How Scalable is Domain Decomposition in Practice?

D. E. Keyes*

1 Introduction

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. The communication overhead per iteration also grows with granularity when global operations, such as inner products, are required as part of the acceleration scheme. The growth is sublinear on reasonable networks, but not *a priori* dismissable as processor counts enter the thousands.

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. Its effects are mitigated by several factors, including pseudo-transient nonlinear continuation and dominant intercomponent coupling that can be captured fully in a point-block ILU preconditioner. Furthermore, complex models may be so dominated by fully concurrent work at each gridpoint that global communication constitutes a small, albeit growing, tax. We illustrate these favorable claims for the practicality of domain decomposition with encouraging scalabilities for a legacy unstructured-grid Euler flow application code, parallelized with the pseudo-transient Newton-Krylov-Schwarz (ΨNKS) algorithm using the freely available PETSc library on contemporary high-end platforms. We note some impacts on performance of the horizontal (distributed) and vertical (hierarchical) aspects of the memory system and consider architecturally motivated algorithmic variations for their amelioration.

*Computer Science Department, Old Dominion University, Norfolk, VA 23529-0162 and ICASE, NASA Langley Res. Ctr., Hampton, VA 23681-2199, keyes@icase.edu. Supported in part by the National Science Foundation under grant ECS-9527169, by NASA under contracts NAS1-19480 and NAS1-97046, and by Argonne National Laboratory under contract 982232402.

2 Scalability

Domain decomposition has been continually reinvented for several reasons: it is natural for task-based divide-and-conquer algorithms on problems with multi-physics and/or multi-discretizations, it provides a framework for combination and reuse of legacy software, and it organizes "out of core"-style access to massive amounts of memory. However, the main motivation for the focus on domain decomposition methods in the past two decades is scalable performance through data parallelism. Few other algorithms for PDEs possess the same combination of *arbitrary concurrency degree* with *modest convergence degradation*.

2.1 Definitions of scalability

Several types of scalability are formally defined in the parallel computing literature, and numerous other operational definitions of scalability exist in different scientific subcommunities, making semantic digression a necessity before any consideration of the scalability of domain decomposition methods. Given a performance metric that is a function of parameters $\{\pi_1, \pi_2, \ldots\}$, we study the variation of the metric as the π_i are varied, independently or on some constrained manifold. If the metric exhibits acceptable behavior over some region of parameter space, performance is said to "scale" over that region. Scalability of interesting numerical algorithms is rarely uniform over the entire parameter space, so claims of scalability should ordinarily be accompanied by a statement of applicable domain.

Consider parameters N, the discrete problem dimension, and P, the number of processors employed, and performance metric η, the parallel efficiency. The variation of $\eta(N,P)$ as P varies with fixed N is the "fixed-problem-size" scaling. The variation of $\eta(N,P)$ as N and P vary with N/P fixed is the "fixed-storage-per-processor" scaling. If computational work is not linear in N, then a fixed-work-per-processor scaling may be useful to consider, in addition. The expression of scalability may be inverted, e.g., the isoefficiency function, $N(P)$, of an algorithm may be sought such that $\eta(N(P),P) = \text{constant}$, as P varies.

Fixed-storage-per-processor is arguably the most interesting limit from an architectural point of view, for three reasons. First, the proportion of memory to processors is typically fixed (at purchase). Second, work and communication are designed to scale as different powers of N/P in domain decomposition methods. If N/P varies, the ratio of communication to work varies, becoming more parasitic as N/P becomes smaller. Third, the performance of a single processor-memory unit may vary considerably with local problem size, due to cache effects. There are often thresholds of workingset size, across which performance jumps (see, e.g., [5]). Careful attention to data locality smooths out these thresholds, providing a range of problem size over which performance is nearly level, but if such thresholds are not controlled for, they may obscure parallel performance evaluation. Keeping N/P constant avoids the possibility of threshold effects. Despite the aesthetic superiority of fixed-storage-per-processor scaling, fixed-size scaling is often performed in practice because, for instance, grid generation

is more responsive to discretization demands than parallelization opportunities. Since fixed-size scalability is more difficult to achieve (because of the second point above), we measure it, rather than fixed-storage-per-processor scalability, in all but one of the examples in Section 4.

