On Parallel ELLPACK for Shared Memory Machines

By Calvin J. Ribbens

TR 90-46

On Parallel ELLPACK for Shared Memory Machines

Calvin J. Ribbens*
Department of Computer Science
Virginia Polytechnic Institute & State University

August 27, 1990

Abstract

This report outlines design goals and criteria for a new version of ELLPACK for shared memory multiprocessors. We discuss several specific issues in detail, and suggest paths for further work. An example is given which illustrates some of the tradeoffs necessary in parallelizing a large and complicated mathematical software package such as ELLPACK. We also raise several important questions that must be dealt with as packages such as ELLPACK evolve along with modern high-performance architectures.

1 Introduction

The purpose of this report is to summarize design goals, current plans, and progress on a version of ELLPACK specifically tailored for shared memory multiprocessors. ELLPACK [25] is a mathematical software system for numerically solving elliptic partial differential equations (PDEs). It consists of a very high-level language for defining PDE problems and selecting methods of solution, and a package of approximately fifty problem solving methods, or *modules*, implemented in approximately 100,000 lines of FORTRAN. It is straightforward to use ELLPACK to solve a single second order, linear, elliptic PDE posed on a general two dimensional domain or a three dimensional box. With some additional programming, the system may also be used to solve higher order equations, nonlinear problems, time-dependent problems, and systems of elliptic equations.

The purpose of the work outlined here is twofold: (1) to design and implement a production-quality version of ELLPACK for a specific target environment, and (2) to understand better the process of efficiently implementing and using large heterogeneous mathematical software packages on currently available parallel architectures. As an immediate and realistic goal, we feel that a version of ELLPACK targeted specifically for shared memory machines is an appropriate and worthwhile first step.

Our comments reflect extensive experience with the current ELLPACK system on both sequential and parallel machines. We have installed the current ELLPACK system on a Sequent Symmetry S81. Although parallelism is not automatically exploited by current versions of ELLPACK, we have carried out several extensive experiments using this environment [7], [13], [23], [22]. In each of these cases we used parallelism in a "brute-force" way. ELLPACK has no capabilities at present to use parallelism in any way without considerable human intervention.

*Supported by Department of Energy grant DE-FG05-88ER25068.

Our comments reflect communication and collaboration with the ELLPACK group at Purdue University. The work there is more ambitious than described here. It is being done in the context of a large project to develop tools for computing with models of physical objects [3]; they are concerned with several additional aspects in addition to the issues addressed in the present report (see [15], [16], [17]). The primary focus thus far of the Purdue ELLPACK group has been on distributed memory machines, and on distributed computing environments (where a variety of machines are connected by a network). Furthermore, there is a large expert-systems component to their work, which we are not pursuing. Their current design for a parallel system stresses a "domain decomposition" approach to parallelism. They have spent considerable time on algorithms and tools for decomposing domains, and for mapping the corresponding problem onto the nodes of a hypercube. Despite certain differences in emphasis, the development of parallel versions of ELLPACK will clearly be a joint effort—headed by the group at Purdue, with contributions from Virginia Tech and elsewhere. The purpose of the present report is to provide early input to this process.

2 Design goals and criteria

We list several general criteria which should govern the development of a Parallel ELLPACK system:

1. Compatible with current system. It should be possible to run an ELLPACK program written for the current sequential (batch) system, without doing anything special. Reasonable speedup should be achieved for these cases. An obvious exception here would be an ELLPACK program that uses modules no longer in the system.

2. Portable across machines. A user's Parallel ELLPACK program should be portable to any shared memory machine on which Parallel ELLPACK is implemented. Virtually no changes should be required on the part of the user, unless some specific feature of a given machine or compiler is being used.

3. Hidden (implicit) and visible (explicit) parallelism. It is not required that the user know details of the parallelism that is occurring when a Parallel ELLPACK program executes. Reasonable performance should be possible when the parallelism is hidden from the user. At the same time, a sophisticated user should be able to explicitly control the parallelism available for a given problem and machine, if desired.

