The Center for Research on Parallel Computation Perspective on Parallel Computing

1 Introduction

Section Editor: Foster

1.1 CRPC and the National Scene (10 pages)

Author: Kennedy

This chapter will provide a historical perspective on the background for the book, discussing the political and technical themes that pervaded the parallel computing community during the period of the CRPC's lifetime.

At the time of the founding of the CRPC, parallel computing was dominated by the bus-based multiprocessors, although a few pioneering projects, such as the Caltech effort to build and program hypercubes, had begun to experiment with more scalable designs.

In the first few years of the CRPC, distributed-memory computing paradigms dominated the discourse on parallel computing. Among the machines of this era were the hypercubes from Intel and the TMC CM-2 and CM-5. The hallmark of this era was the struggle to develop a programming paradigm and methodology for easy-to-use portable programming. Most of these systems employed manufacturer-specific message passing libraries for programming. An important contribution of the CRPC was the launching of the MPI forum to standardize message passing for such machines. This built on work at Caltech and on PVM, the first defacto standard for portable message passing.

In parallel with the development of programming models was the effort to develop suitable scalable algorithms for all areas of scientific and engineering programming. CRPC's algorithmic endeavors were classified in three groups: linear algebra, numerical optimization, and simulation. This chapter will provide an overview of the CRPC-related efforts.

A third major thrust area was the development and standardization of high-level programming interfaces for scalable machines. Among these were HPF, Concurrent C++, HPC++, and OpenMP. Definition and implementation of such languages, along with the development of tools to support them, became an important part of the CRPC effort.

As the CRPC matured, new computing paradigms began to emerge, which gave rise to new research challenges: hardware DSM systems such as the SGI Origin and HP/Convex Exemplar, clusters of symmetric multiprocessors, and distributed heterogeneous collections of processors. In addition, the incorporation of secondary storage into the memory hierarchy has become a major concern.

In retrospect, it is clear that parallel computation is a rich and diverse area, filled with many complexities. The CRPC has made a number of contributions to programming models and algorithms for application development in the field and its work has been supplemented by extensive work by the entire community. However, as the computing platforms become more complex, much more work on software for such systems will be needed.

2 Overview

Section Editor: Foster

2.1 The Future of High-end Computing Applications (10 pages)

Author: White

Societal-scale Applications. Today, computers have taken over some very small fraction of our decision-making responsibility and assist us in more—anti-lock brakes, whether to wear a raincoat or sun block to work, information access on the Web, and fly-by-wire in aircraft are some common examples. Even so, the scope of the decisions in which computer simulation plays a dominant role remains fairly limited.

However, as society's infrastructures have grown increasingly complex, interdependent and fragile, the need for better informed, more accurate and timely decisions has grown correspondingly critical. The plight in which we find ourselves is succinctly stated in the following excerpt from *The Political Limits to Forecasting*,

> "...it is clear that decision aptitudes are sharply challenged. The range of alternatives is greater. The underlying technical facts are more difficult to comprehend because of their sophistication and specialised jargon, and the consequences of error are more lethal and irreversible. Decision-makers are perplexed by new levels of complexity and hyper-interdependence in our society, accompanied by uncertainty, a heightened pace of social change, and discontinuities in utility of past experience."

The last phrase is the most telling—decisions based on rules-of-thumb responses honed over years of trial and error are becoming less and less likely to yield acceptable solutions. The ability to reliably project alternative courses of action (e.g. global climate scenarios, war fighting strategies, electrical power deregulation policies) into the future in order to assess the impacts and consequences of each course of action may be a key to our prosperity, even survival, in the next millennium.

The problems we face affect lives, security, and well being at, potentially, every level of society. Further, the application of computing technology to the solution of these problems demands predictive modeling and simulation at a fidelity, scale, and tempo that are far beyond our current technological ability. Applications discussed in this chapter will include:

Global environmental security: global climate prediction, with down scaling to regional scale (e.g. agricultural production in mid-west), basin scale (e.g. water resources in the Southwest), and local scale (e.g. natural hazards such as wildfire and floods).

Emerging and re-emerging diseases: management of natural or man-made biological threats. These applications connect global epidemiology to molecular biology.

Infrastructure planning and investment: transportation, energy, health care, and telecommunications are the basic infrastructures upon which society rests. The framework for these applications takes the form of networks or graphs rather than the discretization of continuous phenomena.

Crisis management: training, planning, and response: wildfire, earthquakes, floods, severe weather, volcanoes, etc. provide a special challenge for modeling and simulation because of the time urgency and real-time surveillance requirements.

Attacking problems at the societal level places additional requirements on the technology as well as the fundamental theory upon which these simulations are based. Robust threat identification (e.g. wildfire, infectious disease) will require not only peta(fl)ops computing, but also commensurate surveillance and I/O capabilities. Real-time applications will require extraordinary RAS compared to today's scientific computing systems. Quantification of the uncertainty inherent in these simulations will be of paramount importance, as these simulations may very well be used in life and death situations. Finally, we must understand the decision-making process and embed these tools within it. That is, we must provide the end user, the decision-maker, with information they require and trust, in a format, context, time-scale, and location that they can use in support of their activities.

2.2 Parallel Architectures (20 pages)

Author: Stevens

1. Parallel Computer Architectures
   (a) Overview and Motivation
2. Perennial Concerns for Obtaining High-Performance
   (a) CPU speed
   (b) Memory Bandwidth and Latency

2.3 Computing Technologies (10 pages)

Authors: Foster, Kennedy

This chapter will provide an overview of the computing technologies for parallel systems that will be discussed later in the book. These technologies fall generally into two categories: system software including compilers, and parallel numerical algorithms.

In the software section are included base technologies, such as message passing libraries, run-time libraries for parallel computing such as class libraries for HPC++, languages like HPF and HPC++, tools such as Pablo, high-level programming systems, and problem-solving environments.

