CHAPTER 20

Parallel-Vector Computation with High Degree Element-By-Element Methods

E. Barragy*
G.F. Carey*

Abstract. Element-by-element techniques and variants of this class of methods may be interpretted as a form of abstract domain decomposition procedure. The EBE approach, implemented as a block iteration form of domain decomposition, offers greater flexibility than more standard, and restrictive, domain decompositions. Hence it can be incorporated more easily in existing finite element software. Here we consider EBE conjugate gradient solution and some aspects of parallel and vector processing. Of particular interest in the present study are spectral (p) methods, with large element degree p. Numerical results are presented for representative linear and nonlinear model problems.

1. Introduction. Domain decomposition in its simplest form consists of dividing the domain of a problem into a number of subdomains (possibly overlapping) and then interactively adjusting the global solution using local subdomain solutions. This idea was suggested by Schwarz in the 1800's for analytical solution of Poisson's equation using overlapping subdomains [5]. Since it offers a conceptually simple divide and conquer strategy, the idea is also appealing from the standpoint of parallel processing. The number of subdomains, size of subdomain grids and connectivity of the subdomains will influence the performance of the global iteration and also the efficiency on a given parallel computer architecture. For example, the number of subdomains may be an integral multiple of the number of processors and the grid size (number of nodal unknowns) equal for each subdomain to achieve perfect load balance. For practical applications with complex geometries, equilibrating the subdomain grids for load balance is more difficult, and has led to so called "annealing" or grouping procedures where the subdomain interfaces can be quite complicated.

*Department of Aerospace Engineering/Engineering Mechanics, University of Texas at Austin, Austin, TX 78712-1085.

Subdomain iteration strategies of this type are equivalent to block iterative methods with the block partitioning defined by the subdomain prescription [21,22]. Hence, it is useful to examine these domain decomposition methods from this perspective. The finite element procedure begins by discretizing the domain into a union of elements. Hence, individual elements or collections of elements can be used to define the subdomains. If the elements in each subdomain are contiguous, one obtains a standard domain decomposition. If instead, the elements are not so closely associated then the subdomains may be collections of scattered elements. Finally, individual elements may be taken as the subdomains. Each of the above choices may place restrictions on the type of basic block iterative method to be implemented, but each can be accelerated by the method of conjugate gradients or biconjugate gradients.

2. Discussion.

2.1. Element and "Block" Parallelism. The main steps in such a procedure involve element matrix-vector products, accumulation (or extraction) of local and global vectors, and global inner products. The element matrix-vector products are easily parallelized over the elements to achieve near-optimal load balance with both coarse and fine granular parallel systems. Accumulation or extraction of vectors and inner product calculations are more troublesome. Difficulties can arise, for instance, due to communication overhead on distributed parallel systems. The communication overhead depends on the parallel architecture, hardware characteristics, and complexity of the subdomain interfaces. In an element-by-element subdomain procedure the element boundaries are the interfaces. If a large number of simple elements are used, the measure of the interfaces is large. Then the associated interface-related overhead in, for instance, communication will be high. Moreover, the element calculations will be small for simple low-degree elements such as bilinear rectangles ($p = 1$). Hence, for reasonable processors the system will be communication-limited.

One alternative is to group the elements contiguously into subdomains, each containing approximately the same number of elements and the same small interface measure. For example, large square subdomains or rectangular strips might be appropriate for certain simple geometries. This is one

form of the "annealing" technique referred to earlier. Again, the number of subdomains are restricted to be an integer multiple the number of processors. The element-by-element matrix formation, vector products and inner products can then be made within each subdomain for a given processor. Interchanges for global vector accumulation, dot products, etc. then involve only the subdomain interfaces. If the subdomains are "topologically" square, then the interior work increases with the number of elements N and the interface grows with $4\sqrt{N}$. The domain size can therefore be chosen so that the balance between internal subdomain computation and interface operations can be greatly improved. The interior subdomain computations can also be vectorized over the elements in the subdomain.

