Chapter 1

Introduction

Jack Dongarra & Ken Kennedy & Andy White (Geoffrey Fox, editor)

Target Length: 20 pages

1.1 Parallel Computing

"Nothing you can't spell will ever work." Will Rogers

Parallel computing is more than just a strategy for achieving high performance—it is a compelling vision for how computation can seamlessly scale from a single processor to virtually limitless computing power. This vision is decades old, but it was not until the late 1980s that it seemed within grasp. However, the road has been a rocky one and, as of the writing of this book, parallel computing cannot be viewed as an unqualified success.

True, parallel computing has made it possible for the peak speeds of high-end supercomputers to grow at a rate that exceeded Moore's law. Unfortunately, the scaling of application performance has not matched the scaling of peak speed, and the programming burden for these machines continues to be heavy. This is particularly problematic because the vision of seamless scalability cannot be achieved without having the applications scale automatically as the number of processors increases. However, for this to happen, the applications have to be programmed to be able to exploit parallelism in the most efficient possible way. Thus the responsibility for achieving the vision of scalable parallelism falls on the application developer.

The Center for Research on Parallel Computation was founded in 1989 with the goal of making parallel programming easy enough so that it would be accessible to ordinary scientists. To do this, the conducted research on software and algorithms that could form the underpinnings of an infrastructure for parallel

programming. The result of much of this research was captured in software systems and published algorithms, so that it could be widely used in the scientific community. However, the published work has never been collected into a single resource and, even if it had been, it would not incorporate the broader work of the parallel computing research community.

This book is an attempt to fill that gap. It represents the collected knowledge of and experience with parallel computing from a broad collection of leading parallel computing researchers, both within and outside of CRPC. It attempts to provide both tutorial material and more detailed documentation of advanced strategies produced by research over the last two decades.

In the remainder of this chapter we will delve more deeply into three key aspects of parallel computation—hardware, applications, and software—to provide a foundation and motivation for the material that will be elaborated later in the book. We begin with a discussion of the progress in parallel computing hardware. This is followed by a discussion of what we have learned from the many application efforts that were focused on exploitation of parallelism. Finally, we will briefly discuss the state of parallel computing software and the prospects for such software in the future. We conclude with a roadmap that tells how the book can be effectively used by a variety of different kinds of readers.

1.2 Parallel Computing Hardware

In last 50 years, the field of scientific computing has seen a rapid change of vendors, architectures, technologies and the usage of systems. Despite all these changes the evolution of performance on a large scale however seems to be a very steady and continuous process. Moore's Law is often cited in this context. If we plot the peak performance of various computers of the last 5 decades in Figure 1.1 which could have been called the 'supercomputers' of the time we indeed see how well this law holds for almost the complete lifespan of modern computing. On average we see an increase in performance of two magnitudes of order every decade.

In the second half of the seventies the introduction of vector computer systems marked the beginning of modern supercomputing. These systems offered a performance advantage of at least one order of magnitude over conventional systems of that time. Raw performance was the main if not the only selling argument. In the first half of the eighties the integration of vector system in conventional computing environments became more important. Only the manufacturers that provided standard programming environments, operating systems and key applications were successful in getting industrial customers and survived. Performance was mainly increased by improved chip technologies and by producing shared memory multi processor systems.

Fostered by several Government programs massive parallel computing with scalable systems using distributed memory got in the focus of interest end of the eighties. Overcoming the hardware scalability limitations of shared memory systems was the main goal. The increase of performance of standard micro pro-

