Parallel ELLPACK for Shared Memory Multiprocessors*

Calvin J. Ribbens, George G. Pitts, and Layne T. Watson

TR 92-56

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

December 20, 1992

**This work was supported in part by Department of Energy Grant DE-FG05-88ER25068 and Air Force Office of Scientific Research Grant F49620-92-J-0236. The authors also gratefully acknowledge the use of the Argonne Advanced Computing Research Facility.*

PARALLEL ELLPACK FOR SHARED MEMORY MULTIPROCESSORS

Calvin J. Ribbens
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106

George G. Pitts
Department of Mathematics
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0123

Layne T. Watson
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106

ABSTRACT

This paper describes a parallel version of ELLPACK for shared memory multiprocessors. ELLPACK is a system for numerically solving elliptic PDEs. It consists of a very high level language for defining PDE problems and selecting methods of solution, and a library of approximately fifty problem solving modules. Each of the modules performs one of the basic steps in solving an elliptic PDE: discretization, reordering of equations and unknowns, linear system solution, etc. ELLPACK may be used to solve linear elliptic PDEs posed on general two dimensional domains or in three dimensional boxes, nonlinear problems, time dependent problems, and systems of elliptic equations. Earlier work considered three discretization modules (*five point star*, *hodie*, and *hermite collocation*), two linear system solution modules (*linpack spd band* and *jacobi cg*), and a triple module (*hodie fft*) which includes both discretization and solution, all for rectangular domains and simple boundary conditions. Here we describe parallel versions of six additional modules (*hermite collocation, hodie helmholtz, five point star, band ge, sor, symmetric sor cg*) for general boundary conditions and domains, and discuss modifications to the ELLPACK preprocessor, the tool that translates an ELLPACK "program" into FORTRAN. The parallelization strategy is based on kernels, with the machine dependent and performance critical code located in a relatively small number of kernels. We report performance of these parallel ELLPACK modules on a Sequent Symmetry S81 shared memory multiprocessor with 26 processors for two simple test problems, and a problem involving a composite laminate with spatially varying fiber orientations.

1. INTRODUCTION.

Mathematical software packages are an important component of large scale scientific computing today. The usefulness of such packages in building systems that solve large scale numerical problems is widely recognized. The community has come to depend on the availability of good algorithms, implemented in quality software, for a great number of problems or important subproblems. The significance of parallel computation for large scale scientific computing is also widely recognized. The contemporary scientific computing environment is characterized by a wide variety of high performance vector and/or parallel computers—from supercomputers with a few very powerful vector processors, to shared memory machines with tens of processors, to distributed memory machines with hundreds or thousands of processors. As parallel and vector computers become more and more common, and especially as they begin to be used as general purpose scientific computing engines, there is a great need for quality mathematical software packages on these machines.

At present there are only a few examples of large numerical packages available for parallel machines. This situation is not surprising given the rapid evolution in hardware, the unstable and immature software environments on most new parallel machines, and the many difficulties inherent in building good parallel mathematical software. Existing mathematical software packages are difficult to parallelize because they are large, complex, and often have many customers who do not want to see significant changes in a package to which they have grown accustomed. At the same time, building completely new packages designed specifically for a parallel and/or vector machine is a very time consuming and costly proposition. Given the importance of having good mathematical software on parallel machines and the huge investment in existing sequential packages, it is natural to consider ways in which existing packages can be evolved toward efficient implementations on a range of parallel machines.

In recent work[10,12] we have compared several approaches to parallelizing a large mathematical software package for a particular class of machine, namely a shared memory multiprocessor. We have focused on ELLPACK[13], a well-known package for solving elliptic partial differential equations (PDEs). Three general strategies are compared in our work: explicit "by-hand" parallelization using nonportable language extensions, automatic parallelization using a commercially available precompiler (KAP[7]), and a two-level approach based on explicit parallelization of a set of low-level primitives and reformulation (as needed) of the code to use these primitives.

In connection with the third approach, we have proposed a set of primitives or kernels appropriate for PDE software. The set includes the three levels of dense Basic Linear Algebra Subprograms (BLAS)[3,4,8], a few kernels from a proposed set of sparse BLAS[1], plus kernels that are unique to PDE solving software. These PDE solving kernels are described in detail by Ribbens and Pitts[11]. The motivation is to define a set of low-level kernels on top of which sophisticated algorithms can be built. The goal is both portability (in that the algorithms are written in terms of higher level constructs) and efficiency (assuming the kernels themselves are efficiently implemented on various machines).

Our experience is that the two-level approach, with a set of efficiently implemented kernels supporting the rest of the code, is a good one. It seems to represent the best way to balance the conflicting goals of parallel performance, programmer effort, and portability. Some work is certainly required to modify existing software to be based cleanly on the kernels, but we find that this is comparable to the explicit parallelization, with the added benefit that you only have to do it once. From then on, as you move from machine to machine you only need to implement the kernels efficiently. The automatic strategy saves work when it works, but we find it incapable of dealing with typical complex codes. The automatic parallelization strategy has considerable difficulty with most of the code and is unable to achieve reasonable parallel performance without significant assistance from the programmer. A very significant amount of rewriting is necessary to achieve good parallel performance on several of the modules using the explicit parallelization strategy. The two-level approach does require some effort in implementing the kernels efficiently, but the amount of code that needs close attention is manageable, and the resulting parallel performance is very similar to that achieved with the explicit by-hand strategy.