Absolute efficiency on P processors is defined as the speedup divided by P,

$$\eta(N,P) = \frac{1}{P} \cdot \frac{T(N,1)}{T(N,P)}, \tag{1}$$

where $T(N,P)$ is the execution time. The execution time on one processor of a fixed-size problem large enough to be worthy of the combined memories of $P \gg 1$ is often inflated by memory system thrashing, leading to superunitary efficiencies. Over smaller ranges of variation of processor number, it is useful to define the relative efficiency, in going from Q to P processors ($P > Q$),

$$\eta(N,P|Q) = \frac{Q \cdot T(N,Q)}{P \cdot T(N,P)}. \tag{2}$$

In iterative methods, time T can be factored into a total number of iterations, I, and an average cost per iteration, C. Domain decomposition methods are typically implemented in an SPMD style that assigns one subdomain per processor, which has the effect of changing the algorithm, mathematically, when P changes, by inducing finer diagonal blocks in the underlying global preconditioner. Thus, domain decomposition has an algorithmic efficiency:

$$\eta_{alg}(N,P|Q) = \frac{I(N,Q)}{I(N,P)}. \tag{3}$$

The remaining factor in the overall efficiency, is the implementation efficiency:

$$\eta_{impl}(N,P|Q) \equiv \eta/\eta_{alg} = \frac{Q \cdot C(N,Q)}{P \cdot C(N,P)}. \tag{4}$$

Whereas T and I (and hence η and η_{alg}) are measured, C (and hence η_{impl}) is inferred.

2.2 Problems with scalability metrics

Viewing the parallel performance of domain decomposition (or any other) methods for PDEs through such a simple metric as efficiency is fraught with pitfalls. As is well known [1], efficiency can be improved while absolute performance deteriorates at all granularities by employing methods that perform wasteful local arithmetic, and thus drive up the fraction of the overall work that is concurrent. Hence, efficiency should be measured only after tuning subdomain solver arithmetic complexity to a practical minimum. Fixed-storage-per-processor metrics require varying the discrete problem size, N, along with P. However, increasing the resolution of the problem often degrades linear conditioning and increases nonlinear stiffness, e.g., by giving better definition to near-singularities, increasing the difficulty of the discrete problem. From the point of view of solving

the continuous problem, this is highly desirable, and constitutes a main motivation for doing parallel computing in the first place. However, it means that a simple metric like fixed-storage-per-processor scalability is meaningful only for rare problems with self-similarity in the fine scales. Finally, as mentioned above, fixed-problem-size efficiencies can be greater than 100% for hierarchical memory computers when, as P grows, workingsets suddenly drop into cache. They should be measured only over a range of N/P where single-processor performance is fairly flat.

3 Theoretical scalability of DD

In this section, we build an extremely simple model of the scalability of domain decomposition methods, while illustrating a methodology that is extensible to more comprehensive models. Reflecting the factorization of efficiency into algorithmic and implementation terms, we separate convergence and implementation issues. We restrict attention to a scalar, linear, elliptically dominated PDE.

3.1 Convergence of Schwarz methods

For a general exposition of Schwarz methods for linear problems, see [12]. Assume a d-dimensional isotropic problem. Consider a unit aspect ratio domain with quasi-uniform mesh parameter h and quasi-uniform subdomain diameter H. Then $N = h^{-d}$ and, under the one subdomain per processor assumption, $P = H^{-d}$. Consider four preconditioners: point Jacobi, subdomain Jacobi, 1-level additive Schwarz (subdomain Jacobi with overlapped subdomains), and 2-level additive Schwarz (overlapped subdomains with a global coarse problem with approximately one degree of freedom per subdomain). The first two can be thought of as degenerate Schwarz methods (with zero overlap, and possibly point-sized subdomains). Consider acceleration of the Schwarz method by a Krylov method such as conjugate gradients or one of its many generalizations to nonsymmetric problems, e.g., GMRES. Krylov-Schwarz iterative methods typically converge in a number of iterations that scales as the square-root of the condition number of the Schwarz-preconditioned system. Table 1 lists the expected number of iterations to achieve a given reduction ratio in the residual norm. (Here we gloss over unresolved issues in 2-norm and operator-norm convergence definitions, but see [4].) The first line of this table pertains to diagonally-scaled CG, a very common default parallel implicit method, but one which is *not* very algorithmically scalable, since increasing the problem resolution N degrades the iteration count. The results in this table were first derived for symmetric definite operators with exact solves on each subdomain, but they have been extended to operators with nonsymmetric and indefinite components and inexact solves on each subdomain. The intuition behind this table is the following. Errors propagate from the interior to the boundary in steps that are proportional to the largest implicit aggregate in the preconditioner, pointwise or subdomainwise. The use of overlap avoids the introduction of high-energy

Table 1: Iteration count scaling of Schwarz-preconditioned Krylov methods

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

field discontinuities at subdomain boundaries. The 2-level method projects out low-wavenumber errors at the price of solving a global problem.