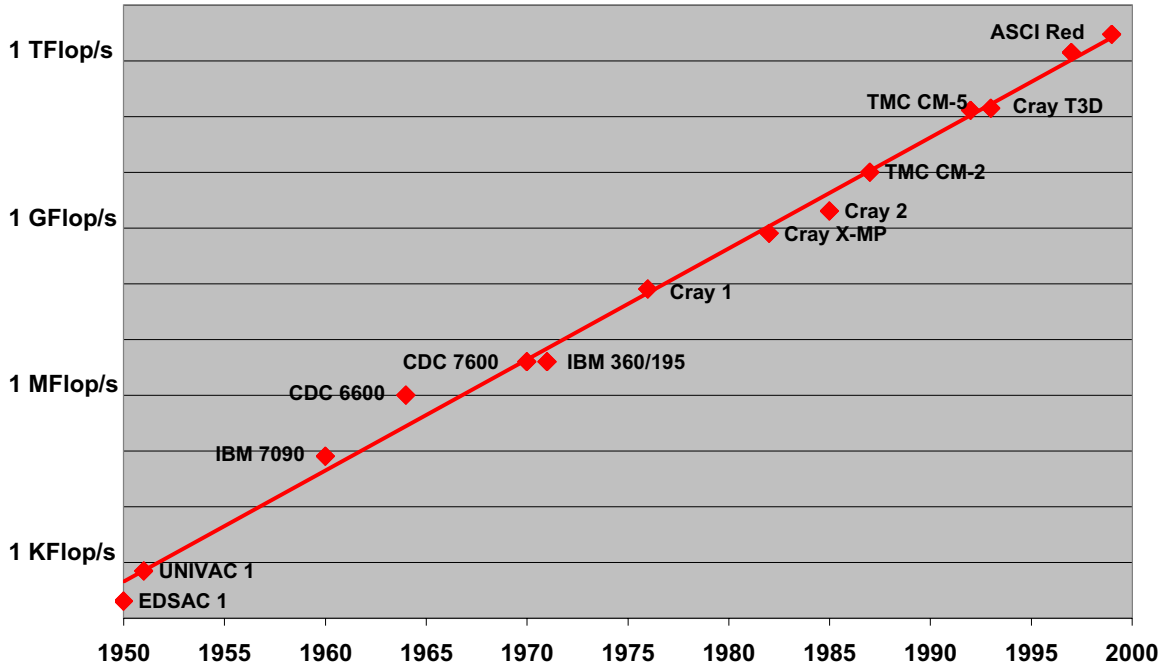

Figure 1.1: **Moores Law and Peak Performance of Various Computers Over Time**

cessors after the RISC revolution together with the cost advantage of large scale productions formed the basis for the "Attack of the Killer Micro". The transition from ECL to CMOS chip technology and the usage of "off the shelf" micro processor instead of custom designed processors for MPPs was the consequence.

Beginning of the nineties while the MP vector systems reached their widest distribution, a new generation of MPP systems came on the market with the claim to be able to substitute of even surpass the vector MPs. To provide a better basis for statistics on high-performance computers, The Top500 [**?**] list was begun. This report lists the sites that have the 500 most powerful computer systems installed. The best Linpack benchmark performance [**?**] achieved is used as a performance measure in ranking the computers. The TOP500 list has been updated twice a year since June 1993. In the first Top500 list in June 1993 there were already 156 MPP and SIMD systems present (31% of the total 500 systems).

The year 1995 saw some remarkable changes in the distribution of the systems in the Top500 for the different types of customer (academic sites, research labs, industrial/commercial users, vendor installations, and confidential sites) Until June 1995, the major trend seen in the Top500 data was a steady decrease of industrial customers, matched by an increase in the number of government-funded research sites. This trend reflects the influence of the different gov-