The purpose of this paper is to describe a parallel version of ELLPACK for shared memory multiprocessors. ELLPACK is a system for numerically solving elliptic PDEs. It consists of a very high level language for defining PDE problems and selecting methods of solution, and a library of

Table 1. Newly parallelized ELLPACK modules.

Module	Purpose
HERMITE COLLOCATION	Discretizes a general elliptic operator with general linear boundary conditions on a two-dimensional rectangular domain. Uses collocation and Hermite bicubic basis functions.
HODIE HELMHOLTZ	Discretizes the generalized Helmholtz equation with general linear boundary conditions on a rectangle. Uses second, fourth, or sixth order accurate compact finite differences (HODIE) depending on the problem.
5-POINT STAR	Discretizes the variable coefficient equation $au_{xx} + cu_{yy} + du_x + eu_y + fu = g$ with general linear boundary conditions on a general two-dimensional domain. Uses classical finite differences.
BAND GE	Solves a real banded system of linear equations. Computes an LU factorization using scaled partial pivoting.
SOR	Classical successive overrelaxation with adaptive selection of relaxation factor (from ITPACK[6]).
SYMMETRIC SOR GC	Symmetric SOR with conjugate gradient acceleration (from ITPACK).

approximately fifty problem solving modules. Each of the modules performs one of the basic steps in solving an elliptic PDE: discretization, reordering of equations and unknowns, linear system solution, etc. It is straightforward to use ELLPACK to solve linear elliptic PDEs posed on general two dimensional domains or in three dimensional boxes. The system may also be used to solve nonlinear problems, time dependent problems, and systems of elliptic equations. The problem solving modules comprise over 100,000 lines of FORTRAN.

Ribbens and Pitts[10] and Ribbens et al.[12] describe parallel versions of six ELLPACK modules and report performance on a Sequent Symmetry S81. The modules previously considered include three discretization modules (*five point star*, *hodie*, and *hermite collocation*), two linear system solution modules (*linpack spd band* and *jacobi cg*), and a triple module (*hodie fft*) which includes both discretization and solution. All these modules were for rectangular domains and simple Dirichlet or Neumann boundary conditions. Here we describe parallel versions of six additional modules (see Table 1) for general domains and general boundary conditions, and discuss modifications to the ELLPACK preprocessor, the tool that translates an ELLPACK "program" into FORTRAN.

Section 2 describes the two-level parallelization strategy introduced above, with the machine dependent and performance critical code located in a relatively small number of kernels. Section 3 describes the three test problems–two simple mathematical test problems and a realistic composite laminate problem with spatially varying fiber orientations. Section 4 describes the BLAS and ELLPACK modules under consideration, and Section 5 presents and discusses the parallel performance results on a Sequent Symmetry S81 shared memory multiprocessor with 26 processors. Section 6 concludes.

2. STRATEGY FOR PARALLELIZATION.

Ribbens and Pitts[10,11] considered in detail three parallelization strategies: (1) explicit manual parallelization, (2) automatic parallelization via compilers, and (3) a two level strategy. In terms of parallel performance, programmer effort, and portability, overall (3) seemed to be the best approach. Therefore the approach we consider here for parallelizing a large mathematical software package is a two-level strategy built on parallel implementations of a set of low level primitives. This approach is motivated in essentially the same way as the three levels of Basic Linear Algebra Subprograms (BLAS)[3,4,8]. In fact, for PDE applications, these kernels themselves are of considerable use. Several of the modules in ELLPACK already make use of Level 1 BLAS, so this approach is quite natural for this package. If the number of kernels can be kept relatively small, if an efficient implementation of the kernels is available on a given machine, and if the most time consuming code in the package can be written in terms of these kernels, then a good balance between performance and portability can be achieved. However, there can be considerable work involved if the existing code makes no use of the kernels.

In Table 2 we list the kernels used for the experiments reported in this paper. The first set are taken from the dense BLAS, sparse BLAS or iterative BLAS[9]. The leading "d" indicates double precision data. The second set of five kernels are simple vector operations. The final set of six kernels are more unique to PDE solving. For example, it is useful to have an operation such as foreach_point which allows a given computation to be performed in parallel at each grid point of a rectangular grid. Similar operations over finite element discretizations are obviously possible. We also need operations that apply a function to each horizontal or vertical grid line. It may also be efficient to organize a foreach_point computation by lines in order to improve granularity. The foreach_proc function is used for initializations that need to be done only once for each process. Note that the members of this last block of operations are not kernels in the traditional sense, in that they do not represent a single simple operation. Instead, they might be termed "operators" or "parallel control structures" since they take an arbitrary function and apply it at each grid point, line, process, etc. For our purposes they do meet the most important criterion for kernels: they hide nonportable details in a small section of code, allowing the large majority of code to remain as is. For purposes of this paper we did no special optimizations of the kernels themselves. We simply implemented them in FORTRAN, using the language extensions available under Sequent's ATS FORTRAN compiler. It is worth mentioning that each of the last set of kernels takes only a small fixed number of parameters. This means that to do complex operations at each point, line, etc., one may have to pass considerable data in COMMON blocks. This is not attractive in many cases. A better alternative might be to allow a variable number of parameters to be passed along with the function or subroutine which is to be applied in parallel. A kernel allowing this would not be implementable in FORTRAN on many systems, however.