Other strategies based on this standard type of domain decomposition are also possible. For example, in the spirit of substructure analysis, the interior unknowns for each subdomain can be eliminated retaining only interface nodes. Each subdomain is then essentially a super-element and the resulting system can be solved using parallel block iteration (a superelement EBE scheme) or otherwise [1,4]. Various strategies involving exact or approximate block inverses can be incorporated. They may involve full factorization, fast direct methods, incomplete factorizations for preconditioners, or a conjugate gradient procedure within each subdomain at each outer iteration [2,3].

2.2 High-Degree Elements. Rather than use subdomains consisting of simple elements, the EBE approach can be applied directly to blocks made up of single high-degree elements or a few high-degree elements. It is well known that for regular solutions the approximation error reduces with decreasing mesh size h and with increasing polynomial degree p. This corresponds to the so-called h and p refinement procedures, respectively. Moreover, the error drops dramatically with increasing p [7,9]. Thus, the use of subdomains containing one or a few high-degree elements will require dramatically fewer unknowns per block and per interface to achieve the same accuracy. However the computation within the subdomain is still relatively large and dense particularly is 2D. Thus a more favorable computation/communication ratio is achieved as p is increased. In addition, for sufficiently large p, computations can be vectorized within the

elements, rather than over the elements. This yields a procedure that is extremely simple to vectorize and parallelize. For example, if biquintic elements ($p = 5$) are used with an EBE domain decomposition (simple element subdomains) then each subdomain (element) has 36 nodes and will generate a dense 36×36 element matrix. Also 20 edge nodes are now fully connected. A 5×5 subdomain mesh of bilinear elements would be less accurate but offer the same number of nodes (and a different sparse connectivity). Efficient vectorization is more difficult to achieve and one faces added book-keeping problems associated with "joining" the 5×5 block of elements into a subdomain.

As element degree p increases, the conditioning degrades and appropriate preconditioners are needed to achieve an efficient solution. (A parallel scheme that is perfectly balanced but interminably slow to converge is of no use.) Moreover, the preconditioner should ideally be parallelizable at the element (or subdomain) level. For certain types of frequently encountered operators, simply choosing a special element basis will suffice [23,24]. In the numerical studies later, we consider a Helmholz operator and use the integrated legendre polynomials. More generally, special bases can be constructed at the element level that exploit (approximately) the orthogonality properties associated with the given operator and produce better conditioned systems. Other preconditioners, such as diagonal Jacobi or element block preconditioning, can be used in conjunction with such schemes to further accelerate convergence [6,8,18,20].

A variable p finite element analysis and an EBE conjugate gradient parallel solver have been developed. Details of our EBE gradient scheme are given in [10] and the parallel ideas in [12,16,17]. For other related work see also [11,19]. In the following numerical study we focus on EBE schemes using single element subdomains. A range of p values is considered. Studies are made of vectorization on a Cray X-MP and parallelization on a shared memory multiprocessor (Alliant FX/8). Results are given for both linear and nonlinear model problems in 2-D.

3. Numerical Studies. We first consider the following elliptic model problem in domain Ω,

$$-\nabla^2 u + \pi^2 u = f \quad \text{in } (0,1)\times(0,1)$$
with $$u = 0 \quad \text{on boundary } \partial\Omega$$

The data f is chosen to correspond to the exact solution

$$u(x,y) = \sin(\pi x)\sin(2\pi y) + (x^2 - x)e^x(y^2 - y)e^y$$

The second problem considered is the stream function vorticity formulation of the 2-D incompressible Navier-Stokes equations,

$$Re(U \cdot \nabla\omega) - \nabla^2\omega = f$$
$$-\nabla^2\psi - \omega = 0 \quad \text{in } (0,1)(0,1)$$

with $\psi = 0$ on boundary $\partial\Omega$ and where $\boldsymbol{U} = (\partial\psi/\partial y, \partial\psi/\partial x)$ denotes the velocity. Details of the mixed method used for the discretization and the application of the vorticity boundary conditions can be found in [13] The data f is chosen to correspond to the exact solution

$$\psi(x,y) = \sin^2(\pi x)\sin^2(\pi y)$$

Newton iteration is used to solve the coupled system for a given value of Re, the Reynold's number. Thus a linear system is solved at each Newton step, and an EBE scheme can be applied for each such solution.

