TRENDS IN ALGORITHMS FOR NONUNIFORM APPLICATIONS ON HIERARCHICAL DISTRIBUTED ARCHITECTURES

DAVID E. KEYES
Department of Mathematics & Statistics
Old Dominion University
Norfolk, VA 23529-0077 USA,
Institute for Scientific Computing Research
Lawrence Livermore National Laboratory
Livermore, CA 94551-9989 USA, &
Institute for Computer Applics. in Science and Engineering
NASA Langley Research Center, MS 132C
Hampton, VA 23681-2199 USA

Abstract.

Scientific programmers are accustomed to expressing in their programs the "who" (variable declarations) and the "what" (operations), in some sequentialized order, and leaving to the systems software and hardware the questions of "when" and "where". This act of delegation is appropriate at the small scales, since programmer management of pipelines, multiple functional units, and multilevel caches is presently beyond reward, and the depth and complexity of such performance-motivated architectural developments are sure to increase. However, disregard for the differential costs of accessing different locations in memory (the "flat memory" model) can put unnecessary amounts of synchronization and data motion on the critical path of program execution. Different organization of algorithms leading to mathematically equivalent results can have very different levels of exposed synchronization and data motion, and algorithmicists of the future will have to be conscious of and adapt to the distributed and hierarchical aspects of memory architecture.

Plenty of examples of architecturally motivated algorithmic adaptations can be given today; we illustrate herein with examples from recent aerodynamics simulations. For this purpose, pseudo-transient Newton-Krylov-Schwarz methods are briefly introduced and their parallel scalability in bulk synchronous SPMD applications is explored. We also indicate some fundamental limitations of bulk synchronous implicit solvers and propose asynchronous forms of nonlinear Schwarz methods as perhaps better adapted both to massively parallel architectures and strongly nonuniform applications. Suitably adapted PDE solvers seem to be readily extrapolated to the 100 Tflop/s capabilities envisioned in the coming decade. Making use of some novel quantitative metrics for the memory access efficiencies of high performance applications ("memtropy") and for the local strength of nonlinearity ("tensoricity") in applications with spatially nonuniform characteristics, we propose a migration path for scientific and engineering simulations towards the distributed and hierarchical Teraflops world, and we consider what simulations in this world will look like.

1. Introduction

By way of introduction, we sketch two fictional on-the-job scenarios that exemplify our vision of highly parallel, adaptive, immersive computations in the aerosciences of the very early 21st century.

1.1. AIRPLANE INDUSTRY SCENARIO

Hector returns from lunch, checks the parallel batch queue, and notices that the first twelve of the twenty 32-processor jobs he launched on his way out are complete, and five more are in various stages of execution.

"Lotta nodes free today...," he hums to himself as he dons the goggles, steps into the CAVE and summons the first set of on-wing pressure coefficient contours. He cycles repeatedly through the dozen images, slowly at first, as they are rendered from disk files and as he checks general features, then faster, as they are cached from main memory of the display engine. Tracking just the foot of the main shock between successive frames, and observing it to advance towards the leading edge as the leading edge thickness parameter increases, he recalls that the thickest wings were towards the end of the batch queue, he snoops on the partially converged latest case. The image materializes in patches at random intervals, which reminds him of the new asynchronous nonlinear Schwarz algorithm to which he recently upgraded the parallel solver. Visualization is a snooping thread of tertiary priority, active only during cycles when the Schwarz solver and the local Jacobian refresh threads are stalled on memory operands, or when the entire global analysis occasionally synchronizes to compute a norm for convergence checking.

"That asynch Schwarz saves me more than an order of magnitude by skipping most of the farfield updates until after the shock position con-

verges," he reflects. "Of course, that's on top of the factor of near 25 in memory I get over most of the farfield by abandoning primitive variables outside of the wake region and replacing those 5×5 *primitive variable Jacobian blocks with scalar elements of the full potential equation." Just thinking about all the memory he used to waste in every Euler run back when memory was the main cost of outfitting a supercomputer embarrasses him. "Of course, memory isn't such a limiting cost anymore, now that we have so enhanced data locality that we can even cover the latency of private disks in out-of-core Schwarz solvers."*

Still, he knows that the senior members of the wing design team don't trust the multi-model Euler/full potential pressure data, so he squeezes a pre-scripted button in the wand, rotates the wing for a view from the outboard end in towards the fuselage, peers spanwise, and superposes the physical three-dimensional shock surface onto an image of the thick fringe of gridpoints that indicates the adaptively chosen transition zone between the Euler model used in shocks and wakes — capable of describing entropy generation and rotation — and the full potential model used in isentropic regions.

"Shock safely contained inside the Euler region," he nods, calling for the same superposition from another thick wing case, which he notices has just finished. "Actually, I trust these Euler-FP runs more *than the full Euler runs," he thinks. "Too much spurious entropy generation from the Euler discretization where there shouldn't be any! Then it convects downstream where it's hard to isolate from local effects by the time it reaches the rudder."*

With the requisite data already in the bag for his afternoon design meeting, he retreats to his desktop to steal an hour for research. He stores the memory reference and execution time traces from a run based his latest meshpoint ordering to the data base of the memtropy *optimizer, and restarts the optimizer to generate a new ordering with improved memory reference locality. He then writes a script to display 3D isosurfaces of the latest form of his* tensoricity *metric. Tensoricity, a localized measure of Fréchet derivative of the Jacobian elements, helps the nonlinear Schwarz adaptively apply effort where the computation is highly nonlinear, where the Newton tangent hyperplanes are constantly shifting. Checking his watch, he pings a cross-country colleague, pops open the remote collaboration tool, frames a midspan cross section, and highlights a crescent of cells near the shock for discussion …*

1.2. AUTOMOBILE INDUSTRY SCENARIO

Helene sits in front of the windtunnel's glass window, slowly steering a wheel that rotates the instrumented automobile model plus or minus 15 degrees *with respect to the oncoming 50 mph wind while listening to headphones, which are connected to microphones at the ear locations of a dummy driver. Simultaneously, she studies the Fourier signature of the broadband noise, as it is projected on the heads-up see-through display panel. Occasionally, she footpedals the smoke tracer to reveal the position of the vortex rolling up along the driver-side window aft of the passenger compartment A-pillar. Standing behind her is an engineering collaborator, similarly equipped with headphones and similarly visually transfixed. Using smoke traces, she settles on a particularly interesting yaw angle and dispatches the coordinates to an offsite parallel engine.*

"Stand by for the simulation," she calls out, and hits a control sequence on the keyboard. Several seconds later a sentinel silence occurs, then a split second of numerically synthesized sound. A corresponding spectrum, much "whiter" than that of the experiment, appears and is captured on the display. Groans follow. "Okay, we knew we might want to increase the coefficient on the dissipation rate term in the turbulence model," says the engineer, at the beginning of a week of dedicated windtunnel timeslots. He remembers the days when a single Navier-Stokes run over such a bluff body could not be done in a week.

Helene, meanwhile, projects the computational grid on the heads-up panel, aligns it to the physical geometry with size and position verniers, and triggers another smoke trace. "It looks like the grid adaptation routine is completely missing the main reattachment zone," she says. "Let's adjust the surface stress refinement indicator before we monkey with the turbulence model." Within the first two hours of tunnel time, the investigators explore many hypotheses, many adaptations of the grid and tunings of the turbulence model, then retreat to a conference room, where other engineers gather. Helene presents a short lecture on their new parallel multi-model technique, in which a turbulent Navier-Stokes simulation is used to generate noise in the vicinity of the A-pillar, which is then Fourier-analyzed, filtered, and propagated throughout the car interior by means of parallel Helmholtz solves at hundreds of component frequencies. Each of the independent Helmholtz solves is then further decomposed for parallel solution on an appropriately resolved grid by means of a discontinuous Schwarz preconditioner, and the signal is reconstructed.

"Tens of thousands of processors are needed to do the work of the microphone in the windtunnel," she laughs, "but if we ever get these simulations to be predictive, we can play with the geometry without folding any metal or molding any clay. But none of this is practical without the multi-model capability or without the data-parallel analysis capability for the individual task-parallel problems. Can't afford to run Navier-Stokes everywhere, and the energy in the noise signal would be lost in the discretization error even

if we could. Solving for the scalar disturbance potential is the obvious way to propagate the noise in the cabin. Linking the models up is where the real research lies. And until it was convenient to cut-and-try with a parallel algorithmic breadboard, we could never consider doing this ... "

1.3. ORGANIZATION OF THIS CHAPTER

With these scenarios as a foretaste, we consider in Section 2 the historical role of algorithmic research and how it is being redefined as a result of its own success. Section 3 summarizes some of the principal forces on contemporary algorithmic research, which point to the centrality of latency tolerance, the subject of Section 4. Section 5 introduces an important class of solution algorithms for the nonlinear systems of PDEs that occur throughout the aerosciences, which have been designed with latency tolerance foremost in mind. Adaptivity or "tuning" of these algorithms is considered within the context of bulk synchronous SPMD applications. The next three sections (6–8) present the need for more radical forms of adaptivity, to extremes of application nonuniformity, memory hierarchy, and distributed memory granularity. Novel metrics are introduced to quantify novel algorithmic design criteria. Asynchronous nonlinear methods, representing a break with the traditional SPMD model, are suggested as a means of providing the additional flexibility required to accommodate these coming extremes. Section 9 defines a class of nonlinear Schwarz methods that can be run in synchronous, pipelined-deferred synchronous, or partially asynchronous modes. We conclude in Section 10 with a summary agenda for research in algorithms for the computational aerosciences, and for algorithm research more generally.

2. The Role of Algorithmic Research

The complementary roles and co-importance of progress in algorithms and progress in computer architecture were canonized in the first of the U.S. federal interagency HPCC initiative "bluebooks" (FCCSET, 1992). The fruits of algorithmic progress considered therein were confined to the realm of operation count reduction, and were assumed to be orthogonal to the fruits of architectural progress, namely faster processing of operations. This implies that that factors of improvement in algorithms and architecture can simply be multiplied in order to arrive at the overall factor of performance improvement. Such simple models are no longer adequate.