Scalable parallel algorithmic research is classified in three groups: linear algebra, numerical optimization, and simulation. This section will provide an overview of these efforts in the CRPC.

Finally, the section will review advanced technologies for new computing paradigms such as DSM, clusters, and distributed heterogeneous grids.

2.4 Numerical Algorithms and Libraries (10 pages)

Authors: Dongarra, Sorensen

Traditional Libraries. The ultimate development of fully mature parallel scalable libraries will necessarily depend on breakthroughs in many other supporting technologies. Development of scalable libraries cannot wait, however, until all of the enabling technologies are in place. The reason is twofold: (1) the need for such libraries for existing and near-term parallel architectures is immediate, and (2) progress in all of the supporting technologies will be critically dependent on feedback from concurrent efforts in library development.

The linear algebra community has long recognized that we needed something to help us in developing our algorithms into software libraries. Several years ago, as a community effort, we put together a *de facto* standard for identifying basic operations required in our algorithms and software. Our hope was that the standard would be implemented on the machines by many manufacturers and that we would then be able to draw on the power of having that implementation in a portable way. We began with those BLAS operations designed for basic matrix computations. Since, on a parallel system, message passing is critical, we have been involved with the development of message passing standards. Both PVM and MPI have helped in the establishment of standards and the promotion of portable software that is critical for software library work.

User Interfaces. As computer architectures and programming paradigms become increasingly complex, it becomes desirable to hide this complexity as much as possible from the end user. The traditional user interface for large, general-purpose mathematical and scientific libraries is to have users write their own programs (usually in Fortran) that call on library routines to solve specific subproblems that arise during the course of the computation. When extended to run on parallel architectures, this approach has only a limited ability to hide the underlying architectural and programming complexity from the user. As we extend the conventional notion of mathematical and scientific libraries to scalable architectures, we must rethink the conventional concept of user interface and devise alternate approaches that are capable of hiding architectural, algorithmic, and data complexity from users.

One possible approach is that of a "problem solving environment," typified by current packages like MATLAB, which would provide an interactive, graphical interface for specifying and solving scientific problems, with both algorithms and data structures hidden from the user because the package itself is responsible for storing and retrieving the problem data in an efficient distributed manner. Such an approach seems especially appropriate in keeping with the trend toward graphical workstations as the primary user access to computing facilities, together with networks of computational resources that include various parallel computers and conventional supercomputers. The ultimate hope would be to provide seamless access to such computational engines that would be invoked selectively for different parts of the user's computation according to whichever machine is most appropriate for a particular subproblem. We envision at least two interfaces for a library in linear algebra. One would be along conventional lines (LAPACK-style) for immediate use in conventional programs that are being ported to novel machines, and the other would be in the form of a problem solving environment (MATLAB-style). The two proposed interface styles are not inconsistent or incompatible: the problem solving environment can be built on top of software that is based on a more conventional interface.

Heterogeneous Networking. Current trends in parallel architectures, high-speed networks, and personal workstations suggest that the computational environment of the future for working scientists will require the seamless integration of heterogeneous systems into a coherent problem-solving environment. Graphical workstations will provide the standard user interface, with a variety of computational engines and data storage devices distributed across a network. The diversity of parallel architectures means that inevitably different computational tasks will be more efficient on some than on others, with no single architecture uniformly superior. Thus, we expect the "problem-solving environment" envisioned above eventually to migrate to a heterogeneous network of workstations, file servers, and parallel computation servers. The various computational tasks required to solve a given problem would automatically and transparently be targeted to the most appropriate computational engine on the network. System resources would be shared among many users, but in a somewhat different manner than conventional timesharing computer systems. We have already made important first steps toward achieving these goals with systems like PVM and MPI, which supply the low-level services necessary to coordinate the use of multiple workstations and other computers for individual jobs. These systems could serve as the foundation for a complete problem-solving environment of the type we envision.

Network computing techniques such as NetSolve offer the ability to look for computational resources on a network for a submitted problem (which can be a single LAPACK, ScaLAPACK, or Matlab function call), choose the best one available, solve the problem (with retry for fault tolerance), and return the answer to the user. This system is available for Fortran, C, and Matlab users.

Software Tools and Standards. An ambitious development effort in scalable libraries will require a great deal of supporting infrastructure. Moreover, the portability of any library is critically dependent on adherence to standards. In the case of software for parallel architectures, precious few standards exist, so new standards must evolve along with the research and development. A particularly important area for scalable distributed-memory architectures is internode communication. The BLAS have proven to be very effective in assisting portable, efficient software for sequential computers and some of the current class of high-performance computers. We are investigating the possibility of expanding the set of standards that have been developed. There is a need for a light-weight interface to much of the functionality of traditional BLAS. In addition, iterative and sparse direct methods require functionality not in traditional BLAS. Numerical methods for dense matrices on parallel computers require high efficiency kernels that provide functionality similar to that in traditional BLAS on sequential machines.

Software tools are also of great importance, both for developers to use in designing and tuning the library software, and for end-users to monitor the efficiency of their applications.

Conclusions.

1. In spite of a lack of enabling technologies, library development cannot wait for research in programming languages, compilers, software tools, and other areas to mature, but must be done in conjunction with work in these areas. The the time to begin is now.

2. The user–library interface needs rethinking. It is not clear that the conventional library interface will be adequate to hide the underlying complexity from the user.

3. Object-oriented programming will be required to develop portable libraries that allow the user to work at an appropriate conceptual level.

4. Work on algorithms, particularly linear algebra, is important and cannot be isolated from general library development.

5. Language standards are important. The lack of language standards is the most significant obstacle to the development of communication libraries. A language standard must emerge before a software tool "development sweep" can begin.

These are some of the major research issues in developing scalable parallel libraries.