For both problems the domain is discretized with various uniform meshes of elements depending on the combination of h and p chosen. For the first problem an integrated Legendre polynomial basis is chosen since this gives reasonable conditioning [23]; for the second problem a Lobatto basis is chosen, also for conditioning considerations [14]. A conjugate gradient iterative solver is employed for the first problem for the parallelization results; a biconjugate gradient solver is used for the vectorization results. For the stream function vorticity problem a biconjugate gradient solver is used in both studies. The initial solution iterate is taken to be zero and the convergence criterion used was $r_n/r_0 < \epsilon = h^{(p+1)}/(p+1)!$ where r_n denotes the residual norm of the preconditioned system for the nth iterate and r_0 for the first iterate. The convergence criterion is based on a simple Taylor series estimate for discretization accuracy. Thus we attempt to solve

each system only up to full discretization accuracy. Unless otherwise noted, Jacobi preconditioning is used.

For these two test cases we are interested in four issues. Most importantly we would like to determine the performance of the EBE formulation in a Fortran "dusty deck" implementation on shared memory parallel and vector processors. The performance is expected to vary greatly with h, p and for the two test cases considered. Note that the element matrices are twice as large for the second problem as for the first. We emphasize that the performance figures shown are for standard Fortran code, no "hand tuning" was applied. Issues of subsidiary interest are the accuracy attainable for various choices of h, p and the associated computational costs, and the level of parallelization possible with sophisticated block preconditioners such as those in [14]. The parallel computations were carried out in double precision on an FX/8 Alliant multiprocessor configured with six processors in the "complex". To determine speedup, we run each discretization for parallel code executing on a single "detached" processor and for parallel code executing on the "complex" with six processors. The speedup is then reported as the ratio of these two run times. This should be considered an upper value for the speedups, as the overhead for parallelization implementation in the single processor code is not subtracted out. Previous experience indicates a reasonable degradation of computed speedups due to the subtraction of this overhead [12] Note that we do not include the grid generation phase of the computations in the timings nor do we include the post processing to compute the L^2 error. The vectorized computations were carried out on a single CPU Cray X-MP. The CFT77 Fortran compiler was used and the performance figures reported were obtained with the PERFTRACE utility. Only the performance for selected routines is reported. Specifically, this includes element formation, matrix vector products, summing the results of the element matrix vector products, and the biconjugate gradient routine, which is mostly BLAS level 1 operations.

The parallelization results are shown in Tables 1 and 2 below, for the first and second problems respectively. Each entry has the value of the L^2 error, the solution time t_1 for a single detached processor and for six processors t_6 in the complex, as well as the speedup S obtained. No entry indicates the case was not run. An element block type preconditioner [14] is used in the second set of values reported ($p_b = 6$).

Table 1: Parallelization of Diffusion Problem.

		$h = 1/6$	$h = 1/12$	$h = 1/24$	$h = 1/48$
$p = 1$	L_2	4.9e-2	1.3e-2	3.2e-3	8.0e-4
	t_1	0.187	0.976	6.680	53.38
	t_6	0.053	0.259	1.89	15.70
	S	3.53	3.77	3.54	3.40
$p = 2$	L_2	3.4e-3	4.2e-4	5.2e-5	
	t_1	0.99	5.94	37.9	
	t_6	0.22	1.32	8.8	
	S	4.52	4.50	4.31	
$p = 3$	L_2	2.0e-4	1.3e-5		
	t_1	4.08	22.17		
	t_6	0.80	4.48		
	S	5.07	4.95		