2.1. BEYOND "OPTIMAL" LIES "ADAPTIVELY OPTIMAL"

The "bluebook" illustration culminated with an algorithm that is arithmetically optimal for an expanding class of applications, namely multigrid. Algorithmic progress of the future will — by definition — not lie in a further reduction of the arithmetic complexity for these applications, but in adaptation to the increasingly severe architectural environment, so that optimal arithmetic complexity translates into optimal execution time. It will also lie in the development of optimal methods for an ever expanding class of applications that are pronouncedly nonuniform in character throughout the simulation domain.

Throughout the history of high performance computing initiatives, it has always been assumed that algorithms can be adapted to satisfy the "boundary conditions" imposed by applications requirements and architectural availability. The exponentially diverging service times of floating point processors and memory systems make this assumption increasingly nontrivial. However, the resulting pressure on algorithms to be memory latency-tolerant and synchronization-tolerant at least gives a clear direction for extrapolation of algorithmic progress. The extrapolations of application requirements and architectural availability into the next 10 years seem relatively straightforward, and algorithms will evolve to attempt to span the gap.

2.2. MINING THE LITERATURE FOR "NEW" ALGORITHMS

We predict that much of what will be called "algorithmic progress" in the next 10–20 years is published already — some of it long ago. Algorithmic advances often spend decades in the literature, dormant with respect to computational science, before triggering revolutionary advances in the latter when adapted to the service of contemporary architecture.

Space-filling Curves were described as topological curiosities in (Peano, 1890) and (Hilbert, 1891), but were recognized by Warren & Salmon (1995) to provide a 1D ordering on points in 3D space that translates physical locality into memory locality. They are now the basis of Bell-prize-winning N-body gravitational simulations for hundreds of millions of particles on thousands of processors and a cost-effective out-of-core Beowulf version of the same simulation (Warren et al., 1998).

The *Schwarz Alternating Procedure* was described as a proof of existence and uniqueness for the solution of elliptic boundary value problems on geometrically irregular regions in (Schwarz, 1869), but was recognized by Dryja & Widlund (1987) to provide a projection operator framework for solving PDEs that satisfies memory locality. It is now the basis for distributed-memory algorithms for solving PDEs with tens of millions of degrees of freedom on hundreds of processors (Kaushik et al., 1998) and of the PDE solver in a major parallel software library (Balay et al., 1998).

The *Method of Conjugate Gradients* was described as a direct method in (Hestenes & Stiefel, 1952), but was recognized by Reid (1971) as an iterative method for large, sparse, well-conditioned problems in linear algebra that does not require workspace for factorization. It is now the basis (through preconditioning) of solutions to linear systems of dimensions in the tens of millions with memory overhead proportional to a small multiple of system size. It has been generalized to Krylov methods for nonsymmetric and indefinite systems (e.g., GMRES (Saad & Schultz, 1986)), and shown to be an effective alternative to direct methods even on well-conditioned dense systems (e.g., arising from integral equations with compact operators). Its property of accessing the matrix only in the form of matrix-vector products makes it amenable to abstract, data structure-neutral implementation.

A common theme in these examples (Table 1) is adaptation to the limitations of contemporary memory systems. We have the luxury of concentrating on memory system bottlenecks only because processing bottlenecks have been conquered through optimal algorithms such as multigrid, fast multipoles, and fast transforms.

TABLE 1. Classical mathematical constructs reborn as important contemporary algorithms

	invented	recognized	motivation
Conjugate Gradients	1952	1970's	memory capacity
Alternating Procedure	1869	1980's	memory locality
Space-filling Curves	1890	1990's	memory locality

There are many other themes in algorithmic progress in which pre-computational classical results have been revived to meet computational demands. Plate, shell, and beam theories from classical mechanics are the basis of "multi-model" finite element libraries, where extreme aspect ratios make full three-dimensional linear elasticity less suited than its asymptotic counterparts in accommodating modeling complexity. Classical Delaunay and Voronoi tesselations are the basis of important contemporary algorithms for constructing unstructured grids and discretizing upon them in accommodating geometric complexity.

The needs of large-scale computational science have also been met from technology that is contemporary, but has its origins in other fields. For instance, data compression methods for network and archival technologies now reduce transmission and storage complexity in scientific codes. Immersive technology from pilot training now reduces the complexity of the human-computer interface.

3. Trends in Algorithmic Research

The preceding examples suggest that many algorithmic breakthroughs are found and adapted at the time of need or opportunity, not necessarily spontaneously created at that time. The organizing principles for this contribution on future trends in algorithmic research are therefore: (1) identification of needs and opportunities, in view of trends in application and architectural (hardware and software) boundary conditions; (2) adaptation of current algorithms; and (3) searching for algorithms to borrow.

3.1. TRENDS IN APPLICATIONS

The important application boundary conditions are well articulated by framing documents from the federal agencies, such as the "Grand Challenge" program (NSF, 1996), the Accelerated Strategic Computing Initiative (DOE, 1997) and the interagency High Performance Computing and Communications initiative (FCCSET, 1992). These are characterized by: first-principles or multiple-scale modeling, billions of degrees of freedom, real-time simulations (in some cases, e.g., to control an on-going process), parametric studies (using analysis codes as fluently as spreadsheets), and immersive interaction. Essentially, applications should be as large, run as fast, and their data sets be as rapidly transmitted and post-processed as possible. In these regards, there is no concern that the customer will become satisfied anytime soon!

3.2. TRENDS IN HARDWARE

The hardware architecture boundary conditions are easily extrapolated by the technology roadmap of the Semiconductor Industry Association (1998). The following approximate trends are reasonably steady: processor speed doubles about every 3 years[1], memory and disk access time halve about every 10 years, memory and disk capacity quadruple about every 3 years, program memory consumption rises to fill capacity (invested in friendlier user interfaces and increased resolution), larger and cheaper means slower at all levels of storage, memory is still the most expensive component of a supercomputer, and disk is two orders of magnitude cheaper than memory

[1]Moore's Law, which states that transistor count on a chip quadruples about every three years, is often misapplied directly to logic speed. In practice, the marginal benefit of the additional transistors to processing rates saturates. A more careful projection of the growth rate of processor speed is approximately a factor of 2.8 in three years.

per byte but three orders of magnitude slower. Essentially, the premiums on anticipating data requirements in advance and on data locality increase dramatically due to the increasing latency of memory system relative to processor capability. These industry-wide trends in commodity microprocessor architecture also apply qualitatively (though with different constants) to special-purpose supercomputer architecture, which is now primarily based on commodity processor and memory components. Cray's own homepage shows that their top-of-the-line Cray T3E suffers a single memory latency of 252 floating point operations — more than 25 times worse than the same dimensionless measure of memory latency for the CDC 7600, which preceded it by 25 years. Whereas the number of floating point operations lost on a cache miss is in the hundreds, the number lost on a message start-up in a contemporary multiprocessor is in the thousands. For the HTMT Petaflops machine, memory latencies are forecast to be in the 10,000's of floating point operations (Sterling et al., 1997). If 1 Pflop/s (10^{15} floating point operations per second) is achieved with one million of today's 1 Gflop/s processors, in*ter*processor latencies will rise with the broadening network. If, instead, it is achieved with ten thousand of tomorrow's 100 Gflop/s quantum logic processors, in*tra*processor latencies will rise with the deepening memory hierarchy. With the increasing cost of accessing memory accompanying either design, there will be no relief from the necessity of user management of storage anytime soon!

3.3. TRENDS IN THE SOFTWARE ENVIRONMENT

The major software architecture boundary conditions are more qualitative than quantitative, and more informally accepted than officially articulated. They include: multilevel design (on a scale from "opaque" to "transparent", in order to provide an appropriate impedance match to users ranging from computationally naive to computationally sophisticated); object-oriented design (at least encapsulation, abstraction, and polymorphism) at all but the lowest levels[2]; adherence to standards and portability across all computers that support the model of message-passing communication through the MPI standard; extensibility to multi-threaded environments; and incorporation of intelligence.

The last two points deserve particular elaboration in the context of computational science. Computational science environments of the near future will offer interactive, immersive visualization via multithreaded memory

[2]For scientific codes, it is well documented that too much performance is left on the table when lowest level scalar arithmetic operations are not bound at compile time. Furthermore, scientific codes are sufficiently limited in the way that their components interact that very little useful expressivity and flexibility of object orientation is sacrificed together with the fourth object-oriented pillar of inheritance.

snooping and computational steering through interactive modification of the flow of control of a program, accompanied by memory modification. Components of multithreaded solvers may run asynchronously, both with respect to the threads enabling these interactive tasks, and with respect to other components of the same multithreaded solver.

"Intelligence" implies more than the forms of adaptivity that are routine today, such as optimization of a relaxation parameter, discovery and retention of an approximate null space for deflation purposes on subsequent systems, or scheduling recomputation of an expensive Jacobian or a Hessian in an inexact Newton or optimization method. These are nontrivial tasks when posed as optimization problems in their own right, but heuristics exist that permit their automation, given relatively modest observations of the local behavior of a code. A typical solver has literally dozens of such tuning parameters associated with it, apart from the parameters of the discretization and the parameters of the physical system (see Section 5.2). The intelligent drivers for future algorithms will take advantage of more global knowledge of a problem-algorithm-architecture class, using information mined from a data base of similar cases. Newly engineered systems are often similar enough to previously engineered systems, that experience with previous systems is valuable. Today this experience resides in the people who *run* the codes, but a team of people is a fragile, distributed, and unreliable reservoir of such experience, and for some purposes it is better replaced by an intelligent agent.

3.4. ALGORITHMIC RESPONSE

In response to the changing climate in applications and architecture, algorithms will adapt. Algorithms already adapt, of course; scientific codes contain a wide variety of compile-time and run-time options, or "tuning parameters," some of which are set automatically. Section 5 reviews the state of the art of one such tunable family, which has already been stretched to close to the limits of its scalability by synchronization latency on the 512-node message-passing supercomputers of 1998. First, we review the central emerging issue in algorithms for high-performance computers.

