

PARALLEL NUMERICAL ALGORITHMS: AN INTRODUCTION

*David E. Keyes*¹

Department of Computer Science	ICASE, MS 132C
Old Dominion University	NASA Langley Research Center
Norfolk, VA 23529-0162	Hampton, VA 23681-0001
keyes@cs.odu.edu	keyes@icase.edu

The rapidly evolving field of parallel computing has seen four eras, substantially overlapping, yet progressive. The first, which we could label the Stone Age, was characterized by having speed-up as its primary objective (“Solve my problem faster...”). The tyranny of Amdahl’s Law, combined with architectural convergence to systems that added memory in fixed proportion to processors, led to the Bronze Age, characterized by pursuit of *scaled* speed-up (“Solve my problem bigger, in constant time...”). The cost associated with special purpose high-performance hardware and the even greater cost of developing reliable special purpose system-level software led to the Iron Age of commodity processors — clusters of full-function computers connected by either a special purpose network or a commodity network like Ethernet or FDDI (“Solve my problem cheaper...”). Though technology and market forces have not been eclipsed in their influence on parallel tool-building and parallel applications, we can now point to a growing influence of algorithmic development and software environment development on the envelope of computability (“Solve my problem smarter...”). Perhaps, through continued progress in architecture-adaptive algorithmic design that improves upon “best fit” flop-for-flop algorithmic porting, we will soon be able to refer to a Golden Age of parallel computing.

The goals of the ICASE program in parallel numerical algorithms are to *accelerate*, *widen*, and *enhance* the use of high performance parallel scientific computing within NASA computational applications. The first goal refers to hastening the transition to parallel

¹This work was supported by the National Aeronautics and Space Administration under NASA Contract NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.

supercomputing, the second to expanding the parallel user community beneath the topmost decile or two of computer fluency in the user community at large, and the third to improving the efficiency of parallel supercomputing throughput — all the way from problem definition to data interpretation. Recognizing that parallel software continues to lag behind hardware, its mission includes: (1) evaluating the parallel scalability of currently important algorithms in scientific and engineering computation, both theoretically and experimentally; (2) developing new algorithms with efficiency trade-offs guided by knowledge of both architectural limitations and mathematical theory; and (3) demonstrating the practicality of implementation of such algorithms on networks of workstations and parallel supercomputers.

The study of parallel numerical algorithms is neither a proper subset of numerical analysis, in which the striving for optimal complexity for a specified accuracy is often blind to memory hierarchy, nor of traditional computer science, which is often blind to numerical magnitudes and thus fails to exploit the fact that not all nonzeros are of equal consequence to the results to be extracted from a given scientific computation. The field of parallel numerical algorithms could be defined as “scientific computing in the presence of nonuniform memory access.” It requires inputs from both numerical analysis and computer science.

Computer science solutions to the problems of partitioning, mapping, and scheduling in the abstract are critical to the effective implementation of scientific computing algorithms on scalable distributed-memory parallel computers. However, even optimal solutions to these problems applied to standard numerical algorithms may require too much communication for scalable performance on current systems. Further progress depends on the recognition that many algorithms contain data dependencies that are *superfluous with respect to the quality required* in an intermediate result. An important example is the output of a preconditioning step in a Krylov method. Guided by mathematical (and ultimately physical) reasoning, such algorithms can be streamlined in communication or arithmetic complexity, or both, resulting in faster, more parallel algorithms. An algorithm whose data dependency graph is first pruned at the mathematical definition stage creates additional possibilities for computer science solutions at the implementation stage.

Though the proper algorithm (or, increasingly, the proper polyalgorithmic combination) is critical to overall performance, some might

object to the adjective “parallel” in the title of this collection. Future computational scientists may find it ironic that people referred to “parallel algorithms” during the 1980’s and 1990’s, as if there were other kinds. After all, a folk theorem states: “The best parallel algorithm for a given problem is also the best serial algorithm.”

