CHAPTER 1

# Introduction to Parallel Computing

The past decade has seen tremendous advances in microprocessor technology. Clock rates of processors have increased from about 40 MHz (e.g., a MIPS R3000, circa 1988) to over 2.0 GHz (e.g., a Pentium 4, circa 2002). At the same time, processors are now capable of executing multiple instructions in the same cycle. The average number of cycles per instruction (CPI) of high end processors has improved by roughly an order of magnitude over the past 10 years. All this translates to an increase in the peak floating point operation execution rate (floating point operations per second, or FLOPS) of several orders of magnitude. A variety of other issues have also become important over the same period. Perhaps the most prominent of these is the ability (or lack thereof) of the memory system to feed data to the processor at the required rate. Significant innovations in architecture and software have addressed the alleviation of bottlenecks posed by the datapath and the memory.

The role of concurrency in accelerating computing elements has been recognized for several decades. However, their role in providing multiplicity of datapaths, increased access to storage elements (both memory and disk), scalable performance, and lower costs is reflected in the wide variety of applications of parallel computing. Desktop machines, engineering workstations, and compute servers with two, four, or even eight processors connected together are becoming common platforms for design applications. Large scale applications in science and engineering rely on larger configurations of parallel computers, often comprising hundreds of processors. Data intensive platforms such as database or web servers and applications such as transaction processing and data mining often use clusters of workstations that provide high aggregate disk bandwidth. Applications in graphics and visualization use multiple rendering pipes and processing elements to compute and render realistic environments with millions of polygons in real time. Applications requiring high availability rely on parallel and distributed platforms for redundancy. It is therefore extremely important, from the point of view of cost, performance, and application requirements, to understand the principles, tools, and techniques for programming the wide variety of parallel platforms currently available.

## 1.1 Motivating Parallelism

Development of parallel software has traditionally been thought of as time and effort intensive. This can be largely attributed to the inherent complexity of specifying and coordinating concurrent tasks, a lack of portable algorithms, standardized environments, and software development toolkits. When viewed in the context of the brisk rate of development of microprocessors, one is tempted to question the need for devoting significant effort towards exploiting parallelism as a means of accelerating applications. After all, if it takes two years to develop a parallel application, during which time the underlying hardware and/or software platform has become obsolete, the development effort is clearly wasted. However, there are some unmistakable trends in hardware design, which indicate that uniprocessor (or implicitly parallel) architectures may not be able to sustain the rate of *realizable* performance increments in the future. This is a result of lack of implicit parallelism as well as other bottlenecks such as the datapath and the memory. At the same time, standardized hardware interfaces have reduced the turnaround time from the development of a microprocessor to a parallel machine based on the microprocessor. Furthermore, considerable progress has been made in standardization of programming environments to ensure a longer life-cycle for parallel applications. All of these present compelling arguments in favor of parallel computing platforms.

### 1.1.1 The Computational Power Argument – from Transistors to FLOPS

In 1965, Gordon Moore made the following simple observation:

"*The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000.*"

His reasoning was based on an empirical log-linear relationship between device complexity and time, observed over three data points. He used this to justify that by 1975, devices with as many as 65,000 components would become feasible on a single silicon chip occupying an area of only about one-fourth of a square inch. This projection turned out to be accurate with the fabrication of a 16K CCD memory with about 65,000 components in 1975. In a subsequent paper in 1975, Moore attributed the log-linear relationship to exponential behavior of die sizes, finer minimum dimensions, and "circuit and device cleverness". He went on to state that:

*"There is no room left to squeeze anything out by being clever. Going forward from here we have to depend on the two size factors - bigger dies and finer dimensions."*

He revised his rate of circuit complexity doubling to 18 months and projected from 1975 onwards at this reduced rate. This curve came to be known as "Moore's Law". Formally, Moore's Law states that circuit complexity doubles every eighteen months. This empirical relationship has been amazingly resilient over the years both for microprocessors as well as for DRAMs. By relating component density and increases in die-size to the computing power of a device, Moore's law has been extrapolated to state that the amount of computing power available at a given cost doubles approximately every 18 months.

The limits of Moore's law have been the subject of extensive debate in the past few years. Staying clear of this debate, the issue of translating transistors into useful OPS (operations per second) is the critical one. It is possible to fabricate devices with very large transistor counts. How we use these transistors to achieve increasing rates of computation is the key architectural challenge. A logical recourse to this is to rely on parallelism – both implicit and explicit. We will briefly discuss implicit parallelism in Section 2.1 and devote the rest of this book to exploiting explicit parallelism.