4. The Holy Grail of Latency Tolerance

Given the demands of applications for more memory, and the weakening ability of the hardware to provide fast access to arbitrary elements of memory, the memory model upon which algorithm performance evaluation is based must evolve from the outdated illusion of a flat memory space to the present and future reality of hierarchical distributed memory.

We may distinguish between the *vertical* latencies within a processor-memory node arising from the hierarchical aspect and the *horizontal* latencies between nodes arising from the distributed aspect. The former is nearly universally under hardware control in today's processors. The latter is either under user control in the message-passing model, or under some combination of hardware and system software control in the cache-coherent shared-memory model. We may also distinguish between the startup (size-independent) part of a data access (when the data itself is ready) and the synchronization cost (when the data is not ready). For many purposes in the analysis of algorithmic performance, these various types of latencies "look the same." The same locality-enhancing blocking practices that reduce vulnerability to message-passing latency on a parallel architecture tend to reduce vulnerability to cache-miss latency on any architecture. In assessing the penalty of message-passing latency, startup and synchronization effects are summed, and special diagnostics are generally required to separate them. Therefore, except when otherwise explicitly mentioned, we use the term "latency" generically in this chapter.

Whenever the latency potentially inherent in a data dependency cannot be removed by removing the data dependency, itself, the latency must be tolerated by:

– Arranging for temporal locality of memory references: once an operand is cached on a processor, use it as many times as practical before sending it "down" or "out".
– Arranging for spatial locality of memory references: when an operand needs to be moved "up" or "across", fill up the slots in the same packet with other operands that will be required soon.
– Arranging for other things to do while stalled for data: perform lightweight multithreading and/or extra work (relative to optimal work complexity) per data transfer-laden "cycle" if it reduces the total number of such cycles.

Fortunately, these arrangements are easier to make for a typical PDE solution algorithm than for many less regular large-scale computational tasks. We claim that certain scalable algorithms for general purpose PDE simulations in use today will *in theory*[3] survive in the future application and architecture climate. But, in practice, we must simultaneously strive improve adaptivity to the application and tolerance of memory latency of the architecture. To these ends, everything in algorithmics and scientific software engineering should be placed on the table as negotiable.

[3] "In theory there is no difference between theory and practice. In practice, there is." — Yogi Berra (1925–), baseball player and philosopher.

"Unstable" methods may get new life, particularly in high precision environments (e.g., 128-bit floats). Optimal arithmetic complexity may need to be sacrificed in favor of optimal overall solution time. Data locality considerations may need to be given priority over functionally clean, modular coding practices. Computational scientists may need to learn about many new areas — architecture, software engineering, control theory, optimization — not to expand their application domain, but just to run in place. Despite the potential discomfort implied by some of these heresies, PDE solution algorithms are still the "easy" problems of the next 10–20 years since ample concurrency with locality is readily exposed. Hardware and systems software will not necessarily expose all of the mathematically available concurrency or locality because allowable program transformations are too restrictive, so users must be given tools to help expose it.

The latency tolerance techniques available to architects fall into two categories (Culler et al., 1998): *amortization* via block transfers that pay the overhead only once for several pieces of data, and *covering* via precommunication (anticipating future needs for data so that they arrive when needed), proceeding past an outstanding communication in the same thread (also known as "hit-under-miss"), and multithreading. The requirements for all of these techniques are excess instruction-level or thread-level concurrency in the program (beyond the number of processors being used) and excess capacity in the memory and communication architecture. When there is excess concurrency, the temporal behavior of the program can be improved at the expense of extra spatial resources.

All of the strategies mentioned above, which are typically under hardware or system software control, have counterparts under user control. In a sufficiently rich programming environment, the algorithmicist can express to potential advantage the "when" and the "where," in addition to the "who" and the "what," of a given algorithm. In addition, algorithmicists have a unique strategy, not available to architects by definition: reformulation of the problem to create concurrency. There are mathematical limits to this opportunity, which are defined by the error to be tolerated in the final result and/or the rate of convergence to be tolerated in achieving it. The opportunity stems from the observation that not all nonzero data dependencies are created equal and some can be dropped or deferred. We may accept extra outer iterations if doing so greatly reduces the latency of the iteration body.

Adapting the algorithm directly requires an understanding of the convergence behavior of the problem, especially the dependence of the convergence behavior on special exploitable structure, such as heterogeneity (region-dependent variation) and anisotropy (direction-dependent variation).

5. Parallel Implicit PDE Solvers and the Newton-Krylov-Schwarz Method

The illustrations of algorithmic adaptivity and the motivation for the innovative features of this chapter come from the implicit solution of nonlinear systems of PDEs from computational aerodynamics. In this section we make parallel complexity estimates for the pseudo-transient Newton-Krylov-Schwarz (ΨNKS) family of algorithms, which is designed for parallel implicit nonlinear applications. Space limitations do not permit self-contained development of ΨNKS methods, but many references are available. For an survey article with an emphasis on software aspects, see (Gropp et al., 1998). For a theoretical introduction to ΨNKS, see (Kelley & Keyes, 1998). For a focus on scalability and large-scale problems, see (Kaushik et al., 1998). Finally, (Cai et al., 1998) is a reasonably self-contained discussion of a model application.

If $\mathbf{u}^\ell$ represents state variable (unknown) vector at the ℓ^{th} step in an iterative process converging to the solution of $\mathbf{f}(\mathbf{u}) = 0$, where $\mathbf{f}$ is a vector of residuals of conservation laws, a pseudo-transient implicit method solves successively

$$\frac{\mathbf{u}^\ell}{\Delta t^\ell} + \mathbf{f}(\mathbf{u}^\ell) = \frac{\mathbf{u}^{\ell-1}}{\Delta t^\ell},$$

for timesteps Δt^ℓ, $\ell = 1, 2, \ldots$, such that $\Delta t^\ell \to \infty$ as $\ell \to \infty$. Each step $\mathbf{u}^\ell$ is obtained from an Newton iteration with Jacobian matrix $(\frac{1}{\Delta t^\ell} I + \frac{\partial \mathbf{f}}{\partial \mathbf{u}})$. These linear systems are solved via a matrix-free Krylov method preconditioned by a Schwarz method. Pseudo-code for the complete algorithm is shown in Fig. 1. Operations shown in uppercase customarily involve global synchronizations.

For a three-dimensional problem with a discrete size of N, domain-decomposed into P subdomains, each subdomain containing N/P points, the concurrency is pointwise, $\mathcal{O}(N)$, for all aspects of the problem except for the Schwarz preconditioner, where it is subdomainwise, $\mathcal{O}(P)$. The communication-to-computation ratio is surface-to-volume (apart from global reduction steps), namely $\mathcal{O}\left((\frac{N}{P})^{-1/3}\right)$. Therefore, it remains constant if N and P are scaled in proportion. The communication is mainly nearest-neighbor, but convergence checking, orthogonalization/conjugation steps, and any hierarchically coarsened problems included in the Schwarz preconditioner add nonlocal communication. Depending upon implementation, the synchronization frequency is often more than once per concurrent mesh-sweep, up to K, the Krylov dimension, namely $\mathcal{O}\left(K(\frac{N}{P})^{-1}\right)$. If K can be kept independent of problem size and granularity this, too, is constant. Typically, K grows slowly with problem size.

```
do l = 1, n_time
    SELECT TIME-STEP
    do k = 1, n_Newton
        compute nonlinear residual and Jacobian
        do j = 1, n_Krylov
            doall i = 1, n_Precon
                solve subdomain problems concurrently
            enddoall
            perform Jacobian-vector product
            ENFORCE KRYLOV BASIS CONDITIONS
            update optimal coefficients
            CHECK LINEAR CONVERGENCE
        enddo
        perform DAXPY update with robustness conditions
        CHECK NONLINEAR CONVERGENCE
    enddo
enddo
```

Figure 1. Pseudo-code for the ΨNKS method

5.1. PARALLEL COMPLEXITY ESTIMATION FOR BULK-SYNCHRONIZED STENCIL COMPUTATIONS

Given complexity estimates of the leading terms of the concurrent computation (taking intra-node memory latencies into account), the communication-to-computation ratio, and the synchronization frequency, as well as a model of the architecture including internode communication (namely, the network topology and protocol reflecting horizontal memory structure), we can formulate optimal concurrency and optimal execution time estimates. This can be done on a per-iteration basis or overall (by taking into account any granularity-dependent convergence rate). Let there be n grid points in each direction, with storage $\mathcal{O}(N) = \mathcal{O}(n^3)$, and p processors in each direction, with total processors $P = p^3$, giving memory per node requirements of $\mathcal{O}(N/P)$. The execution time per iteration is An^3/p^3, where coefficient A lumps together factors including number of components at each point, number of points in stencil, number of auxiliary arrays, and the reciprocal of the effective per node computation rate. With n/p grid points on a side of a single processor's subdomain the neighbor communication per iteration, apart from latency, is Bn^2/p^2. The cost of an individual global reduction is assumed to be logarithmic or fractional power in p. The cost per iteration, $C \log p$ or $Cp^{1/d}$, includes synchronization frequency as a factor. Coefficients A, B, and C are all expressed in the same dimensionless units, for instance, multiples of the scalar floating point multiply-add.

For a 3D stencil-based computation with tree-based global reductions, the total wall-clock time per iteration is

$$T(n,p) = A\frac{n^3}{p^3} + B\frac{n^2}{p^2} + C\log p \ . \tag{1}$$

The optimal p is found at $\frac{\partial T}{\partial p} = 0$, or $-3A\frac{n^3}{p^4} - 2B\frac{n^2}{p^3} + \frac{C}{p} = 0$, or (with $\theta \equiv \frac{32\cdot B^3}{243\cdot A^2 C}$),

$$p_{opt} = \left(\frac{3A}{2C}\right)^{1/3} \left(\left[1 + (1-\sqrt{\theta})\right]^{1/3} + \left[1 - (1-\sqrt{\theta})\right]^{1/3}\right) \cdot n \ . \tag{2}$$

It is apparent that p can grow with n without "speeddown." In limit as $B/C \to 0$, $p_{opt} = (3A/C)^{1/3} \cdot n$.