It would be surprising if this “theorem” held for all specifiable transformations of input to output, because parallel algorithms are subject to practical constraints in their access of non-local data to which serial algorithms are not subject. It would be miraculous if some good algorithms were not excluded upon reducing the feasible space of all possible algorithms to just those algorithms that make predominant use of local memory accesses when mapped to parallel hardware. In fact, there are many elementary counterexamples in which forcing concurrency onto a series of arithmetic operations causes deterioration of the convergence rate of an iterative method, relative to a less concurrent evaluation order.

However, the theorem is provocative, especially when the focus is reversed: Is there any such thing as a *serial* algorithm? A serial algorithm might be defined as an algorithm for which data accesses occur sequentially, and for which the global memory space can be thought of as flat, meaning that it costs a constant amount of time to access data from anywhere in the global address space of the problem. For sufficiently large problem size, no buildable architecture supports this model of memory. Going out of cache in a serial machine is analogous to going off board in a message-passing machine. If the three most important criteria in selecting an algorithm are data locality, data locality, and data locality, then for sufficiently large problems, serial algorithms that are not also parallel algorithms are irrelevant.

If future computational scientists will find the title “Parallel Numerical Algorithms” anachronistic, they will at least concur with the importance of “parallel algorithmic thinking,” and readers of this volume will likely have theirs improved. The best parallel algorithms tend to have in common not only a restructuring of data access patterns to fit into local basins of fast access memory; they also make strategic use of global information. Global information — such as a global maximum operation to find the best pivot, or a global inner product to find an optimal step length, or a direct solution on an agglomerated coarse grid version of a problem as part of a preconditioner — is always going to have a nonnegligible communication cost associated with it, but that cost can be traded off against the

cost in numerical stability or the cost in convergence rate of making due with only locally available information. This trade-off is where the battle for the best algorithms is won or lost, and it is likely to be machine specific as well as problem specific. Thus, parallel algorithmic thinking is not a series of mechanized transformations, but intellectually creative.

We briefly annotate some aspects of parallel algorithmic thinking in the chapters to follow, using the workshop presentation order as an outline.

Linear Systems

- *Dense Matrix Methods*, James Demmel, Departments of Mathematics and Computer Science, University of California, Berkeley. This chapter expands on and updates the dense matrix sections of the survey *Parallel Numerical Linear Algebra*, Acta Numerica, 1993, and specifically emphasizes development of the LAPACK and ScaLAPACK linear algebra libraries (see URLs below), and the BLAS (Basic Linear Algebra Subroutines) and PBLAS (Parallel BLAS) on which they are built. Standing first in this collection, it contains careful definitions of some of the key terminology of parallel computing, and establishes the motivation for locality of reference and for reuse of data elements communicated to the processor from memory. The relative merits of the various distributed data layouts for a dense matrix (as shown in Figure 4) are evaluated with respect to the data access patterns of Gaussian Elimination (as shown in Figures 8 and 9). An example of a parallel complexity analysis taking into account the overhead of communication by means of a two-parameter latency-bandwidth model is developed in Section 5.2 for the publicly available ScaLAPACK Gaussian elimination factorization routine for a distributed memory computer. The theoretical model compares very well with experiment in the scaled speed-up limit, as shown in Figure 10. Asymptotic scaled speedups increase with the size of the problem computed on each node, achieving 50% at as little as 2MB/node, for example, over a 16 to 128 processor range on the Intel Gamma.
- *Sparse Direct Solution Methods*, Michael T. Heath, Department of Computer Science, University of Illinois, Urbana-Champaign.

