorithms

(0) Problems requiring extremely high computational power.
Example. NP-complete problems. Tower of Hanoi problem.

(1) Weather prediction
Example. Forcasting the weather requires the solution of general circulation model equations. Three dimensional space, plus time. 270 miles grid. 100 billion operations for a 24-hour fourcast. 100 minutes in a supercomputer capable of performing 100 million operations per second. Half the grid, 16-fold increase in the number of computations. $16 \times 100 = 1600$ minutes.

(2) Artificial intelligence
Friendly input and/or output. Need to recognize voice, pictures, natural languages. To be handled in real-time needs enormous amount memory and computational power. Why? Consider picture recognition.

(3) Remote sensing
Example. A map from a satellite has 8 images. Each image is a 6000 (pixel) by 6000 square, each pixel represented by 8 bits. A single picture needs 288 megabytes, or 2284 megabits.

(4) Nuclear reactor control
Simulate and verify the corrective measures.

(5) Military usage
Real-time missile control

2. Methods used to achieve higher performance computing

(1) From bit-serial to bit-parallel arithmetic

(2) I/O processors (channels)

(3) Interleaving memory
Memory: a bottleneck in the system; matching the speed of different components
Lower-order interleaving vs. higher-order interleaving

(4) Cache memory
Locality of reference (temporal locality, spatial locality)

(5) Instruction look-ahead (or instruction buffering)

(6) Multiple functional units
Int add, floating multiplication, boolean, shift and etc.

(7) Instruction pipelining
The use of pipelining to allow multiple instructions to in the execution stage at the same time. Divide an execution of each instruction into a number of phases, such as fetch, decode, oprand fetch, and instruction execution.

(8) Pipelined vector processors

(9) Processor array

(10) Multiprogramming, timesharing

(11) Multiprocessors
A computer with shared-memory multiple-CPU

(12) Multicomputers
A computer with multiple-CPU but without shared-memory

(13) Data Flow computers
Control-driven vs. data-driven
Data flow graph

(14) Time chart of computer performance improvement of four classes of computers since 1965 (Fig. 1-2, p.4). The performance has increased by a factor of 10 every 5 years

3. Terminologies

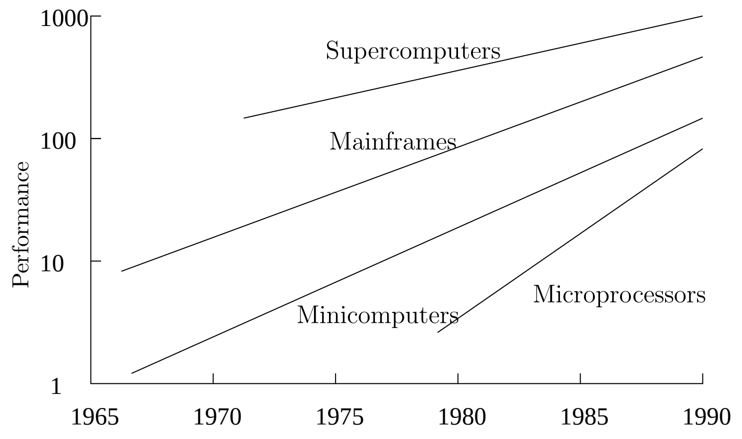

Figure 1: Performance growth of four classes of computers since 1965. (Courtesy Hennessy and Patterson 1990.) (Fig.1-2, p.4 of textbook)

(1) **Parallel processing**: concurrent processing of data elements that belong to one or more processes that solve a single problem.

(2) **Parallel computers**: a parallel computer is a multiprocessor computer capable of parallel processing.

(3) **Supercomputers**: a supercomputer is a general purpose computer capable of solving individual problems at extremely high computational speeds, compared with other computers available during the same period of time.

(4) **Throughput**: the number of results produced per unit time.

(5) **Data parallelism**: is the use of multiple functional units to apply the same operation simultaneously to elements of a data set. A k-fold increase in the number of functional units leads to a k-folds increase in the throughput of the system, if there is no overhead involved.

(6) **Speedup**: the ratio between the time needed for the most efficient sequential algorithm to solve a problem and the time needed to solve the same problem on a parallel or pipelining machine.

4. Pipelining vs. parallelism

(0) **Pipelining** and **parallelism** are two main mechanism to achieve concurrency.

(1) Concurrency level:

- User level (time sharing);
- Job or program level (multiprogramming);

    - Process level (Multiprocessor);
    - Instruction level (pipelining).

(2) **Segments** (or *stages*): a pipelined computation is divided into a number of steps, called segments or stages. Each segment works at full speed on a particular part of a computation.

