Understanding the Parallel Scalability of An Implicit Unstructured Mesh CFD Code

W. D. Gropp[1], D. K. Kaushik[1], D. E. Keyes[2], and B. F. Smith[1]

[1] Math. & Comp. Sci. Division, Argonne National Laboratory, Argonne, IL 60439,
{gropp,kaushik,bsmith}@mcs.anl.gov
[2] Math. & Stat. Department, Old Dominion University, Norfolk, VA 23529,
ISCR, Lawrence Livermore National Laboratory, Livermore, CA 94551,
and ICASE, NASA Langley Research Center, Hampton, VA 23681,
keyes@icase.edu

Abstract. In this paper, we identify the scalability bottlenecks of an unstructured grid CFD code (PETSc-FUN3D) by studying the impact of several algorithmic and architectural parameters and by examining different programming models. We discuss the basic performance characteristics of this PDE code with the help of simple performance models developed in our earlier work, presenting primarily experimental results. In addition to achieving good per-processor performance (which has been addressed in our cited work and without which scalability claims are suspect) we strive to improve the implementation and convergence scalability of PETSc-FUN3D on thousands of processors.

1 Introduction

We have ported the NASA code FUN3D [1, 3] into the PETSc [4] framework using the single program multiple data (SPMD) message-passing programming model, supplemented by multithreading at the physically shared memory level. FUN3D is a tetrahedral vertex-centered unstructured mesh code originally developed by W. K. Anderson of the NASA Langley Research Center for compressible and incompressible Euler and Navier-Stokes equations. FUN3D uses a control volume discretization with a variable-order Roe scheme for approximating the convective fluxes and a Galerkin discretization for the viscous terms. In reimplementing FUN3D in the PETSc framework, our effort has focused on achieving small time to convergence without compromising scalability, by means of appropriate algorithms and architecturally efficient data structures [2].

The solution algorithm employed in PETSc-FUN3D is pseudo-transient Newton-Krylov-Schwarz (ψNKS) [8] with block-incomplete factorization on each subdomain of the Schwarz preconditioner and with varying degrees of overlap. This code spends almost all of its time in two phases: flux computations (to evaluate conservation law residuals), where one aims to have such codes spent almost *all* their time, and sparse linear algebraic kernels, which are a fact of life in implicit methods. Altogether, four basic groups of tasks can be identified based on the criteria of arithmetic concurrency, communication patterns, and the ratio of operation complexity to data size within the task. These four distinct phases, present in most implicit codes, are: vertex-based loops, edge-based loops, recurrences, and global reductions. Each of these groups of tasks

stresses a different subsystem of contemporary high-performance computers. Analysis of PETSc-FUN3D shows that, after tuning, the linear algebraic kernels run at close to the aggregate memory-bandwidth limit on performance, the flux computations are bounded either by memory bandwidth or instruction scheduling (depending upon the ratio of load/store units to floating point units in the CPU), and parallel efficiency is bounded primarily by slight load imbalances at synchronization points [6, 7].

Achieving high sustained performance, in terms of solutions per second, requires attention to three factors. The first is a scalable implementation, in the sense that time per iteration is reduced in inverse proportion to the the number of processors, or that time per iteration is constant as problem size and processor number are scaled proportionally. The second is algorithmic scalability, in the sense that the number of iterations to convergence does not grow with increased numbers of processors. The second factor arises since the requirement of a scalable implementation generally forces parameterized changes in the algorithm as the number of processors grows. However, if the convergence is allowed to degrade the overall execution is not scalable, and this must be countered algorithmically. The third is good per-processor performance on contemporary cache-based microprocessors. In this paper, we only consider the first two factors in the overall performance in Sections 3 and 4, respectively. For a good discussion of the optimizations done to improve the per processor performance, we refer to our earlier work [7, 6].

2 Large Scale Demonstration Runs

We use PETSc's profiling and logging features to measure the parallel performance. PETSc logs many different types of events and provides valuable information about time spent, communications, load balance, and so forth, for each logged event. PETSc uses manual counting of flops, which are afterwards aggregated over all the processors for parallel performance statistics. In our rate computations, we exclude the initialization time devoted to I/O and data partitioning. Since we are solving large fixed-size problems on distributed memory machines, it is not reasonable to base parallel scalability on a uniprocessor run, which would thrash the paging system. Our base processor number is such that the problem has just fit into the local memory.