3. TEST PROBLEMS.

This section describes three test problems of varying complexity: a Helmholtz problem, a variable coefficient self-adjoint problem on a general region, and a realistic composite laminate problem. Not every ELLPACK module is applicable to every problem, e.g., the *hodie helmholtz* module is only applicable to the Helmholtz PDE. Similarly, not every solution module can follow any discretization module. Only *band ge* can be used with *hermite collocation* on general problems, but all three solution modules *band ge*, *sor*, *symmetric sor cg*, can be applied to the *five point star* discretization of a self-adjoint differential operator.

Table 2. Parallel kernels for pde solving.

Name	(Source)	Function
daxpy	(BLAS1)	vector update: $y \leftarrow \alpha x + y$
dcopy	(BLAS1)	copy one vector to another: $y \leftarrow x$
ddot	(BLAS1)	dot product of two vectors: $x^T y$
dgbmv	(BLAS2)	band matrix vector multiply: $y \leftarrow \alpha Ax + \beta y$
dgemm	(BLAS3)	matrix matrix multiply: $C \leftarrow \alpha AB + \beta C$
dgemv	(BLAS2)	matrix vector multiply: $y \leftarrow \alpha Ax + \beta y$
dger	(BLAS2)	rank one update: $A \leftarrow \alpha xy^T + A$
dgthr	(SPBLAS)	vector gather: $x_i \leftarrow y_{k_i}$
dscal	(BLAS1)	scale a vector by a constant: $x \leftarrow \alpha x$
dsctr	(SPBLAS)	vector scatter: $y_{k_i} \leftarrow x_i$
dsyrk	(BLAS3)	perform a symmetric rank k update: $C \leftarrow \alpha AA^T + \beta C$
dtbsv	(BLAS2)	solve a single banded triangular system: $x \leftarrow A^{-1}x$
dtrsm	(BLAS3)	solve triangular systems of equations: $B \leftarrow \alpha A^{-1}B$
dyasx2	(ITBLAS)	sparse matrix-vector multiply: $y_i \leftarrow y_i + \alpha \sum_{j=1}^{k} a_{i,m_j} x_{m_j}$
dvadd		componentwise vector addition: $y_i \leftarrow x_i + y_i$
dvfill		vector fill: $x_i \leftarrow \alpha$
dvmult		componentwise vector multiplication: $x_i \leftarrow x_i y_i$
dvrecp		componentwise vector reciprocal: $y_i \leftarrow 1/x_i$
dvsqrt		componentwise vector square root: $y_i \leftarrow \sqrt{x_i}$
eval_grid		evaluate a real function on a grid, returning a matrix
eval_grid_int		evaluate an integer function on a grid, returning a matrix
foreach_point		execute a subroutine once for each point in grid
foreach_hline		execute a subroutine once for each horizontal grid line
foreach_vline		execute a subroutine once for each vertical grid line
foreach_proc		execute a subroutine once for each process

3.1. Helmholtz problem.

We have chosen a Helmholtz problem in which the right side is entire but nearly singular and for which the solution has a boundary layer:

$$u_{xx} + u_{yy} - 100u = 0.5\left(\alpha^2 - 100\right)\frac{\cosh(\alpha y)}{\cosh(\alpha)}$$

with Dirichlet boundary conditions imposed on the unit square. The parameter α adjusts the strength of the y-side boundary layer and the solution is nearly singular:

$$u(x,y) = 0.5\left(\frac{\cosh(10x)}{\cosh 10} + \frac{\cosh(\alpha y)}{\cosh \alpha}\right).$$

All three discretization modules, *hodie helmholtz*, *five point star*, and *hermite collocation*, and all three solution modules are applicable to this problem.

3.2. Variable coefficient self-adjoint problem.

Consider the problem (in self-adjoint form)

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

with Dirichlet boundary conditions imposed on the quarter unit circle in the first quadrant, where f is chosen so that the true solution is

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

The applicable discretization modules are *five point star* and *hermite collocation*, although for the latter we must use a rectangular domain.

3.3. Composite laminate problem.

This is a problem recently considered in a paper by Zafer Gürdal and Reynaldo Olmedo[5], in which solutions to the plane elasticity problem for a symmetrically laminated composite panel with spatially varying fiber orientations are computed. The fiber angles vary along the length of the composite laminate, resulting in stiffness properties that change as a function of location. This in-plane response of a variable stiffness panel is governed by a system of coupled elliptic partial differential equations:

$$\begin{aligned}
A_{11}(x)\frac{\partial^2 u}{\partial x^2} + A_{66}(x)\frac{\partial^2 u}{\partial y^2} + \frac{\partial A_{11}(x)}{\partial x}\frac{\partial u}{\partial x} &= p(x,y)\\
A_{66}(x)\frac{\partial^2 v}{\partial x^2} + A_{22}(x)\frac{\partial^2 v}{\partial y^2} + \frac{\partial A_{66}(x)}{\partial x}\frac{\partial v}{\partial x} &= q(x,y)
\end{aligned}$$