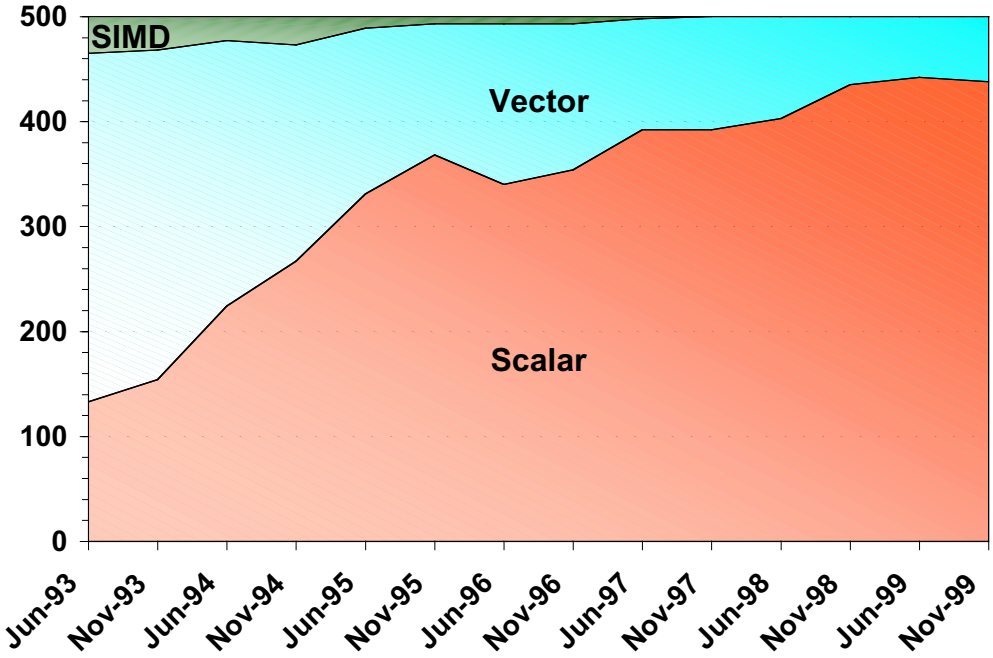

Figure 1.2: **Processor Design Used as Seen in the Top500**

ernmental HPC programs that enabled research sites to buy parallel systems, especially systems with distributed memory. Industry was understandably reluctant to follow this step, since systems with distributed memory have often been far from mature or stable. Hence, industrial customers stayed with their older vector systems, which gradually dropped off the Top500 list because of low performance.

Beginning in 1994, however, companies such as SGI, Digital, and Sun started to sell symmetrical multiprocessor (SMP) models of their major workstation families. From the very beginning, these systems were popular with industrial customers because of the maturity of these architectures and their superior price/performance ratio. At the same time, IBM SP2 systems started to appear at a reasonable number of industrial sites. While the SP initially was sold for numerically intensive applications, the system began selling successfully to a larger market, including database applications, in the second half of 1995.

It is instructive to compare the growth rates of the performance of machines at fixed positions in the Top 500 list with those predicted by Moore's Law. To make this comparison, we separate the influence from the increasing processor performance and from the increasing number of processor per system on the total accumulated performance. (To get meaningful numbers we exclude the SIMD systems for this analysis, as they tend to have extreme high processor numbers and extreme low processor performance.) In Figure 1.3 we plot the

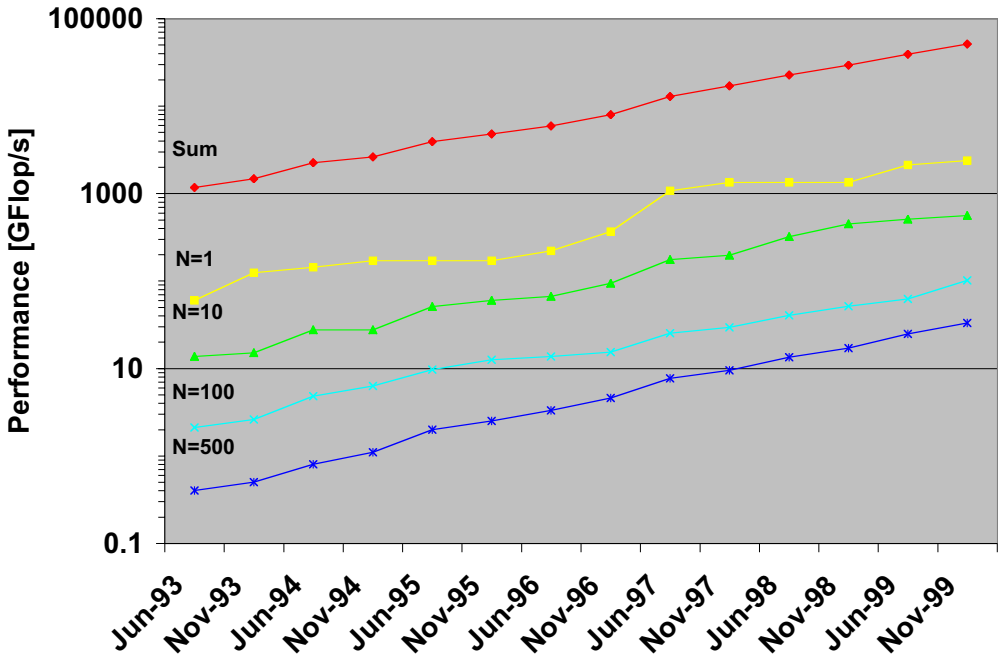

Figure 1.3: **Performance Growth at Fixed Top500 Rankings.**

relative growth of the total processor number and of the average processor performance defined as the quotient of total accumulated performance by the total processor number. We find that these two factors contribute almost equally to the annual total performance growth factor of 1.82. The processor number grows per year on the average by a factor of 1.30 and the processor performance by 1.40 compared 1.58 of Moore's Law.