4. Minimize change to existing code. While substantial change to some of the code in the problem solving modules is unavoidable, if at all possible these changes should be confined to relatively small kernels. The motivation is not only to avoid extra work (and the introduction of bugs), but to avoid widespread system dependencies. Since the BLAS routines [20] form the kernel of several current ELLPACK modules, it is expected that parallelized versions of these routines will be a significant first step in this process. Recently a second and third level of the BLAS (see [9], [10])—based on matrix-vector and matrix-matrix operations, respectively—has been defined. It is likely that these will prove extremely useful in our context.

5. Learn from the experience. Since one of the overall purposes of this project is to understand better the process of parallelizing large mathematical software packages, it is important that lessons be learned from this effort. For example, considerable effort is currently being made toward developing tools to help people program in parallel. Representative tools of this

kind should be evaluated for our task. Additional tools may be suggested by the work as well. Furthermore, it is important that Parallel ELLPACK be flexible enough to evolve into subsequent generations of problem solving environments. Clearly, the current scientific computing environment is not likely to remain unchanged for very many years. Mathematical software packages will continue to be needed as environments change (new hardware, new user interfaces, new computational demands, new research paradigms). It is important to have some idea of how (or how not) to design packages so as to increase the chances of a smooth evolution path.

3 Specific issues

In this section we briefly discuss six important issues affecting Parallel ELLPACK. In some cases we make specific recommendations; in other cases we simply raise questions and examine possible solutions.

3.1 Language and preprocessor

The changes required to the ELLPACK language for a parallel system are not significant. In fact, one could argue that one of the strengths of a "very high level language" is that it allows problem solving strategies to be stated at a sufficiently high level of abstraction, so that questions of parallelism are hidden at lower levels (i.e., in the FORTRAN implementation of the problem solving modules). While an ELLPACK program does indicate a sequential order of execution, the individual statements, or *segments*, each represent the execution of an entire module. In most cases, the most efficient use of parallelism will be *within* that module, so that the ELLPACK program itself does not change. The question arises however, whether new control constructs should be added to the ELLPACK language to allow segments to be organized into parallel loops or to indicate that multiple copies of a given segment should be executed in parallel. This seems to be unnecessary. The Parallel ELLPACK language will still be an extension of FORTRAN, and thus arbitrary sections of FORTRAN code—including whatever parallel constructs or compiler directives are available in the FORTRAN dialect being used—may still be used to indicate parallelism at the highest level.

The parallel ELLPACK system being developed at Purdue [16] currently includes several new types of segments (e.g., "mesh", "decomposition", "subdomain", "interface"). It is not clear that each of these should be a new segment. Rather, it appears that these are simply *procedures* in current ELLPACK terminology—that is, they are computations that do a useful task related to setting up for, or analyzing, a given solution. Furthermore, elevating them to the status of segment suggests that they are each fundamental (or at least, common) steps in the numerical solution of a PDE, when they are instead only fundamental steps in one approach to solving PDES—namely, domain decomposition. This may be only a semantic distinction; but care should be taken before the relatively clean organization of ELLPACK is cluttered by unnecessary additions.

A few small additions to the ELLPACK language are obviously needed. Options to indicate machine type and number of processors are clearly necessary. (The current prototype at Purdue actually has a new segment "machine", with options "machine name", "number of pes", and "topology"). Another option which should be considered is "granularity". As discussed in more detail in Section 3.5, this option would allow changes in the size of the subtasks to be done in parallel.

If so few changes are needed to the ELLPACK language, then the preprocessor will clearly

require few modifications. Even fewer changes would be needed if most of the new segments were simply procedures (or perhaps hidden in *triple* segments). Modifications to the ELLPACK *template* are clearly needed, of course. The more varied the class of machines on which ELLPACK is implemented, the more complicated the template will have to be, since control programs must be generated for several different dialects of FORTRAN. We have considerable experience with this type of situation (e.g., the template for Interactive ELLPACK [12] includes code for several types of graphics workstations). However, some thought should be given to ways to minimize, and perhaps automate, the work involved in modifying the template for a new machine.