### 1.1.2 The Memory/Disk Speed Argument

The overall speed of computation is determined not just by the speed of the processor, but also by the ability of the memory system to feed data to it. While clock rates of high-end processors have increased at roughly 40% per year over the past decade, DRAM access times have only improved at the rate of roughly 10% per year over this interval. Coupled with increases in instructions executed per clock cycle, this gap between processor speed and memory presents a tremendous performance bottleneck. This growing mismatch between processor speed and DRAM latency is typically bridged by a hierarchy of successively faster memory devices called caches that rely on locality of data reference to deliver higher memory system performance. In addition to the latency, the net effective bandwidth between DRAM and the processor poses other problems for sustained computation rates.

The overall performance of the memory system is determined by the fraction of the total memory requests that can be satisfied from the cache. Memory system performance is addressed in greater detail in Section 2.2. Parallel platforms typically yield better memory system performance because they provide (i) larger aggregate caches, and (ii) higher aggregate bandwidth to the memory system (both typically linear in the number of processors). Furthermore, the principles that are at the heart of parallel algorithms, namely locality of data reference, also lend themselves to cache-friendly serial algorithms. This argument can be extended to disks where parallel platforms can be used to achieve high aggregate bandwidth to secondary storage. Here, parallel algorithms yield insights into the development of out-of-core computations. Indeed, some of the fastest growing application areas of parallel computing in data servers (database servers, web servers) rely not so much on their high aggregate computation rates but rather on the ability to pump data out at a faster rate.

### 1.1.3 The Data Communication Argument

As the networking infrastructure evolves, the vision of using the Internet as one large heterogeneous parallel/distributed computing environment has begun to take shape. Many applications lend themselves naturally to such computing paradigms. Some of the most impressive applications of massively parallel computing have been in the context of wide-area distributed platforms. The SETI (Search for Extra Terrestrial Intelligence) project utilizes the power of a large number of home computers to analyze electromagnetic signals from outer space. Other such efforts have attempted to factor extremely large integers and to solve large discrete optimization problems.

In many applications there are constraints on the location of data and/or resources across the Internet. An example of such an application is mining of large commercial datasets distributed over a relatively low bandwidth network. In such applications, even if the computing power is available to accomplish the required task without resorting to parallel computing, it is infeasible to collect the data at a central location. In these cases, the motivation for parallelism comes not just from the need for computing resources but also from the infeasibility or undesirability of alternate (centralized) approaches.

## 1.2 Scope of Parallel Computing

Parallel computing has made a tremendous impact on a variety of areas ranging from computational simulations for scientific and engineering applications to commercial applications in data mining and transaction processing. The cost benefits of parallelism coupled with the performance requirements of applications present compelling arguments in favor of parallel computing. We present a small sample of the diverse applications of parallel computing.

### 1.2.1 Applications in Engineering and Design

Parallel computing has traditionally been employed with great success in the design of airfoils (optimizing lift, drag, stability), internal combustion engines (optimizing charge distribution, burn), high-speed circuits (layouts for delays and capacitive and inductive effects), and structures (optimizing structural integrity, design parameters, cost, etc.), among others. More recently, design of microelectromechanical and nanoelectromechanical systems (MEMS and NEMS) has attracted significant attention. While most applications in engineering and design pose problems of multiple spatial and temporal scales and coupled physical phenomena, in the case of MEMS/NEMS design these problems are particularly acute. Here, we often deal with a mix of quantum phenomena, molecular dynamics, and stochastic and continuum models with physical processes such as conduction, convection, radiation, and structural mechanics, all in a single system. This presents formidable challenges for geometric modeling, mathematical modeling, and algorithm development, all in the context of parallel computers.

Other applications in engineering and design focus on optimization of a variety of processes. Parallel computers have been used to solve a variety of discrete and continuous optimization problems. Algorithms such as Simplex, Interior Point Method for linear optimization and Branch-and-bound, and Genetic programming for discrete optimization have been efficiently parallelized and are frequently used.