Fig. 1 shows aggregate flop/s performance and a log-log plot showing execution time for the 2.8 million vertex grid on the three most capable machines to which we have thus far had access. In both plots of this figure, the dashed lines indicate ideal behavior. Note that although the ASCI Red flop/s rate scales nearly linearly, a higher fraction of the work is redundant at higher parallel granularities, so the execution time does not drop in exact proportion to the increase in flop/s.

3 Implementation Scalability

Domain-decomposed parallelism for PDEs is a natural means of overcoming Amdahl's law in the limit of fixed problem size per processor. Computational work on each evaluation of the conservation residuals scales as the volume of the (equal-sized) subdomains, whereas communication overhead scales only as the surface. This ratio is fixed when

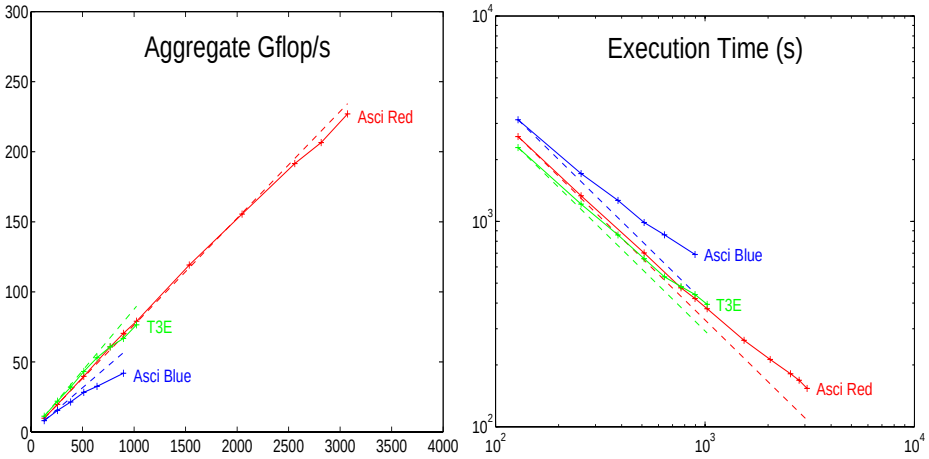

Fig. 1. Gigaflop/s ratings and execution times on ASCI Red (up to 3072 2-processor nodes), ASCI Pacific Blue (up to 768 processors), and a Cray T3E (up to 1024 processors) for a 2.8M-vertex case, along with dashed lines indicating "perfect" scalings.

problem size and processors are scaled in proportion, leaving only global reduction operations over all processors as an impediment to perfect performance scaling.

When the load is perfectly balanced (which is easy to achieve for static meshes) and local communication is not an issue because the network is scalable, the optimal number of processors is related to the network diameter. For logarithmic networks, like a hypercube, the optimal number of processors, P, grows directly in proportion to the problem size, N. For a d-dimensional torus network, $P \propto N^{d/d+1}$. The proportionality constant is a ratio of work per subdomain to the product of synchronization frequency and internode communication latency.

3.1 Scalability Bottlenecks

In Table 1, we present a closer look at the relative cost of computation for PETSc-FUN3D for a fixed-size problem of 2.8 million vertices on the ASCI Red machine, from 128 to 3072 nodes. The intent here is to identify the factors that retard the scalability. The overall parallel efficiency (denoted by $\eta_{overall}$) is broken into two components: η_{alg} measures the degradation in the parallel efficiency due to the increased iteration count (Section 4) of this (non-coarse-grid-enhanced) NKS algorithm as the number of subdomains increases, while η_{impl} measures the degradation coming from all other non-scalable factors such as global reductions, load imbalance (implicit synchronizations), and hardware limitations.

From Table 1, we observe that the buffer-to-buffer time for global reductions for these runs is relatively small and does not grow on this excellent network. The primary factors responsible for the increased overhead of communication are the implicit synchronizations and the ghost point updates (interprocessor data scatters).

Table 1. Scalability bottlenecks on ASCI Red for a fixed-size 2.8M vertex mesh. The preconditioner used in these results is block Jacobi with ILU(1) in each subdomain. We observe that the principle nonscaling factor is the implicit synchronization.