3.2 Problem solving modules

The design of a new ELLPACK system is a good opportunity to modify the set of problem solving modules. Modules that are virtually unused or unusable should be dropped. The original ELLPACK project had an important performance-evaluation component—thus, there are included several methods which otherwise might not be present. This is a good time to omit those methods from future versions of ELLPACK. We avoid offending any particular author at this point by omitting a suggested list for removal.

At the same time, recent years have seen the development of many new methods for PDEs. Examples are preconditioned conjugate gradient methods and spectral methods. It is unrealistic to demand a complete new set of methods. But Parallel ELLPACK should at least have a stronger representation of methods which are particularly attractive in terms of parallelism. Domain decomposition and multigrid methods are obvious candidates. In no particular order, we list several types of algorithms that should be considered (given sufficient time and resources) for inclusion in Parallel ELLPACK. (See Rice [24] for a rough prioritization of some of these methods.)

- Domain decomposition algorithms.
- Preconditioned conjugate gradient methods.
- New versions of existing modules (e.g., block-oriented algorithms for linear equations, reordering and pivoting strategies appropriate for parallel computation).
- Adaptive methods. The author believes strongly that a much greater use of adaptation should be made, especially when multiple processors are available.
- New collocation algorithms which are more attractive for parallelism (see Christara, et. al [6] and Houstis, Rice and Vavalis [18]).

As was done in the original ELLPACK project, contributions of software from other researchers should be welcomed. In fact, as discussed in Section 3.4 below, it is even more important to solicit contributions from others in the case of parallel algorithms. The task of parallelizing some of the code currently in ELLPACK would be formidable indeed for someone not extremely familiar with that code.

One final question regarding problem solving modules is the degree to which the use of the BLAS kernels should be encouraged or enforced. As discussed elsewhere in this report, one attractive approach to achieving reasonable parallel efficiency across many architectures, without extensive use of nonportable code, is a flexible use of the three levels of the BLAS. Unfortunately, the current ELLPACK system contains only a few modules that are written in terms of BLAS routines. It is likely that a much greater use of BLAS routines will make implementing parallel versions of

ELLPACK much easier, both for the short term and the long term, as architectures and computing environments evolve.

3.3 Portability

A good balance among portability, initial development effort, ease of subsequent installation, and efficiency is clearly a difficult challenge. Simply writing everything in a universal dialect of FORTRAN is no longer possible if a wide variety of parallel machines are targeted (even if only shared memory machines are of interest). In this section we comment on several avenues that might be pursued in an effort to achieve a reasonable degree of portability at manageable expense.

- BLAS routines. Again, we argue that an extensive use of the full set of BLAS routines will go a long way toward achieving portability and performance across a significant range of machines. If most of the performance-critical sections of code are written in terms of BLAS calls, and if the BLAS are efficiently implemented for a particular machine, then with minimal effort a reasonable level of performance should be achievable. Of course, the BLAS are not a panacea for all problems of portability vs. performance and effort. There will doubtless be situations where using a completely different algorithm (i.e., with changes at a higher level than even BLAS-3) will be required in order to achieve reasonable performance.

- Portable parallel programming tools or packages. One could consider implementing parallel versions of ELLPACK on top of a more extensive layer of software than the BLAS. Recently a few packages have begun to appear that seek to help Fortran programmers develop portable parallel applications. An example is SCHEDULE [11], which allows users to express subroutine level parallelism in terms of a task dependency graph. Once a package such as SCHEDULE is implemented for a given machine, the user's programs can be run on that machine with virtually no change. For our purposes however, this approach does not seem promising. Our experience with SCHEDULE (see [13]) is that it is difficult to use for complicated dependency graphs, particularly when the overall computation is iterative, or when the exact flow of control is not known until runtime. The developers of SCHEDULE do not claim that it completely handles all situations, of course. Nonetheless, it is not realistic to expect all the code in ELLPACK to be rewritten in terms of something like SCHEDULE.