Only the 2-level method scales perfectly in convergence rate (constant, independent of N and P), like a traditional multilevel iterative method. However, the 2-level method shares with multilevel methods a nonscalable cost-per-iteration from the necessity of solving a coarse-grid system of size $\mathcal{O}(P)$.

3.2 Per-iteration parallel complexity of Schwarz methods

Krylov-Schwarz methods fit perfectly into the bulk synchronous model of parallel complexity, consisting of balanced concurrent computations on each subdomain, with costs that may be taken as proportional to the volume, N/P. Assume a distributed-memory architecture with a toroidal mesh interconnect and a global reduction time of $CP^{1/d}$. Assume no coarse-grid solve. For simplicity, we neglect the cost of neighbor-only communication relative to arithmetic and global reductions.

The first line of Table 2 shows the estimated execution time per iteration in the left column and the overall execution time (factoring in the number of iterations for 1-level additive Schwarz from Table 1) in the right column. All of the work terms (matrix-vector multiplies, subdomain preconditioner sweeps or incomplete factorizations, DAXPYs, and local summations of inner product computations) are contained in A, and, since it is given in units of time, A also reflects per-processor floating-point performance, including local memory system effects. C includes the cost of synchronization, including algorithmic synchronization frequency relative to the work term and the actual cost of each synchronization. If the network architecture changes, the functional form in P of the synchronization term changes. A fuller model would contain a term of the form $B(N/P)^{2/3}$.

The second line of Table 2 shows the optimal number of processors to employ on a problem of size N, based on the parallel complexity in the first line. The work term falls in P and the communication term rises; setting the P-derivative of their sum to zero yields the P that minimizes overall execution time. The marginal value of the $P_{opt}{}^{\text{th}}$ processor may be small, but $\mathcal{O}(N^{d/d+1})$ processors may be employed without encountering the phenomenon of "speeddown."

Isoefficiency scaling, under these algorithmic and architectural assumptions, is $P \propto N^{\frac{d}{d+1}}$, or $N \propto P^{\frac{d+1}{d}}$. Observe that this requires more memory per

Table 2: Execution time scaling of Schwarz-preconditioned Krylov methods

	Time per iter.	Time overall
$T(N,P)$	$A\left(\frac{N}{P}\right)+CP^{1/d}$	$[A\left(\frac{N}{P}\right)+CP^{1/d}]\cdot P^{1/d}$
P_{opt}	$(dA/C)^{\frac{d}{d+1}}\cdot N^{\frac{d}{d+1}}$	$((d-1)A/C)^{\frac{d}{d+1}}\cdot N^{\frac{d}{d+1}}$
$T(N,P_{opt})\vert_{d=2}$	$\propto A^{1/3}C^{2/3}\cdot N^{1/3}$	$\propto A^{2/3}C^{1/3}\cdot N^{2/3}$
$T(N,P_{opt})\vert_{d=3}$	$\propto A^{1/4}C^{3/4}\cdot N^{1/4}$	$\propto A^{1/2}C^{1/2}\cdot N^{1/2}$

processor as processors increase. If, on the other hand, as is architecturally customary, $N \propto P$ and we solve the largest possible problem as we scale, then in 3D, as C dominates,

$$\eta_{impl}(P|1) = \frac{AN+C}{AN+CP^{1+(1/d)}} \sim P^{-1/d}, \tag{5}$$

$$\eta_{overall}(P|1) = \frac{AN+C}{[AN+CP^{1+(1/d)}]\cdot P^{1/d}} \sim P^{-2/d}. \tag{6}$$

Execution time increases in inverse proportion to efficiency in the fixed-storage-per-processor scaling.

4 Practical scalability of DD

In practical applications of domain decomposition to large-scale nonlinear problems, the preceding linear scaling estimates for elliptically dominated systems are baselines, at best. Nonlinear problems are approached through a sequence of linear problems, each with its own Jacobian matrix. Nonlinear problems typically employ parameter continuation, such as pseudo-transience, which adds diagonal dominance to these Jacobians, making the linear convergence estimates above *pessimistic*. Nonlinear problems often become "stiffer" as they are better resolved, dragging out the continuation process, requiring more linear systems to be solved, and making the overall estimates above *optimistic* for the overall scaling, inasmuch as they were predicated on a single linear system. Because of these complications that lie outside of any convenient theory, the value of domain decomposition in practical problems is ultimately problem-specific.