With this value the optimal execution time in the limit as $B/C \to 0$ is

$$T(n, p_{\mathrm{opt}}(n)) = C\left[\log n + \frac{1}{3}\log\frac{A}{C} + \mathit{const.}\right] \ . \tag{3}$$

(This analysis is on a per iteration basis; fuller analysis would multiply this cost by an iteration count estimate, which may generally depend upon n and p and affect the optimal scaling. The optimal execution time is directly proportional to coefficient C, which contains the synchronization latency and synchronization frequency.)

The estimates above are based upon Additive Schwarz preconditioning of the Jacobian $J = (\frac{1}{\Delta t^\ell}I + \frac{\partial \mathbf{f}}{\partial \mathbf{u}})$. The preconditioner for J is built out of (approximate) local solves on (overlapping) subdomains. Figure 2(a) shows a square domain Ω divided into nine square subdomains Ω_i, which overlap each other to a depth of $3h$, where h is the width of a subinterval. The overlapped domains are denoted Ω_i'. Two horizontally adjacent subdomains Ω_i and Ω_j are shown in Fig. 2(b). The right solid boundary of Ω_i coincides with the left solid boundary of Ω_j, but the domains are pulled apart to show the nearest neighbor exchange buffers that each one fills for the other prior to performing a subdomain solve its overlapped region. If R_i and R_i^T represent Boolean gather and scatter operations, mapping between a global vector and its extended i^{th} subdomain support then the action of the Additive Schwarz preconditioner can be specified algebraically as $\sum_i R_i^T \tilde{J_i}^{-1} R_i$, where, in turn, subdomain Jacobian block J_i is $R_i J R_i^T$. The "~" in the formula indicates that an approximate solution with the local Jacobian block, rather than a full direct solution, may be carried out in parallel on each extended subdomain.

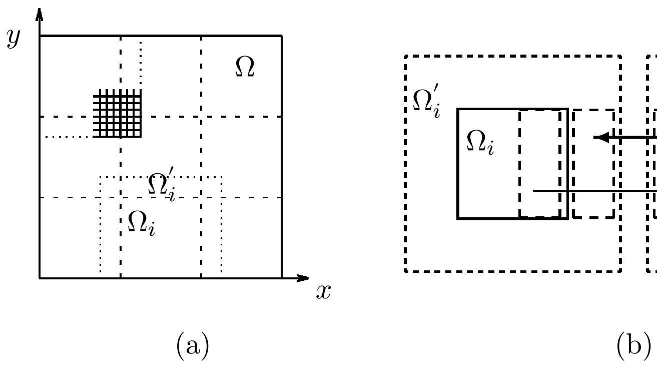


Figure 2. Overlapping Additive Schwarz domain decomposition and detail of overlapped data buffering

Iteration count estimates from the Schwarz theory for elliptic PDEs (Smith et al., 1996) are put in the context of point Jacobi and nonoverlapping subdomain block Jacobi in Table 2. Preconditioned Krylov iterative methods typically converge in a number of iterations that scales as the square-root of the condition number of the preconditioned system. Observe that the Schwarz methods converge at a rate independent of the size of the discrete system, and a 2-level version of the Schwarz method (Dryja & Widlund, 1987) converges at a rate independent of the number of processors. This optimal convergence rate is often nearly achieved even without a 2-level preconditioner for parabolic problems, including the pseudo-transient parabolization of an elliptic problem.

TABLE 2. Iteration counts for preconditioned Krylov iteration on elliptic problems

Preconditioning	*2D*	*3D*
Point Jacobi	$\mathcal{O}(N^{1/2})$	$\mathcal{O}(N^{1/3})$
Subdomain Block Jacobi	$\mathcal{O}(N^{1/4}P^{1/4})$	$\mathcal{O}(N^{1/6}P^{1/6})$
1-level Additive Schwarz	$\mathcal{O}(P^{1/2})$	$\mathcal{O}(P^{1/3})$
2-level Additive Schwarz	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Armed with these convergence estimates, we can repeat the estimates of p_{opt} and T_{opt} above on an overall execution time basis, rather than a per-iteration basis (see Keyes et al. (1998)).

5.2. ALGORITHMIC TUNING

The goal of adaptivity is to take advantage of data-dependent features that are known at problem definition or discovered during runtime to minimize some combination of computational resources required to arrive at a result of given quality. A disadvantage of adaptive algorithms is that it may be difficult for inexperienced users to tune the parameters that give them their adaptive power. A mistuned algorithm may perform much less efficiently than a nonadaptive version with conservative defaults. The ΨNKS family of methods contains numerous parameters, which can be associated primarily with different loop levels of the pseudo-code in Fig. 1.

The outermost pseudo-transient continuation relies on a sequence of timesteps, Δt^ℓ, and a steady-state residual tolerance for convergence.

The inexact, matrix-free Newton iteration requires a Fréchet differencing parameter for the Jacobian-vector products, a damping method, a nonlinear residual tolerance for convergence at each time step, a recovery mechanism (feeding back on timestep selection) in case Newton's method fails to converge, and possibly additional continuation devices (e.g., continuation in boundary condition enforcement or dimensionless physical parameters).

The Krylov iteration requires a maximum Krylov subspace dimension, an orthogonalization frequency (with direct effect on coefficient C in (3)), a preconditioner refresh frequency, linear tolerance for convergence at each Newton step, and a failure recovery mechanism in case the Krylov method stagnates.

The innermost Schwarz preconditioner contains parameters that directly affect the overall concurrency in the algorithm and the communication cost of each iteration. Subdomain granularity P is directly related to processor granularity. Subdomain overlap width enters into B in (1). Subdomain solution quality is a leading contributor to the work per iteration in A of (1). The presence of a coarse grid and its density affect A and C and are not considered in the simple estimates leading to (3). Finally, partitioning governing subdomain orientation and aspect ratio in nonuniform problems can have a strong effect on the quality of the linear conditioning and therefore the number of iterations.

Besides the obvious adaptation of the Schwarz preconditioner to the granularity of the architecture, there are several choices and trade-offs in the implementation of an ΨNKS code which can effect substantial rebalancing between coefficients A, B, and C, or the total number of iterations. We can think of these decisions as architectural tuning parameters, including: inspector/executor trade-offs, dispatch/merge trade-offs, buffer/recopy trade-offs, and store/recompute trade-offs. Blocking parameters and data orderings for cache, and loop unrolling parameters for registers also substantially affect A. In Section 9, we bring out two additional architectural tuning parameters: the number of user-managed threads per processor, and frequency of deferred synchronization for convergence testing and timestep selection.

Some of these parameters, like timestep, Newton damping parameter, and Fréchet differencing parameter, are continuous and supported by sound theory or heuristics that allow their selection to be automated. Others, like subdomain partitioning, are presently chosen intuitively. Still others, like the use of a coarse grid and the amount of overlap in a Schwarz preconditioner are tuned by trial and error. Many examples of such tunings in the context of a 3D transonic Euler flow code are given in (Gropp et al., 1998).

More systematic choices can be made in problem and architecture-specific environments by exhaustive search or more efficient parallel direct search methods (Dennis & Torczon, 1991). We mention the PHiPAC project at Berkeley (Bilmes et al., 1998) and the Atlas project at the University of Tennessee (Whaley & Dongarra, 1998) as extreme examples of architectural tuning for the blocking of kernel loops. PHiPAC, for instance, reports attainment of 90% or better of theoretical peak for the BLAS3 routine DGEMM on all major RISCstations, sometimes improving upon the vendor implementations. The application of optimization techniques with execution time as the objective function and the parameters mentioned above as the design space would automate the adaptivity, relieving users of it though possibly at considerable resource cost.

Besides optimization, we mention control theory as a likely source for automated parametric tuning of dynamically adaptive computations. Söderlind (1998) has pointed out the disproportionate amount of effort that goes into stiff ODE integrators compared to the effort that goes into timestep selection for the integrators, even though many of the practical difficulties that with which the integrators must cope arise from poor timestep selection analogous to rudimentary "deadbeat" control. More sophisticated strategies are already available, but relatively uknown in the computational science community.

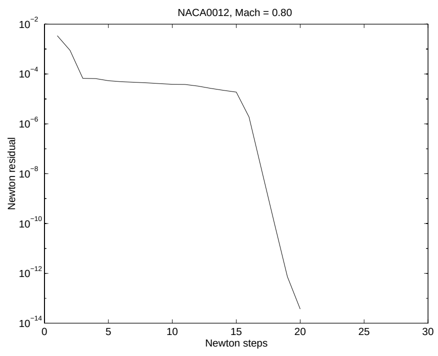
6. Adaptivity to Problem Nonuniformity

The theoretical scalability of ΨNKS methods, a representative bulk synchronous SPMD application, was presented in Section 5.1. This analysis shows that on a network with sufficiently fast global reductions, ΨNKS can be within a logarithmic factor constant efficiency, as problem size and processor number are scaled in proportion. It is not easy to improve upon this scalability for any parallel implicit method for systems of elliptic boundary value problems. Nevertheless, it is very easy to imagine improving upon the total execution time of a global Newton method. The example of this section is only one of a plethora of problems with nonuniform characteristics throughout the spatial domain that could be employed to illustrate this point.