		$h = 1/3$	$h = 1/6$		
$p = 6$	L_2	2.0e-6	1.6e-8		
	t_1	22.1	113.2		
	t_6	5.1	20.5		
	S	4.3	5.5		
$p_b = 6$	L_2	2.0e-6	1.6e-8		
	t_1	21.3	129.0		
	t_6	5.1	25.1		
	S	4.1	5.1		

Examining the relative L^2 errors for a given p, we first note that they all appear to converge in accordance with the standard theoretical estimate for regular problems ($||u - u^*||_{L_2} = Ch^{(p+1)}/(p+1)!$, where u is the analytic solution and u^* is the finite element solution). This is to be expected as both solutions are quite smooth. Next note that if a one percent L^2 error tolerance was desired for the solution of the first problem, the best choice here is $(h, p) = (1/12, 1)$. If a 0.1 percent error tolerance is desired, then we could choose $(1/24, 1)$ or $(1/6, 2)$. However, choosing the latter discretization results in a savings factor of 6 in the execution time. For the

Table 2: Parallelization of Stream Function Vorticity Problem.

		$h = 1/6$	$h = 1/12$	$h = 1/24$	$h = 1/48$
$p = 1$	L_2	1.57	0.40	9.9e-2	2.5e-2
	t_1	7.59	92.2	980.0	9429.
	t_6	1.93	24.0	258.6	2528.
	S	3.93	3.85	3.79	3.73
$p = 2$	L_2	0.11	1.4e-2	1.8e-3	
	t_1	109.8	1225.	6291.	
	t_6	23.9	264.	1377.	
	S	4.59	4.64	4.57	
$p = 3$	L_2	6.7e-3	4.3e-4		
	t_1	642.	7320.		
	t_6	128.	1452.		
	S	5.0	5.04		

		h = 1/3	h = 1/4	h = 1/5	
$p = 6$	L_2	6.7e-5	9.1e-6		
	t_1	1679.	5277.		
	t_6	391.	1066.		
	S	4.29	4.95		
$p_b = 6$	L_2	6.7e-5	9.1e-6	1.9-6	
	t_1	1090.	3121.	6255.	
	t_6	264.	655.	1388.	
	S	4.13	4.77	4.51	

extreme error tolerance of 10^{-4} one could choose from three discretizations: $(1/48, 1)$, $(1/12, 2)$, or $(1/6, 3)$. Comparing the best $(p = 3)$ and worst $(p = 1)$ execution times the savings factor is over 13. For the second problem, where the L^2 error of the vorticity is given, we find a savings factor of 8 for a one percent error tolerance ($(h, p) = (1/48, 1), (h, p) = (1/12, 2)$), and 5 for a 0.1 percent error tolerance ($(h, p) = (1/24, 2), (h, p) = (1/6, 3)$). As expected, p convergence is much more efficient than h convergence for the two problems considered.

Examining the results of the first problem, we find speedups of about 3.5 to 3.9 for $p = 1$, falling off as the mesh is refined. The speedups increase with p up to a level of 5.0 at $p = 3$ and 5.5 at $p = 6$. (We do not consider the cases of $(h, p) = (1/3, 6), (1/4, 6), (1/5, 6)$ as these are not perfect load balance discretizations.) Applying Amdahl's Law to fit the observed speedups for the first problem, the performance is "equivalent" to perfect parallelization with 14% serial code at $p = 1$, 7% at $p = 2$, 4% at $p = 3$ and 2% at $p = 6$. The fraction of serial code for the second problem would generally be slightly less. These results are quite disastrous when compared to previous work (e.g., [12]) which showed good speedups obtained with biquadratic elements on similar problems. The answer lies partially in the routine which takes element matrix vector product results and sums them into global results. In this work, the routine which does the "summation" was not parallelized. In the previous study it was heavily parallelized by forming the elements into disjoint groups and performing the sum in parallel within each group. Comparing floating point operation counts for the element matrix vector products and for the "summation" operation we find a ratio given as $2(p + 1)^2$. Correcting for the added CPU times in the element formation routines we estimate fractions of serial code of $8\%, 3\%, 2\%$ for $p = 1, 2, 3$, which accounts for a substantial part of the observed serial code. Next we note that only "user" time is reported in the results in the tables. Substantial "system" time was observed for $p = 1$, decreasing as p increased but increasing as the mesh is refined. Beyond $p = 3$ there is little system time involved. Generally large "system" times indicate considerable paging activity in the program. Based on this observation, we deduce that the rest of the serial code deficit is a result of memory conflicts and simple page faulting, although we have no page fault statistics to prove this.