4.1 Background on a legacy CFD code

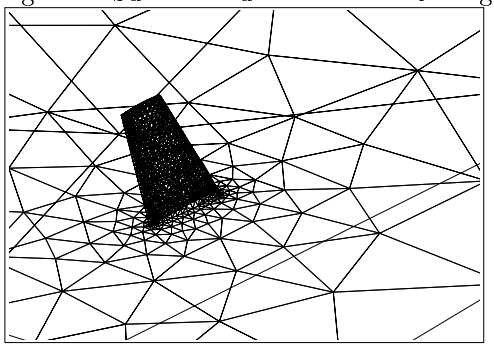
FUN3D is a tetrahedral vertex-centered unstructured grid code developed by W. K. Anderson (LaRC) for compressible and incompressible Euler and Navier-Stokes equations and described elsewhere in these proceedings [8]. It employs pseudo-timestepping for nonlinear continuation towards steady state. One of the solvers in the legacy vector-oriented code is Newton-Krylov with global point-block-ILU preconditioning, which is competitive with FAS multigrid in 2D sequential contexts. When rewritten in the ΨNKS framework (see, e.g., [10] for the continuation theory, [6] for software engineering aspects, [9] for architectural

aspects, and [3] for an aerodynamic application of the 2-level method) using PETSc [2], its flow of control can be represented by the following four-deep nest of loops, the innermost of which is a concurrent `doall`:

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

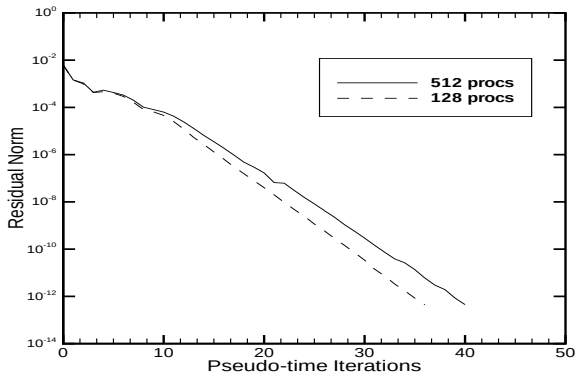
The code is applied to the M6 wing, a coarse surface triangulation of which appears in Fig. 1. Sample convergence histories for a fixed-size incompressible

Figure 1: Surface visualization of M6 wing



case of approximately 2.8 million vertices (approximately 11 million degrees of freedom) on 128 and on 512 processors of a T3E-900 are shown in Fig. 2. To translate the history of the 2-norm of the nonlinear steady-state residual back to the domain decomposition theory, it is necessary to note that the linear Newton correction GMRES iterations are truncated to a fixed (timestep-independent) Krylov dimension. Thus, a weaker preconditioner leads to a poorer Newton correction for a given amount of work, which ultimately increases the number of

Figure 2: M6 wing convergence history on T3E



pseudo-timesteps to convergence. Contrary to the $P^{1/3}$ scalar elliptic theoretical scaling a four-fold increase in P leads to only a 10% increase in overall iterations. (The results in Fig. 2 represent a significant improvement over earlier results on the same problem in [9]. More aggressive CFL advancement and less linear work per Newton correction make the difference.)

A fuller set of performance profiles for this problem on 128, 256, and 512 processors of a T3E appears in Fig. 3.

Though our limited catalog of M6 grids does not permit a perfect fixed-storage-per-processor scaling, Table 3 comes close. In the final column may be seen a good demonstration of scalable implementation efficiency, and in the first two columns to the right of the double bar may be seen a good illustration of the non-self-similarity of nonlinear problems.