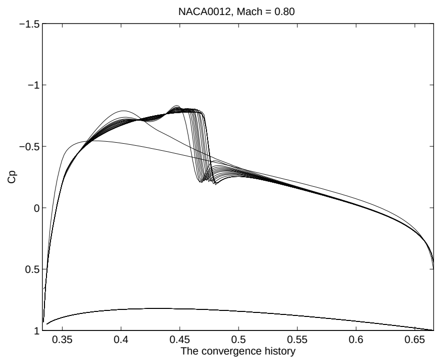
As an example of algorithmic adaptivity to application nonuniformity, consider the nonlinear full potential model for steady flow over a transonic flow over a smooth two-dimensional airfoil described in (Cai et al., 1998). Figure 3(a), from (Cai et al., 1998), shows the norm of the nonlinear residual for a finite element discretization of this problem, resolved with approximately a quarter million degrees of freedom and solved with a straight NKS method (no pseudo-transient continuation) in twenty iterations. During the first three iterations, substantial progress is made towards the final solution from a uniform flow initial iterate. During the final five iterations, Newton's method makes rapid progress towards the machine-precision convergence tolerance attainable for this discrete problem. Between iterations 3 and 15, very little progress is made. Figure 3(b) reveals why. This plot of dimensionless pressure distributions over the upper surface of the airfoil, with iterations 1 through 20 superposed, shows that during the initial three iterations the distribution evolves rapidly in shape, during the final five iterations, it does not change at all to pixel resolution, and during the middle "plateau" iterations, the shock that forms approximately 2/3rds of the way down the airfoil is moving about one grid point per iteration from the point it sets up until the point where it converges. During this "plateau" phase, the nearly linear ambient flow barely changes; the globally computed Newton updates are nearly zero over most of the domain. In the ODE initial value problem context, where the progress parameter is time, such a phenomenon — a critical mode of the solution holding the remaining modes hostage — is called stiffness. By analogy, in this context where the progress parameter is iteration index, we call this retarding of the global update by a critical feature "nonlinear stiffness."

In analogy with the practice of adaptive refinement, in which a scalar field, the error estimator, indicates that subset of the domain over which a better resolution is needed, we would like to compute a scalar field that indicates the local degree of nonlinearity of a solution iteration. We may use such an indicator to focus extra work on the relevant subdomain in an automatically adaptive manner, just as we would apply h, p, or r refinement in an automatically adaptive manner, thus saving work (and storage) relative to uniform refinement.

Practical use of such a pointwise metric requires that the cost of its evaluation be subdominant to the cost of solving the problem. We propose



(a)



(b)

Figure 3. $||\mathbf{f}(\mathbf{u}^\ell)||$ and $C_p(x)^\ell$ $\ell = 1, 2, \ldots, 20$, for transonic full potential flow over a NACA0012 airfoil

a metric based on absolute ratios of finite differences of Jacobian elements:

$$\tau_i = \frac{||\frac{\partial f_i}{\partial \mathbf{u}}(\mathbf{u}^{k+1}) - \frac{\partial f_i}{\partial \mathbf{u}}(\mathbf{u}^k)||}{||\mathbf{u}^{k+1} - \mathbf{u}^k|| \cdot ||\frac{\partial f_i}{\partial \mathbf{u}}(\mathbf{u}^k)||} \ .$$

We propose the name "tensoricity" for this quantity, defined for each component i, because it is a directional derivative of Jacobian elements, related to the tensor term in the multivariate Taylor expansion upon which Newton's method is based (see Dennis & Schnabel (1973)). Tensoricity uses only by-product norm information; in practice, it would not be recomputed at every step k, but only when the preconditioner for the Jacobian is recalculated. Tensoricity is dimensionless and satisfies the null test, in the sense that τ_i vanishes if f_i is linear in all of the components of $\mathbf{u}$.

Our initial test of the potential utility of tensoricity is in a model problem arising in industrial metal cutting, approximately described by the nonlinear elliptic BVP in the unit square Ω (see (Ierotheou et al., 1998) for a fuller context):

$$-\nabla \cdot (\kappa(u) \nabla u) = 0 \in \Omega \ , \tag{4}$$

$$\text{where} \ \ \kappa(u) \equiv 1/(1 + cu^2) \ , \tag{5}$$

with boundary conditions

$$u(x, y) = A \exp(-\sigma x^2), \ \text{for } (x, y) \text{ on } \partial\Omega \ . \tag{6}$$

This problem provides a tunably steep and narrowly confined Gaussian ridge in the temperature field u at the left end of the interval. Applying

Newton's method (for instance, in the form of NKS) to the subdomain of high tensoricity in the left half of the subdomain, iterating to convergence, using the resulting iterate to help initialize the full domain, and iterating to convergence on the full domain is approximately half the work of solving the global problem with Newton's method. As the nonlinear regions becomes more and more narrowly confined at the left edge of the domain, the ratio of computational work can be made almost arbitrarily low, by confining the first Newton process to a smaller and smaller portion of the domain. The main savings are in the linear system work and in global synchronization steps in the inner Krylov iterations. Jacobian preconditioner formation is also reduced. It is needed globally only at the end of the computation. The total number of Newton steps on the active subregion is virtually the same as the total number of Newton steps on the global problem, without exploiting tensoricity. This observation reinforces the view that the rate of convergence of the global problem is determined by the most "nonlinear" subdomain. Quantitative execution time advantages will be presented elsewhere, in joint work with Gropp, Lai & Palansuriya.

In the literature of computational aerodynamics, there is considerable blurring of attribution of problem difficulty when it comes to nonlinearity. In the Navier-Stokes equations, the principal source of nonlinearity is the first-order advection, which is also the principal contributor of nonsymmetry to the Jacobian. In addition, the nonlinearity of the advection is associated with the near singularity of shocks and the resulting stress on the resolution capabilities of computational aerodynamics discretizations. Often these twin difficulties of nonsymmetry and near singularity have been blamed upon the nonlinearity, even though it is possible generate both of these phenomena independently of nonlinearity. The intrinsic difficulty of nonlinearity is rapidly shifting tangent hyperplane (Jacobian) approximations, of which high tensoricity is symptomatic. There is a need for special attention to all three of the algorithmic problems associated with advective terms, and we hope that tensoricity will permit better isolation of the root cause and more efficient allocation of computational resources via the nonlinear Schwarz methods presented in Section 9.

7. Adaptivity to Hierarchical Memory

As an example of algorithmic adaptivity to the vertical aspects of the memory system, in overcoming cache-miss-related memory latencies, consider the incompressible inviscid flow over an M6 wing described in (Kaushik et al., 1998). Table 3, from (Kaushik et al., 1998), shows a seven-fold serial execution time performance benefit derived from three successively applied memory locality strategies, in an unstructured grid problem of 22,677 vertices (90,708 unknowns) small enough to run on a single IBM P2SC processor. Unenhanced version "1" is the code in its original multicolored vector-based ordering.

TABLE 3. Execution times and improvement ratios for three strategies for locality enhancement and their combinations in an unstructured Euler flow code (run in 4-component incompressible and 5-component compressible formulations) on a 120 MHz P2SC "thin" processor

	Enhancements			Results			
	Field	Structural	Edge	Incompressible		Compressible	
	Interlacing	Blocking	Reordering	Time/Step	Ratio	Time/Step	Ratio
1				165.7s	—	237.6s	—
2	×			62.1s	2.67	85.8s	2.77
3	×	×		50.0s	3.31	65.7s	3.62
4			×	43.3s	3.82	67.5s	3.52
5	×		×	33.5s	4.95	50.8s	4.68
6	×	×	×	22.1s	7.51	32.2s	7.37

The first two enhancements, interlacing and blocking, are well-known in the high-performance computing literature, especially the compiler literature. The third, a greedy edge-reordering applied to a vertex-centered control volume scheme, is intuitive. There may be many additional locality-enhancing transformations available in structured and unstructured computational science codes that are neither well-known nor intuitive, which may need to be discovered more systematically in an optimization process, such as a direct search or genetic process. Such optimization may even ultimately be implementable in optimizing compilers or adaptive runtime systems of the future. However, to automate locality-enhancement through optimization, a scalar objective function is required, which permits determination of whether a given move is favorable or unfavorable, short of actually executing the code.

In an attempt to thus quantify locality, we propose the abstract concept of memtropy, a real scalar function of a sequence of memory references, which can be assumed without loss of generality (for uniformly sized data objects) to be a subset of the nonnegative integers. By design, a set of well ordered references should have lower memtropy than the same set of references ordered poorly. If memtropy is to be a useful concept, it must be shown to correlate monotonically with real-world metrics, like cache miss frequency.

A possible form of the proposed metric is a windowed 1-norm of the cumulative jumps in successive memory references, m_i, $i = 1, 2, \ldots$:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=0}^{i-1} e^{-\alpha j^2} \ |m_i - m_{i-1-j}| \ .$$

We define $m_0 = m_1$ to keep the formula simple while insuring that the first reference m_1 incurs no penalty. The index space is triangular and can be rewritten as

$$\mu = \frac{1}{n} \sum_{j=0}^{n-1} e^{-\alpha j^2} \sum_{i=1}^{n-j} |m_i - m_{i-1-j}| \ .$$

It is important to record the actual memory references rather than the just the size of the jumps between successive references, since a series of n uniformly sized jumps could either bounce back and forth between two locations or visit n distinct locations, and the latter should intuitively be more penalized. The difference term captures spatial locality while the Gaussian window captures temporal locality by weighting recent differences more heavily than event-separated differences.

The metric satisfies the null test by vanishing if all references are the same, is small if all references cycle within a small set, is invariant with respect to translations of the sequence, and in the case of two sequences with a fixed relative pattern of jumps, it is smaller for the sequence with smaller magnitude jumps. The normalization by n tends (modulo α effects), to produce a μ for a periodic sequence of references that is independent of the number of periods. The following examples illustrate the main points of this paragraph. The subscript on the final brace indicates the number of terms, n, in the sequence of memory references whose pattern is within the braces. In the first eight sequences, $\alpha = 1$.