where

$$\begin{aligned}
p(x,y) &= -\left[A_{12}(x) + A_{66}(x)\right]\frac{\partial^2 v}{\partial x \partial y} - \frac{\partial A_{12}(x)}{\partial x}\frac{\partial v}{\partial y},\\
q(x,y) &= -\left[A_{12}(x) + A_{66}(x)\right]\frac{\partial^2 u}{\partial x \partial y} - \frac{\partial A_{66}(x)}{\partial x}\frac{\partial u}{\partial y},
\end{aligned}$$

and the A's are determined using the invariants for an orthotropic lamina. The boundary conditions, corresponding to Case 2 in Gürdal and Olmedo[5], are

$$\begin{array}{lll}
\text{at } x = 0, & u = 0, & \text{(symmetry)}\\
 & \dfrac{\partial v}{\partial x} = 0, & \text{(symmetry, no shear)}\\
\text{at } x = a/2, & u = u_0, & \text{(applied displacement)}\\
 & \dfrac{\partial v}{\partial x} = 0, & \text{(no shear)}\\
\text{at } y = 0, & v = 0, & \text{(symmetry)}\\
 & \dfrac{\partial u}{\partial y} = 0, & \text{(symmetry)}\\
\text{at } y = b/2, & v = 0, & \text{(restrained)}\\
 & \dfrac{\partial u}{\partial y} = 0, & \text{(no shear).}
\end{array}$$

The only applicable ELLPACK discretization module is *hermite collocation.*

4. ELLPACK MODULES.

In this paper we focus on six modules of ELLPACK (*hermite collocation, hodie helmholtz, five point star, band ge, sor,* and *symmetric sor cg*) which, although only a fraction of the code in ELLPACK, are representative of important classes of codes in the package. The first three, *hermite collocation, hodie helmholtz,* and *five point star*, are discretization modules, while the last three, *band ge, sor,* and *symmetric sor cg,* are solution modules. We give a short description of these modules here; for complete details on these and other modules in ELLPACK see Rice and Boisvert[13]. In the following n_x is the number of x grid points with spacing h_x, n_y is the number of y grid points with spacing h_y, and if $h_x = h_y$ then the spacing is simply referred to as h.

Hermite collocation.

This module, written by William F. Mitchell[13], discretizes a general elliptic operator with general linear boundary conditions on a two-dimensional rectangular domain D:

$$Lu = au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu = g$$

on D subject to the boundary conditions

$$Mu = s\left(\frac{\partial u}{\partial n}\right) + ru = pu_x + qu_y + ru = \phi$$

on ∂D, and a, b, c, d, e, f, g, p, q, r, s, and ϕ are functions of x and y.

This module implements finite elements to obtain a Hermite bicubic piecewise polynomial approximation to the solution. The coefficients of the approximation are determined by satisfying the problem exactly at a set of interior collocation points. Similarly the boundary conditions are satisfied at a set of boundary collocation points. The mathematical formulation is the same as *hermite collocation* with simple boundary conditions except for the collocation equations corresponding to the boundary. The equation cannot be in self adjoint form and must have a unique solution. The resulting linear system has bandwidth $4n_y$. The majority of the work is in function evaluations and computation of the discretization coefficients. The discretization error is generally $O(h^4)$.

Five point star.

This module was written by Ronald F. Boisvert and John J. Nestor, III[13]. It discretizes a general linear elliptic partial differential equation with no cross derivatives subject to a combination of Dirichlet or Neumann boundary conditions on a general two-dimensional domain D:

$$Lu = au_{xx} + cu_{yy} + du_x + eu_y + fu = g$$

on D subject to the boundary conditions

$$Mu = s\left(\frac{\partial u}{\partial n}\right) + ru = pu_x + qu_y + ru = \phi$$

on ∂D, and a, c, d, e, f, g, p, q, r, s, and ϕ are functions of x and y.

The approximation is determined by classical second order finite differences. Centered difference formulas are used to approximate derivative boundary conditions, with fictitious points outside the boundary eliminated by applying the PDE on the boundary. The bandwidth of the resulting linear system is n_x. For smooth problems on uniformly spaced grids the discretization error is $O(h^2)$, while for nonuniformly spaced grids it reduces to $O(h)$. Performance is dominated by function evaluations and computation of the coefficients of the discretization.

HODIE Helmholtz.

This module was also written by Boisvert and Nestor. It discretizes the Helmholtz equation on a rectangular domain D subject to boundary conditions having no tangential derivative components with constant boundary condition coefficients or periodic boundary conditions:

$$Lu = u_{xx} + u_{yy} + fu = g$$

on D subject to the boundary conditions

$$\begin{gathered} Mu = pu_x + su = \phi \text{ on } D_1, \\ Nu = qu_y + tu = \psi \text{ on } D_2 \text{ and/or} \\ \text{periodic on } D_3, \end{gathered}$$

where $\partial D = D_1 \cup D_2 \cup D_3$ and f, g, ϕ, and ψ are functions of x and y, and p, q, s, and t are constants unless order 2 is requested, in which case p, q, s, and t may be variable. The method uses HODIE compact finite differences (3×3 stencil) and may obtain discretization errors of $O(h_x^k + h_y^k)$ where $k = 2, 4$, or 6. However, as a higher order is requested the class of applicable problems decreases.