Based on the current Top500 data which cover the last 6 years and the assumption that the current performance development continue for some time to come we can now extrapolate the observed performance and compare these values with the goals of the mentioned government programs. In figure 1.4 we extrapolate the observed performance values using linear regression on the logarithmic scale. This means that we fit exponential growth to all levels of performance in the Top500. These simple fitting of the data shows surprisingly consistent results. Based on the extrapolation from these fits we can expect to have the first 100 TFlop/s system by 2005 which is about 1–2 years later than the ASCI path forward plans. By 2005 also no system smaller then 1 TFlop/s should be able to make the Top500 any more.

There are two general conclusions we can draw from these figures. First, parallel computing is here to stay. It is the primary mechanism by which computer performance can keep up with the predictions of Moore's law in the face of the increasing influence of performance bottlenecks in conventional processors. Sec-

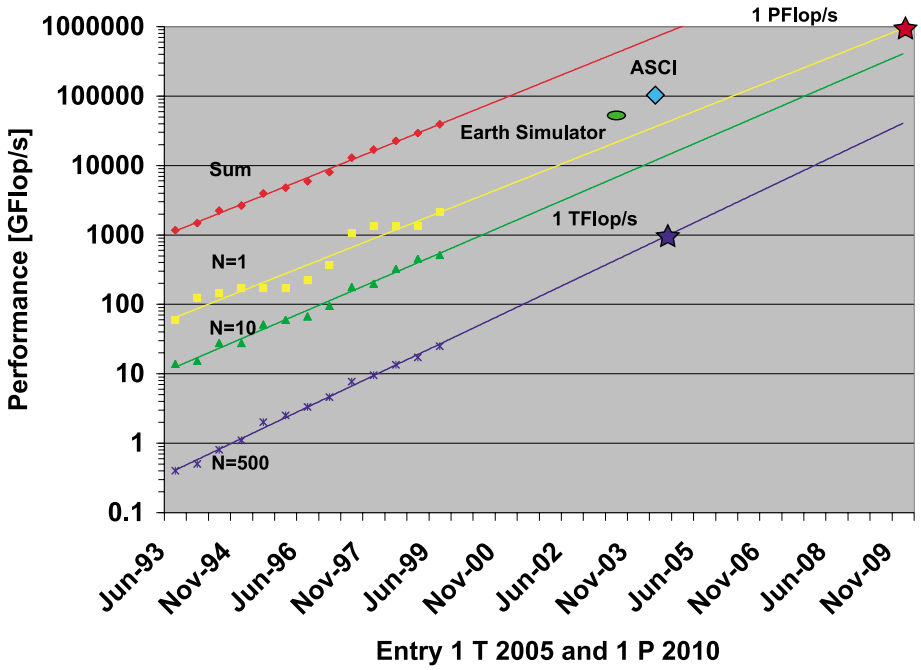

Figure 1.4: **Extrapolation of Top500 Results.**

ond, the architecture of high performance computing will continue to evolve at a rapid rate. Thus, it will be increasingly important to find ways to support scalable parallel programming without sacrificing portability. This challenge must be met by the development of software systems and algorithms that promote portability while easing the burden of program design and implementation.

Looking even further in the future we could speculate that based on the current doubling of performance every year the first Petaflop system should be available around 2009. Due to the rapid changes in the technologies used in HPC systems there is however at this point in time no reasonable projection possible for the architecture of such a system at the end of the next decade. Even as the HPC market has changed it's face quite substantially since the introduction of the Cray 1 three decades ago, there is no end in sight for these rapid cycles of re-definition.

1.3 What Have We Learned from Applications?

Remarkable strides have been taken over the last decade in utilization of high-end, that is parallel, computers. Federal agencies, most notably NSF, DOE, NASA, and DoD have provided increasingly powerful, scalable resources to scientists and engineers across the country. We will discuss a handful of lessons

learned that punctuate the lifetime of the CRPC and provide important context for the next millennium.

Transformation of science and engineering. Scalable, parallel computing has transformed a number of science and engineering disciplines, including cosmology, environmental modeling, condensed matter physics, protein folding, quantum chromodynamics, device and semiconductor simulation, seismology, and turbulence [**?**]. Of these, one of the most striking paradigm shifts has occurred in cosmology – the study of the universe, its evolution and structure [**?**]. A number of new, tremendously detailed observations deep into the universe are available from such instruments as the Hubble Space Telescope and the Digital Sky Survey. However, until recently, it has been difficult, except in relatively simple circumstances, to tease from mathematical theories of the early universe enough information to allow comparison with observations.