$\mu(\{0, 0, 0, 0, \ldots\}_{128}) = 0$
$\mu(\{0, 1, 0, 1, \ldots\}_{128}) = 1.016$
$\mu(\{3, 4, 3, 4, \ldots\}_{128}) = 1.016$
$\mu(\{0, 4, 0, 4, \ldots\}_{128}) = 4.062$
$\mu(\{0, 1, 2, 3, 0, 1, 2, 3, \ldots\}_{128}) = 2.244$
$\mu(\{0, 1, 2, 3, 3, 2, 1, 0, \ldots\}_{128}) = 1.325$
$\mu(\{0, 1, \ldots, 30, 31, 31, 30, \ldots, 1, 0, \ldots\}_{128}) = 1.730$
$\mu(\{0, 1, 2, 3, 3, 2, 1, 0, \ldots\}_{64}) = 1.329$
$\mu(\{0, 1, 2, 3, 3, 2, 1, 0, \ldots\}_{256}) = 1.316$

More generally, α is a real parameter, $0 \leq \alpha < \infty$, which may be taken as small as zero to remove temporal locality effects. α can be manipulated to mimic cache capacity effects by lengthening or shortening the "memory" of previous references. With $\alpha = 2$, e.g.,

$\mu(\{0, 1, 2, 3, 3, 2, 1, 0, \ldots\}_{128}) = 0.950$

For less trivial examples of memtropy, consider standard 5-point stencil operations applied to the unblocked and the blocked orderings shown below. At each of 64 points in labeled order, including those on the boundary, all defined N, S, E, and W neighbors are read, together with central point, and the central point is written (up to 5 reads and 1 write per point; these two types of references are not differentiated). Considering boundary effects, there are 352 total memory references in each of the differently ordered sets. The memtropies shown under each configuration are for $\alpha = 1$. Blocking leads to smaller differences in successive m_i for stencil points in the interior of the blocks. As the domains grow in size beyond those illustrated, blocking becomes more and more important.

57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

7.516

45	46	47	48	61	62	63	64
41	42	43	44	57	58	59	60
37	38	39	40	53	54	55	56
33	34	35	36	49	50	51	52
13	14	15	16	29	30	31	32
9	10	11	12	25	26	27	28
5	6	7	8	21	22	23	24
1	2	3	4	17	18	19	20

7.383

For an example with greater relevance to the CFD results in this section, consider 5-point stencil operations applied to each of two field components for the segregated and interlaced orderings shown below. At each of 16 points, for each component, both components at all defined N, S, E, and W neighbors are read, together with central point, and central point of each component is written (up to 10 reads, 1 write per component per point). Considering boundary effects, there are 288 total memory references Memtropies shown under each configuration are for $\alpha = 1$. Memtropy shows interlacing to be strongly favorable.

13	14	15	16	|	29	30	31	32
9	10	11	12	|	25	26	27	28
5	6	7	8	|	21	22	23	24
1	2	3	4	|	17	18	19	20

16.736

(25,26)	(27,28)	(29,30)	(31,32)
(17,18)	(19,20)	(21,22)	(23,24)
(9,10)	(11,12)	(13,14)	(15,16)
(1,2)	(3,4)	(5,6)	(7,8)

6.053

As implied by the examples, memtropy is expected to be useful in ordering a given set of references, but not in ranking absolutely the memory-system friendliness of different sets of references. Furthermore, as a machine-parameter-independent measure of memory locality, memtropy cannot be expected to help in fine-tuning for a given architecture. Cache capacity for a fully associative cache can be very crudely reflected in the α, but other associativity policies, replacement policy, cache line size, multilevel effects, cannot be specifically be represented in the primitive form of memtropy described above. Therefore, critical threshold effects in realistic cache trace simulations cannot be represented (see, e.g., (Culler et al., 1998)).

On the other hand, cache simulators, which are in principle capable of performance prediction on a specific memory system configuration, have complementary weaknesses. Cache simulators are fixed to a machine with specific associativity and replacement policies, capacity, line size, etc. Running a simulator is a fairly expensive discrete event simulation, which is, in itself, a Grand Challenge problem in computer science. Running a simulation for a set of memory traces is ordinarily much slower than running the code from which the traces were generated on the hardware of interest. Simulators are useful for insight and for design of new hardware, not for exhaustive searches of optimal memory access patterns. Furthermore, memory traces from a given application program predict real cache histories rather imperfectly — other processes interfere in the cache, including the operating system, itself.

Although locality is difficult to define and therefore to measure, it is a major key to latency tolerance, helping with both amortization and covering. With strong locality, latency can be conquered directly with large block transfers. Sufficient locality permits not only fast out-of-cache implementations but even acceptable of out-of-core implementations (Warren et al., 1998). Strong locality can also be used to cover latency in the following way. Latency can be covered with extra concurrency by switching to another thread. Extra cache memory allows the data for many threads to be co-resident; concurrency can therefore be bought with extra memory. Better locality "looks like" extra cache memory (in the sense that fewer memory access miss the cache) without increasing cost or penalizing performance.

8. Adaptivity to Distributed Memory

As a motivator for algorithmic adaptivity to the horizontal aspects of the memory system, in overcoming synchronization-related message latencies, consider again the incompressible inviscid flow over an M6 wing described in (Kaushik et al., 1998), this time the cache-efficient code of line 5 in Table 3 run in parallel on 128, 256, and 512 nodes of a Cray T3E-900.

Table 4 shows apparently very good adaptivity to varying numbers of processors, as evidenced by the relatively slight degradation in convergence rate with the four-fold increase in concurrency, from 37 pseudo-transient steps to 41. Sustained flop rate per processor decreases only slightly from 71.5 Mflop/s to 68.3 Mflop/s as the surface-to-volume ratio increases with a four-fold reduction in computational volume per processor. Aggregate flop rate for the system increases from 9.1 Gflop/s to 35.0 Gflop/s with the four-fold increase in processor power. However, Table 4 also shows deterioration in fixed-problem-size efficiency, from 94% of a two-fold processor increase to 84% of a four-fold processor increase, for an unstructured tetrahedral grid of 2,761,774 vertices (11,047,096 unknowns). As highlighted, the principal nonscaling feature is the global inner products (whose share of total execution time grows to 9% of execution time for 512 nodes), due mostly to synchronization delays. (Separate runs on 512 nodes, with barriers before the inner products and norms, show that the time required by software overhead and hardware transmission time of these global operations alone is only about 1% of the total execution time, meaning that synchronization delays account for the remaining 8%.)

TABLE 4. Performance data for fixed-problem-size scaling of an unstructured Euler flow code on a T3E

					communication		sustained	sustained
no. procs.	no. its.	exec. time	speed up	overall eff.	inner prod.	halo exch.	Mflop/s per proc.	total Gflop/s
128	37	2,811s	1.00	1.00	6%	3%	71.5	9.1
256	38	1,495s	1.88	0.94	8%	3%	69.7	17.8
512	41	834s	3.37	**0.84**	**9%**	4%	68.3	**35.0**

The inefficiency attributable to synchronization may be reduced in any combination of three ways:

– Reduce penalty of each synchronization step: load balance surface work phase simultaneously with dominant volume work phase.
– Reduce frequency of synchronizations: employ more speculative control for fewer total norms required, and/or less stable numerics in projection steps for fewer inner products.
– Reduce globality of each synchronization step: replace global Newton linearization with regional Newton processes inside of an outer loosely synchronous nonlinear Picard iteration.

Recent work by Karypis and Kumar (1998) is providing a means of dealing with the first strategy listed above. The load imbalance, which grows with the increasing percentage of surface vertices (those whose edges are cut by a subdomain partition boundary) to volume vertices (those whose edges are entirely contained within a subdomain partition) as a fixed-size problem is decomposed for greater concurrency, can be addressed by multiple distribution weights. At present, we balance based on volume (computational) work alone, without regard for the disparity between processors in surface (communication) work. Computation work is dominant, so a single-objective load balance is almost adequate. However, as surface work becomes more important, a secondary objective becomes a candidate for simultaneous balance.

Implementation of the second and third strategies for reducing synchronization inefficiency above invites departures from the traditional bulk synchronous SPMD application model. This call beyond bulk synchronous programming is strengthened by the results of Section 6 and forms the motivation for Section 9.

9. Synchronous and Asynchronous Forms of Nonlinear Schwarz and Schur

The transonic potential example considered in Section 6 shows the algorithmic inefficiency of a global linearization, because of useless work done in areas far from the key nonlinearity that limits the progress of Newton iteration. The incompressible flow example considered in Section 8 shows the architectural inefficiency of frequently synchronization in the linear algebraic methods required to solve the global Newton correction equations. The extension of nonlinear solution algorithms to ASCI-scale platforms (10,000 or more processors) may require that computational resources be deployed less rigidly for greater efficiency. We are at the beginning of exploring two such nonlinear domain decomposition methods: Schwarz-Newton and Schur-Newton.

Schwarz-Newton methods are also known as “nonlinear Schwarz.” Recognizing that degree of nonlinearity is a form of local “stiffness” to which global Newton problems should not be held hostage, Schwarz-Newton methods invert the orders of linearization and decomposition, putting decomposition on the outside, and wrapping a nonlinear Picard iteration around Newton solutions on individual subdomains that cover the overall problem domain. They are seemingly universally relevant in continuum mechanics: in aerodynamics, acoustics, combustion, plasticity, geophysics, and most other nonlinear applications. Many problems in these fields have in common the embedding of a strongly nonlinear near-field problem in an ambient weakly nonlinear far-field problem. Schur-Newton methods, in which the nonlinear iteration is reduced to a complex lower-dimensional interface, are the nonoverlapping analogs of Schwarz-Newton methods, just as Schur complement methods are the analogs of Schwarz methods in the linear theory (Keyes & Gropp, 1987).