(3) **Example.** A widget assemlby line.

   a. Fig.1-4, p.7. 3 time units to assemble a toy. 3 steps, A, B, and C, each take 1 time unit.

      - A sequential assembly machine takes 3 time units to produce a toy, 6 time units to produce two toys, and so on. The throughput is 1/3.
      - In a pipelining machine, 3 steps. First machine for task A, 2nd for B, and 3rd for C. 1 toy in 3 time units, 1 toy every other time unit thereafter. Throughput is 1/3, 2/4, 3/5, 4/6 and so on.
      - Three sequential machines operate in parallel. 3 toys in 3 time units, 6 in 6 time units.

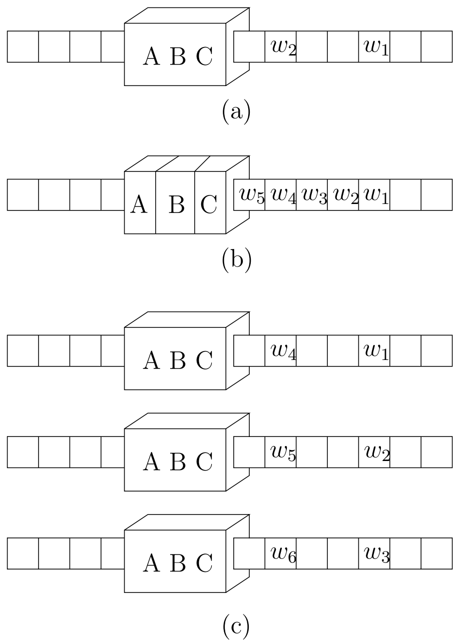

Figure 2: Three methods to assemble widgets: (a) A sequential widget-assembly machine produces one widget every three units of time. (b) A three-segment pipelined widget-assembly machine produces first widget in three units of time and successive widgets every time unit thereafter. (c) A three way data-parallel widget-assembly machine produces three widgets every three units of time. (Fig.1-4, p.7 of textbook)

b. Comparison of speedups by parallel and pipelined method: Fig.1-5,p.8.

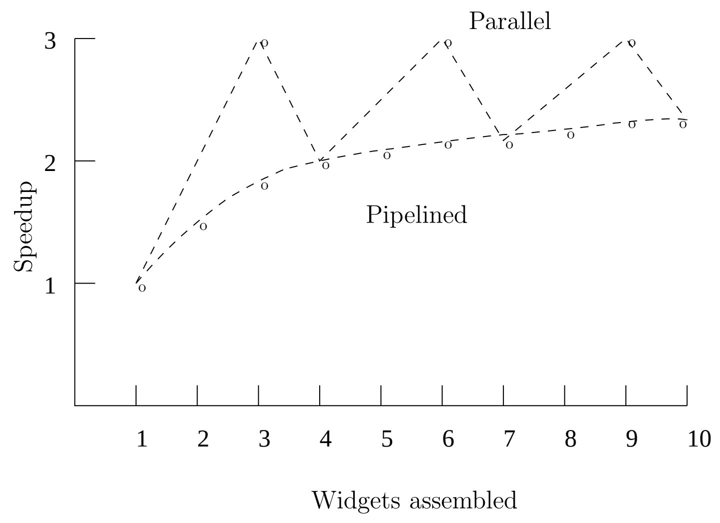

Figure 3: Speedup achieved by pipelined and parallel widget-assembly machines. Note that speedup is graphed as a function of problem size (number of widgets assembled). This is unusual. Speedup is typically graphed as a function of number of processors used. (Fig.1-5, p.8)

5. Control parallelism and scalability

(1) **Control parallelism**: is achieved by applying different operations to different data elements simultaneously.

(2) The data-flow graph of a control-parallel algorithm can be arbitrarily complex. Pipelining is a special case of the control-parallel algorithms – its data-flow graph is a simple directed path.

(3) Most realistic problems can exploit both data parallelism and control parallelism

(4) **Example** (bot. of p.8 to p .9, Fig. 1-6): the weekly landscape maintenance problem.

a. Four tasks: mowing the lawn, edging the lawn, checking the sprinklers, weeding the flower beds.

b. Observations:

(a) Except for checking sprinklers, other three tasks can be performed in parallel.

(b) The task of edging lawn have to be performed after the task of mowing lawn.