This chapter expands on and updates the sparse matrix section of the survey *Parallel Numerical Linear Algebra*, Acta Numerica, 1993, and provides an introduction to graph-based language for describing sparse matrices and to task scheduling and ordering problems for the irregular data structures of sparse matrices. Choices are enumerated and evaluated in ordering the three loops of the Gaussian elimination factorization process, in ordering the sparse matrix, itself, and in mapping subtrees of the elimination tree to processors to minimize communication and balance load. There is a trade-off between minimizing fill-in and maximizing concurrency. The state-of-the-art for the symmetric positive definite sparse problem seems recently to have converged on a multifrontal approach with a two-dimensional block-cyclic mapping. A parallel case study on a CM-5 using a code that is publicly available as part of the ScaLAPACK distribution (see URL below) in a constant work per processor scaling is described, showing reasonable scalability in the 16 to 128 processor range.

- *Sparse Iterative Solution Methods*, Henk Van der Vorst, Department of Mathematics, University of Utrecht. This chapter prepends considerable motivational material to the iterative matrix section of the survey *Parallel Numerical Linear Algebra*, Acta Numerica, 1993, which it also updates with respect to the recent flurry of invention of Krylov methods based on 3-term bi-orthogonality relations and hybrids of full-orthogonality methods (such as GMRES) with other methods. The only intrinsic impediment to parallelism in such methods, apart from preconditioning (which is treated in an independent chapter), is the global reduction necessitated by inner products. Means of reordering the sequential algorithms to overlap the communication of inner products with useful computation, or of clustering the inner products so that their latencies are paid only once per several steps are explored. The trade-off in the latter case is against stability, but the price is often acceptable. By way of example, experiments reported on a Parsytec machine for a fixed-size model 2D PDE-based problem showed a doubling of the speed-up when inner products are combined, from 108 to 223 (on 400 processors).
- *Sparse Iterative Eigensystem Methods*, Dan Sorensen, Depart-

ment of Computational and Applied Mathematics, Rice University. This chapter concentrates on primarily new material on implicitly restarted Arnoldi/Lanczos methods for large-scale eigenproblems. It serves as an authoritative tutorial for ARPACK, which is now available as part of the ScaLAPACK distribution (see URL below). The algorithmics are built on Krylov iteration, which is presented in the previous chapter. Attention is given in Section 7 to the software issue of allowing a user-supplied distributed data structure for the (usually) sparse matrix operator of the Krylov method, instead of requiring the user to conform to an internally supplied data structure to be communicated several levels down a complex calling tree from the point of user invocation to the point of application. This is a critical issue for the separation of expertise of user and algorithm designer, and is hence critical to the portability of any library of Krylov solvers. In ARPACK, this issue is resolved by a reverse communication interface, in which the eigensolver returns to the user code each time a matrix-vector product on a given vector is required. (This approach is complementary to the use of a context variable for data-hiding in the PETSc software library mentioned below.) Tables 3 through 5 illustrate scaled and fixed-size speed-ups for ARPACK on idealized problems designed to expose scalability. Scaled efficiencies of 70% are realized over a range of 1 to 256 processors.

Preconditioning

- *Incomplete and Approximate Factorizations*, Tony Chan, Department of Mathematics, University of California, Los Angeles. The battle for reduced wall-clock execution time or scaled efficiency for a parallel iterative method is usually won or lost in the preconditioner. This chapter examines many ways of trading off algebraic preconditioning quality against parallelizability, which is the main manifestation in parallel supercomputing of the general trade-off in any iterative technique between the number of steps and the cost per individual step. Using primarily the five-point finite-difference operator as a tutorial example, a variety of incomplete factorizations are defined algorithmically, and three means of extracting greater parallelism from

them are compared in Section 4. The primary focus is on re-ordering a recurrence to obtain either long fronts of concurrent operations or multiple smaller fronts. A complementary technique to parallelizing the intrinsically sequential recurrences of incomplete factorizations is to approximate them with products or sums of operators that require only highly concurrent sparse matrix-vector multiplies. Finally, on problems that arise from PDEs, techniques based on domain decomposition, leading to independently formed preconditioners on different subdomains are put into the incomplete factorization context. (These techniques are outlined more comprehensively in the chapter on domain decomposition.) Extensive literature citations illustrate the problem dependence and architecture dependence of any conclusions about the overall effectiveness of these methods. Nevertheless, their general applicability causes them to remain the methods of choice for many problems.