- A virtual parallel machine. An even more ambitious level of software could be used as the platform on which Parallel ELLPACK is built. For example, one could adopt the philosophy that all "machines" are made up of an unlimited number of processors that communicate via messages (e.g., the Actor model [1]). Or perhaps the Linda [5] approach might be used to abstract away from messy machine details. The significant initial effort, and the expected performance penalties that would result, make this seem like a very unattractive approach. Even the more modest approach of simply taking Parallel ELLPACK implemented for the hypercube and simulating message passing via shared memory seems unnecessarily inefficient (although it probably should at least be considered). Treating a shared memory machine such as the Alliant, with a few very powerful processors, as if it were a hypercube would seem to be a big mistake.

- Our own level of software. In an effort to enhance portability one normally attempts to organize that portion of code that is *not* portable into the smallest possible set of subprograms. We could consider identifying our own set of subprograms that would need to be implemented

differently for different machines. This also seems like a time-consuming and problematic task. While a set of "basic pde-solving subprograms", analogous to the BLAS, sounds like a clever idea, it is probably overly ambitious and unrealistic at this point.

3.4 Costs vs. benefits of parallelism

It is important to remember that parallelizing a large, complicated mathematical software package is qualitatively different from parallelizing a single algorithm or program. There are some similarities, of course. In both cases it is important to identify sections of code that dominate the execution time, and ignore sections that take very little time. In the case of a large package however, it is important to identify a huge amount of code—many subprograms, in fact—that may safely be ignored in terms of parallelism. There simply is not enough time to carefully consider parallelism for every piece of the package; and the payoffs for most of such an effort would be negligible. Furthermore, not every combination of methods (e.g., ELLPACK modules) is important. Some ability to identify the important computational paths through the ELLPACK software, and ignore insignificant or even illegal paths, is important. This is so because efficient parallelism for a particular sequence of modules may require certain choices for algorithms, data structures, or data distribution. If a particular sequence is not likely to be used, it should not be allowed to influence the design of the package. Finally, some commitment to modularity, maintainability, and a "software-parts" approach to mathematical software design must not be abandoned. Necessarily, some efficiency will be lost if we do not pursue every last ounce of parallelism in ELLPACK. But we believe that these other issues remain centrally important, and are not worth sacrificing for another 10% in performance.

It is also important to recognize that even parallelizing those few sections of code that are most important is very difficult in our setting. Loops containing diagnostic output statements, conditional branches out of loops, unnecessary use of global variables (in COMMON blocks), and loops that simply do not parallelize make the task extremely tedious for typical scientific code. There are tools and environments being developed (see [14], [19], [4] for example) which might help in this process. It is expected though, that these tools will prove most useful when a new parallel algorithm is being designed from scratch. They seem less promising for attacking a large body of existing code; and clearly these tools are not able to suggest a complete redesign of the algorithm. Nonetheless, it would be a valuable experiment to try such tools.

Soliciting contributions of parallel code form other authors seems to be even more strongly indicated in the case of Parallel ELLPACK than it was for the original system. Obviously, it is much easier for the original authors to parallelize a given piece of software. Even better, authors who are willing to contribute entirely new algorithms and implementations designed for efficient parallel execution should be encouraged. Possibilities that come immediately to mind are the group at University of Texas for parallel versions of ITPACK, the LAPACK project [2] for recent descendants of LINPACK and EISPACK routines, Bill Gropp and David Keyes for implementations of widely-used domain decomposition algorithms, and the group at Yale for PCGPAK and SMPAK (Krylov subspace methods and a descendent of the Yale Sparse Matrix Package, respectively).

The question still remains how to decide when "enough" of the package is parallelized, or when sufficient parallel efficiency is reached. It is impossible to give a final answer to this, of course. The best approach is simply to prioritize the set of modules and routines, and deal with as many as resources (and outside contributions) allow. Such a prioritization itself is nontrivial. In view of the above comments, and remembering that ELLPACK is a large, heterogeneous, and general-purpose package, we are left with the following comments: (1) outstanding parallel efficiency simply

cannot be guaranteed for every piece of code and every sequence of modules; (2) the user may have to fine-tune something if there is a relatively obscure section of code that heavily influences the performance of his or her application; (3) Parallel ELLPACK should remain an open and accessible system so that sophisticated users can do their own modifications and additions to ELLPACK (this seems to be even more important in a parallel environment, and one of the real strengths of the ELLPACK design).