3 Applications

Section Editor: Fox

This section of the book contains 5 chapters. Chapter 1 is a general discussion; The second chapter is a set of about 20 short case studies and chapters 3 to 5 are each long case studies. One goal is that a reader should be able to take their favorite application and find a "near-match" somewhere in the five chapters, and that will help them to start parallel computing.

3.1 General Application Issues (20 Pages)

Author: Fox

The first application chapter contains an introduction to the other four chapters followed by a discussion of general strategies that have been found helpful in parallelizing applications. We will describe an application as a general set of linked entities (a.k.a. a complex system) and initially contrast artificial systems such as financial instruments with physical simulations. In the latter case, we contrast microscopic or macroscopic entities and discuss how the different states of matter (fields, classical particle, or quantum mechanical) lead to different numerical challenges. We note that some characteristics, such as multiple physical scales and phase transitions, are pervasive. We describe broad issues in applications and algorithms including partial differential equations, particle dynamics, circuits, ordinary differential equations, Monte Carlo, domain decomposition, pleasingly parallel, metaproblems, heuristic algorithms, data analysis, preconditioning, synchronous, loosely synchronous, regular, and irregular. We will add to this list and organize it. We discuss differences between illustrative and real applications by contrasting Laplace's equation with Navier Stokes and complex physics in climate simulations. We will discuss typical problem sizes and why tera- and peta-(fl)op machines are relevant. Typical issues governing performance are discussed. We will philosophize as to whether computational science is a science or an art by illustrating how much experimentation is needed to find reliable numerical methods and how different approaches are in seemingly similar Application areas. These general remarks should tie to the discussion in following four chapters which will be summarized in a suitable set of tables. Note that the discussion of basic numerical methods (PDE, ODE, Monte Carlo etc.) could go to § 5.

3.2 List of Application Overviews: (about 1.5 to 2 pages each – Total about 40 pages)

Author: Fox

1. Black holes (Matzner, Fox) - *start*
2. Astrophysics (Salmon) - *start*
3. Earthquakes (Rundle, Fox) - *start*
4. Climate - (LANL - White, Malone) - *start*
5. Computational Chemistry - (Kuppermann, Goddard, McKoy, PNL) - *start*
6. QCD (Fox, Rajan, Ceperley) - *start*
7. Accelerators (LANL - Ryne)
8. Plasma Physics (Reynders) - *start*
9. MDO (Fox)
10. Financial modeling (Fox)
11. Weather (CAPS, NASA)
12. Computational Biology (Keck Center, Rice) - *start*
13. Astronomy (Prince)

14. Scheduling (Bixby) - *start*
15. Materials (Holian, Lomdahl, Goddard)
16. Combustion (Butler, LANL - Colella)
17. Networks (LANL)
18. Structural, solid mechanics (DOD, Ortiz)
19. Forces modeling (Fox, CACR)
20. CFD (Meiron, Keller) - *start*
21. Energy and environment (Wheeler) - *start*
22. Computational Electromagnetics (DoD Modernization)
23. Signal Processing (DoD Modernization)
24. Electrical Transmission Lines (Fox)

Designation as *"start"* implies that one starts with this subset to give exemplars that can be used to show others how to represent their field. Note that some of those areas are quite broad and could generate several distinct summaries. For instance, "Computational Chemistry" could generate separate overviews corresponding to applications typified by Charmm, Gaussian, and Mopack.

Template for each application overview:

1. Application overview and summary—field discussion
2. Focused case study—what was parallelized, technology discussion and results
3. References and resources
4. Computational issues including algorithms, software, and comments on performance needs and hardware dependencies
5. What has been done and what needs to be done

3.3 Parallel Computing in CFD (20 pages)

Authors: Meiron, Keller

The basic equations of fluid mechanics are presented. A brief overview is provided of some of the common physical regimes described by these equations (e.g. steady vs. unsteady flow, compressible vs. incompressible flow, inviscid vs. viscous flow) and the associated dimensionless parameters associated with these physical regimes (e.g. Reynolds number, Mach number, etc.). The need to utilize high performance computation to solve these equations in many cases of interest is motivated via some examples of applications.

The particular computational difficulties associated with incompressible viscous and inviscid CFD are described and some examples of the application of high performance parallel computation are provided for very simple geometries. For complex geometries that are of practical interest, special attention is paid to the application of the spectral element method and its parallel implementation.

A variety of new techniques have been devised and employed to study two specific viscous incompressible flows. The first is Taylor-Couette flow, which has been studied theoretically, computationally, and experimentally for many years; it is the flow between two rotating coaxial cylinders. The second is Kolmogorov flow, devised by him to study the onset of turbulence. This is a model flow in a periodic box with a periodic body force in one coordinate direction. Our calculations use a three-dimensional box and reveal a huge variety of bifurcations. We have also been able to compute the spiral flow observed by Taylor and Coles in their experiments. In both of these flows we have been able to employ many of the methods discussed in the chapter on "Continuation and Bifurcation in Scientific Computation" (§ 5.5). But, we have also devised computational techniques based on a new theory of Differential Algebraic Equations, which is applicable to the spatially discretized Navier-Stokes equations. We shall describe these methods and indicate how they were employed with the aid of the Recursive Projection Method. Concurrent computing was fundamental in being able to solve these huge three-dimensional steady and unsteady flow problems.

A brief overview is presented of approaches to the numerical simulation of compressible CFD. It is argued that the need to resolve fine-scale features such as shock waves makes the use of adaptive mesh refinement essential especially in three dimensions. The difficulty of establishing load balancing and scalability for such calculations is discussed. The chapter concludes with a brief discussion of some future computational challenges for CFD and an assessment of the computational resources required to overcome these challenges.