9.1. NONLINEAR SCHWARZ AND SCHUR METHODS FOR TWO SUBDOMAINS

Given $u \in \mathcal{R}^n$, coming from a discretization on Ω, and $f : \mathcal{R}^n \to \mathcal{R}^n$, such that $f(u) = 0$ is a governing system (including implicitly posed BCs) on Ω, we define a nonlinear Schwarz method for a two-subdomain partition of Ω as follows. Let Ω be partitioned into overlapping subdomains Ω_1 and Ω_2 that cover the original domain, inducing a partioning of unknown vector u into u_1 and u_2 and of residual vector f into f_1 and f_2. Given initial iterates $u_1^{(0)}$ and $u_2^{(0)}$, iterate $k = 0, 1, \ldots$, until convergence:

Solve $f_1(u_1, u_2^{(k)}) = 0$ for $u_1^{(k+1)}$;
Solve $f_2(u_1^{(k+1)}, u_2) = 0$ for $u_2^{(k+1)}$.

This is a "multiplicative" synchronous version; an "additive" synchronous version is also possible, in which the second equation is replaced with $f_2(u_1^{(k)}, u_2) = 0$, breaking the data dependence upon the output of the first equation. Both versions are synchronous since the $k+1^{st}$ iterates are based upon the most recently available data from the current or previous iterations, whether sequentially or concurrently undertaken. Let Ω_1 be a subdomain drawn reasonably compactly around the strongly nonlinear region, and Ω_2 include its complement. In the multiplicative version, all processors could be first assigned to Ω_1, and then remapped to Ω_2, if the individual stages were iterated sufficiently many times to justify the dynamic repartitioning. In the additive version, processors could be allocated the respective subdomain problems based on a load balancing that took into account total work (cost per iteration and number of iterations) between synchronization stages. Each subdomain solution process on which Newton iterations take place between Schwarz updates can be further partitioned for parallel solution by NKS. Thus, we can have data parallelism within the task parallelism of the separate subdomain solutions. The region of overlap region can coincide with an entire subdomain; i.e., subdomain Ω_2 can be the entire domain. In addition to all of the algorithmic parameters requiring specification in the individual NKS subdomain contexts, there is an interesting new parameter for theorists to explore: the convergence tolerance of the Newton methods, and the effect of incomplete convergence in early iterations upon the progress of the Picard iterations.

The Schur-Newton method (described by different names in Lai, et al. (1997; 1998)) for two subdomains is similar. Let the same Ω be partitioned instead into nonoverlapping subdomains Ω into Ω_1 and Ω_2, with bounding curve Γ, inducing partionings of u into u_1, u_2, and u_Γ and of f into f_1, f_2, and f_Γ. Given initial iterates $u_1^{(0)}$, $u_2^{(0)}$, and $u_\Gamma^{(0)}$, iterate $k = 0, 1, \ldots$ until convergence:

Solve $f_\Gamma(u_1^{(k)}, u_2^{(k)}, u_\Gamma) = 0$ for $u_\Gamma^{(k+1)}$;
where $f_i(u_i^{(k)}, u_\Gamma^{(k)}) = 0$ for $i = 1, 2$.

As with Schwarz-Newton, each subdomain solve, regarded as coarsely task-parallel, can be further partitioned for parallel solution by inner NKS (or inner Newton-Schur) methods. The Jacobian matrix for the system condensed to u_Γ, $\partial f_\Gamma / \partial u_\Gamma$, is dense. Its action on Krylov vectors can be evaluated by fully converged subroutine calls to Newton methods in adjoining subdomains. It can be preconditioned by Broyden (or Broyden-Schubert) updates. This is a straightforward nonlinear analog of the Schur complement method; it reduces identically to block Gaussian elimination of the Jacobian matrix down to the interface unknowns if $f(u)$ is linear. Based on the extensive theory for linear Schur complement methods (Smith et al., 1996), better preconditioners may be derived from this limiting observation.

9.2. USER-SPECIFIABLE THREADS IN COMPLEX IMPLICIT NONLINEAR METHODS

Practical nonlinear methods (inexact, modified, quasi Newton) are somewhat “dirty” in comparison to textbook methods. The Jacobian and/or preconditioner matrices for it are frequently built from crude, inexpensive discretizations (relative to the discretization of $\mathbf{f}$) and not necessarily frequently updated. The practical reason for this is that Jacobian construction and the construction of its (approximate) factors can easily dominate the computational complexity (both arithmetic and communication) of the Newton process. It is understood theoretically and observed experimentally that approximate and/or “stale” Jacobians can be used in place of the true Jacobian, with a net benefit in time-to-convergence, after the cheaper iterations and the convergence rate penalty trade-offs are accounted for.

Since Jacobian blocks are computationally intensive and rarely needed urgently, we are motivated to defer their computational and recomputation to a background mode, off the critical path of the execution. The Jacobian can be recomputed lazily, while the processor or the memory system or the network is stalled on some critical-path computation. Many other noncritical-path tasks could also make use of stalled resources. However, in codes written for flat memory systems, these tasks are almost invariably placed on the critical execution path.

In a typical nonlinear implicit method, the minimal critical path is:

```
..., solve, bound_step, update, solve, bound_step, update, ...
```

"Off the path" tasks include: Jacobian refresh, convergence testing, continuation and algorithmic parameter adaptation, interprocessor communication and disk I/O, visualization, filtering, compression, data mining, etc. Some of these "off the path" tasks enjoy considerable locality. Parallelizability of "naked" sparse linear problems may lead to unrealistically pessimistic conclusions about the scalability of implicit methods. However the full-scale simulations requiring the largest parallel resources can typically cover the latencies of the critical path solvers with many types of useful work — work that would be required anyway, but that may, from a purely algorithmic perspective, be executed concurrently with the critical path work with very little penalty in terms of overall convergence rate or parallel overhead.

There are two reasons why the critical path is unnecessarily burdened with such tasks in today's codes: (1) a lack of experience in thinking asynchronously, and (2) lack of programming model support. Programming models for parallel implicit solvers of the future will support multithreaded execution — at least at the software level. (Direct hardware support of multithreading has also arrived in the Tera machine, and will become more widespread.) Combined with a thread-safe message passing system, this environment will provide many opportunities to exercise the strategy of both covering and deferring synchronization (the second strategy at the end of of Section 8), endowing implicit solvers with considerably more latency tolerance than even the best non-threaded implementations currently enjoy.

Given the availability of threads for partitioning of the tasks local to each processor to those that are on the critical path and those that are not, we can further exploit this environment for greater latency tolerance of the critical path code by following the third and final strategy at the end of Section 8: reducing the globality of the synchronizations that do, of necessity, still occur. This leads us to our final recommendation that classical asynchronous algorithms be considered for future nonuniform applications on distributed architectures.

Asynchronous methods have a long history in parallel computing. Chazan and Miranker (1969) showed that the linear fixed point iteration, $x \leftarrow Ax + b$, may be solved asynchronously by partitioning the elements of x and the corresponding rows of A and b and updating each partition of x based on the best currently available values, but not necessarily the most recently computed values anywhere in the memory space. Each value of x is migrated at some nondeterministic rate from the processor where it is updated to the processors where it is consumed. (The set of processors where it is consumed is determined by the column sparsity structure of the rows of A on other processors; if A is dense, it is consumed everywhere.)

Chazan and Miranker showed that there are examples for which such asynchronous ("chaotic") relaxation diverges, and provided a theory for the case in which $\rho(|A|) < 1$ and the degree of staleness of off-processor unknowns is bounded. (This rules out the case, for instance, in which one processor fails ever to update and broadcast its local unknowns to those that depend upon them.)

Miellou (1975) generalized the asynchronous theory to nonlinear fixed points: $x \leftarrow F(x)$. This is sufficiently general to include Newton's method. He provided theoretical convergence results for the case of "contracting" operators, namely operators $F(x)$ with contracting Lipschitz matrices: $|F(x) - F(y)| \leq A|x-y|$ for some nonnegative A such that $\rho(A) < 1$.

Baudet (1978) relaxed various theoretical hypotheses of the earlier papers and performed experiments on a 6-processor C.mmp at CMU; and so forth. Bersekas (1990) is a book-length monograph on aynchronous methods.

We do not need to contemplate full asynchrony to derive a latency-tolerant benefit from such relaxed-synchrony methods. It is sufficient for distributed-memory purposes to employ pipelined deferred synchronous methods, in which the remotely owned values arrive not at random intervals, but at staged intervals, depending upon their effective network distance from a given consuming processor. Generalizations of this relaxed synchrony are straightforward special cases of fully asynchronous fixed point methods. Whether there are deferred synchrony Krylov methods is a different question of potentially major significance.

Newton-Krylov-Schwarz and Schwarz-Newton methods can both be cast as global fixed point iterations, but the hypotheses of the convergence theorems cited above will generally be unverifiable in important problems. Experience is needed. Proof of convergence is not especially relevant, since *fast* convergence is needed to justify the use of these techniques in high-performance applications. Physical intuition and experience will ultimately guide use of the methods. Polyalgorithms will emerge, in which the opening game is played with Schwarz-Newton methods, and the endgame with more rapidly convergent Newton-Krylov-Schwarz methods. The object-oriented software environment will make such fundamental changes of context within the same execution sequence easy to contemplate.

10. Selected Ripe Agendas in Algorithms

Summarizing our attempt to extrapolate algorithmic progress, based upon the easy to extrapolate gap between applications demand and architectural availability, we believe that support for highly nonuniform applications and high latency architectures will be critical. Both aspects will benefit from programmer-managed multithreading, at a high level. (System-software and hardware-managed multithreading at lower levels is also likely to be strongly beneficial.) Both aspects will also benefit from algorithms that require the concurrent processes to synchronize less frequently than is customary at present.

To take advantage of the considerable new flexibility in programming in such an environment, it is necessary to have new quantitative metrics to guide the allocation of work to processes and threads and new metrics to guide the allocation of data items to distributed, hierarchical memory. The examples of new metrics we have given herein are meant to be suggestive. We hope that they will be examined, improved upon, and supplemented with many others.