3.5 Granularity

Another implication of the heterogenaity of a package such as ELLPACK is that one level of granularity is not sufficient for every algorithm or every application. Even for a single piece of code, there are cases when a single loop should be run in parallel (i.e., spread across available processors) and other cases when it should be run sequentially—because, for example, other copies of the same code are running concurrently on other processors. Consider the various levels of the BLAS, for example. It is easy to think of typical computations where parallelism at each of the three levels is appropriate. Computing acceleration parameters for an iterative linear equations solver often requires that a single vector operation (e.g., dot product) be done very quickly. At a higher level, a single sweep of an iterative linear equations solver is easily viewed as a matrix-vector operation. At a higher level still, block-methods for solving linear equations and tensor product alternating direction methods are efficiently implemented in terms of matrix-matrix operations. One could even consider a higher level of granularity, in which an entire PDE solve was viewed as an atomic task (e.g., Schwarz splitting methods, or systems of equations). The point is that fixing the level of granularity, at say the matrix-vector level, is a mistake. It would be better to allow the best level of granularity to be selected, depending on the given computation.

An ELLPACK option "granularity" should be considered. For discussion sake, it could have four possible values corresponding to the four levels illustrated in the above paragraph. Users could set this option statically by setting the option in their ELLPACK program, or (since there would be a corresponding FORTRAN variable) change granularity at runtime. The functionality of this option would be implemented by modifying the BLAS routines (for example) so that they always executed sequentially, unless the granularity option was set to their level. For example, the "saxpy" routine would do its work in parallel only if granularity=1; for other values it would execute sequentially (the typical situation being that parallelism was occurring at higher levels of granularity). This idea obviously needs further work. But it seems that something along these lines is needed to allow efficient parallel execution of the wide variety of computations for which ELLPACK is designed.

3.6 Data structures

Issues involving data structures are made much easier by the assumption of shared memory. We simply mention that the current ELLPACK design, with several standard "interfaces", through which modules communicate, should be maintained. This strategy is a key to modularity and the ease with which other people's software can be included in ELLPACK. It is likely that a new interface will need to be defined to keep track of information that relates specifically to parallelism.

One other small point about data structures is the cost of copying data between different data structures. ELLPACK does this frequently (e.g., copying sparse linear systems from ELLPACK storage format to LINPACK band format). On many machines, this step may become relatively more of a performance bottleneck than it has been previously. Shared memory busses are notorious

Table 1: Time in seconds, parallel speedup (Sp) and efficiency (Ef) as a function of the number of processors (P) for three attempts at parallelizing an example PDE computation.

	Attempt 1			Attempt 2			Attempt 3		
P	Time	Sp	Ef	Time	Sp	Ef	Time	Sp	Ef
1	734			734			734		
2	416	1.76	0.88	406	1.81	0.90	401	1.83	0.91
3	298	2.46	0.82	286	2.57	0.86	278	2.65	0.88
4	244	3.01	0.75	229	3.20	0.80	218	3.36	0.84
5	210	3.50	0.70	195	3.77	0.75	183	4.02	0.80
6	188	3.91	0.65	173	4.24	0.71	160	4.59	0.77
7	172	4.27	0.61	156	4.70	0.67	143	5.12	0.73
8	161	4.57	0.57	143	5.11	0.64	131	5.60	0.70
9	152	4.83	0.54	135	5.45	0.61	122	6.00	0.67

for becoming clogged when every processor transfers a large amount of data concurrently. If this proves to be a serious problem, consideration should be given to alleviating this situation in some way (e.g., avoiding more of the copying, or overlapping the computation and communication to a greater extent).

4 An example

In the final section of this report we present an example of an "everyday" PDE computation on an "everyday" machine. We believe that while the grand challenges of computational science are important and exciting, it is also important to have available efficient parallel versions of general purpose packages such as ELLPACK on the current generation of widely available parallel machines. With this exercise we want to illustrate some of the points made in the preceding sections, and begin to understand the costs and benefits of developing a parallel version of ELLPACK.