3.4 Parallel Computing in Environment and Energy (20 pages)

Author: Wheeler

We describe the relevance of and computational issues found in real-world problems in the environment and energy fields. These include simulation of the circulation patterns in bays and estuaries, groundwater contamination, reservoir management, remediation of polluted soils and aquifers, as well as estimating the productivity of oil and gas wells diagenesis and related geological processes. These are posed numerically as large sets of coupled differential equations and require parallel computing to be able to achieve results that are meaningful for either industry or societal applications.

The general approach requires a multicomponent, multiphase flow and transport simulator. Efficient results need domain decomposition, whose use on parallel machines is described in detail and compared with multigrid. Other important algorithmic issues discussed are different time-stepping methods, linear and non-linear solution techniques, and the need for mixed methods (finite element and cell-centered finite difference methods) for diffusive processes and velocity computations. We note that locally adaptive, nonmatching multiblock logically rectangular grids are able to resolve geologic features such as pinch-outs and faults.

We give results using codes developed as part of the CRPC and other activities. These include UTCHEM and UTCOMP, which have been efficiently parallelized from sequential flood simulation codes developed at the University of Texas. The Parallel Subsurface Simulator Parssim has been developed in our group embodying our experience on algorithms and parallel systems.We discuss the parallel speedups associated with various domain decomposition and multigrid techniques. One needs to carefully address load balancing, especially for locally intensive computations such as those arising from wells, phase behavior, chemical reactions, and general geometry.

3.5 Computational Cosmology (20 pages)

Author: Salmon

A very brief review of the problem domain emphasizes the different length scales and relevant physics. Length scales encompass stars, galaxies, clusters, filaments, sheets, and voids. Observations provide redshifts, ages, masses, and correlations in position and velocity. The "missing mass" implies a dominant dark matter component. Theory allows for a density parameter, a cosmological constant and various flavors of dark matter in addition to "normal" baryonic matter. At the largest scales, Newtonian gravity dominates and numerical methods that follow gravitating particles are good models of the real Universe. Analytic approaches are hampered by the nonlinearity, positive feedback and negative heat-capacity of gravitating systems.

Supercomputers have played a significant role for at least the last 30 years in understanding the large scale structure of the Universe. Computational methods include Poisson solvers, $O(N^2)$, and "Fast" (O(NlgN) and O(N)) methods. Parallel implementations exist for each of these algorithmic approaches. Poisson solvers can use FFT methods or multi-grid (both of which exist in parallel). $O(N^2)$ methods have perhaps the best parallel efficiency of any non-trivial parallel application (Tcomm/Tcalc is proportional to P/N). At least two approaches to Fast methods have been shown to be workable. Orthogonal Recursive Bisection with explicit assembly of "locally essential" data works when the locally essential data is predictable in advance. It has difficulty supporting improved sequential algorithms with dynamic "acceptability criteria", however. A new approach based on space-filling curves is required. It has the additional benefit of being "cache friendly". It also naturally supports both out-of-core and parallel computations, which is important because one consequence of "Fast"ness is that one tends to run out of memory before one runs out of patience.

Computation has allowed us to understand the implications of different scenarios and parameter sets, including the effects of the input power spectrum of fluctuations, the evolution of non-linear clustering and correlation functions, and the effects of the density parameter and the cosmological constant on observational data. Computations continue to be challenged by ever improving observations, but they are important tools for understanding and effectively rule out many scenarios by demonstrating them to be inconsistent with observation.

Similar numerical methods (indeed the first "Fast" method) apply to potential and scattering problems for the Poisson, Helmholz, and Maxwell equations. Other application areas include electrostatic interactions in chemical dynamics, stress-strain interactions in solid mechanics/geophysics and vortex dynamics from incompressible CFD.

4 Computing Technologies

Section Editor: Kennedy

4.1 Base Technologies (20 pages)

Authors: Kesselman, Foster

This chapter will cover the basic software technologies that are used to support the design and implementation of parallel programs. We will focus on the middle layer of abstractions that sit between the low-level mechanisms provided by the underlying parallel hardware, covered in previous chapters, and the high-level abstractions provided by application-specific libraries and tools, which are discussed in the following chapters.

We will conclude with an overview of the technology that is emerging to support the Grid.

Programming Models. This section is an overview of the parallel programming models which the basic technologies must support. This will include SIMD, SPMD, BSP, pipelines, object parallelism, general task parallelism, message passing, and shared memory.

Low-Level Tools. This section will discuss low-level infrastructure that has been developed to support very high performance applications. Focus is on low-latency and high-bandwidth. Tools covered will include the Reactive Kernal, Active Messages, and Fast Messages. Discussion will also cover one-sided communication primitives, such as the SHMEM primitives provided on the Cray T3E.

Message Passing. This section provides an overview of message passing libraries, with a focus on PVM and MPI.

Shared Memory Tools. This section is an overview of support for programming in the shared memory model. It will cover two main topics: control primitives and memory models.

The discussion of control primitives will consider how threads of control are created and how they interact with one another. Systems to be covered will include the POSIX threads model, SGI sproc and arenas, and Nexus.

The memory model discussion will review the various consistency models that have been proposed and how those models have been supported, with a focus on the distributed shared memory systems.

Task Parallel Systems. This section will cover the infrastructure that has been developed to support explicitly task parallel applications. Topics covered will include Nexus and parallel object-oriented systems.

Parallel I/O. In this section, we will provide an overview of the various systems that have been proposed to support input and output in parallel systems. The discussion will provide a historical review and cover commercial and research systems, including PIOFS, PPIO, Passion, etc. This section will also include a discussion of HPSS, and MPI-IO.

This section will include a brief discussion of checkpointing libraries, as a special case of parallel I/O.

Resource Management. In this section, we will provide an overview of how resources in parallel systems have been managed. We will provide a brief overview of queuing systems, and then focus on the use of space-sharing and time-sharing resource models to parallel machines.

Grid Infrastructure. We conclude the chapter with a discussion of emerging Grid technologies, with the focus on integrating high-performance resources into a distributed environment. Topics covered will include Globus, CORBA, and Legion.