Number of Processors	Its	Time	Speedup	Efficiency $\eta_{overall}$	η_{alg}	η_{impl}
128	22	2,039s	1.00	1.00	1.00	1.00
256	24	1,144s	1.78	0.89	0.92	0.97
512	26	638s	3.20	0.80	0.85	0.94
1024	29	362s	5.63	0.70	0.76	0.93
2048	32	208s	9.78	0.61	0.69	0.89
3072	34	159s	12.81	0.53	0.65	0.82

Number of Processors	Percent Times for Global Reductions	Implicit Synchronizations	Ghost Point Scatters	Scatter Scalability Total Data Sent per Iteration (GB)	Application Level Effective Bandwidth per Node (MB/s)
128	5	4	3	3.6	6.9
256	3	6	4	5.0	7.5
512	3	7	5	7.1	6.0
1024	3	10	6	9.4	7.5
2048	3	11	8	11.7	5.7
3072	5	14	10	14.2	4.6

Interestingly, the increase in the percentage of time (3% to 10%) for the scatters results more from algorithmic issues than from hardware/software limitations. With an increase in the number of subdomains, the percentage of grid point data that must be communicated also rises. For example, the total amount of nearest neighbor data that must be communicated per iteration for 128 subdomains is 3.6 gigabytes, while for 3072 subdomains it is 14.2 gigabytes. Although more network wires are available when more processors are employed, scatter time increases. If problem size and processor count are scaled together, we would expect scatter times to occupy a fixed percentage of the total and load imbalance to be reduced at high granularity.

The final column in Table 1 shows the scalability of the "application level effective bandwidth" that is computed by dividing the total amount of data transferred by the time spent in scatter operation. It includes the message packing and unpacking times plus any contention in the communication. That is why it is far lower than the achievable bandwidth (as measured by the "Ping-Pong" test from the message passing performance (MPP) [10] tests) of the networking hardware. The Ping-Pong test measures the point to point unidirectional bandwidth between any two processors in a communicator group. It is clear that the Ping-Pong test results in Table 2 are not representative of the actual communication pattern encountered in the scatter operation. To better understand this issue, we have carried out the "Halo" test (from the MPP test suite) on 64 nodes

Table 2. MPP test results on 64 nodes of ASCI Red. The Ping-Pong results measure the unidirectional bandwidth. The Halo test results (measuring the bidirectional bandwidth) is more representative of the communication pattern encountered in the scatter operation.

Message	Bandwidth, MB/s	
Length, KB	Ping-Pong	Halo
2	93	70
4	145	94
8	183	92
16	235	106
32	274	114

of the ASCI Red machine. In this test, a processor exchanges messages with a fixed number of neighbors, moving data from/to contiguous memory buffers. For the Halo test results in Table 2, each node communicated with 8 other nodes (which is a good estimate of the neighbors a processor in PETSc-FUN3D will need to communicate). The message lengths for both these tests (Ping-Pong and Halo) have been varied between 2KB to 32 KB since the average length of a message in the runs for Table 1 varies from 23 KB to 3 KB as the number of processor goes up from 128 to 3072. We observe that the bandwidth obtained in the Halo test is significantly less than that obtained in the Ping-Pong test. This loss in performance perhaps can be attributed to the fact that a processor communicates with more than one neighbor at the same time in the Halo test. In addition, as stated earlier, the scatter operation involves the overhead of packing and unpacking of messages at the rate limited by the achievable memory bandwidth (about 145 MB/s as measured by the STREAM benchmark [14]). We are currently investigating the impact of these two factors (the number of pending communication operations with more than one neighbor and the memory bandwidth) on the performance of scatter operation further.

3.2 Effect of Partitioning Strategy

Mesh partitioning has a dominant effect on parallel scalability for problems characterized by (almost) constant work per point. As shown above, poor load balance causes idleness at synchronization points, which are frequent in implicit methods (e.g., at every conjugation step in a Krylov solver). With NKS methods, then, it is natural to strive for a very well balanced load. The p-MeTiS algorithm in the MeTiS package [12], for example, provides almost perfect balancing of the number of mesh points per processor. However, balancing work alone is not sufficient. Communication must be balanced as well, and these objectives are not entirely compatible. Figure 2 shows the effect of data partitioning using p-MeTiS, which tries to balance the number of nodes and edges on each partition, and k-MeTiS, which tries to reduce the number of noncontiguous subdomains and connectivity of the subdomains. Better overall scalability is observed with k-MeTiS, despite the better load balance for the p-MeTiS partitions. This is due to the slightly poorer numerical convergence rate of the iterative NKS algorithm with the p-MeTiS partitions. The poorer convergence rate can be explained by the fact that