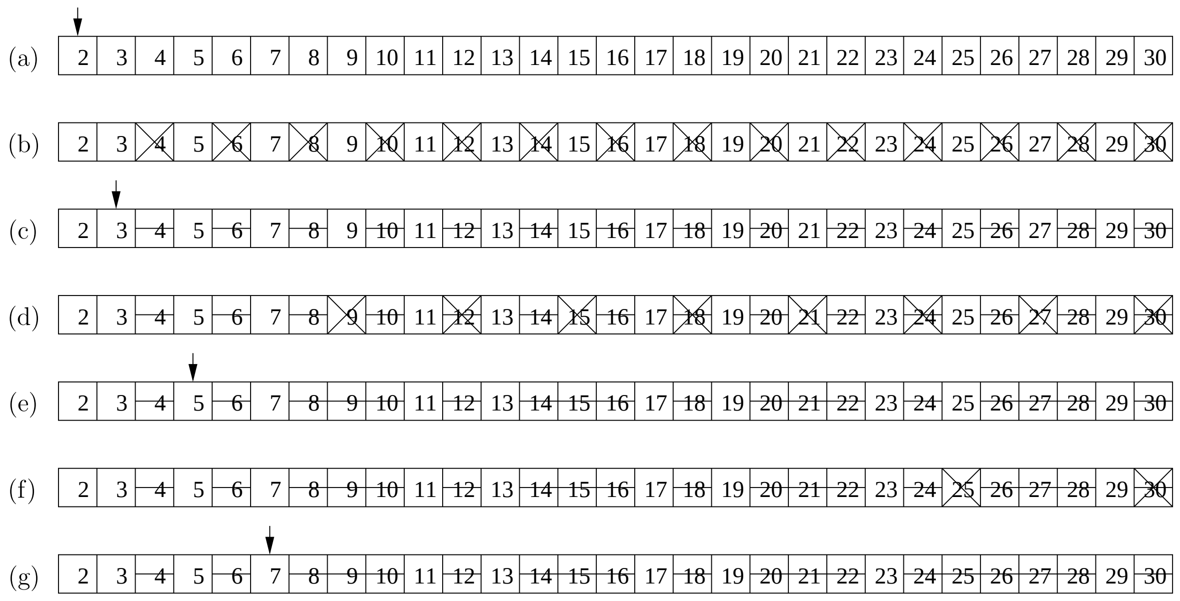

Figure 4: Use of Sieve of Eratosthenes to find all primes less than or equal to 30. (a) Prime is next unmarked natural number – 2. (b) Strike all multiples of 2, beginning with 2^2. (c) Prime is next unmarked natural number – 3. (d) Strike all multiples of 3, beginning with 3^2. (e) Prime is next unmarked natural number – 5. (d) Strike all multiples of 5, beginning with 5^2. (f) Prime is next unmarked natural number – 7. Since 7^2 is greater than 30, algorithm terminiates. All remaining unmarked natural numbers are also primes. (Fig.1-7, p.10)

SPEEDY LANDSCAPE, INC

Work assignments – Medici Manor

Mow lawn	**Edge lawn**
Allan	Frances
Bernice	Georgia
Charlene	**Weed garden**
Dominic	Hillary
Edward	Irene
Check sprinklers	Jose
Allan	

Fig.1-6 Most realistic problems have data parallelism, control parallelism, and preceddence constraints between tasks

(5) Scalability

a. Algorithmic and architectural scalability.

(a) A parallel algorithm is scalable if the level of parallelism increases at least linearly with the problem size.

(b) A parallel architecture is scalable if it continues to yield the same performance per processor, albeit used on a larger problem size, as the number of processors increases. In other words, increasing the number of processors will speedup the computation.

b. Data-parallel algorithms are more scalable than control-parallel algorithms due to the fact that:

(a) the level of control parallelism is usually a constant, independent of the problem size.

(a) the level of data parallelism is usually a function of the problem size.