We consider solving Problem 1 from the set of test problems described in [26]:

$$(e^{xy}u_x)_x + (e^{-xy}u_y)_y - \frac{1}{1+x+y}u = f(x,y),$$

where f is chosen so that the true solution is

$$u(x,y) = 0.75e^{xy}\sin(\pi x)\sin(\pi y),$$

and we have Dirichlet boundary conditions on the unit square. A uniform mesh with spacing $h = 1/128$ is placed on the domain, and ELLPACK modules 5 POINT STAR (second-order centered finite differences) and LINPACK SPD BAND (Cholesky factorization) are used. All computations are in double precision on a Sequent Symmetry S81.

Three attempts at parallelizing this computation are described. Each attempt builds on the preceding ones. Table 1 summarizes parallel performance for the entire computation resulting from the three attempts. A plot of parallel efficiency vs. number of processors is given in Figure 1.

Attempt 1. The factor routine of LINPACK SPD BAND is parallelized. Specifically, the algorithm is modified to compute the elements of the upper triangular Cholesky factor by rows

Table 2: Time in seconds, parallel speedup (Sp) and efficiency (Ef) as a function of the number of processors (P) for the triangular solves with Attempt 3.

P	Time	Sp	Ef
1	26.8		
2	18.9	1.42	0.71
3	16.3	1.65	0.55
4	14.3	1.87	0.47
5	13.4	2.00	0.40
6	12.4	2.16	0.36
7	12.5	2.14	0.31
8	13.2	2.04	0.25
9	12.8	2.09	0.23

instead of by columns This modification of the original LINPACK [8] routine is needed to get any parallelism at all out of the inner loop. A special round-robin scheduling strategy is used so that the same processors are not always assigned the longest dot products within this loop. Note that parallelizing the dot products themselves yields virtually no speedup—the granularity is too small.

Attempt 2. The generation of equations in 5 POINT STAR is parallelized. Specifically, the numbering of equations is done first, and then equations corresponding to each row of mesh points are generated in parallel.

Attempt 3. The triangular system solves in LINPACK SPD BAND are parallelized. It is well known that triangular solves are difficult to do efficiently in parallel for many combinations of machine configuration and system size and sparsity. For this example, we employed a block matrix version of the column-sweep algorithm (see [21]). The parallel performance of just the forward and back solves for this problem (i.e., ELLPACK routine Q5LSSL) is summarized in Table 2.

We now make several comments by way of discussing the results summarized in Tables 1 and 2 and Figure 1.

- The programming effort needed to make the necessary modifications for Attempts 1 and 2 was modest (though *not* trivial). The effort was substantial for Attempt 3—a couple of man-days, at least; in fact it is likely that there is a more efficient way to do the triangular solves than the method used.

- Note however that these routines were *not* parallelized in a portable manner. The syntax is specific to Sequent Fortran.

- The example illustrates the potential importance of parallelizing more than "just" the linear algebra. The improvement in Attempt 2 over Attempt 1 is due to parallelizing the discretization phase, for example. Other steps in the computation were not parallelized at all—namely, the setup for LINPACK SPD BAND, and the output module MAX. These steps amounted