- *Multilevel Methods*, Steven F. McCormick, Department of Applied Mathematics, University of Colorado, Boulder. Multilevel methods are solvers in their own right, and optimal serial algorithms on an ever increasing variety of problems, including some with no continuous origin. However, we may also think of them as preconditioners by employing fewer cycles than it would take for convergence. In problems for which it is difficult to obtain optimal multilevel convergence rates, multilevel methods may still be very effective in combination with a Krylov iterative method. By the metric of efficiency, parallelization of standard sequential multigrid is challenging, because of the small denominator in the communication to computation ratio. Within and between each smoothing process on each level, communication is required that cannot be overlapped with computation to any useful degree, and the ratio of communication to computation worsens with each coarsening of the grid. Section 2 examines data parallelism within a level of a standard sequential multilevel approach, as well as two other opportunities for concurrency: multiple concurrent coarse grids to maintain the density of computation at coarse levels and methods that smooth concurrently on different levels. The latter discussion leads naturally to a brief exposition of the AFAC algorithm. The bibliography was drawn up at

a session of the 1995 Copper Mountain Conference on Multigrid Methods. (For future updates from MGNet, see the URL below.)

- *Domain Decomposition Methods*, Barry F. Smith, Mathematics and Computer Science Division, Argonne National Laboratory. Domain decomposition is perhaps the ultimate hybrid method. It may take on the form of a block iterative method, a block direct elimination method, or a (usually truncated) multilevel method. What makes domain decomposition deserving of a separate chapter is its continuous problem origin, which translates insight from the underlying PDE into the partitioning and solution process. The contemporary interest in domain decomposition methods comes from their promise as highly concurrent preconditioners for a Krylov method. This chapter surveys the state-of-the-art and serves as a brief introduction to the author's new book on the subject, as well as to the PETSc software library (see URL below).

“Fast” Application of Operators

- *Fast Fourier Transforms*, Richard B. Pelz, Department of Mechanical and Aerospace Engineering, Rutgers University. The FFT, and its generalization to other orthogonal polynomials besides sines and cosines, is a kernel operation in many areas of large-scale computing. A prominent example, and one that motivates this chapter, is spectral PDE algorithms, which provide very high resolution for applications such as turbulence simulation. Parallelization of the FFT is challenging because of the all-to-all communication required to transform an input sequence into an output sequence. This chapter describes FFT algorithms for distributed memory processors with a focus on communication patterns and on networks that support their basic blocks: the i-cycle and the transpose. Significant communication savings can be realized if the transformed sequence can be used out of order, which may often be the case in practice. Multidimensional FFTs raise a variety of mapping options, for which the optimal topology varies with parallel granularity.

- *Fast Multipole Methods*, Jacob White, Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology. The FMM is a kernel operation in boundary integral techniques for potential problems, where it improves the serial solution complexity from N^2 to N , where N is the number of boundary elements in the discretization. The costs come from counting the mutual interactions of each element with the others. $\mathcal{O}(N)$ evaluation complexity implies that only constant time is required to evaluate the interaction of each of N elements with all of the others, to an acceptable accuracy. These $\mathcal{O}(N)$ matrix-free applications of a linear operator are building blocks from which a Krylov solver may be constructed. The numerical conditioning is such that a Krylov solver such as GMRES converges in a bounded number of iterations without any further preconditioning. The FMM is an example of an algorithm that achieves an order improvement in complexity by discarding arithmetic operations (and, perhaps more significantly, the communication that is required to set up their operands in a distributed environment) based on *a priori* bounds on the magnitude of their affect on the desired result. The key idea is that most mutual interactions between the discrete elements are long-range in nature, and that long-range interactions may be coarse-grained. Long-range interactions are represented in a multipole expansions, which are truncated to number of terms that is small relative to the number of elements effectively being summed over. The octree-type data structures required to efficiently implement the FMM are briefly described in Section 3. The FASTLAP code is publicly available (see URL below).
- *Fast Multiresolution Application of Operators*, Amiram Harten, formerly Department of Mathematics, Tel-Aviv University. Multiresolution methods are generalized wavelet transforms that, like FFTs, transform a sequence of coefficients from one basis to another which is more convenient for some computational purpose and, like FMMs, permit a significant fraction of the arithmetic to be skipped with negligible consequence to the accuracy of the result. The data compression obtained through multiresolution transforms can be used either to sparsify the transformed sequence or to sparsify the operator itself (through a two-sided transform). Harten's presentation is de-