Band GE.

This is a solution module derived from the LINPACK[2] band solver in which row equilibration has been added by Wayne R. Dyksen[13]. It solves a real banded system by producing an LU factorization using Gaussian elimination with scaled partial pivoting. An ELLPACK interface routine reformats the linear system into LINPACK band storage in which diagonals are stored as rows.

SOR and Symmetric SOR CG.

These are iterative solution modules based on routines from ITPACK 2C and modified for ELLPACK by David R. Kincaid *et al.*[13] SOR solves a linear system of equations of the form

$$Au = b,$$

where A is both positive definite and either symmetric or nearly symmetric. These iterative methods correspond to a splitting of the matrix A in the form

$$A = Q - (Q - A),$$

where the nonsingular matrix Q is called the splitting matrix. To derive the basic iterative method we rewrite the linear system as

$$Qu = (Q - A)u + b,$$

and the iterative method is

$$Qu^{(n+1)} = (Q - A)u^{(n)} + b.$$

The splitting matrix for SOR is

$$\omega^{-1}(D - \omega C_L)$$

while for symmetric SOR it is

$$[\omega(2 - \omega)]^{-1}(D - \omega C_L)D^{-1}(D - \omega C_U)$$

where

$$A = D + C_L + C_U$$

and D is diagonal, C_L is strictly lower triangular, and C_U is strictly upper triangular. Conjugate gradient acceleration may then be used when both A and Q are symmetric positive definite in the form

$$u^{(n+1)} = \rho_{n+1}\left[\gamma_{n+1}(Gu^{(n)} + k) + (1-\gamma_{n+1})u^{(n)}\right] + (1-\rho_{n+1})u^{(n-1)}$$

with $\rho_1 = 1$ and

$$\rho_{n+1} = \left[1 - \gamma_{n+1}(\gamma_n\rho_n)^{-1}\langle\delta^{(n)}, Q\delta^{(n)}\rangle / \langle\delta^{(n-1)}, Q\delta^{(n-1)}\rangle\right]^{-1},$$

$$\gamma_{n+1} = \left[1 - \langle\delta^{(n)}, QG\delta^{(n)}\rangle / \langle\delta^{(n)}, Q\delta^{(n)}\rangle\right]^{-1},$$

where $G = I - Q^{-1}A$, $k = Q^{-1}b$, and the pseudo-residual vector is given by $\delta^{(n)} = Gu^{(n)} + k - u^{(n)}$.

5. PERFORMANCE RESULTS.

Tables 3–6 present computation times in seconds on a Sequent Symmetry S81 for various numbers of processors, along with speedup S_p = (sequential time)/(time for p processors) and efficiency $\omega = S_p/p$. Tables 3 and 4 present various discretization times for the Helmholtz and self-adjoint problems. Table 5 presents the total time for the composite laminate problem, for which only *band ge* was parallelized. Finally, Table 6 presents times for three solution modules. For the Helmholtz and self-adjoint problems we used a grid size of 128×128, resulting in linear systems of dimension 65536 for *hermite collocation* and 15876 for *hodie helmholtz* and *five point star*. For the composite laminate problem a grid size of 15×15 was used, with a total of 10 outer iterations. It is clear that the computation times of the solution modules dominate the discretization times. For the self-adjoint problem, the parameter $\alpha = 20$. This parameter does not affect run times, but only the singularity of the problem and the accuracy of the computed solution.

The various timings were obtained on the 26 processor Sequent S81 at Argonne National Laboratory. All computation times reported in the tables are the average of at least three runs; the variation between separate runs is generally less than 3%. Note that the sequential times, on which the speedup and efficiency are based, are times taken by the standard ELLPACK implementation. This partially explains the speedups greater than one, since sometimes rewriting the ELLPACK code to use the kernels results in an improvement even for one processor. The parallelization of each discretization and solution module is described below, along with remarks on our results.

Hermite collocation.

There are two versions of *hermite collocation* in ELLPACK—one assumes homogeneous boundary conditions and the other does not. We parallelized the former module in connection with an earlier paper[10]; here we discuss the work required to construct a parallel implementation of the more general version of *hermite collocation*, using the parallel kernels. Although the two versions of the module are similar, there are significant differences in how they handle the boundary conditions. Our parallel implementation is based on two primary steps: numbering the equations and unknowns and generating the equations. Each step is organized in such a way that a single parallel task corresponds to a vertical column of square elements. Thus, the equations for a given column of elements are numbered and generated independently from the equations for another column. This

Table 3. Helmholtz problem.

	Hermite collocation			five point star			HODIE Helmholtz		
p	time	S_p	ω	time	S_p	ω	time	S_p	ω
1	38.13	1.02	1.02	7.88	.92	.92	11.00	1.07	1.07
2	19.25	2.03	1.01	4.03	1.80	.90	5.68	2.08	1.04
4	9.87	3.95	.99	2.08	3.48	.87	3.12	3.78	.95
8	5.25	7.43	.93	1.08	6.71	.84	1.78	6.64	.83
16	3.10	12.59	.79	.65	11.15	.70	1.17	10.10	.63
24	2.30	16.97	.71	.57	12.72	.53	1.12	10.74	.45

Table 4. Variable coefficient self-adjoint problem.