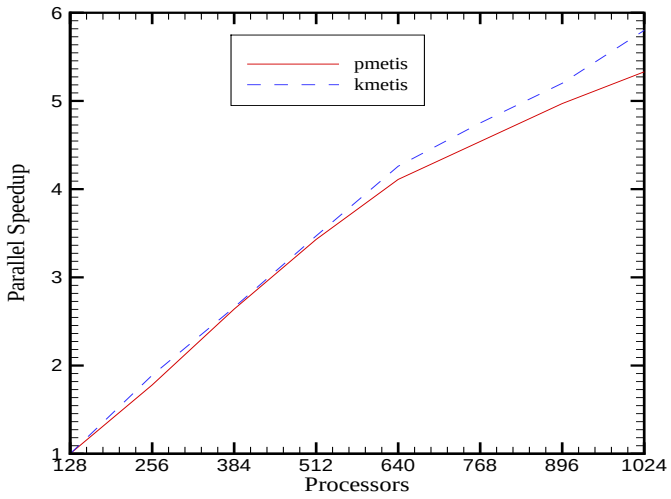

Fig. 2. Parallel speedup relative to 128 processors on a 600 MHz Cray T3E for a 2.8M vertex case, showing the effect of partitioning algorithms k-MeTiS, and p-MeTiS.

the p-MeTiS partitioner generates disconnected pieces within a single "subdomain," effectively increasing the number of blocks in the block Jacobi or additive Schwarz algorithm and increasing the size of the interface. The convergence rates for block iterative methods degrade with increasing number of blocks, as discussed in Section 4.

3.3 Domain-based and/or Instruction-level Parallelism

The performance results above are based on subdomain parallelism using the Message Passing Interface (MPI) [9]. With the availability of large scale SMP clusters, different software models for parallel programming require a fresh assessment. For machines with physically distributed memory, MPI has been a natural and successful software model. For machines with distributed shared memory and nonuniform memory access, both MPI and OpenMP have been used with respectable parallel scalability. For clusters with two or more SMPs on a single node, the mixed software model of threads within a node (OpenMP being a special case of threads because of the potential for highly efficient handling of the threads and memory by the compiler) and MPI between the nodes appears natural. Several researchers (e.g., [5, 13]) have used this mixed model with reasonable success.

We investigate the mixed model by employing OpenMP only in the flux calculation phase. This phase takes over 60% of the execution time on ASCI Red and is an ideal candidate for shared-memory parallelism because it does not suffer from the memory bandwidth bottleneck (see next section). In Table 3, we compare the performance of

Table 3. Execution time on the 333 MHz Pentium Pro ASCI Red machine for function evaluations only for a 2.8M vertex case, comparing the performance of the hybrid (MPI/OpenMP) and the distributed memory (MPI alone) programming models.

	MPI/OpenMP Threads per Node		MPI Processes per Node	
Nodes	1	2	1	2
256	483s	261s	456s	258s
2560	76s	39s	72s	45s
3072	66s	33s	62s	40s

this phase when the work is divided by using two OpenMP threads per node with the performance when the work is divided using two independent MPI processes per node. There is no communication in this phase. Both processors work with the same amount of memory available on a node; in the OpenMP case, it is shared between the two threads, while in the case of MPI it is divided into two address spaces.

The hybrid MPI/OpenMP programming model appears to be a more efficient way to employ shared memory than are the heavyweight subdomain-based processes (MPI alone), especially when the number of nodes is large. The MPI model works with larger number of subdomains (equal to the number of MPI processors), resulting in slower rate of convergence. The hybrid model works with fewer chunkier subdomains (equal to the number of nodes) that result in faster convergence rate and shorter execution time, despite the fact that there is some redundant work when the data from the two threads is combined due to the lack of a vector-reduce operation in the OpenMP standard (version 1) itself. Specifically, some redundant work arrays must be allocated that are not present in the MPI code. The subsequent gather operations (which tend to be memory bandwidth bound) can easily offset the advantages accruing from the low latency shared memory communication. One way to get around this problem is to use some coloring strategies to create the disjoint work sets, but this takes away the ease and simplicity of the parallelization step promised by the OpenMP model.