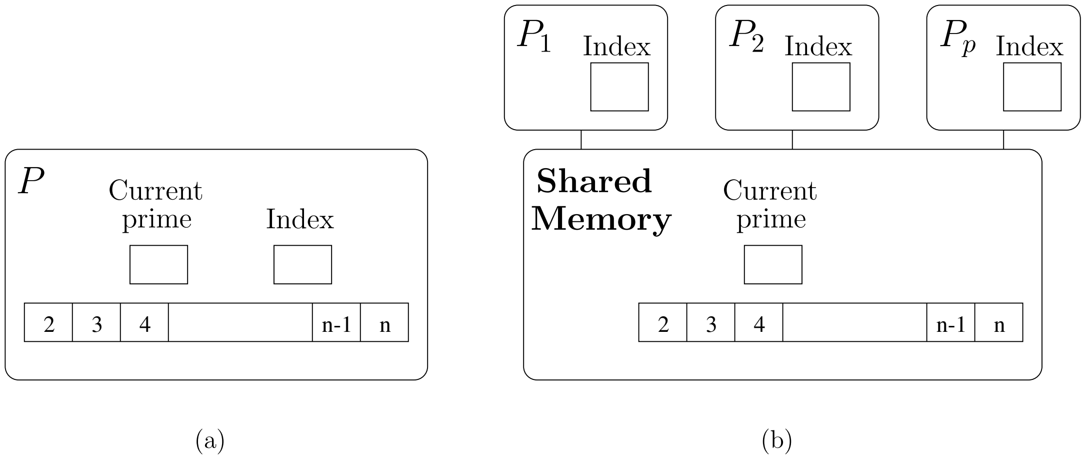

Figure 5: Shared memory model for parallel Sieve of Eratosthenes algorithm. (a) Sequential algorithm maintains array of natural numbers, variable storing current prime, and variable storing index of loop iterating through array of natural numbers. (b) In parallel model each processor has its own private loop index and shares access to other variables with all processors. (Fig.1-8, p.11)

6. The Sieve of Eratosthenes

(1) The problem

a. The classical finding prime numbers problem: find all prime numbers less than or equal to some given integer n.

b. Prime numbers: have only two factors: itself and 1.

c. Eratosthenes Sieve algorithm: Striking multiples of 2, 3, 5, and successive primes (Fig. 1-7, p.10).

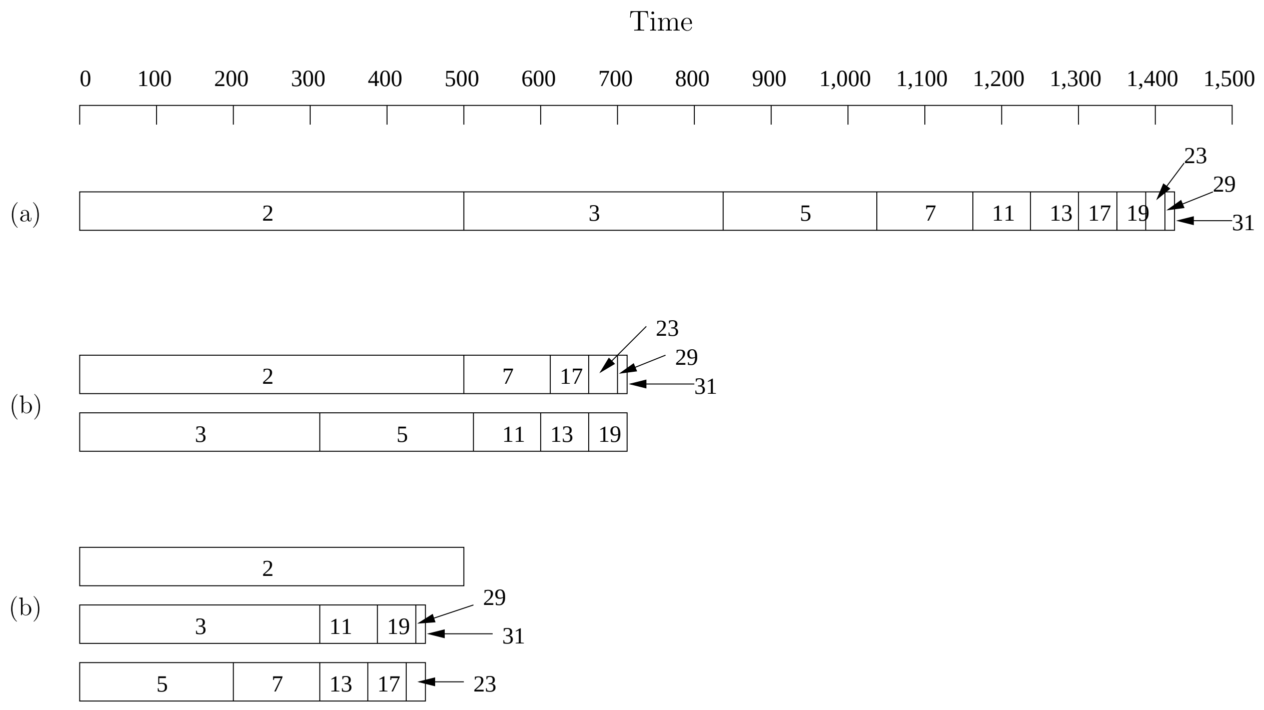

Figure 6: Study of how adding processors reduces the execution time of the control-parallel Sieve Eratosthenes algorithm with $n = 1,000$. The number in the bar is the time needed to strike these multiples. (a) Single processor strikes out all composite numbers in 1,411 units of time. (b) With two processors execution time drops to 706 time units. (c) With three or more processors execution time is 499 time units, the time needed for a processor to strike all multiples of 2. (Fig.1-9, p.13)

d. Sequential Eratosthenes algorithm is not practical for testing primality of "interesting" numbers – those with hundreds of digits – because the time complexity is $\Omega(n)$, and n increases exponentially with the number of digits. Many existing algorithms make use of the sieve technique.

e. Sequential implementation: Fig. 1-8, p. 11.
Three key data structures: a boolean array whose elements correspond to the natural numbers being sieved, an integer corresponding to latest prime number found, and an integer used as a loop index incremented as multiples of the current prime are marked as composite numbers (Fig. 1-8a).