### 1.2.2 Scientific Applications

The past few years have seen a revolution in high performance scientific computing applications. The sequencing of the human genome by the International Human Genome Sequencing Consortium and Celera, Inc. has opened exciting new frontiers in bioinformatics. Functional and structural characterization of genes and proteins hold the promise of understanding and fundamentally influencing biological processes. Analyzing biological sequences with a view to developing new drugs and cures for diseases and medical conditions requires innovative algorithms as well as large-scale computational power. Indeed, some of the newest parallel computing technologies are targeted specifically towards applications in bioinformatics.

Advances in computational physics and chemistry have focused on understanding processes ranging in scale from quantum phenomena to macromolecular structures. These have resulted in design of new materials, understanding of chemical pathways, and more efficient processes. Applications in astrophysics have explored the evolution of galaxies, thermonuclear processes, and the analysis of extremely large datasets from telescopes. Weather modeling, mineral prospecting, flood prediction, etc., rely heavily on parallel computers and have very significant impact on day-to-day life.

Bioinformatics and astrophysics also present some of the most challenging problems with respect to analyzing extremely large datasets. Protein and gene databases (such as PDB, SwissProt, and ENTREZ and NDB) along with Sky Survey datasets (such as the Sloan Digital Sky Surveys) represent some of the largest scientific datasets. Effectively analyzing these datasets requires tremendous computational power and holds the key to significant scientific discoveries.

### 1.2.3 Commercial Applications

With the widespread use of the web and associated static and dynamic content, there is increasing emphasis on cost-effective servers capable of providing scalable performance. Parallel platforms ranging from multiprocessors to linux clusters are frequently used as web and database servers. For instance, on heavy volume days, large brokerage houses on Wall Street handle hundreds of thousands of simultaneous user sessions and millions of orders. Platforms such as IBMs SP supercomputers and Sun Ultra HPC servers power these business-critical sites. While not highly visible, some of the largest supercomputing networks are housed on Wall Street.

The availability of large-scale transaction data has also sparked considerable interest in data mining and analysis for optimizing business and marketing decisions. The sheer volume and geographically distributed nature of this data require the use of effective parallel algorithms for such problems as association rule mining, clustering, classification, and time-series analysis.

### 1.2.4 Applications in Computer Systems

As computer systems become more pervasive and computation spreads over the network, parallel processing issues become engrained into a variety of applications. In computer security, intrusion detection is an outstanding challenge. In the case of network intrusion detection, data is collected at distributed sites and must be analyzed rapidly for signaling intrusion. The infeasibility of collecting this data at a central location for analysis requires effective parallel and distributed algorithms. In the area of cryptography, some of the most spectacular applications of Internet-based parallel computing have focused on factoring extremely large integers.

Embedded systems increasingly rely on distributed control algorithms for accomplishing a variety of tasks. A modern automobile consists of tens of processors communicating to perform complex tasks for optimizing handling and performance. In such systems, traditional parallel and distributed algorithms for leader selection, maximal independent set, etc., are frequently used.

While parallel computing has traditionally confined itself to platforms with well behaved compute and network elements in which faults and errors do not play a significant role, there are valuable lessons that extend to computations on ad-hoc, mobile, or faulty environments.

## 1.3 Organization and Contents of the Text

This book provides a comprehensive and self-contained exposition of problem solving using parallel computers. Algorithms and metrics focus on practical and portable models of parallel machines. Principles of algorithm design focus on desirable attributes of parallel algorithms and techniques for achieving these in the contest of a large class of applications and architectures. Programming techniques cover standard paradigms such as MPI and POSIX threads that are available across a range of parallel platforms.

Chapters in this book can be grouped into four main parts as illustrated in Figure 1.1. These parts are as follows:

**Fundamentals** This section spans Chapters 2 through 4 of the book. Chapter 2, Parallel Programming Platforms, discusses the physical organization of parallel platforms. It