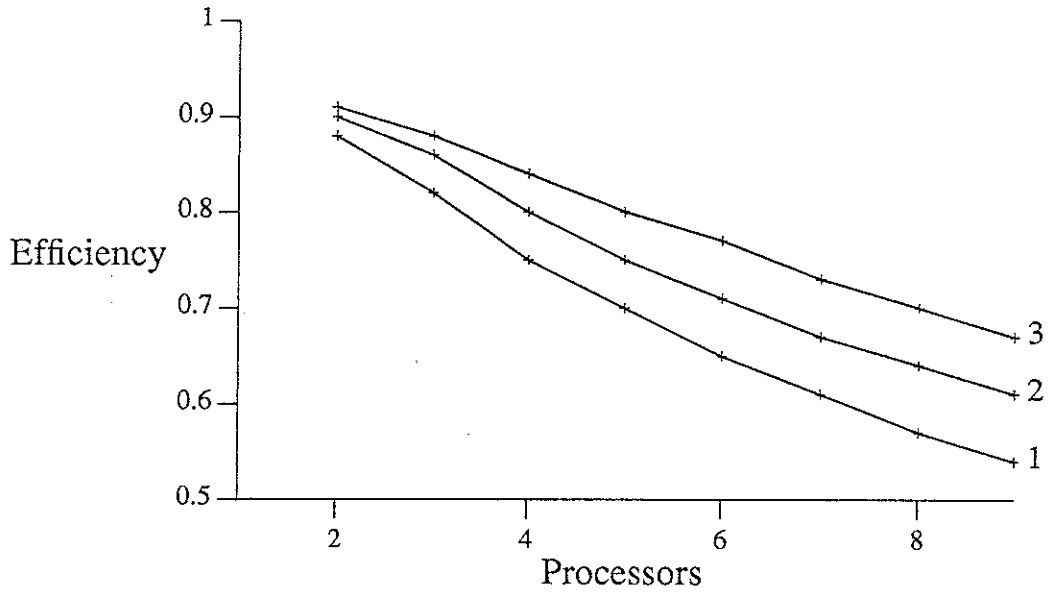

Figure 1: Parallel efficiency for three attempts at parallelizing an example PDE computation.

to only 8 seconds in the above example, however. The function evaluations required to evaluate the solution at many points is easy to parallelize and can be quite significant (e.g., a high-resolution plot of the solution requires thousands of function evaluations).

- If a great deal of emphasis is placed on "scaled speedup" (i.e., the problem size grows with the number of processors), then simply parallelizing the factorization step gives you most of the performance you expect. However, it is not always sufficient to argue that scaled speedup makes all inefficiencies disappear. This is especially true for shared memory machines with modest numbers of processors. Frequently, one simply does not want to solve a problem that is big enough to achieve good parallel efficiencies. This may be true even when the overall computation is very substantial—if, for example, the PDE solve is the inner loop of an optimization or control problem. We need the ability to achieve reasonably efficient parallelism even for these modest PDE calculations.

- The triangular solves are clearly a potential bottleneck; Attempt 3 succeeds only marginally in speeding them up. For our example, the penalty for slow triangular solves is not too serious (altough for fixed problem size, as the number of processors grows, the triangular solves do become more of a problem). However, other methods (e.g., preconditioned conjugate gradients) may require many more triangular solves.

- There is a 20% improvement in time from Attempt 1 to Attempt3 with nine processors; and the difference in parallel efficiency is growing slowly. While not a dramatic improvement, it does seem significant enough to make the extra work needed for Attempt 3 worthwhile.

- At the same time, the first two attempts illustrate that a reasonable level of performance is achievable with relatively modest effort on a few important routines.

References

[1] G. Agha, *A Model of Concurrent Computation in Distributed Systems*, MIT Press, 1986.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, "LAPACK: A portable linear algebra library for high-performance computers", University of Tennessee, Computer Science Report CS-90-105, May 1990.

[3] C. Bajaj, W. R. Dyksen, C. M. Hoffman, E. N. Houstis, J. T. Korb and J. R. Rice, "Computing about physical objects", Technical Report CSD-TR-696, Computer Sciences Department, Purdue Universit, July 1987.

[4] J. C. Browne, "Framework for formulation and analysis of parallel computation structures", *Parallel Comput.*, 3(1986), pp 1–9.

[5] N. Carriero and D. Gelernter, "Linda in context", *Commun. ACM*, 32(1989), pp 444–458.

[6] C. C. Christara, E. N. Houstis and J. R. Rice, "A parallel spline collocation-capacitance method for elliptic partial differential equations", Computer Sciences Department Report CSD-TR-735, Purdue University, 1988.

[7] B. Cleveland and C. J. Ribbens, "Schwarz splitting using ELLPACK", Dept. of Computer Science Report 88-2, Virginia Polytechnic Institute & State University, 1988.