4 Convergence Scalability

The convergence rates and, therefore, the overall parallel efficiencies of additive Schwarz methods are often dependent on subdomain granularity. Except when effective coarse-grid operators and intergrid transfer operators are known, so that optimal multilevel preconditioners can be constructed, the number of iterations to convergence tends to increase with granularity for elliptically controlled problems, for either fixed or memory-scaled problem sizes. In practical large-scale applications, however, the convergence rate degradation of single-level additive Schwarz is sometimes not as serious as the scalar, linear elliptic theory would suggest (e.g. see the 2nd column in Table 1). Its effects are mitigated by several factors, including pseudo-transient nonlinear continuation and dominant intercomponent coupling. The former parabolizes the operator, endow-

ing diagonal dominance. The latter renders the off-diagonal coupling less critical and, therefore, less painful to sever by domain decomposition. The block diagonal coupling can be captured fully in a point-block ILU preconditioner.

4.1 Additive Schwarz Preconditioner

Table 4 explores two quality parameters for the additive Schwarz preconditioner: subdomain overlap and quality of the subdomain solve using incomplete factorization. We exhibit execution time and iteration count data from runs of PETSc-FUN3D on the ASCI Red machine for a fixed-size problem with 357,900 grid points and 1,789,500 degrees of freedom. These calculations were performed using GMRES(20), one subdomain per processor (without overlap for block Jacobi and with overlap for ASM), and ILU(k) where k varies from 0 to 2, and with the natural ordering in each subdomain block. The use of ILU(0) with natural ordering on the first-order Jacobian, while applying a second-order operator, allows the factorization to be done in place, with or without overlap. However, the overlap case does require forming an additional data structure on each processor to store matrix elements corresponding to the overlapped regions.

From Table 4 we see that the larger overlap and more fill helps in reducing the total number of linear iterations as the number of processors increases, as theory and intuition predict. However, both increases consume more memory, and both result in more work per iteration, ultimately driving up execution times in spite of faster convergence. Best execution times are obtained for any given number of processors for ILU(1), as the number of processors becomes large (subdomain size small), for zero overlap.

The execution times reported in Table 4 are highly dependent on the machine used, since each of the additional computation/communication costs listed above may shift the computation past a knee in the performance curve for memory bandwidth, communication network, and so on.

4.2 Other Algorithmic Tuning Parameters

In [7] we highlight some additional tunings (for ψNKS Solver) that have yielded good results in our context. Some subsets of these parameters are not orthogonal, but interact strongly with each other. In addition, optimal values of some of these parameters depend on the grid resolution. We are currently using derivative-free asynchronous parallel direct search algorithms [11] to more systematically explore this large parameter space.

5 Conclusions

Unstructured implicit CFD solvers are amenable to scalable implementation, but careful tuning is needed to obtain the best product of per-processor efficiency and parallel efficiency. In fact, the cache blocking techniques (addressed in our earlier work) employed to boost the per-processor performance helps to improve the parallel scalability as well by making the message sizes longer. The principle nonscaling factor is the implicit synchronizations and not the communication. Another important phase is the

Table 4. Execution times and linear iteration counts on the 333 MHz Pentium Pro ASCI Red machine for a 357,900-vertex case, showing the effect of subdomain overlap and incomplete factorization fill level in the additive Schwarz preconditioner. **The best execution times for each ILU fill level and number of processors are in boldface in each row.**

ILU(0) in Each Subdomain						
Number of Processors	Overlap					
	0		1		2	
	Time	Linear Its	Time	Linear Its	Time	Linear Its
32	688s	930	**661s**	816	696s	813
64	**371s**	993	374s	876	418s	887
128	**210s**	1052	230s	988	222s	872
ILU(1) in Each Subdomain						
Number of Processors	Overlap					
	0		1		2	
	Time	Linear Its	Time	Linear Its	Time	Linear Its
32	598s	674	**564s**	549	617s	532
64	**334s**	746	335s	617	359s	551
128	**177s**	807	178s	630	200s	555
ILU(2) in Each Subdomain						
Number of Processors	Overlap					
	0		1		2	
	Time	Linear Its	Time	Linear Its	Time	Linear Its
32	**688s**	527	786s	441	—	—
64	**386s**	608	441s	488	531s	448
128	**193s**	631	272s	540	313s	472