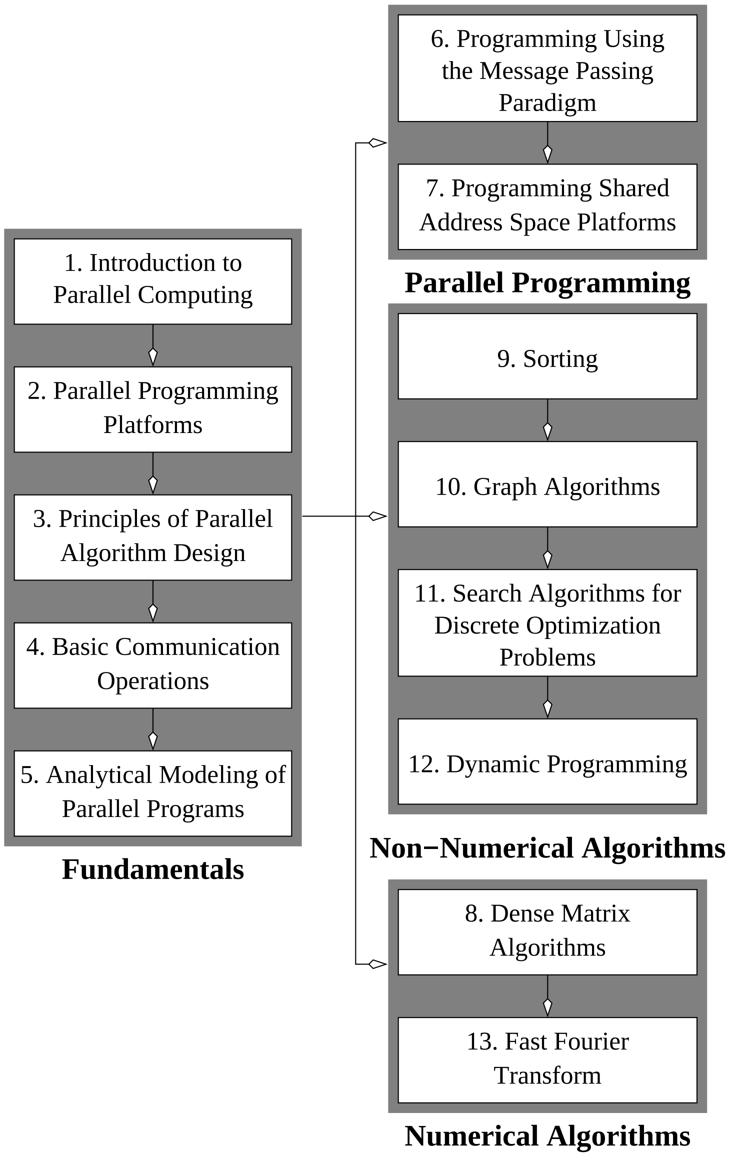
6. Programming Using the Message Passing Paradigm
7. Programming Shared Address Space Platforms
Parallel Programming
1. Introduction to Parallel Computing
2. Parallel Programming Platforms
3. Principles of Parallel Algorithm Design
4. Basic Communication Operations
5. Analytical Modeling of Parallel Programs
Fundamentals
9. Sorting
10. Graph Algorithms
11. Search Algorithms for Discrete Optimization Problems
12. Dynamic Programming
Non−Numerical Algorithms
8. Dense Matrix Algorithms
13. Fast Fourier Transform
Numerical Algorithms

**Figure 1.1** Recommended sequence for reading the chapters.

establishes cost metrics that can be used for algorithm design. The objective of this chapter is not to provide an exhaustive treatment of parallel architectures; rather, it aims to provide sufficient detail required to use these machines efficiently. Chapter 3, Principles of Parallel Algorithm Design, addresses key factors that contribute to efficient parallel algorithms and presents a suite of techniques that can be applied across a wide range of applications. Chapter 4, Basic Communication Operations, presents a core set of operations that are used throughout the book for facilitating efficient data transfer in parallel algorithms. Finally, Chapter 5, Analytical Modeling of Parallel Programs, deals with metrics for quantifying the performance of a parallel algorithm.

**Parallel Programming** This section includes Chapters 6 and 7 of the book. Chapter 6, Programming Using the Message-Passing Paradigm, focuses on the Message Passing Interface (MPI) for programming message passing platforms, including clusters. Chapter 7, Programming Shared Address Space Platforms, deals with programming paradigms such as threads and directive based approaches. Using paradigms such as POSIX threads and OpenMP, it describes various features necessary for programming shared-address-space parallel machines. Both of these chapters illustrate various programming concepts using a variety of examples of parallel programs.

**Non-numerical Algorithms** Chapters 9–12 present parallel non-numerical algorithms. Chapter 9 addresses sorting algorithms such as bitonic sort, bubble sort and its variants, quicksort, sample sort, and shellsort. Chapter 10 describes algorithms for various graph theory problems such as minimum spanning tree, shortest paths, and connected components. Algorithms for sparse graphs are also discussed. Chapter 11 addresses search-based methods such as branch-and-bound and heuristic search for combinatorial problems. Chapter 12 classifies and presents parallel formulations for a variety of dynamic programming algorithms.