(2) Control parallel approach (p. 10-11, Fig. 1-8, 1-9)

a. Basic idea: every processor repeated goes the two-step process of finding the next prime number and striking from the list multiples of that prime,

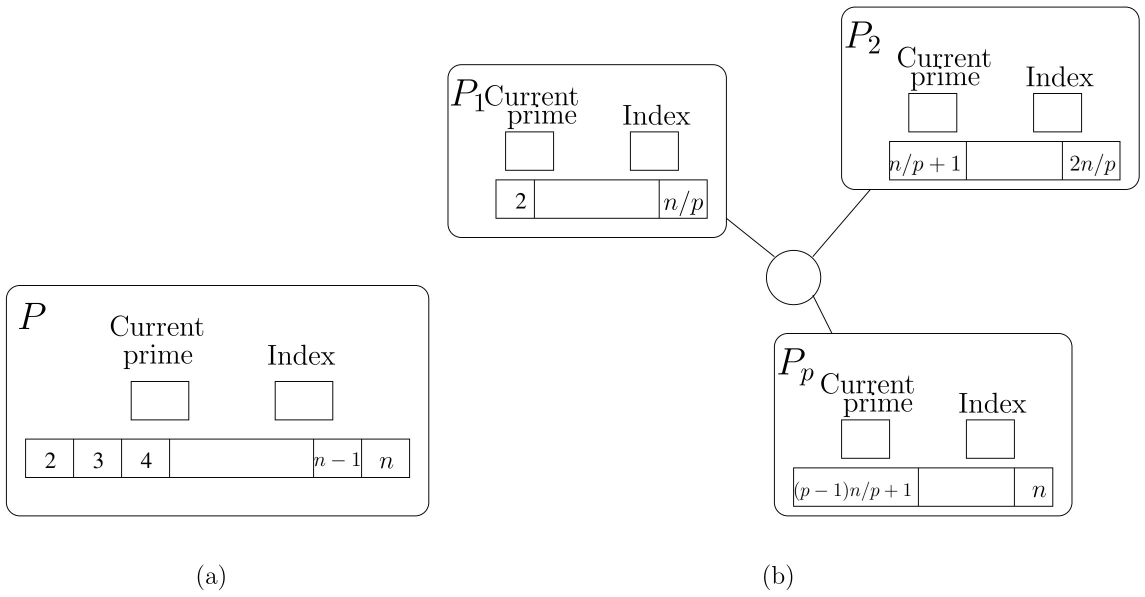

Figure 7: Private memory model for parallel Sieve of Eratosthenes algorithm. (a) Sequential algorithm maintains array of natural numbers, variable storing current prime, and variable storing index of loop iterating through array of natural numbers. (b) In parallel model each processor has its own copy of the variables containing the current prime and the loop index. Processor 1 finds primes and communicates them to the other processors. Each processor iterates through its own portion of the array of natural numbers, marking multiples of the prime. (Fig.1-10, p.14)

beginning with its square. The processors continure until a prime whose value is greater than $\sqrt{n}$ is found.

b. Shared: *current prime* and the *boolean array* (Fig.1-8(b)).

c. Access to the shared memory must be controled, or inefficiencies or errors may occur:

  * Two processors may end up using the same prime to sieve through the array. Inefficient.
  * A processor may end up sieving multiples of a composite number. Inefficient.

d. Maximum achievable speedup by this parallel algo.

  (a) Assume that it takes 1 unit of time for a processor mark a cell. Assume that there are k primes less than or equal to $\sqrt{n}$: $\pi_1, \pi_2, \cdots, \pi_k$.