4.2 Libraries (20 pages)

Authors: Reynders, Gannon

In the chapter on "Numerical Algorithms and Libraries," Dongarra and Sorensen described the challenges and progress that has been made on scalable, parallel numerical libraries. In this chapter we approach the design of libraries for parallel, scientific computation from a different, but complementary perspective. These libraries are based on object-oriented and generic programming, and are strongly related to the design of application-specific programming languages.

Object-oriented programming taught us the power of abstraction through encapsulation of data and function. By learning to organize software in ways that exploit inheritance and polymorphism, we gain in our ability to maintain and reuse important code. This is because we can separate the interface of an object from its implementation. It also gives us a more powerful tool for factoring a computation into multiple levels of functionality. Generic programming takes us in a seemingly different direction. It encourages us to think about algorithms in a manner that is independent of the data structure we use to represent information. In scientific computing, this concept is reflected in the numerical templates approach where one is shown the "generic" design of an algorithm, such as conjugate gradient, in a manner that is independent of the implementation of the matrices and vectors. It is also reflected in the work of Chandy on archetypes. But it was the work of Alexander Stepanov that pioneered the use of generic programming in C++ with the introduction of the standard template library (STL). And it was the work of Todd Veldhuizen that showed us how to use the C++ template library to write "template expressions" that could run as fast as optimized Fortran.

The libraries described here borrow from both object-oriented and generic programming principles and evolved over the life of the CRPC to be used in a number of applications. POOMA (Parallel Object-Oriented Methods and Applications) is an object-oriented framework for applications in computational science requiring high-performance parallel computers. It is a library of C++ classes designed to represent common abstractions in field simulation and other applications. POOMA provides a data-parallel programming model, but it runs on both distributed memory and shared memory multiprocessors through multi-threaded execution. POOMA hides the details of parallel computation in a flexible "Evaluator" architecture. For the user, this means that a program can be written in a highly-abstract data-parallel form, tested and debugged in serial mode, and then run in parallel with very little effort.

The other library we describe is called HPC++. HPC++ was originally a programming languages project funded by DARPA to design a parallel programming extension to C++. However, as the C++ language evolved with the introduction of templates, it was realized that many of the HPC++ goals could be achieved by an approach similar to that of POOMA. That is, provide core functionality as generic parallel algorithms expressed as C++ templates. This included a parallel version of the STL. In addition, HPC++ became heavily influenced by the design of Compositional C++ and Java, so templated class libraries were added to the language to support CC++ and Java style programming.

In this chapter, we describe the core feature of the POOMA architecture and HPC++ in its current version, and describe the way in which each are in use today. The chapter concludes with a discussion of the limitations of this approach and some research problems that must be solved in order to move this approach to the next level of scalable parallel performance.

4.3 Languages (20 pages)

Authors: Kennedy, Chandy

This chapter will discuss the progress made on languages and compilers for high performance computing systems during the lifetime of the CRPC, with a special emphasis on work sponsored by the CRPC. The principal foci of this discussion will be on Fortran and C++, with some discussion of Java and compositional languages such as PCN and Strand.

In the Fortran section, topics will include:

- Automatic Parallelization. Continued progress on the parallelization of plain Fortran applications, especially for symmetric multiprocessors.
- Data Parallel Languages. This will focus primarily on High Performance Fortran and other distribution-based languages.
- Task Parallel Languages. This will discuss a number of strategies for representing task parallelism in Fortran, including pthreads and OpenMP, a derivative of the original PCF Fortran, and a discussion of Fortran M (Ian Foster).

The discussion will conclude with an assessment of the impact of these models on practice and a discussion of which models work well in various cases.

In the C++ arena, the principal focus will be on Compositional C++ and HPC++. The latter emphasis will concentrate on the part of the language efforts which were not based on libraries (which will be expanded in § 4.2. The topics will include:

- Concurrent Objects: From C++ to CC++ (Carl Kesselman, from Carl's paper in book)
- Data-Parallel Objects and Extensions: HPC++ (Dennis Gannon)
- Compositional Concurrent Languages (Ian Foster from Ian's ACM paper)
- Java for Distributed Scientific Applications: Issues (Geoffrey Fox, Mani Chandy)

The goal of this chapter is to provide a survey of progress with hints to the user that will help in selecting the right high-level programming model for a given application.

4.4 Programming Environments (20 pages)

Authors: Dongarra, Fox

Parallel computers or more generally distributed resources will be little used if they are not easily accessible to ordinary users. In this chapter, we examine how the complexities of programming can be reduced through the use of application-specific tools and toolkits comprising what is commonly called a Problem Solving Environment (PSE). These toolkits enable a programmer to specify their problems at a high level, frequently using abstractions tailored to a specific application domain. The details of mapping this high-level description onto back end compute resources are left to the application tool. For example, issues of resource discovery (both hardware and software), problem decomposition, scheduling, application code locate, etc. can all be managed by the application-specific tool without user intervention.

This chapter is divided into three parts. First we describe general issues in PSEs—discussing two types of activities, building the components from which many different PSEs can be constructed, and using these components in particular application domains. Then we illustrate these general ideas by two particular examples, Netsolve and WebFlow, which have been successfully applied in several application areas.

We note that the remote resources harnessed by network-enabled application-specific toolkits are just as likely to be software as hardware. Numerical libraries, software development systems, and problem solving systems have become increasingly sophisticated. Users normally do not know where these resources are and, once located, they are tedious to obtain and/or use. Hence, techniques are required for identifying and locating appropriate software, delivering that software to the user, identifying an appropriate compute server, and testing and evaluating software. This is one example of the many services a PSE must offer. We discuss these in terms of a middleware of networked servers accessed directly or via agents, as in NetSolve from the client. Some of the ideas of current distributed object technology (such as CORBA and RMI) are relevant and we illustrate this in the discussion of the WebFlow architecture.