Examining the parallel performance of the block preconditioner we find speedups of about 5 at $p_b = 6$, which results in a serial code fraction of 3.5%. Thus the block preconditioner is highly parallelizable. However, it is not as parallelizable as Jacobi iteration, which yields a speedup of 5.5. Speedups of about 4.5 for the block precondiditoner and 5.0 for Jacobi are found for the second problem. Note that the number of elements precludes perfect load balancing for the second problem, thus we see smaller speedups. In order to assess the iterative performance of the block preconditioner we

can compare the t_1 timings for $p = 6, p_b = 6$ in Tables 1 and 1. In Table 1 we find approximately equal performance: the block preconditioner is 4% faster for $h = 1/3$ and 14% slower for $h = 1/6$. In Table 2 we find the block preconditioner 35 and 41% faster for $h = 1/3$ and $h = 1/6$. Thus the block preconditioner is highly effective for the i coupled stream function vorticity problem.

The results of the vectorization study are given in Tables 3 and 4 for the first and second problems respectively. The tables show MFLOP rates and CPU time spent in a routine, for four selected routines which comprise most of the solver in the code. The routines are: ELEM, which forms the element matrices, AX/AXT which takes matrix vector products and transposes, SPRAY which combines elemental matrix vector product results, and BCG which is the biconjugate gradient driver routine. In addition, the average MFLOP rate for the solver is given under AVG.

Table 3: Vectorization of Diffusion Problem.

h, p	ELEM	AX	SPRAY	BCG	AVG
1/24, 1	16.8	7.5	3.5	78.	11.9
	.087	.163	.003	.004	
1/12, 2	21.9	17.6	9.0	78.2	21.1
	.074	.076	.003	.004	
1/6, 4	46.1	46.5	23.1	81.6	47.0
	.115	.088	.004	.007	
1/4, 6	75.9	72.5	41.7	84.5	74.6
	.212	.168	.003	.012	
1/3, 8	91.0	84.2	43.4	85.5	88.0
	.436	.302	.003	.016	

The vectorization results for the diffusion problem indicate basically poor performance for p less than 3, reasonable performance for $p = 4$ and good performance for $p = 6, 8$. This is quite expected given the way the routines are coded, *i.e.*, on an element-wise basis. Only when $p = 6$ or more do the inner loops in the routines become large enough for effective vectorization. At an average speed of 75 MFLOPs for $p = 6$ and 88 for $p = 8$ we feel that our "dusty deck" is doing remarkably well in terms

Table 4: Vectorization of Stream Function Vorticity Problem.

h, p	ELEM	AX	SPRAY	BCG	AVG
$1/24, 1$	15.6	6.3/5.9	5.2	109.	16.1
	5.72	12.1	2.2	.29	
$1/12, 2$	20.4	33.9/36.3	10.5	110.	30.8
	4.4	7.83	.72	.34	
$1/6, 4$	43.3	71.7/54.5	22.9	111.	56.4
	5.67	11.6	.31	.46	
$1/4, 6$	60.2	92.3/63.6	32.1	111.	70.5
	9.3	19.1	.23	.53	
$1/3, 8$	76.2	106.6/68.5	32.9	112	80.1
	19.5	26.8	.21	.54	