We note that even traditional forms of algorithmic adaptivity (e.g., grid and discretization adaptivity) have proved challenging to accommodate in parallel implementations because of the dynamic load-balancing problem. Dynamically balanced load tends to be difficult to prescribe cost-effectively and may be difficult to support in underpowered communication networks.[4] Less synchronous styles of programming take a little of the heat off of the quality of the load balancing, but only a little, since we still expect bulk synchronous methods to be employed within larger asynchronous tasks. Multiobjective dynamic load balancing remains high on the algorithmic agenda.

We expect much of the software infrastructure required for solution algorithms to be developed for other layers of the overall problem-solving environment. For instance, parallel I/O, visualization, and steering will be accomplished off the critical path for computation, and will pioneer portable multithreaded asynchronous features. Application programmers will use these features to go beyond monolithic models and discretizations, to multi-models and multi-discretizations (both conceptually, and in terms of practical software engineering).

As the complexity of solvers grows, we expect more built-in use of control theory and optimization to perform automated parameter selection in algorithms. With the advance of meaningful performance metrics, formal optimization techniques may permeate lower levels of the software — run-time systems and compilers.

[4]If we have failed to mention the problems of limited bandwidth and concentrated only on problems of high latency, it is primarily because we have not paid enough attention to the load balancing problem. In any reasonable PDE solution process, surface-based communication will always be subdominant to volume-based computation, but volume-based communication may also become important in the load balancing process. Fortunately, high bandwidth does not seem to be as daunting to achieve on future architectures as low latency.

Finally, expect to see more "intelligence" in tomorrow's solvers — intelligence that goes beyond the forms of routine adaptivity available today, which is used to make local decisions. Tomorrow's algorithms will access more global knowledge of a problem-algorithm-architecture class, using information stored from previous iterations of a given execution or mined from a data base of similar completed executions. Many fields that are considered distant cousins to numerical analysis in the world of computer science today will heavily be drawn upon in support of the computational aerosciences — and the computational sciences — of the 21st century.

Acknowledgments

The author wishes to acknowledge the valuable contributions to this chapter — codes, examples, and discussions — from Dinesh Kaushik (ODU), Choi-Hong Lai (University of Greenwich, UK), and members of the PETSc team at Argonne National Laboratory: Satish Balay, Bill Gropp, Lois McInnes, and Barry Smith. He is grateful to Manny Salas and Kyle Anderson, organizers of the "Computational Aerosciences for the 21st Century" workshop for the unnatural impetus to contemplate a computational world 10–20 years into the future, however imperfectly, as it may turn out. Steve Ashby (LLNL), Peter Brown (LLNL), Dennis Bushnell (NASA Langley), Xiao-Chuan Cai (UC-Boulder), Rob Falgout (LLNL), Alex Pothen (ODU), and Bob Voigt (College of William & Mary) have all contributed discussion essential to this chapter. Of course, none of the acknowledged can be held accountable for subsequent twisting of their ideas. Work on this chapter or work reported in this chapter has been sponsored by the National Aeronautics and Space Administration (NAS1-97046 and NAG-1-1692), by the National Science Foundation (ECS9527169), Argonne National Laboratory () Lawrence Livermore National Laboratory (B341996), and the Engineering and Physical Sciences Research Council (EPSRC) of the U.K. on a travel grant through the University of Greenwich.

References

Balay, S., Gropp, W.D., McInnes, L.C. and Smith, B.F. (1998) PETSc 2.0 users manual, Technical Report ANL-95/11 - Revision 2.0.22, Argonne National Laboratory, April 1998.

Baudet (1978) Asynchronous Iterative Methods for Multiprocessors, *J. of the ACM* **25**, pp. 226–244.

Bersekas (1990) Partially Asynchronous Parallel Algorithms for Network Flow and Other Problems, *SIAM J. Control and Optimization* **28**, pp. 678–710.

Cai, X.-C., Gropp, W.D., Keyes, D.E., Melvin, R.G. and Young, D.P. (1998) Parallel Newton-Krylov-Schwarz algorithms for the transonic full potential equation, *SIAM J. Scientific Computing* **19**, pp. 246–265.

Chazan and Miranker (1969) Chaotic Relaxation, *Linear Algebra and its Applications* **2**, pp. 199–222.

Culler, D.E., Singh, J.P. and Gupta, A. (1998) *Parallel Computer Architecture*. Morgan-Kaufman, 1998.

Bilmes, J., Asanović, K., Chin, C.-W. and Demmel, J. (1998) Optimizing matrix multiply using PHiPAC: A Portable High-Performance ANSI C Methodology, in *Proceedings of the International Conference on Supercomputing*, Vienna, Austria, July 1997 (ACM SIGARC) and `http://www.icsi.berkeley.edu/~bilmes/phipac`.

Dennis, J.E. and Schnabel, R. (1973) *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, 1973.

Dennis, J.E. and Torczon, V. (1991) Direct search methods on parallel machines, *SIAM J. Optimization* **1**, pp. 448–474.

Department of Energy (1998) Accelerated Strategic Computing Initiative, `http://www.llnl.gov/asci/overview`.

de Sturler, E. and van der Vorst, H. A. (1987) Reducing the Effect of Global Communication in GMRES(m) and CG on Parallel Distributed Memory Computers, *Applied Numerical Mathematics* **18**, pp. 441–459.

Dryja, M. and Widlund, O.B. (1987) An Additive Variant of the Schwarz Alternating Method for the Case of Many Subregions, Technical Report #339, Courant Institute, NYU.

Federal Coordinating Council For Science, Engineering, and Technology (1992) High Performance Computing and Communications Initiative. (See also `http://www.hpcc.gov/blue98`.)

Gropp, W.D., Keyes, D.E., McInnes, L.C. and Tidriri, M.D. (1998) Globalized Newton-Krylov-Schwarz Algorithms and Software for Parallel Implicit CFD, ICASE Technical Report 98-24, 36 pp. [To appear in *Int. J. for High Performance Comput. Applics.*]

Hestenes and Stiefel (1952) Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Stand.* **49**, pp. 409–435.

Hilbert, D. (1891) Uber die stetige Abbildung einer Linie auf ein Flächenstück, *Mathematische Annalen* **38**, pp. 459–460.

Ierotheou, C., Lai, C.-H., Palansuriya, C.J. and Pericleous, K.A. (1998) Simulation of 2-D metal cutting by means of a distributed algorithm. *The Computer Journal* **41**, pp. 57–63.

Karypis, G. and Kumar, V. (1998) Multilevel Algorithms for Multi-Constraint Graph Partitioning, Technical Report 98-019, CS Department, University of Minnesota.

Kaushik, D.K., Keyes, D.E. and Smith, B.F. (1998) On the interaction of architecture and algorithm in the domain-based parallelization of an unstructured grid incompressible flow code, in *Proceedings of the Tenth International Conference on Domain Decomposition Methods* (Mandel, J., et al., eds.), AMS, pp. 311–319.

Kelley, C.T. and Keyes, D.E. (1998) Convergence analysis of pseudo-transient continuation, *SIAM J. Numerical Analysis* **35**, pp. 508–523.

Keyes, D.E. and Gropp, W.D. (1987) A Comparison of Domain Decomposition Techniques for Elliptic Partial Differential Equations and their Parallel Implementation, *SIAM J. Scientific and Statistical Computing* **8**, pp. s166–s202.

Keyes, D.E., Kaushik, D.K. and Smith, B.F. (1998) Prospects for CFD on petaflops systems, in *CFD Review 1997* (Hafez, M., et al., eds.), Wiley (to appear).

Lai, C.-H. (1997) An application of quasi-Newton methods for the numerical solution of interface problems. *Advances in Engineering Software* **28**, pp. 333–339.

Lai, C.-H., Cuffe, A.M. and Pericleous, K.A. (1998) A defect equation approach for the coupling of subdomains in domain decomposition methods. *Computers and Mathematics with Applications* **35**, pp. 81–94.

Miellou (1975) Itérations chaotiques à retards, *Comptes Rendus Ser. A* **280**, pp. 233–236.

National Science Foundation (1996) Grand Challenges, National Challenges, Multidisciplinary Computing Challenges. `http://www.cise.nsf.gov/general/grand_challenge.html`.

Peano, G. (1890) Sur une courbe qui remplit toute une aire plane, *Mathematische An-*

nalen, **36**, pp. 157–160.

Reid, J.K. (1971) On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations, in *Large Sparse Sets of Linear Equations*, J.K. Reid, ed., Academic Press, pp. 231–254.

Saad, Y. and Schultz, M.H. (1986) GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scientific and Statistical Computing* **7**, pp. 856–869.

Semiconductor Industry Association (1998) The National Technology Roadmap for Semiconductors, 1997 Edition, `http://notes.sematech.org/97pelec.htm`.

Smith, B.F., Bjorstad, P.E. and Gropp, W.D. (1996) *Domain Decomposition Methods.* Cambridge University Press, Cambridge.

Schwarz, H.A. (1890) Uber einen Grenzubergang durch Alternierenden Verfahren, [originally published in 1869] in *Gesammelte Mathematische Abhandlungen*, Springer Verlag, **2**, pp. 133–134.

Söderlind, G. (1998) The Automatic Control of Numerical Integration, Technical Report LU-CS-TR:98-200, Lund Institute of Technology (Lund, Sweden).

Gao, G.R., Theobald, K.B., Marquez, A. and Sterling, T. (1997) The HTMT Program Execution Model, CAPSL TM-09, ECE Department University of Delaware. [See also `http://htmt.cacr.caltech.edu/publicat.htm`]

Warren, M. and Salmon, J. (1995) A parallel, portable and versatile treecode, in *Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pp. 319-324, SIAM, Philadelphia.

Warren, M., Salmon, J.K., Becker, D.J., Goda, M.P., Sterling, T. and Winckelmans, G.S. (1997) Pentium Pro inside: I. A treecode at 430 Gigaflops on ASCI Red. II. Price/performnace of $50/Mflop on Loki and Hyglac, in *Supercomputing '97*, IEEE Computer Society, Los Alamitos.

Whaley, R.C. and Dongarra, J. (1998) Automatically Tuned Linear Algebra Software. `http://www.netlib.org/atlas/index.html`.