However, scalable parallel computers with large memories have changed all of that. Now, cosmologists can simulate the principal physical processes at work in the early universe over space-time volumes sufficiently large to determine the large-scale structures predicted by the models. With such tools, some theories can be discarded as being incompatible with the observations. High performance computing has allowed comparison of theory with observation and, thus, has transformed cosmology into a hard science: "The theories currently being discussed are true scientific theories, capable of verification or falsification." [**?**].

To port or not to port. That is not the question. "Porting" a code to parallel architectures is an opportunity to reformulate the basic code and data structures and, more importantly, to reassess the basic representation of the processes or dynamics involved. In the case of modeling the ocean, standard Bryan-Cox-Semtner (BCS) ocean model was retargeted from Cray vector architecture to the CM-2 and CM-5 [**?**]. The BCS model was inefficient in parallel for two reasons: the primary loop structure needed to be reorganized and global communications were required by the stream-function formulation of the BCS representation. The later feature of the BCS model required that independent line integrals be performed around each island in the calculation. The model was reformulated in surface-pressure form, where the solution of the resulting equations does not require line integrals around islands and is better conditioned than the mathematically equivalent stream-function representation. An additional change to the original model **reference? [Implicit free-surface method for the Bryan-Cox-Semtner ocean model, ???]** relaxed the 'rigid-lid' approximation which suppressed surface-gravity waves (and allowed longer time-steps) in the BCS model.

In addition to a more efficient code on either a vector or parallel architecture, this reformulation brought several remarkable benefits:

1. islands were treated with simple, point-wise boundary conditions, thereby allowing all island features to be included at a particular resolution;

2. unsmoothed bottom topography could be used without adverse effects on convergence; and

3. a free-surface boundary at ocean-air interface made the sea-surface height a prognostic variable and allowed direct comparison with Topex-Poseidon satellite altimeter data.

Satellite data has become a key feature in the verification and validation of global ocean models [**?**].

Answering challenges to society. Computational science has just begun to make an impact on problems with direct human-interest and on systems whose principal actors are not particles and aggregations of particles, but rather are humans and collections of humans. Perhaps the most oft-asked and rarely-answered question about scientific computing concerns predicting the weather. However, there are some things that can be said. Hurricane tracks are being more accurately predicted [**?**], which directly reduces the cost of evacuations and which indirectly reduces loss of life. This increased fidelity is equal parts computation and observation - more accurate and detailed data on hurricane wind-fields is available using dropwindsondes **?** which report not only the meteorological variables of interest, but also an accurate position by using GPS. Another significant development over the last decade has been the Advanced Regional Prediction System [**?**] developed by the NSF Science and Technology Center for the Analysis and Prediction of Storms `[www.caps.out.edu]`.

However, basically this work still concerns modeling of physical systems, in this case severe weather, which have significant impact on society. A more difficult job is effectively modeling society itself, or a piece thereof. For example, the Environmental Protection Agency (EPA) requires detailed environmental impact statements **[reference???]** prior to any significant change in metropolitan transportation systems. In order to meet this regulatory requirement, the Department of Transportation commissioned a development of a transportation model. The result, TRANSIMS, models traffic flow by representing all of the traffic infrastructure (e.g. streets, freeways, lights, stop signs), developing a statistically consistent route plan for the area's population, and then simulating the movement of each car, second by second. The principal distinction here is that we believe that precise mathematical laws exist that accurately characterize the dynamics and interplay of physical systems. No such systems of laws, with the possible exception of Murphy's, is contemplated for human-dominated systems.

Its not the hardware, stupid. The focus has often been on computing hardware. The reasons are straight-forward: they cost a lot of money and take a lot of time to acquire; they have measurable, often mysterious except to the fully initiated, properties; and we wonder how close they are getting to the most famous computer of all, HAL. However, if we contrast the decade of the Crays to the tumultuous days of the MPPs, we realize that it was the consistency

of the programming model, not the intricacies of the hardware, that made the former 'good old' and the latter 'interesting'.