of utilizing the machine's resources effectively. The results for the stream function vorticity problem are somewhat worse. The code structure for the nonlinear problem is quite different than that for the linear diffusion problem, and this leads to many more categories of significant computational content. For example, the application of the boundary conditions for the stream function vorticity problem is a respectable part of the total CPU time, but it is not for the diffusion problem. Rather than give a detailed breakdown with many new categories, the same timing categories used for the diffusion problem in Table 4 are also used in Table 4. We have lumped the various timings for the stream function vorticity problem into the closest applicable functional category. For example, the application of the boundary conditions is included in the ELEM (element formation) category. Examining the AVG results in Table 4 we see similar performance to that of Table 4 for the $p = 1$, better performance for $p = 2, 4$ and worse performance for $p = 6, 8$. Overall, this is quite unexpected, as the vector lengths should be twice as long for the stream function vorticity problem as the diffusion problem. However, examining the code structure, it is clear that loops are often not configured to take advantage of this extra vector potential. Rather they are written with other considerations in mind, such as readability and code maintainability. It is still suprising that such routines as SPRAY and ELEM exhibit lower performance on the stream

function vorticity problem. This may indicate added memory conflicts, or compiler difficulty in utilizing the gather/scatter hardware. Perhaps most interesting are the MFLOP rates for routines AX and AXT. These routines are expected to perform better as they are configured to take advantage of the longer vector lengths. For $p = 1$ we find worse performance, but for $p = 2, 4$ we find $50 - 100\%$ better performance. For $p = 6, 8$ we get approximately equal performance. However, examining the rates for AX and AXT spearately, we see that the performance problems are due entirely to the transpose product routine, AXT. It is unclear what is causing the problem at this time.

Effective vectorization of the low p discretizations requires that the inner loop in the routine run over the number of elements in the discretization. Generally this is reasonable for an h type finite element method as many elements will be required. Performance of these types of schemes has been reported in [15]. Such schemes could be incorporated into the existing code at the expense of added bookkeeping, thereby significantly improving performance in the low p regime. This is essential if $h - p$ type discretizations are to be considered. It should also be noted that although the results presented here were obtained for a rectangular grid, similar performance should be expected from any mesh. In contrast, while many authors have obtained speeds of 150+ MFLOPS for similar problems [15,16], generally their results have involved many more unknowns: $5000 - 20,000$ as compared to 625 here, and are generally for problems on rectangular grids.

4. Conclusion. Parallel and vector performance results for a standard spectral EBE scheme were presented for both a linear and a nonlinear problem. The results indicate a high degree of parallelization and vectorization in the formulation for $p \geq 6$. No hand tuning of the code was allowed to obtain the results, thus reinforcing the theme that the EBE method *naturally* introduces parallelization and vectorization into a finite element code, given a sufficiently high p. Previous work, and the work of other authors, indicate that parallelism and vectorization considered separately are naturally introduced for any choice of p, provided that the code is properly restructured. The present work shows that both can be achieved with a high degree of efficiency in a single code. Calculations are performed in parallel over single element subdomains and vectorized within each element.

This suggests that the method is ideally suited for Class VI supercomputers belonging to the shared memory coarse grain vectorizing multiprocessor variety, such as the Cray Y-MP. To produce a general purpose code that performs well on parallel, vector and parallel/vector machines for $h-p$ type FEMs will require some restructuring of the low level algorithm coding to accomodate low degree elements. The EBE formulation however, remains intact. Examples of such low level recoding include: do loop unrolling, inner loops of matrix vector products over the element matrices for low p elements, and the partition of the elements into disjoint subsets for the parallel accumulation of element residual vectors into global residual vectors. The results also indicate the utility of element block preconditioners for coupled problems such as the stream function vorticity example. Not only are such preconditioners effective on uniprocessors, they parallelize quite nicely also.

REFERENCES

[1] O. Axelsson, G. Carey, and G. Lindskog, *On a class of preconditioned iterative methods on parallel computers*, Int. J. Num. Meth. Eng., 27, 1989, (in press).

[2] T. Chan (ed.), *Proc. Second Int. Symp. on Deomain Decomposition Methods for PDE's*, SIAM Publications, Phil., 1988.