	Hermite collocation			five point star		
p	time	S_p	ω	time	S_p	ω
1	54.22	1.08	1.08	10.53	1.05	1.05
2	27.33	2.15	1.07	6.50	1.70	.85
4	13.90	4.23	1.06	3.73	2.98	.74
8	7.20	8.18	1.02	2.32	4.78	.60
16	3.88	15.17	.95	1.63	6.81	.43
24	3.03	19.43	.81	1.40	7.93	.33

requires considerable code modifications in the original sequential version. The parallel kernels used are *dvfill* and *foreach_vline.*

The data in Tables 3 and 4 show good parallel performance for our implementation. There is some advantage over the sequential code even for one processor, and we see good reductions in the time taken by the module as processors are added. The speedups for the variable coefficient problem are slightly better simply because there is a bit more parallelizable work to do (i.e., evaluating PDE coefficients). Note that *hermite collocation* requires a rectangular domain, so the data in Table 4 is from the same differential operator posed on the unit square. ELLPACK has a discretization module, *collocation*, which allows nonrectangular domains; its parallelization would be similar to what was done for *hermite collocation.* We did not use the parallel version of *hermite collocation* for the composite laminate problem (Table 5) because the code defining the problem coefficients was relatively complicated and assumed sequential execution. COMMON blocks were used in such a way that the PDE coefficient functions could not be evaluated at different points concurrently. This points to a potential problem in parallel PDE software: users' code must often be written in such a way that it can be executed concurrently. It is not always possible to build the system so that the user can be oblivious to the parallelism that is occurring.

HODIE Helmholtz.

The usual function evaluations necessary to compute the right and left hand side stencils used by the HODIE method are minimal since the coefficients are constant. The method does, however, require extra right hand side function evaluations in applying the right hand side stencil. Hence one can see that there is more time spent in this discretization than in *five point star* in which no such extra work is required. This extra work, however, results in a fourth order method for

Table 5. Composite laminate problem.

	Hermite collocation + band GE		
p	time	S_p	ω
1	480.3	0.98	0.98
2	369.0	1.28	.64
4	242.8	1.95	.49
8	181.8	2.60	.33
16	150.5	3.14	.20
24	144.7	3.27	.14

hodie helmholtz while *five point star* is only second order accurate. For the chosen mesh size, approximately 10% of the time was spent in initialization, 70% was spent in discretization, and 20% was spent in post-processing. In the initialization we used the PDE kernels *dvfill* and *ivfill* (cf. Table 2). In the discretization code the main loop over the $n \times n$ mesh of points. The outer loop copies right hand side data to a FORTRAN COMMON work area; hence to avoid making COMMON changes, the inner loop was parallelized rather than the outer loop, affecting the granularity and resulting performance. The inner loop used *foreach_proc.* The outer loop used *dcopy.* Post-processing transfers the known boundary conditions to the right hand side of the resulting linear system, and required both *foreach_proc* and some vector clearing with *dvfill.*

The results in Table 3 reflect some speedup in the parallel code with only one processor over the sequential version of the module. We see however that the improvements tail off very sharply after 16 processors, and that the speedups are not quote as good as for *hermite collocation* or *five point star.* The part of the code that had to be left sequential, and the loss in granularity due to the complicated way in which COMMON blocks are used in the code, account for the degradation in parallel performance.

Five point star.

There are two versions of *five point star* in ELLPACK—one for rectangular domains and one for general domains. We considered the simpler rectangular domain case in Ribbens and Pitts[10]. For the general domain case one can only parallelize the generation of the interior equations. Very significant code modifications would be needed to achieve any parallelism at all in the generation of the boundary equations. Even to achieve the parallelism that we did, considerable code modifications were required.

The main problem is that the numbering of equations and unknowns is an inherently sequential process when the domain is irregular. There is no way to know what unknown number to assign to a given grid point until all "previous" grid points have been visited. Here "previous" is defined by some regular ordering of the grid points, such as the left-to-right, bottom-to-top ordering used by *five point star.* Since the interaction of the domain boundary and the grid is potentially very complicated and, in general, impossible to anticipate, one must simply visit the grid points in order, deciding which will correspond to equations and unknowns, and which will not. Our strategy is to do this in an initial (sequential) pass that does as little work as possible. However, we do generate the boundary equations during this first pass, because you have to do much of the work required to generate an equation in order to decide if one is necessary. Then in a second (parallel) pass over the grid points we generate the interior equations. We use *foreach_hline,* so that generating the

Table 6. Solvers on 5-point star discretization of Helmholtz problem.

	band GE			SOR			symmetric SOR CG		
p	time	S_p	ω	time	S_p	ω	time	S_p	ω
1	1730.2	1.00	1.00	216.4	1.15	1.15	828.7	1.08	1.08
2	1012.5	1.72	.86	112.0	2.23	1.11	457.8	2.04	1.02
4	564.9	3.08	.77	58.1	4.30	1.07	250.5	3.66	.89
8	321.2	5.36	.67	32.6	7.66	.96	158.0	5.68	.71
16	265.1	6.56	.41	20.8	12.00	.75	112.7	8.00	.50
24	245.1	7.20	.30	17.5	14.35	.60	99.6	8.97	.38

interior equations corresponding to a single horizontal grid line comprises a single parallel task. The kernel *dvfill* is also used for some initialization.