4.5 Tools (20 pages)

Authors: Reed, Aydt

As parallel applications become more complex, they grow more irregular, with data-dependent execution behavior, and more dynamic, with time-varying resource demands. Consequently, even small changes in application structure can lead to large changes in observed performance. This performance sensitivity is a direct consequence of resource interaction complexity and growing hardware complexity (e.g., deep memory hierarchies, and complex communication networks). To support creation of high-performance applications, we believe one must tightly integrate compilers, languages, libraries, algorithms, problem-solving environments, runtime systems, schedulers, and performance tools. With this deep integration, one can measure, analyze, visualize, and tune all aspects of application code, compilation strategies, and resource management.

5 Numerical Algorithms and Libraries

Section Editor: Dongarra

5.1 Templates and Linear Algebra (20 pages)

Authors: Dongarra, Sorensen

The increasing availability of advanced-architecture computers has had a significant effect on all spheres of scientific computation, including algorithm research and software development in numerical linear algebra. Linear algebra—in particular, the solution of linear systems of equations—lies at the heart of most calculations in scientific computing. This chapter discusses some of the recent developments in linear algebra designed to exploit these advanced-architecture computers. We discuss two broad classes of algorithms: those for dense matrices and those for sparse matrices. A matrix is called sparse if it has a substantial number of zero elements, making specialized storage and algorithms necessary.

Much of the work in developing linear algebra software for advanced-architecture computers is motivated by the need to solve large problems on the fastest computers available. In this chapter, we focus on four basic issues: (1) the motivation for the work; (2) the development of standards for use in linear algebra and the building blocks for libraries; (3) aspects of algorithm design and parallel implementation; and (4) future directions for research.

As representative examples of dense matrix routines, we will consider the Cholesky and LU factorizations. These factorization routines will be used to highlight the most important factors that must be considered in designing linear algebra software for advanced-architecture computers. We use these factorization routines for illustrative purposes not only because they are relatively simple, but also because of their importance in several scientific and engineering applications that make use of boundary element methods. These applications include electromagnetic scattering and computational fluid dynamics problems.

For the past 15 years or so, there has been a great deal of activity in the area of algorithms and software for solving linear algebra problems. The goal of achieving high performance on codes that are portable across platforms has largely been realized by the identification of linear algebra kernels, the Basic Linear Algebra Subprograms (BLAS). We will discuss the EISPACK, LINPACK, LAPACK, and ScaLAPACK libraries which are expressed in successive levels of the BLAS.

The key insight of our approach to designing linear algebra algorithms for advanced architecture computers is that the frequency with which data are moved between different levels of the memory hierarchy must be minimized in order to attain high performance. Thus, our main algorithmic approach for exploiting both vectorization and parallelism in our implementations is the use of block-partitioned algorithms, particularly in conjunction with highly-tuned kernels for performing matrix-vector and matrix-matrix operations (the Level 2 and 3 BLAS). In general, the use of block-partitioned algorithms requires data to be moved as blocks, rather than as vectors or scalars, so that although the total amount of data moved is unchanged, the latency (or startup cost) associated with the movement is greatly reduced because fewer messages are needed to move the data.

A second key idea is that the performance of an algorithm can be tuned by a user by varying the parameters that specify the data layout. On shared memory machines, this is controlled by block size; on distributed memory machines it is controlled by block size and the configuration of the logical process mesh.

The sparse linear systems that result from partial differential equations need very different techniques from those used for dense matrices. While direct methods have the virtue of reliability, they also take copious amounts of space and time. Iterative methods, of one type or another, are considerably more frugal in their space demands, but on difficult problems their convergence may be slow, and is not even guaranteed.

5.2 Parallel Continuous Optimization (20 pages)

Authors: Dennis, Wu

Abstract. We discuss parallel computation methods for continuous optimization and their application in science and engineering. We describe optimization problems and algorithms and their associated parallelism at different computational levels. In particular, we review parallel methods for local and global optimization, and compare strategies for large, sparse versus small-but-expensive problems. We introduce special techniques for parallel optimization including direct search, domain decomposition, and variable and constraint distribution. We survey application areas where parallel optimization is key to the solution of the problems. We conclude with comments and suggestions for future research directions.

Introduction. Optimization has broad applications in engineering, science, and management. Many of these applications either have large numbers of variables or require expensive function evaluations. In some cases, there are also many local minimizers while a global or nearly global minimum is demanded. As a result, the optimization problems arising in these applications require intensive computation which traditional architectures often cannot afford. On the other hand, parallel high-performance computing has provided unique powerful tools for solving these problems to significant degrees of difficulty, which would otherwise be impossible.

Example applications where parallel optimization plays an important role include aircraft shape design (Cramer, Dennis), macromolecular modeling (Coleman, Schnabel, Moré and Wu), and airline crew scheduling (Schneider and Wise). In aircraft shape design, certain design features such as the pressure distribution need to be optimized with respect to the shape variables. The number of shape variables is in the order of hundreds, but they are constrained by two systems of PDEs. In order to obtain a feasible solution, the two systems need to be solved alternatively to their equilibrium, while both systems require expensive PDE solves for millions of grid points, one for the air flow and the other for the structural change. Clearly, the problem is computationally very intensive, and the performance of the optimization procedure will be improved dramatically if parallel computation is employed. In macromolecular modeling, molecular structure is determined by minimizing a given potential energy function. One of the most important applications is the determination of protein structures in structural molecular biology. The challenge for solving this problem is that the potential energy function has many local minimizers, while the structure to be determined corresponds to a global or nearly global optimal solution to the minimization problem. Global optimization algorithms have been developed to solve the problem. However, the algorithms require substantial computing resources, which only parallel high-performance architectures can provide. The final example for airline crew scheduling is where parallel linear programming becomes necessary. Applications in this area typically have hundreds of thousands of variables and thousands of constraints. Linear programming is employed to obtain an approximate solution to the scheduling problem, which is already very costly in terms of computing resources. Efforts have been made to implement various linear programming codes on vector and shared-memory machines to speed up the time consuming scheduling routine in airline industry.