[3] R. Glowinski, G. Golub, G. Meurant, and J. Periaux (eds.), *Proc. First Intl. Symp. on Domain Decomposition Methods for PDE's*, SIAM Publications, Phil., 1987.

[4] B. Noor-Omid and B. N. Parlett, *Element preconditioning using splitting techniques*, SIAM J. Sci. Stat. Comp., 6, (1985), pp. 761–771.

[5] H. A. Schwarz, *Über einige abbildungsaufgaben*, Geo. Math. Abb., 11, (1869), pp. 65–83.

[6] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

[7] I. BABUSKA, B. A. SZABO, AND I. N., KATZ, *The p-version of the finite element method*, SIAM J.N.A., 18, 3, (1981), pp. 515–544.

[8] I. BABUSKA, A. CRAIG, J. MANDEL, AND J. PITKARANTA, *Efficient Preconditioning for the p Version Finite Element Method in Two Dimensions*, preliminary version, 1989.

[9] A. T. PATERA, *A spectral element method for fluid dynamics; laminar flow in a channel expansion*, J. Comput. Phys, 54, (1984), pp. 468–477.

[10] G. F. CAREY AND B. JIANG, *Element-by-element preconditioned conjugate gradient algorithm for compressible flow*, in Innovative Methods for Nonlinear Problems, W.K. Liu, T. Belytschko, and K.C. Park, eds., Pineridge Press, Swansea, UK, (1984), pp. 41–49.

[11] T. J. R. HUGHES, I. LEVIT, AND J. WINGET, *Element-by-element solution algorithm for problems of structural and solid mechanics*, Comp. Meth. Appl. Mech. Eng., 36, (1983), pp. 241–254.

[12] E. BARRAGY AND G. F. CAREY, *A parallel element-by-element solution scheme*, IJNME, 26, (1988), pp. 2367–2382.

[13] E. BARRAGY AND G. F. CAREY, *Stream function vorticity boundary conditions and block element preconditioning*, (in preparation), 1989.

[14] E. BARRAGY AND G. F. CAREY, *Preconditioners for high degree elements*, (submitted), 1989.

[15] L. J. HAYES AND P. DEVLOO, *A Vectorized version of a sparse matrix vector multiply*, IJNME, 23, (1986), pp. 1043-1056.

[16] G. F. CAREY, R. MCLAY, M. SHARMA, AND E. BARAGGY, *Element-by-element vector and parallel computation*, CANM, 4, (1988), pp. 299-307.

[17] G. F. CAREY, *Parallelism in finte element modelling*, CANM, 2, (1986), pp. 281–287.

[18] I. GUSTAFSSON, *A class of first order factorization methods*, BIT, 18, (1978), pp. 42–156.

[19] L. J. HAYES AND P. DEVLOO, *An Element-by-element block iterative method for large nonlinear problems*, in ASME-WAM, New Orleans, W. K. Liu, K. C. Park, T. Balytschko, eds., *Innovative Method for Nonlinear Behavior*, Pineridge Press, 1985.

[20] J. MANDEL, *Two-level domain decomposition preconditioning for the p-version finite element method in three dimension*, in Fourth Copper Mountain Conference on Multigrids, April 9-14, 1989.

[21] P. E. BJORSTAD AND O. B. WIDLUND, *To overlap or not to overlap: a note on a domain decomposition method for elliptic problems*, SIAM J. Sci. Stat. Comput., 10, 5, (1989), pp. 1053–1061.

[22] T. CHAN AND D. RESASCO, *Analysis of domain decomposition preconditioners on irregular regions*, in Advances in Computer Methods for Parital Differential Equations, R. Vichnevetsky and R. Stepleman, eds., IMACS, 1987.

[23] G. F. CAREY AND E. BARRAGY, *Basis function selection and preconditioning high degree finite element and spectral methods*, BIT, 1989 (to appear).

[24] I. BABUSKA, M. GRIEKEL, AND J. PITKARANTA, *The problem of selecting the shape functions for a p-type finite element*, IJNME, 28, (1989), pp. 1891–1908.