[8] J. J. Dongarra, J. R. Bunch, C. B. Moler and G. W. Stewart, *LINPACK Users' Guide*, SIAM Press, Philadelphia, 1979.

[9] J. J. Dongarra, J. DuCroz, S. Hammarling and R. Hanson, "An extended set of fortran basic linear algebra subprograms", *ACM Trans. Math. Softw.*, 14(1988), pp 1–32.

[10] J. J. Dongarra, J. DuCroz, I. Duff and S. Hammarling, "A set of level 3 basic linear algebra subprograms", Technical Report MCS-P1-0888, Argonne National Laboratory, August 1988, (to appear *ACM Trans. Math. Softw.*).

[11] J. J. Dongarra and D. C. Sorensen, "A portable environment for developing parallel fortran programs", *Parallel Computing*, 5(1987), pp 175–198.

[12] W. R. Dyksen and C. J. Ribbens, "Interactive ELLPACK: an interactive problem-solving environment for elliptic partial differential equations", *ACM Trans. Math. Softw.*, 13(1987), pp 113–132.

[13] J. Gamble and C. J. Ribbens, "Performance comparison of three parallel implementations of a Schwarz splitting algorithm", VPI & SU, Department of Computer Science Report TR 89-37, October 1989.

[14] D. Gannon, D. Attapattu, M. H. Lee and B. Shei, "A software tool for building supercomputer applications", in *Parallel Processing for Scientific Computing*, (G. Rodrigue, ed.), SIAM, Philadelphia, 1989, pp 301–305.

[15] E. N. Houstis, J. R. Rice, and T. S. Papatheodorou, "Parallel ELLPACK: An expert system for parallel processing of partial differential equations", Computer Sciences Department Report CSD-TR-831, Purdue University, 1988.

[16] E. N. Houstis, J. R. Rice, N. P. Chrisochoides, H. C. Karathanasis, P. N. Papachiou, M. K. Samartzis, E. A. Vavalis and K-Y Wang, "Parallel (//) ELLPACK PDE solving system", Computer Sciences Department Report CSD-TR-912, Purdue University, October 1989.

[17] E. N. Houstis, J. R. Rice, N. P. Chrisochoides, H. C. Karathanasis, P. N. Papachiou, M. K. Samartzis, E. A. Vavalis and K-Y Wang, "//ELLPACK: a numerical simulation programming environment for parallel mimd machines", Computer Sciences Department Report CSD-TR-949, Purdue University, January 1990.

[18] E. N. Houstis, J. R. Rice and E. A. Vavalis, "A Schwarz splitting variant of cubic spline collocation methods for elliptic pdes", Computer Sciences Department Report CSD-TR-745, Purdue University, 1988.

[19] K. Kennedy, "Programming systems for parallel supercomputers", in T. F. Chan, R. Glowinski, J. Périaux and O. Widlund (eds.), *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM, Philadelphia, 1990.

[20] C. Lawson, R. Hanson, D. Kincaid and F. Krogh, "Basic linear algebra subprograms for fortran users", *ACM Trans. Math. Softw.* 5(1979), pp 308–325.

[21] J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

[22] C. J. Ribbens, C. Y. Wang, L. T. Watson and K. A. Alexander, "Vorticity induced by a moving elliptic belt", Department of Computer Science Report TR 89-35, Virginia Polytechnic Institute & State University, 1989. (submitted to *Computers & Fluids*).

[23] C. J. Ribbens and L. T. Watson, "The parallel performance of schwarz splitting", Department of Computer Science Report TR 90-34, Virginia Polytechnic Institute & State University, October 1989.

[24] J. R. Rice, "Potential evolution directions for ELLPACK", Computer Sciences Department Report CSD-TR-752, Purdue University, March 1988.

[25] J. R. Rice and R. F. Boisvert, *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, New York, 1985.

[26] J. R. Rice. E. N. Houstis and W. R. Dyksen, "A population of linear, second order, elliptic partial differential equations on rectangular domains", *Math. Comp.*, 36(1981), pp 475–484.