lightly algorithmic in its essence, and complemented in an interleaved fashion by a somewhat formal description in terms of “reconstruction” and “decimation” operators. A link with the theory of wavelets is made in Section 3. Integral operator evaluation and hyperbolic initial value problems are considered as applications. The data-dependence of the sparsity pattern complicates the parallelism of multiresolution codes, as in any adaptive method with a dynamic data structure.

Parallel Tools, Environments, and Benchmarking

- *Decomposition and Ordering Methods*, Alex Pothén, Department of Computer Science, Old Dominion University. Scalable data parallelism begins with a partitioning that balances workload among processors, and it is refined as communication is minimized in some metric under a combined partitioning and mapping. Whereas the partitioning and mapping problem often has trivial solutions in problems with regular data structures, such as logically Cartesian grids, it can be harder than the problem of actually solving the discrete equations in irregular problems. This chapter surveys the surprising variety of techniques that have proved useful in different applications in recent years, and shows their relationship with the related problem of ordering. Partitioning may at the same time be considered a tool for parallel computing and an application for parallel computing, since for large problems, the partitioning problem may itself need to be solved in parallel. A geometric partitioning method having good parallelizability is discussed, as is a recursive spectral bisection method that often does an excellent job in minimizing the number of edges cut. The state-of-the-art in partitioners seems to be converging to a multilevel algorithm based on recursive coarsening, refinement, and uncoarsening. Multilevel and other partitioners are available in the METIS and CHACO libraries (see URLs below).
- *Performance “Debugging”*, William D. Gropp, Mathematics and Computer Science Division, Argonne National Laboratory. The implementation of a parallel algorithm is usually justified only in situations in which there is high utilization of the hardware. However, high utilization is a relative concept. Evalua-

tion of the parallel performance of an algorithm-machine pairing requires a quantitative model of parallel execution and experimental monitoring capability. This chapter outlines issues in the rapidly evolving art of performance debugging. Performance debugging logically separates into per node performance analysis of the traditional serial computing type, emphasizing issues of memory hierarchy and pipelining, and communication performance analysis, emphasizing properties of the network and the possibility for overlapping communication with useful computation. The state of the art being very far from satisfactory, this chapter gives some principles and some tutorial examples of performance debugging. Some specific tools are illustrated: the profiling library of the Message Passing Interface (MPI) and the Upshot program, which can interpret log files created by MPI (see URLs below).

- *Performance Metrics and Benchmarking*, David H. Bailey, NAS Applied Research Branch, NASA Ames Research Center. This chapter serves as a brief overview of the NAS Parallel Benchmarks, which have been devised to provide quantitative rankings of parallel computer performance on a variety of kernels (such as the FFT, multigrid, and conjugate gradients) and simulated fuller applications from scientific computing. Like the LINPACK benchmarks, the NAS Parallel Benchmarks are updated periodically with the assistance of users of various systems around the world. The latest versions of both surveys are available on-line (see URLs below). The NAS Parallel Benchmarks are specified not with code, but in a pencil and paper fashion that encourages benchmarkers to select the language constructs and implementation techniques best suited for a particular system.