Substantial research efforts on parallel optimization have been undertaken in the past ten years, some focusing on special applications and some exploring more general parallel schemes. Optimization has close relationships with numerical linear algebra and partial differential equations. For example, a typical optimization procedure requires solving a linear system to obtain a search direction in every iteration; function or constraint evaluation often requires a solution for a partial differential equation. Development of parallel optimization algorithms and software certainly benefits from great advances in parallel numerical linear algebra and partial differential equations. However, there are also structures specific to optimization that can be exploited for design of parallel optimization algorithms and software. Work along this line includes parallel function evaluation (Averrick and Moré), parallel gradient and Hessian estimate (Byrd and Schnabel), parallel multiple line search (Nash and Sofer), parallel inexact Newton step computation (Nash and Sofer), etc. General algorithms have also been developed such as parallel direct search methods by Dennis and Torczon, domain decomposition methods for parallel parameter identification by Dennis, Li, and Williamson, and variable and constraint distribution schemes by Ferris and Mangasarian. An even more active research area is parallel global optimization, which has extensive applications in chemical and biological disciplines such as cluster simulation and protein modeling. Algorithms and software developed in recent years include parallel stochastic global optimization algorithms for molecular conformation and protein folding by Byrd and Schnabel, parallel global continuation software DGSOL for protein structure determination with NMR distance data by Moré and Wu, and parallel effective energy simulated annealing for protein potential energy minimization by Coleman, Shalloway, and Wu.

Optimization problems have many different forms depending on applications. They can be linear or nonlinear, constrained or unconstrained, and local or global. For those concerned with parallel computation, they can also be either large, sparse or small but very expensive. For different types of optimization problems or applications, different parallel algorithms may be required, even with or without using existing parallel linear algebra and PDE software. For some situations, the choice of architecture is also important aspect to achieve high parallel performance. For example, if the problem is large but sparse, in order to exploit the sparsity, a shared-memory system may be a better choice, for otherwise the distribution of a sparse, irregular structure over multiprocessors may cause load imbalance and severe communication overheads. On the other hand, most of global optimization algorithms are coarsely parallel. They can be implemented on distributed-memory architectures or even loosely connected networks of workstations, and still maintain scalability.

In this chapter, we will discuss various parallel optimization methods in greater detail. We describe optimization problems and algorithms and their associated parallelism at different computational levels: function evaluation, algebraic calculation, and optimization. In particular, we review parallel methods for local and global optimization, and compare strategies for large, sparse versus small but expensive problems. Parallel techniques including parallel direct search, domain decomposition, and variable and constraint distribution are introduced. Application areas where parallel optimization is critical to the solution of the problems are surveyed. Comments and suggestions for future research development are given.

5.3 The Traveling Salesman Problem: A Case Study in Parallel Discrete Optimization (20 pages)

Authors: Applegate, Bixby, Chvatal, & Cook

Abstract. The traveling salesman problem (TSP) consists of finding a cheapest way to visit each of a finite number of cities and return to the starting point. We will use the TSP to discuss the application of parallel computing techniques in discrete optimization.

Estimated Length. 10 to 20 pages.

Outline.

1. Introduction

   TSP, Mixed Integer Programming, Parallel Computation

2. Heuristics

   Local Search (Lin-Kernighan), Subproblems (Rohe), Branchwidth, Numerical Results (linkern and branchwidth)

3. Cutting-plane Methods

   Small Polytopes (Christof and Reinelt), ABCC's "subtour" code, ABCC's "concorde" code, Vehicle Routing (Jennifer Rich, Ted Ralphs)

4. Branch and Bound

   Combinatorial and Held-Karp (German group, CMU group), Juenger's code, Networks of Desktops ("subtour"), Concorde and Parallel Cutpool, Jen's VRP work with Treadmarks

5.4 Automatic Differentiation (20 pages)

Author: Carle

The goal of scientific computing is the development of efficient computer simulations that accurately predict complex physical and non-physical phenomenon. Derivatives play key roles in the development and subsequent use of simulations:

- Derivatives are used in the solution of *inverse problems* to calibrate the initial state of a computer model to match experimentally observed data.
- Derivatives are used in *sensitivity analysis* studies to verify robustness of the simulation with respect to small changes to the input parameters and to verify that the model behaves as suggested by experimental data.
- Derivatives are used in *uncertainty analysis* studies to identify the primary sources of uncertainty in the results of the simulation.
- Derivatives are used in *design optimization* activities to identify optimal settings of design parameters to minimize a cost function.

Unfortunately, few simulations provide derivatives, forcing users to rely on finite difference approximations to derivatives (approximate the derivative of the function f at x, evaluate $f(x)$ and $f(x-h)$ for some small h, and then compute $(f(x) - f(x-h))/h$). Unless used with extreme caution, finite difference approximations can be quite poor.

This chapter examines a maturing technology for computing derivatives of simulations that is known as *Automatic Differentiation*. Automatic differentiation "augments" computer codes with derivative computations by: (1) applying compiler-based techniques to transform a computer code into a new code that incorporates explicit statements to compute the required derivatives, or (2) using operator overloading to extend elementary operations (such as multiply and divide) and elementary functions (such as sine and cosine) to compute the required derivatives. In either case, derivatives are computed by mechanically applying the familiar rules of calculus.