Tables 3 and 4 indicate that our parallel *five point star* module performs relatively well on the Helmholtz problem but slightly less efficiently on the variable coefficient self-adjoint problem. The reason is due to the geometry, not the differential operator. For the data in Table 4 the domain used was a quarter of the unit circle, while for Table 3 the domain was the unit square. We see that the difficulty in parallelizing the general domain case makes a noticeable difference in the parallel performance, especially for 16 or more processors.

Band GE.

The *band ge* module is a modification of LINPACK's *dgbfa* and *dgbsl*. The work is dominated by vector operations (e.g., *daxpy* from BLAS1) used to update the lower right submatrix as Gaussian elimination proceeds. This step is equivalent to a rank one update of the form $A = A + \alpha x y^T$, implemented in the BLAS2 kernel *dger*. Hence, it is relatively straightforward to implement a parallel version of *band ge* based on parallel kernels. The main modification required is that a row interchange, which the original version does deep within the loop we want to parallelize, must be pulled out of the loop and done before the main parallel step. There are also a few other BLAS1 kernels which are used in the parallel version. As is typical, as the number of processors grows an increasingly serious bottleneck are the triangular system solves. Almost no speedup is possible in these routines. For example, with a 96×96 grid (8836 equations) and $p = 1$ processor, 16.4 seconds out of 535.5 (3%) are spent in the triangular solves, while with $p = 8$ processors 8.8 out of 84.8 seconds (10%) are spent on this step. One step in the computation that is not parallelized is the search for a pivot at each step. This could be parallelized a bit, but the overall improvement would be small.

The parallel performance of *band ge*, illustrated by the data in Table 6, is reasonably good up through eight processors. For more processors, however, the sequential bottlenecks start to become serious. It should be noted that the parallel performance of *band ge* is better on matrices with wider bandwidth, as would be generated by *hodie helmholtz* and *hermite collocation*, for example. The reason is that the work in the dominant parallelized step of the computation grows like the square of the bandwidth.

Regarding the composite laminate problem, notice that since *band ge* is the only parallel step of this computation, unimpressive speedups are the result. For this problem the systems which *band ge* solves are only of dimension 900. Hence, it is not surprising that we achieve poor speedups. Of the 480 seconds taken by one processor on the composite laminate problem, 436 seconds are used by *band ge* solves; with $p = 16$ processors, 109 seconds of the 150 are spent in *band ge*. So in addition to the poor speedups in the linear solves, there are approximately 40 seconds of purely sequential computation.

SOR.

The parallel performance of *sor* depends heavily on what "indexing" module is used to reorder the equations and unknowns. ELLPACK allows a matrix generated by any discretization module to be reordered arbitrarily, essentially by specifying permutation matrices to be applied on the right and left. For the standard five point star discretization with the so-called natural ordering (left-to-right, bottom-to-top) *sor* parallelizes very poorly. The reason is that a triangular solve is one of the dominant computational steps of the algorithm in this case; and it is well known that triangular solves parallelize very poorly. On the other hand, if red-black ordering is used the algorithm parallelizes quite well because the tridiagonal solve is replaced by a matrix vector multiply and a solve with a diagonal matrix. The matrix vector product is implemented in the parallel kernel *dyasx2*. Several BLAS1 kernels are also used in our implementation.

The data in Table 6 show that the parallel code with one processor is significantly faster than the sequential code. This is due to some loop reordering that was done in the parallel case which makes a vector *gather* unnecessary, and due to the increased use of BLAS1 routines, several of which are implemented with an eye toward sequential efficiency (e.g., loop unrolling, etc.). Parallel efficiency for *sor* is good up through 16 processors; going to 24 processors for this problem does not bring much improvement. The *sor* module does not parallelize quite as well as the even simpler ITPACK module *jacobi cg*[10]. The reason is that with red-black ordering the amount of data for many of the parallel tasks is cut in half, and because there is some additional sequential work in adaptively selecting the *sor* parameter ω.

Symmetric SOR CG.

The parallel performance of *symmetric sor cg* suffers in the same way as *sor* if natural ordering is used. The data in Table 6 are based on red-black ordering. Unfortunately, it is not uncommon that this module works better sequentially if natural ordering is used. The Helmholtz test problem used here is a dramatic example of this pattern. With natural ordering *symmetric sor cg* converges in only 31 iterations; 202 iterations are needed if red-black ordering is used. Since red-black ordering yields by far the better parallel speedups, it is very difficult to determine *a priori* which choice gives the best parallel algorithm.

Our parallel *symmetric sor cg* is very similar to the parallel *sor* module. The primary work is in the parallel kernel *dyasx2* (sparse matrix-vector multiply) and several BLAS1 kernels. The parallel speedups are somewhat worse than for *sor* because there is more sequential computation here in computing the parameters for conjugate gradient acceleration. There are also a few more dot products, for which most of the work is done in parallel, but which do require a sequential combination of the results from each processor into a single value. This becomes a bottleneck as the number of processors grows. We see that beyond 8 processors the parallel efficiencies drop sharply for our example.