**Numerical Algorithms** Chapters 8 and 13 present parallel numerical algorithms. Chapter 8 covers basic operations on dense matrices such as matrix multiplication, matrix-vector multiplication, and Gaussian elimination. This chapter is included before non-numerical algorithms, as the techniques for partitioning and assigning matrices to processors are common to many non-numerical algorithms. Furthermore, matrix-vector and matrix-matrix multiplication algorithms form the kernels of many graph algorithms. Chapter 13 describes algorithms for computing Fast Fourier Transforms.

## 1.4 Bibliographic Remarks

Many books discuss aspects of parallel processing at varying levels of detail. Hardware aspects of parallel computers have been discussed extensively in several textbooks and monographs [CSG98, LW95, HX98, AG94, Fly95, AG94, Sto93, DeC89, HB84, RF89, Sie85, Tab90, Tab91, WF84, Woo86]. A number of texts discuss paradigms and languages for programming parallel computers [LB98, Pac98, GLS99, GSNL98, CDK$^+$00, WA98, And91, BA82, Bab88, Ble90, Con89, CT92, Les93, Per87, Wal91]. Akl [Akl97], Cole [Col89], Gibbons and Rytter [GR90], Foster [Fos95], Leighton [Lei92], Miller and Stout [MS96], and Quinn [Qui94] discuss various aspects of parallel algorithm design and analysis. Buyya (Editor) [Buy99] and Pfister [Pfi98] discuss various aspects of parallel computing using clusters. Jaja [Jaj92] covers parallel algorithms for the PRAM model of computation. Hillis [Hil85, HS86] and Hatcher and Quinn [HQ91] discuss data-parallel programming. Agha [Agh86] discusses a model of concurrent computation based on *actors*. Sharp [Sha85] addresses data-flow computing. Some books provide a general overview of topics in parallel computing [CL93, Fou94, Zom96, JGD87, LER92, Mol93, Qui94]. Many books address parallel processing applications in numerical analysis and scientific computing [DDSV99, FJDS96, GO93, Car89]. Fox et al. [FJL$^+$88] and Angus et al. [AFKW90] provide an application-oriented view of algorithm design for problems in scientific computing. Bertsekas and Tsitsiklis [BT97] discuss parallel algorithms, with emphasis on numerical applications.

Akl and Lyons [AL93] discuss parallel algorithms in computational geometry. Ranka and Sahni [RS90b] and Dew, Earnshaw, and Heywood [DEH89] address parallel algorithms for use in computer vision. Green [Gre91] covers parallel algorithms for graphics applications. Many books address the use of parallel processing in artificial intelligence applications [Gup87, HD89b, KGK90, KKKS94, Kow88, RZ89].

A useful collection of reviews, bibliographies and indexes has been put together by the Association for Computing Machinery [ACM91]. Messina and Murli [MM91] present a collection of papers on various aspects of the application and potential of parallel computing. The scope of parallel processing and various aspects of US government support have also been discussed in National Science Foundation reports [NSF91, GOV99].

A number of conferences address various aspects of parallel computing. A few important ones are the Supercomputing Conference, ACM Symposium on Parallel Algorithms and Architectures, the International Conference on Parallel Processing, the International Parallel and Distributed Processing Symposium, Parallel Computing, and the SIAM Conference on Parallel Processing. Important journals in parallel processing include IEEE Transactions on Parallel and Distributed Systems, International Journal of Parallel Programming, Journal of Parallel and Distributed Computing, Parallel Computing, IEEE Concurrency, and Parallel Processing Letters. These proceedings and journals provide a rich source of information on the state of the art in parallel processing.

## Problems

**1.1** Go to the Top 500 Supercomputers site (`http://www.top500.org/`) and list the five most powerful supercomputers along with their FLOPS rating.

**1.2** List three major problems requiring the use of supercomputing in the following domains:

1. Structural Mechanics.
2. Computational Biology.
3. Commercial Applications.

**1.3** Collect statistics on the number of components in state of the art integrated circuits over the years. Plot the number of components as a function of time and compare the growth rate to that dictated by Moore's law.

**1.4** Repeat the above experiment for the peak FLOPS rate of processors and compare the speed to that inferred from Moore's law.