Topics covered in this chapter will include:

- Overview of Automatic Differentiation. This will outline the forward and reverse modes of automatic differentiation and describe the challenges that arise in implementing robust automatic differentiation software.
- Parallelism in Automatic Differentiation. Two subtopics will be covered: (1) use of parallelism in the computation of derivatives for sequential simulation codes, and (2) issues that arise in the application of automatic differentiation to explicitly parallel codes.
- Applications of Automatic Differentiation. Two applications of automatic differentiation will be highlighted: (1) the NEOS Network-Enabled Optimization System, and (2) a NASA Langley/Boeing/Rice collaboration to develop a derivative-enhanced version of the MPI-based CFL3D computational fluid dynamics code for use in aerodynamic shape optimization.
- Software for Automatic Differentiation. Available automatic differentiation software will be described.

5.5 Continuation and Bifurcations in Scientific Computing (20 pages)

Author: Keller

The discretization of most problems in science and technology leads to large systems of nonlinear equations containing one or more parameters. Solutions are desired for some range of the parameters. Thus if $u \in \mathbb{R}^N$ represents the values of all the unknowns which satisfy the discretized problem, say

$$G(u, \lambda) = 0 \qquad ,$$

and λ is a scalar parameter, then $u = u(\lambda)$ traces out a path in $\mathbb{R}^N$ as λ varies over some interval, I. Powerful methods have been devised to compute such paths for broad ranges of problems in which the dimension of the unknowns may be of the order of $N = 10^6$ or larger. Parallel processors are imperative for these very large problems.

Of course, as the path $\Gamma : \{u = u(\lambda),\ \lambda \in I\}$ is traversed, difficulties may arise due to some singular behavior. There are numerous different kinds of singularities that can occur and methods to circumvent or explore the singular phenomena have been devised. Indeed, it is frequently the case that the location and nature of the singular points on a solution path are the most important aspects to the scientist or engineer.

We shall describe some of the basic methods for following paths and for treating the singular points that arise. These include folds, bifurcations, and Hopf bifurcations. A powerful set of software known as AUTO, which employs most of these techniques, has been developed by E. Doedel and his co-workers. The latest versions of this code were developed by CRPC researchers at Caltech and have been parallelized. They also include methods for the analysis of dynamical systems containing heteroclinic and homoclinic orbits.

6 Bringing It All Together and Futures (20 pages)

Section Editor: White

This chapter will first discuss the state of computational science in the mid-1980s: both the heyday and beginning-of-the-end for traditional vector architectures. This time period began with small parallel symmetric multiprocessors, employing a few tens of processors for the most part. The CRPC's initial focus was here. Large-scale scientific computing was the purview of highly integrated, flat-memory vector computers such as Cray YMPs, which appeared on the stage about the same time as the CRPC. This was the third generation of this successful line of computers from Cray Research, Inc.; OS, compilers, tools, and utilities were is pretty good shape; the NSF Supercomputer Centers were on-line and many science and engineering groups around the country were making very significant progress using these impressive tools.

Enthusiasm for computational science grew with each success. Scientists in chemistry, high-energy physics, astrophysics, numerical weather prediction, the oil and gas industry, aerospace, and many other fields were realizing the possibilities and wanted more computational power. Algorithms often provided an effective increase in resources at least equal to gains in hardware performance. CRPC principals played a leadership role in many areas, such as solving linear and non-linear systems of equations, optimization problems, and discretization techniques. Unfortunately, neither algorithms nor hardware appeared likely to satisfy the demand.

Next, we will discuss Massively Parallel Processors (MPPs) and the work over the next decade that proved that this could be an effective dimension in which to grow computing resources. Machines such as the Intel iPSC and TMC CM-2 provided an initial testbed upon which to examine the potential for tremendous increases in computing power, if only this power could be made accessible to scientists and engineers. Parallel algorithms and algorithm templates were a critical development over this time period; these efforts provided foundational capabilities, e.g. Level 2 and Level 3 BLAS, and developed new capabilities required for parallel computers, e.g. domain decomposition. Data movement and data locality mechanisms (e.g. MPI and HPF), key ingredients in effectively using MPPs, were pioneered by the CRPC. We will recap (from § 3) some of the principal applications successes that occurred over this time period. In particular, we will look at astrophysics, weather prediction, and statistical mechanics to provide metrics for how much progress was actually made in some fields.

Predictably, as the resources grew, the complexities of the architectures themselves, of the memory hierarchies, of the programming models, and of the algorithms grew even faster. Another pioneering effort over this period was in the use of high-level languages, toolkits, and problem-solving environments to help manage the difficulties of developing and maintaining applications. In addition, much progress was made on meta-algorithms whose development, through the CRPC, was directly integrated with parallel computation. We will discuss optimization and continuation methods.

Unfortunately, the market for high-end supercomputers began a precipitous slide downhill in 1991 and MPPs were doomed. The strategy of providing small-scale versions of machines specifically designed for the high-end technical market never took flight. Today, none of the leaders from the early part of the decade—Cray Research, Thinking Machines, Intel, Convex—compete in this market. However, rather than an end to the vision of thousands of processors cooperating on a single problem, this extinction marked the real beginning. Now, many companies sell powerful, parallel servers, which range from tens to a few hundreds of processors each. These servers are the switch-based successors of the SMPs upon which CRPC was initially focused. These SMPs exist in a marketspace that is predicted to double in size over the next five years. Clusters of these servers (SMPs) at NCSA, NPACI, Los Alamos, and Livermore have taken up where MPPs left off, reaching into the terascale regime. And the beauty of it all is that much of the work by the CRPC, its affiliates, and others—application templates, parallel algorithms, C++ frameworks, message-passing, F90, domain decomposition—provides an firm foundation for computing on clusters of SMPs.

Looking into the future we will examine

- computational science as a tool of discovery in science and engineering,
- predictive modeling and simulation as a tool for decision and policy makers, and
- the intersection of computational science and information technology.

In addition, we will look at the algorithms, tools, frameworks, and problem-solving environments that will enable these applications.