  (b) The total amount of time a single processor spends to strike out com-

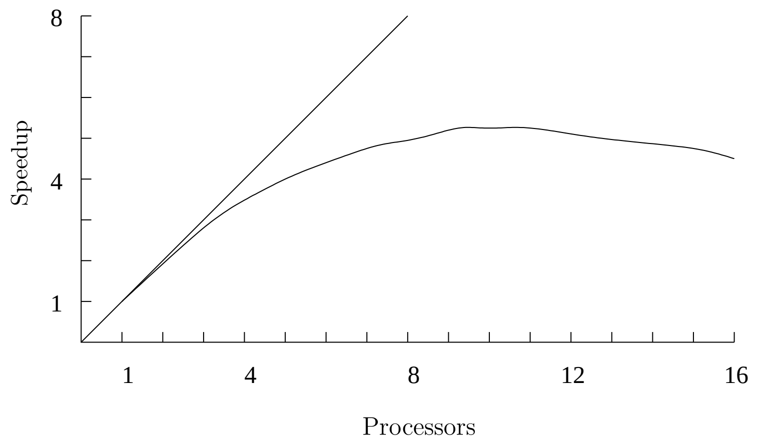

Figure 8: Estimated speedup of the data-parallel Sieve of Estorathenes algorithm, assuming $n = 1,000,000$ and $\lambda = 100\chi$. Note that the speedup is graphed as a function of number of processors used. This is typical. (Fig.1-11, p.15 of textbook)

posite numbers is:

$$\lceil\frac{(n+1)-\pi_1^2}{\pi_1}\rceil+\lceil\frac{(n+1)-\pi_2^2}{\pi_2}\rceil+\lceil\frac{(n+1)-\pi_3^2}{\pi_3}\rceil+\cdots+\lceil\frac{(n+1)-\pi_k^2}{\pi_k}\rceil$$

$$= \lceil\frac{n-3}{2}\rceil + \lceil\frac{n-8}{3}\rceil + \lceil\frac{n-24}{5}\rceil + \cdots + \lceil\frac{(n+1)-\pi_k^2}{\pi_k}\rceil$$

(c) There are $(n-3)/2$ multiples of 2 in the range 4 through n, $(n-8)/3$ multiples of 3 in the range 9 to n, and so on. For $n = 1,000$, the sum is 1,411.

(d) Time taken by the parallel algorithm: Fig.1-9.
Speedup = 2 (1,411/706) with two processors
Speedup = 2.83 (1,411/499) with three processors
It is clear that the parallel execution time will not decrease if more than three processors are used, because the time needed for a single processor to sieve all multiples of 2 determines the paralle execution time.
Hence speedup is limied to 2.83 for $n = 1,000$

(e) Increasing n does not significantly raise the upper bound on speedup imposed by a single processor striking all multiples of 2 (Prob.1-10).

(3) Data parallel approach (p. 13-15, Fig.1-10, 1-11, 1-12)

a. Basic idea: processors work together to strike out multiples of each newly found prime. Every processor is responsible for a segment of array representing the natual numbers.

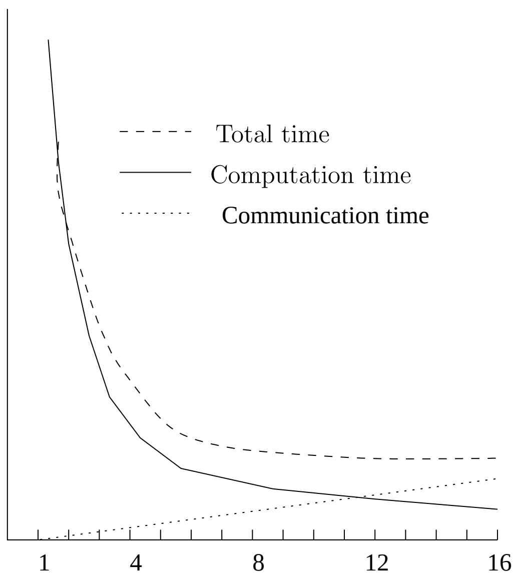

Figure 9: Total execution time of the data-parallel Sieve of Eratosthenes algorithm is the sum of the time spent computing and the time spent communicating. Computation time is inversely proportional to the number of processors; communication time is directly proportional to the number of processors. (Fig.1-12, p.15 of textbook)

b. Speedup: for data-parallel algorithm based on the shared memory model of Fig.1-8(b), the speedup is easy to calculate and ls left as an exercise.

c. Message passing based data parallel approach.

   (a) Basic idea (Fig.1-10,p.14): p processors, each is assigned no more than n/p natural numbers. Assume that $p << \sqrt{n}$ so that all primes less than $\sqrt{n}$ and the first prime greater than $\sqrt{n}$ are in the list of natural numbers controlled by the 1st processor.

   (b) Processor 1 finds the next prime, and broadcast its value to the other processors.

   (c) All processors then strike out from their lists of natural numbers all multiples of the newly found prime.

   (d) This process continues until the first processor reaches a prime greater than $\sqrt{n}$.

