1 Introduction

Section Editor: Foster

1.1 CRPC and the National Scene (20 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 CRPC's lifetime.

At the time of the founding of CRPC, parallel computing was dominated by the bus-based multiprocessors, although a few pioneering projects, such as the Caltech effort to program hypercubes (someone give me the official title), had begun to experiment with more scaleable 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 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 scaleable 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 scaleable machines. Among these were HPF, Concurrent C++, HPC++, and OpenMP (should I discuss this, as our contribution was minimal – I was on PCF). 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 (20 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 corresponding 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 demand 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, 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 which they can use in support of their activities.

2.2 Parallel Architectures (20 pages)

Author: Stevens

No response from author Re: description & attendance at June CRPC book review meeting

2.3 Computing Technologies (20 pages)

Authors: Kennedy, Foster

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, and high-level programming systems.

Scaleable parallel algorithmic research is classified in three groups: linear algebra, numerical optimization, and simulation. This section will provide an overview of these efforts in 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 (20 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 rather 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 in fact 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 supplies the low-level services necessary to coordinate the use of multiple workstations and other computers for individual jobs, and this system could serve as the foundation for a complete problem-solving environment of the type we envision.

Network computing techniques such as NetSolve offers 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 it (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 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 additional 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. Appl1 is a general discussion; Appl2 is a set of about 20 short case studies and Appl3, Appl4 and Appl5 are three 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 so be ready to start parallel computing!

3.1 Appl1: General Application Issues (20 Pages)

Author: Fox

The first application chapter contains an introduction to the other four 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 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. (This list is very mixed up of course – it will be added to and organized). We discuss differences between illustrative and real applications by contrasting Laplace's equation with Navier Stokes and complex physics in climate simulations. We will try to discuss typical problem sizes and why tera- and peta-(fl)op machines are relevant. Typical issues governing performance are discussed. Maybe we can 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 discussion in following four chapters which will be summarized in a suitable set of tables. The discussion of basic numerical methods (PDE, ODE, Monte Carlo etc.) could go to § 5.

3.2 Appl2: 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. Comp. Chemistry - (Kuppermann, Goddard, McKoy, PNL) - start
6. QCD (Geoffrey, Rajan, Ceperley) - start
7. Accelerators (Ryne - LANL)
8. Plasma Physics (Reynders) - start
9. MDO (Geoffrey)
10. Financial modeling (Geoffrey)
11. Weather (CAPS, NASA)
12. Comp. Biology - Keck Center, Rice - start
13. Astronomy - T. 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 - (GCF, CACR)
20. CFD (Dan, Herb) - start
21. Energy and environment (Mary Wheeler) - start
22. Computational Electromagnetics (DoD Modernization)
23. Signal Processing (DoD Modernization)
24. Electrical Transmission Lines (Geoffrey)

Designation as "Start" implies that one starts with this subset to give exemplars that can used to show others how to do their field. Note some of those areas are quite broad and could generate several distinct summaries. For instance, 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 Appl3: Parallel Computing in CFD (20 pages)

Authors: Meiron, Keller

Dan Meiron has responded and will provide the requested description.
Description from Geoffrey Fox:

> special to CRPC
> Intro - context for CFD - impact of CFD on industry
> Incompressible CFD, Homotopy, Compressible CFD
> Specific mention of AMR

3.4 Appl4: Parallel Computing in Environment and Energy (20 pages)

Author: Wheeler

Mary Wheeler has responded and will provide requested description.

Description from Geoffrey Fox:

> Intro - impact
> Specific mention of domain decomposition - multigrid

3.5 Appl5: Computational Astrophysics (20 pages)

Author: Salmon

No response from author Re: description & attendance at June CRPC book review meeting

Description from Geoffrey Fox:

> Specific mention of multipole techniques
> Describe Beowulf and why this can be used here and in other applications

4 Computing Technologies

Section Editor: Kennedy

4.1 Base Technologies (20 pages)

Author: Kesselman

No response from author Re: description & attendance at June CRPC book review meeting

4.2 Libraries (20 pages)

Authors: Reynders, Gannon

Dennis Gannon has responded and will provide the requested description.

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 CRPC. The principal foci of this discussion will be on Fortran and C++, with some discussion of Java.

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.

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 Concurrent C++ and HPC++. The latter emphasis will concentrate on the part of the language efforts that were not based on libraries (which will be expanded by Gannon and Reynders in their section). *****Mani, would you like to finish this?*****

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 language 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. 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 desribe general issues in Problem Solving Environments (PSE) discussing two types of activities – building the components from which many different PSE's can be constructed and secondly using these components in particular application domains. Then we illustrate these general ideas by two particular examples – Netsolve and WebFlow which have been succesfully 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 and 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.

Keywords: source code correlation; instrumentation; adaptive control; multilevel measurement; performance visualization

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 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, 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, and these 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 the block size, while on distributed memory machines it is controlled by the 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 Constrained Optimization (20 pages)

Authors: Dennis, Wu

John Dennis has responded and will provide the requested description.

5.3 Discrete Optimization (20 pages)

Authors: Bixby, Cook

Bill Cook has responded and will provide the requested description.

5.4 Automatic Differentiation (20 pages)

Author: Carle

Alan Carle has responded and will provide the requested description.

5.5 Nonlinear Equations (20 pages)

Author: Keller

No response from author Re: description & attendance at June CRPC book review meeting

6 Bringing It All Together and Futures (20 pages)

Section Editor: White

Andy White wants to review the abstract before composing this description.