A case in point is seismic processing [**?**]. Schlumberger acquired two, 128-node CM-5s to provide seismic processing services to their customers. They were successful simply because it was possible, in this instance, to write an efficient post-stack migration code for the CM-5, provide commercial quality services to their customers, all within the 2-4 year operational window of any given high-end hardware platform [**check with John Ingram???**]. Those programs or businesses that could not profitably, or possibly, write new applications for each new hardware system were forced to continue in the old ways. However, the Schlumberger experience teaches us an important lesson: a successful high-end computing technology must have a stable, effective programming model which persists over the lifetime of the application. In the case of Stockpile Stewardship, this is on the order of a decade.

In conclusion, applications have taught us much over the last ten years.

1. Entire disciplines can move to firm scientific foundation by using scalable, parallel computing to expand and elucidate mathematical theories, thus allowing comparison with observation and experiment.

2. High-end computing is beginning to make an impact on everyday life, on society by providing more accurate, detailed, and trusted forecasts and predictions, even on human-dominated systems.

3. New approaches to familiar problems, taken in order to access high capacity, large memory parallel computers, can have tremendous ancillary benefits beyond mere restructuring of the computations.

4. A persistent programming model for scalable, parallel computers is absolutely essential if computational science and engineering is to realize even a fraction of its remarkable promise.

5. The increase in the accuracy, detail, and volume of observational data goes hand in hand with these same improvements in the computational arena.

1.4 Software and Algorithms

As we indicated at the beginning of this Chapter, the widespread acceptance of parallel computation has been impeded by the difficulty of the parallel programming task. First, the expression of an explicitly parallel program is difficult—in addition to specifying the computation and how it is to be partitioned among processors, the developer must specify the synchronization and data movement needed to ensure the the program computes the correct answers and achieves high performance.

Second, because the nature of high-end computing systems changes rapidly, it must be possible to express programs in a reasonably machine-independent

way, so that moving to new platforms from old ones is possible with a relatively small amount of effort. In other words, parallel programs should be portable between different architectures. However, this is a difficult ideal to achieve because the price of portability is often performance.

The goal of parallel computing software systems should be to make parallel programming easier and the resulting applications more portable while achieving the highest possible performance. This is clearly a tall order.

A final complicating factor for parallel computing is the complexity of the problems being attacked. This complexity requires extraordinary skill on the part of the application developer along with extraordinary flexibility in the developed applications. Often this means that parallel programs will be developed using multiple programming paradigms and often multiple languages. Interoperability is thus an important consideration in choosing the development language for a particular application component.

The principal goal of the Center for Research on Parallel Computation (CRPC) has been the development of software and algorithms that address programmability, portability, and flexibility of parallel applications. Much of the material in this book is devoted to the explication of technologies developed in CRPC and the community to ameliorate these problems. These technologies include new language standards and language processors, libraries that encapsulate major algorithmic advances, and tools to assist in the formulation and debugging of parallel applications.

In the process of carrying out this research we have learned a number of hard but valuable lessons. These lessons are detailed in the next few paragraphs.

Portability is elusive. When CRPC began, every vendor of parallel systems offered a different application programming interface. This made it extremely difficult for developers of parallel applications because the work of converting an application to a parallel computer would need to be repeated for each new parallel architecture. One of the most important contributions of CRPC was an effort to establish cross-platform standards for parallel programming. The MPI and HPF standards are just two results of this effort.

However, portability is not just a matter of implementing a standard interface. In scientific computing most users are interested in *portable performance*, which means the ability to achieve a high fraction of the performance possible on each machine from the same program image. Because the implementations of standard interfaces were not the same on each platform, portability even for programs written in MPI or HPF was not automatically achieved. Typically the implementor would need to spend significant amounts of time tuning for each new platform.

This tuning burden extended to programming that used portable libraries, such as ScaLAPACK. Here the CRPC approach was to isolate the key performance issues in a few kernels that could be rewritten by hand for each new platform. Still the process was tedious.

Parallelism isn't everything. One of the big surprises on parallel computers was the extent to which poor performance arises because of factors other than insufficient parallelism. The principal problem on scalable machines other than parallelism is data movement. Thus, the optimization of data movement between processors is a critical factor in performance of these machines. If this is not done well, a parallel application is likely to run poorly no matter how powerful the individual processors are. A second and increasingly important issue affecting performance is the bandwidth from main memory on a single processor. Many parallel machines use processors that have so little bandwidth relative to the processor power that the processor cycle time could be dialed down by a factor of two without affecting the running time of most applications. Thus as parallelism levels have increased, algorithms and software have had to increasingly deal with memory hierarchy issues, which are now fundamental to parallel programming.