We conclude this introduction with two lists of universal ?? resource locators (URLs) for complementing this volume with world wide web resources. We list sites referred to above in the chapter annotations and we list the home pages of key contributing authors, who in most cases offer their technical reports expanding upon their chapters over the web.

Technical WWW sites:

<http://netlib2.cs.utk.edu/>

The header page for the netlib repository of publicly available software (which is shadowed elsewhere on the Internet), and many other repositories of interest to computational scientists and engineers.

<http://netlib2.cs.utk.edu/lapack/index.html>

The index for the LAPACK library.

<http://netlib2.cs.utk.edu/scalapack/index.html>

The index for the ScaLAPACK library.

http://www.netlib.org/lapack/lug/lapack_lug.html

A hypertext version of the LAPACK User's Guide – Release 2.0, including an electronically browsable index.

http://www.netlib.org/scalapack/scalapack_home.html

The ScaLAPACK home page, including links to the PBLAS and performance plots for ScaLAPACK routines on parallel supercomputers and workstation clusters.

http://www.netlib.org/linalg/html_templates/Templates.html

A hypertext version of the book *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*

<http://casper.cs.yale.edu/mgnet/www/mgnet.html>

The MGNet home page, including links to an enormous bibliography on multilevel and domain decomposition methods, publicly available multigrid codes, and research papers.

<http://www.mcs.anl.gov:80/petsc/petsc.html>

The PETSc home page, including a link to publicly available code with a bundled version of Upshot.

<ftp://rle-vlsi.mit.edu/pub/fastlap/>

The boundary integral method-based FASTLAP solver.

<http://www.cs.umn.edu/~karypis/metis/metis.html>

The METIS partitioner library.

<http://www.cs.sandia.gov/HPCCIT/chaco.html>

The CHACO partitioner library.

<http://www.mcs.anl.gov/mpi/index.html>

The public domain MPICH version of MPI

<http://www.netlib.org/liblist.html>

The LINPACK benchmark repository, including the NAS Parallel Benchmark specification (for an up-to-date copy of the NAS Parallel Benchmark results, see the home page of David Bailey, below).

Key contributing authors, e-mails, and home pages:

David Bailey

dbailey@nas.nasa.gov

<http://www.nas.nasa.gov/~dbailey/dbailey.html>

Tony F. Chan

chan@math.ucla.edu

<http://www.math.ucla.edu/~chan/>

James Demmel

demmel@cs.berkeley.edu

<http://HTTP.CS.Berkeley.EDU/~demmel/>

William D. Gropp

gropp@mcs.anl.gov

<http://www.mcs.anl.gov:80/home/gropp/>

Michael T. Heath

heath@ncsa.uiuc.edu

http://www.cs.uiuc.edu/CS_INFO_SERVER/DEPT_INFO/CS_FACULTY/FAC_HTMLS/heath.html

Jim E. Jones

`jjones@icase.edu`
`http://www.icase.edu/~jjones/index.html`

Steven F. McCormick

`stevem@boulder.colorado.edu`
`http://amath-www.colorado.edu/appm/faculty/stevem/`
`Home.html`

Richard B. Pelz

`pelz@jove.rutgers.edu`
`http://saturn.rutgers.edu:80/~pelz/`

Alex Pothen

`pothen@cs.odu.edu`
`http://www.cs.odu.edu/~pothen/pothen.html`

Barry F. Smith

`bsmith@mcs.anl.gov`
`http://www.mcs.anl.gov:80/home/bsmith/`

Danny C. Sorensen

`sorensen@rice.edu`
`http://www.caam.rice.edu/caam/sorensen.html`

Henk A. Van der Vorst

`vorst@math.ruu.nl`
`http://www.math.ruu.nl/people/vorst/`

Jacob K. White

`white@rle-vlsi.mit.edu`
`http://rleweb.mit.edu/P-WHIT.HTM`