   (e) Estimation of the execution time:

      i. Assume time taken to find next prime is ignorable. Focus on time spent striking composite numbers and communication cost (this is extra for this model).

      ii. Assume χ time units needed for a processor to mark a multiple of

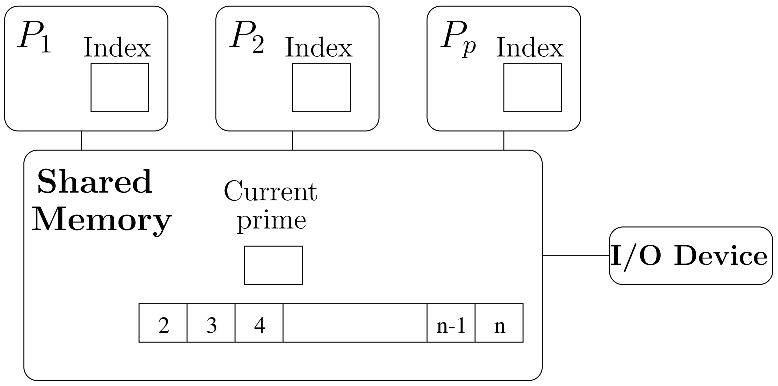

Figure 10: A shared memory model incorporating a sequential I/O device. (Fig.1-13, p.16)

a prime as a composite number. Total time a processor spends to strike out multiples is no greater than:

$$(\lceil\frac{\lceil n/p\rceil}{2}\rceil+\lceil\frac{\lceil n/p\rceil}{3}\rceil+\lceil\frac{\lceil n/p\rceil}{5}\rceil+\cdots+\lceil\frac{\lceil n/p\rceil}{\pi_k}\rceil)\chi$$

iii. Communication cost: assume λ time units are needed for a processor to pass a prime to another processor.
The total communication time for all k primes is $k(p-1)\lambda$ for processor 1.

iv. Consider the case where $n = 1,000,000$. There are $k = 168$ primes less than 1,000. The largest of these is 997. Therefore the maximum execution time spent on striking out primes is:

$$(\lceil\frac{\lceil 1,000,000/p\rceil}{2}\rceil+\lceil\frac{\lceil 1,000,000/p\rceil}{3}\rceil+\cdots+\lceil\frac{\lceil 1,000,000/p\rceil}{997}\rceil)\chi$$

And the communication cost is $k(p-1)\lambda = 168(p-1)\lambda$.

(f) Estimating the speedup. The total execution time by the message passing data-parallel algorithm shown in Fig.1-10 is

$$(\lceil\frac{\lceil 1,000,000/p\rceil}{2}\rceil+\lceil\frac{\lceil 1,000,000/p\rceil}{3}\rceil+\cdots+\lceil\frac{\lceil 1,000,000/p\rceil}{997}\rceil)\chi+168(p-1)\lambda$$

Assume that $\lambda = 100\chi$ (i.e. the time to passing a number to another processor is 100 times of that marking a composite number), we have the following total execution time:

$$(\lceil\frac{\lceil 1,000,000/p\rceil}{2}\rceil+\lceil\frac{\lceil 1,000,000/p\rceil}{3}\rceil+\cdots+\lceil\frac{\lceil 1,000,000/p\rceil}{997}\rceil)\chi+168(p-1)*100\chi$$

i. Estimated speedup: Fig.1-11, p.15.

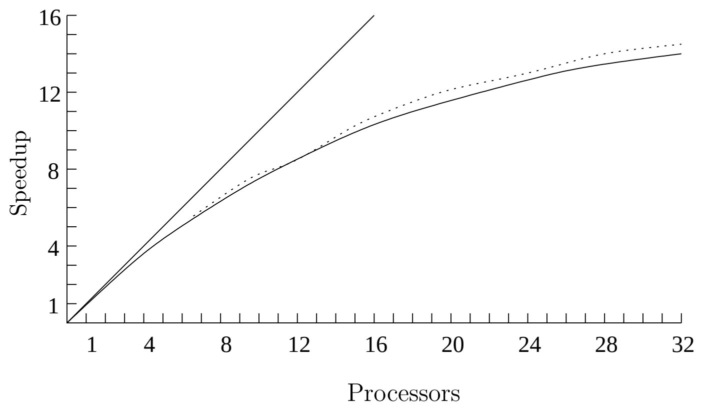

Figure 11: Expected speedup of data-parallel Sieve of Eratosthenes algorithm that outputs primes to an I/O device. Solid curve is speedup predicted from analysis. Dashed curve is maximum speedup as determined by Amdahl's law. This graph is for the case when $n = 1,000,000$ and $\beta = \chi$. (Fig.1-14, p.16 of textbook)

ii. Observation: the speedup is not directly proportional to the number of processors used. Speedup is maximum when the number of processors is 11. With more processors, speedup actually declines.

iii. Reason: with more processors, although the computation time decreases, the communication cost dominates. Fig.1-12 shows the two components – computation time and communication time – in relation to the total execution time.