scatter/gather operation that seems limited more by the achievable memory bandwidth than the network bandwidth (at least on ASCI Red). Given contemporary high-end architecture, critical research directions for solution algorithms for systems modeled by PDEs are (1) multivector algorithms and less synchronous algorithms, and (2) hybrid programming models. To influence future architectures while adapting to current ones, we recommend adoption of new benchmarks featuring implicit methods on unstructured grids, such as the application featured here.

Acknowledgments

We are indebted to Lois C. McInnes and Satish Balay of Argonne National Laboratory, to W. Kyle Anderson, formerly of the NASA Langley Research Center, for collaborations leading up to the work presented here. Gropp and Smith were supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Kaushik's support was provided by a GAANN Fellowship from the U.S. Department of Education and by Argonne National Laboratory under contract 983572401. Keyes was supported by the National Science Foundation under grant

ECS-9527169, by NASA under contracts NAS1-19480 and NAS1-97046, by Argonne National Laboratory under contract 982232402, and by Lawrence Livermore National Laboratory under subcontract B347882. Debbie Swider of Argonne National Laboratory was of considerable assistance in performing ASCI platform runs. Computer time was supplied by Argonne National Laboratory, Lawrence Livermore National Laboratory, NERSC, Sandia National Laboratories, and SGI-Cray.

References

1. W. K. Anderson and D. L. Bonhaus. An implicit upwind algorithm for computing turbulent flows on unstructured grids. *Computers and Fluids*, 23:1–21, 1994.
2. W. K. Anderson, W. D. Gropp, D. K. Kaushik D. E. Keyes, and B. F. Smith. Achieving high sustained performance in an unstructured mesh CFD application. In *Proceedings of SC'99*. IEEE Computer Society, 1999. Gordon Bell Prize Award Paper in Special Category.
3. W. K. Anderson, R. D. Rausch, and D. L. Bonhaus. Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids. *J. Computational Physics*, 128:391–408, 1996.
4. S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. The Portable Extensible Toolkit for Scientific Computing (PETSc) version 28. `http://www.mcs.anl.gov/petsc/petsc.html`, 2000.
5. S. W. Bova, C. P. Breshears, C. E. Cuicchi, Z. Demirbilek, and H. A. Gabb. Dual-level parallel analysis of harbor wave response using MPI and OpenMP. *Int. J. High Performance Computing Applications*, 14:49–64, 2000.
6. W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. Toward realistic performance bounds for implicit CFD codes. In D. Keyes, A. Ecer, J. Periaux, N. Satofuka, and P. Fox, editors, *Proceedings of Parallel CFD'99*, pages 233–240. Elsevier, 1999.
7. W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. Performance modeling and tuning of an unstructured mesh CFD application. In *Proceedings of SC2000*. IEEE Computer Society, 2000.
8. W. D. Gropp, L. C. McInnes, M. D. Tidriri, and D. E. Keyes. Globalized Newton-Krylov-Schwarz algorithms and software for parallel implicit CFD. *Int. J. High Performance Computing Applications*, 14:102–136, 2000.
9. William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd edition. MIT Press, Cambridge, MA, 1999.
10. William D. Gropp and Ewing Lusk. Reproducible measurements of MPI performance characteristics. In Jack Dongarra, Emilio Luque, and Tomàs Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1697 of *Lecture Notes in Computer Science*, pages 11–18. Springer Verlag, 1999. 6th European PVM/MPI Users' Group Meeting, Barcelona, Spain, September 1999.
11. P. D. Hough, T. G. Kolda, and V. J. Torczon. Asynchronous parallel pattern search for nonlinear optimization. Technical Report SAND2000-8213, Sandia National Laboratories, Livermore, January 2000. Submitted to SIAM J. Scientific Computation.
12. G. Karypis and V. Kumar. A fast and high quality scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20:359–392, 1999.
13. D. J. Mavriplis. Parallel unstructured mesh analysis of high-lift configurations. Technical Report 2000-0923, AIAA, 2000.
14. J. D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, 1995. `http://www.cs.virginia.edu/stream`.