Algorithms are not always portable. An issue impacting portability is that an algorithm does not always work well on every machine architecture. The differences arise because of number and granularity of processors, connectivity and bandwidth, and the performance of the memory hierarchy on each individual processor. In many cases, portable algorithm libraries must be parameterized to do algorithm selection based on the architecture on which the individual routines are to run. This makes portable programming even more difficult.

Community acceptance is essential to the success of software. Technical excellence alone cannot guarantee that a new software approach will be successful. The scientific community is generally conservative in the sense that they will not risk their effort on software strategies that are likely to fail. To achieve widespread use, there has to be the expectation that a software system will survive the test of time. Standards are an important part of this, but cannot alone guarantee success. A case in point is High Performance Fortran (HPF). In spite of the generally acknowledged value of the idea of distribution-based languages and a CRPC-led standardization effort, HPF failed to achieve the level of acceptance of MPI because the commercial compilers did not mature in time to gain the confidence of the community.

Good commercial software is rare at the high end. Because of the small size of the high-end supercomputing market, commercial software production is difficult to sustain unless it also supports a much broader market for medium-level systems, such as symmetric multiprocessors. OpenMP has succeeded because it targets that market, while HPF was focused on the high end. The most obvious victim of market pressures at the high end are tools—tuners and debuggers—which are usually left until last by the vendors and often abandoned. This has seriously impeded the widespread acceptance of scalable

parallelism and has led to a number of community-based efforts to fill the gap based on open software. Some of these efforts are described in later chapters.

1.5 How to Use This Book

This book is aimed at students and practitioners of technical computing who need to understand both the promise and practice of high performance and parallel computing. It can be used as a resource by both computer science and application researchers. It and the attached web site can be used in computational science and parallel computing education and training. The principal goal of this book is to make it easy for those entering the field of parallel computing with a good background in applications or computational science to understand the technologies available and how to apply them.

Over the last ten years, while CRPC has been at the forefront of research on parallel computation, the field has matured significantly. The authors of the individual chapters each describe the field from their experiences in this time of great change. In the process, they cover key contributions from researchers both within and outside CRPC. The book is aimed at the users of high performance systems whose architectures span the range of small desktop SMP's and PC clusters to the high-end supercomputers costing $100M or more. The book sets context within this scenario and describes the external forces from the Internet to the HPCC Presidential Initiative that have brought this to pass.

For the most part, the book focuses on software technologies and numerical algorithms, along with the large-scale applications enabled by them. In each area, the discussion contains a general discussion of the state of the field followed by detailed descriptions of key technologies or methods. In some cases such as MPI for message passing, this is the dominant approach whereas in others such as the discussion of problem solving environments, the authors choose systems representing key concepts in an emerging area.

The book is organized into four major sections. This first section provides a tutorial introduction to the field of parallel computers and computing. It is followed by detailed sections on parallel applications and of software and algorithm technologies. The final section discusses futures from both a technology and application perspective. There is a related web site with a set of community resources, CRPC papers and links to other sites of interest.

The application section is designed to help new users learn if and how high performance techniques can be applied in their area. It consists of an overview of the process by which one identifies appropriate software and algorithms and the issues involved in implementation. Some twenty vignettes briefly describing successful approaches to use of parallel systems in different areas illustrate these general comments. These have been chosen to cover a broad range of both scientific areas and numerical approaches. This overview material is complemented by three in-depth studies in the areas of computational fluid dynamics, environmental engineering, and astrophysical particle simulations. The applications are cross-referenced to the following sections, which cover in depth the needed

software technologies and algorithms.

The computer technologies section will discuss the progress made on a variety of software technologies, including message passing libraries, run-time libraries for parallel computing, such as class libraries for HPC++, languages like HPF and HPC++, performance analysis and tuning tools such as Pablo, and high-level programming systems. The goal of this section is to provide a survey of progress with hints to the user that will help in selecting the right technology for use in a given application. The software technologies section also treats numerical algorithms and covers parallel numerical algorithms for a variety of problems in science and engineering including linear algebra, continuous and discrete optimization, and simulation. Each chapter will cover a different algorithmic area. The goal here is to serve as a resource for the application developer seeking good algorithms for difficult problems.

The final section of the book is a discussion of important future problems for the high performance science and engineering community, including distributed computing in a grid environment.