(4) Data parallel approach with I/O (p. 16-17, Fig.1-13, 1-14, 1-15)

a. Problems with the previous two parllel algorithms solving the Sieve of Eratosthenes problem is unrealistic because there is no output at the end of the algorithms.

b. Basic idea: augment the shared memory model for control-parallel shown in Fig.1-8: Fig.1-13. The new model now producing output. Observe: I/O operations must be sequential.

c. Assume that $i\beta$ as the time needed for a processor to transmit i prime numbers to the I/O device.

d. Assume that $n = 1,000,000$. There are 78,498 primes less than 1,000,000. The total computation time:

$$(\lceil\frac{\lceil 1,000,000/p\rceil}{2}\rceil + \lceil\frac{\lceil 1,000,000/p\rceil}{3}\rceil + \cdots + \lceil\frac{\lceil 1,000,000/p\rceil}{997}\rceil)\chi$$

and add to it the total I/O time, $78,498\beta$.

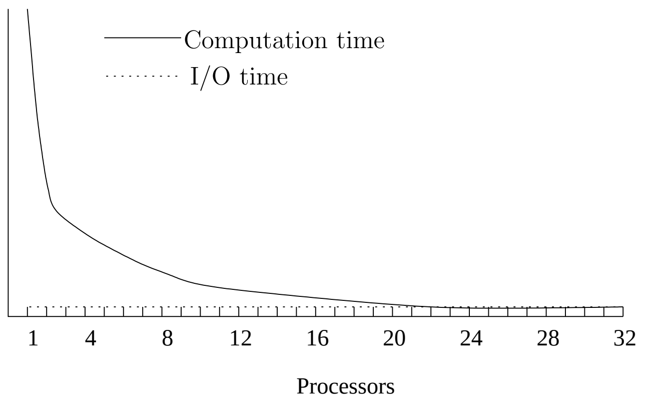

Figure 12: Total execution time of the data-parallel output-producing Sieve of Eratosthenes algorithm as a function of its two components, computation time and I/O time. This graph is for the case where $n = 1,000,000$ and $\beta = \chi$. (Fig.1-15, p.17 of textbook)

e. Fig.1-14 shows the expected speedup for 1,2,...,32 processors, assuming $\beta = \chi$.

f. Fig.1-15 shows the two components of parallel execution time: computation time and output time.

g. Output must be performed sequentially – it restricts the speedup – the Amdahl's law.

7. Amdahl's Law (Amdahl, 1967)

f: fraction of operations that must be performed in sequence;
p: number of parallel processes;
S: speedup.

$$S \leq \frac{1}{f + (1-f)/p}$$

(Question: how to obtain Amdahl's law?)

(1) Essence of Amdahl's law: Don't try to parallelize problems that are inherently sequential.

(2) **Example**. The above data parallel algorithm with I/O.

– $n = 1,000,000$, the sequential algorithm marks $2,122,048$ cells, and outputs 78,498 primes.

– Assumes that these two operations take the same amount of time. The the total sequential time is 2,200,546, and $f = 78,498/2,200,546 = .0357$.

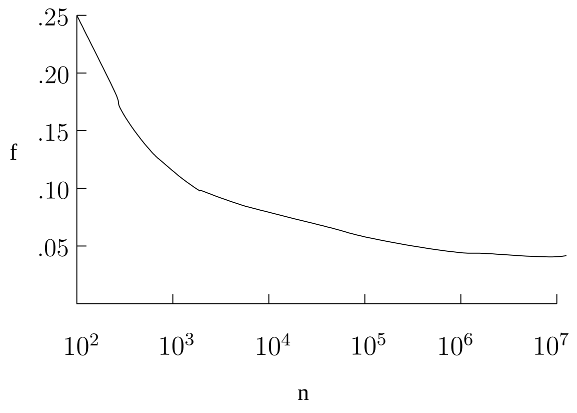

Figure 13: Fraction f of inherently sequential operations in data-parallel sieve algorithm with output, as a function of n, the size of the list of natural numbers, assuming $\beta = \chi$. (Fig.1-16, p.18 of textbook)

- An upper bound on the speedup with p processors:

$$\frac{1}{.0357 + .9643/p}$$

- The dotted curve in Fig.1-14 shows the upper bound on the speedup as predicated by Amdahl's law.

(3) **Amdahl effect**. Often, as the size of the problem increases, the fraction f of inherently sequential operation decreases. Fig.1-16 plots f as a function of n for the data parallel sieve algorithm with output, assuming $\beta = \chi$.