6. CONCLUSIONS.

Although only a small number of ELLPACK routines have been parallelized here and in previous work, and only three test problems have been considered, some valid and significant conclusions can be drawn.

- *The user cannot be completely insulated from parallelization concerns.* The composite laminate problem illustrates two real difficulties in doing nontrivial engineering calculations with a parallel mathematical software package: (1) The user must be at least somewhat aware of the parallelism. In this case the ELLPACK code from Gürdal and Olmedo[5] defining the

PDE coefficients was written in a way that assumed sequential execution. (2) This is not a trivial calculation, yet the most easily parallelized step (the linear system solves) is not very substantial — only 900 equations. The cost of the overall computation is due to the outer loop (used to solve a system of coupled nonlinear equations), which is inherently sequential. Other algorithmic approaches to the problem might parallelize better. For example, instead of dividing the inner loop into two steps (first solving one equation and then the other), the granularity could be increased by constructing one big system to solve in the inner loop. This then requires more sophisticated programming, and stretches the capability of ELLPACK.

- *Efficient use cannot easily be made of large numbers of shared memory processors.* For our test cases it was rarely efficient to go beyond 16 processors. More processors would only be useful if the problems were even bigger, but then memory and swap space on the file system become significant issues.
- *Iterative solvers are better suited to parallelization.* Both here and in previous work we saw that iterative solvers have better sequential times (this is well known for elliptic PDEs), and they have better speedups. The advantage in speedup is reduced as bandwidth grows, though, and as the iterative methods get more sophisticated (Jacobi to SOR to SSOR CG).
- *Geometry (general domains) can cause serious problems for parallelizing discretization modules.* This is true for any computation that is governed by geometry. Complicated boundaries introduce some steps that are inherently sequential—you don't know what to do next until you've seen a certain amount of the "previous" boundary. Domain decomposition ideas could help here, but only with a considerable increase in algorithmic complexity.
- *The kernel-based approach is still viable.* Previously Ribbens and Pitts[10] showed that the kernel-based approach achieves virtually the same speedups as doing the parallelization by hand (nonportably), but locates the parallel code in a relatively small number of routines. It does take considerable work in some cases to re-organize the sequential code to use the kernels, but once that has done, re-implementing the kernels on another shared memory machine should be straightforward. In principle the same remark applies to distributed memory machines, although as mentioned above, perhaps the entire algorithm and computation needs to be reorganized to take full advantage of a distributed memory architecture, where data distribution is a much more serious and difficult problem. A way to quantify the advantage of locating all the parallel code in the kernels is that there are approximately 100,000 lines of code in ELLPACK modules, but the kernels (implemented for the Sequent) only amount to about 2,000 lines. Other advantages of the kernel approach (such as even improving the sequential time) are mentioned in Ribbens and Pitts[10,11].

Finally, while the kernel-based approach to parallel mathematical software, and the use of mathematical software packages for real engineering problems are advocated, the results here suggest that an efficient parallel production environment for engineering computations remains a distant goal.

7. REFERENCES.

[1] D. S. Dodson and J. G. Lewis, "Proposed sparse extensions to the basic linear algebra subprograms," ACM Signum Newsletter **20:1**, 22–25 (1985).

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

[3] J. J. Dongarra, J. DuCroz, S. Hammarling, and I. Duff, "A set of level 3 basic linear algebra subprograms," ACM Trans. Math. Softw. **16**, 1–17 (1990).

[4] J. J. Dongarra, J. DuCroz, S. Hammarling, and R. J. Hanson, "An extended set of FORTRAN basic linear algebra subprograms," ACM Trans. Math. Softw. **14**, 1–17 (1988).

[5] Z. Gürdal and R. Olmedo, "In-plane response of laminates with spatially varying fiber orientations: 'variable stiffness concept'," AIAA J., to appear.

[6] D. R. Kincaid, J. R. Respess, D. M. Young, and R. G. Grimes, "ITPACK 2C: A FORTRAN package for solving large sparse linear systems by adaptive accelerated iterative methods," ACM Trans. Math. Softw. **8**, 302–322 (1982).

[7] Kuck & Associates, *KAP/Sequent User's Guide*, Champaign, IL, 1989.

[8] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for FORTRAN usage," ACM Trans. Math. Softw. **5**, 308–323 (1979).

[9] T. C. Oppe and D. R. Kincaid, "Are there iterative BLAS?," in *Proceedings of the Copper Mountain Conference on Iterative Methods*, 1990.

[10] C. J. Ribbens and G. G. Pitts, "Strategies for parallelizing PDE software," In *Advances in Computer Methods for Partial Differential Equations VII*, G. Richter, ed., IMACS, 1992.

[11] C. J. Ribbens and G. G. Pitts, "Strategies for parallelizing PDE software," Tech. Rep. 92–34, Dept. of Computer Science, Virginia Polytechnic Institute & State Univ., Blacksburg, VA, 1992.

[12] C. J. Ribbens, L. T. Watson, and C. deSa, "Toward parallel mathematical software for elliptic partial differential equations," ACM Trans. Math. Softw., 1992, to appear.

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