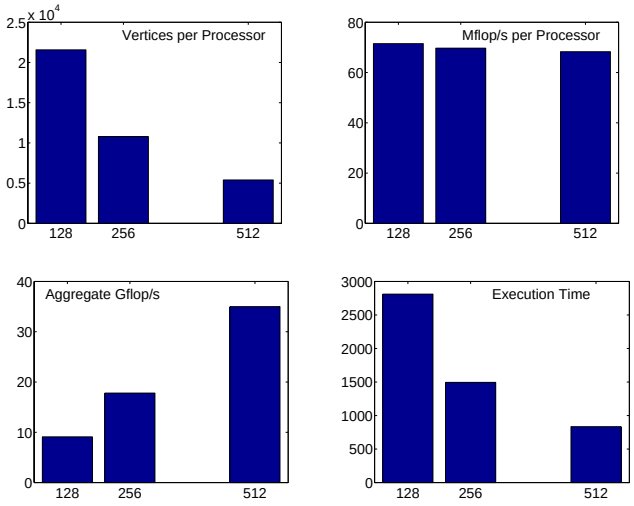
Table 3: Parallel scalability of M6 Euler flow with fixed storage per processor

Vertices, N	P	N/P	Steps	Time	Time/Step
357,900	80	4,474	78	559.93s	7.18s
53,961	12	4,497	36	265.72s	7.38s
9,428	2	4,714	19	131.07s	6.89s

5 Algorithmic issues in improving scalability

Though the results of Section 4 (and in the companion paper [8]) are encouraging, there is reason to hope for significant improvements in computational performance on problems of contemporary scale (10^6 vertices on 10^2 proces-

Figure 3: (upper left) Vertices per processor, inverse in P; (upper right) Mflop/s per processor, nearly flat over this range; (lower left) Aggregate Gflop/s, linear in P; (lower right) Overall execution time, nearly inverse in P



sors) and there is reason to strive earnestly for further improvements at the next scale, typical of the ASCI vision, namely 10^8 vertices on 10^4 processors. Improvements will have to come largely from more complex algorithms since systems software is advancing very slowly at the high-performance end, and most architectural advances drive the percentage of peak available to PDE applications lower, while raising the "theoretical" peak performance. We briefly describe three such improvements algorithmic under investigation; for additional detail, see [11].

5.1 Improvement in algorithmic efficiency

A region of strong nonlinearity embedded in a region of nearly linear behavior is characteristic of many important PDE problems (e.g., noise, flame, and crack propagation). Such problems may be said to be "nonlinearly stiff"; progress of Newton's method is held hostage to a low-dimensional feature and the expensive-to-obtain Newton correction is small over most of the domain. For an illustration involving shocked transonic flow, see [3].

For such problems, we propose cyclicly permuting Newton-Krylov-Schwarz methodology into Schwarz-Newton-Krylov. Also known as "nonlinear Schwarz," its two-domain description is as follows:

> Given: $u \in \mathcal{R}^n$, coming from a discretization on Ω, and $f : \mathcal{R}^n \rightarrow \mathcal{R}^n$, with $f(u) = 0$ a governing system (including implicitly posed BCs) on Ω. Given: an overlapping partition of Ω into Ω_1 and Ω_2,

with induced partionings on u into u_1 and u_2 and f into f_1 and f_2. Given: initial iterates $u_1^{(0)}$ and $u_2^{(0)}$. Iterate $k = 0, 1, \ldots$, until convergence:

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

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

This is the multiplicative version; the additive companion has $u_1^{(k)}$ instead of $u_1^{(k+1)}$ in the second equation. Each subdomain solve (partitioned based on isolation of strongly and weakly nonlinear regions) can be further partitioned for parallel solution by NKS. The overlap region can coincide with an entire inner subdomain. There is a nonoverlapping analog known as Schur-Newton-Krylov, in which the iteration is reduced to a lower-dimensional interface. Automatic detection of the small regions responsible for the nonlinear stiffness may be possible through an indicator such as tensoricity [11].

5.2 Improvement in parallel implementation efficiency

Profiling the runs of Fig. 3, we observe that the efficiency of the 512-processor case relative to the 128-processor case is 84%, and the percentage of execution time devoted to global inner products increases to 9%, from 6% in the 128-processor case. The growing percentage of execution time consumed by the global reduction step of inner products is the dominant effect on overall efficiency. Independent measurements with barriers indicate that only about 1% of the 9% penalty is attributable to "time on the wire"; the rest is slight load imbalance in subdomain surface work, amplified by a granularity of 512. Extrapolation to 10^4 processors without amelioration of this problem is unacceptable.

We look, in part, to multiobjective load balancing [7], and, in part, to multithreading for synchronization relief. The typical critical path in a nonlinear implicit method is `..., solve, bound_step, update, solve, bound_step, update, solve, ...` There are other useful operations to perform, off the critical path, such as refreshing the Jacobian, testing convergence, adapting continuation and algorithmic parameters, performing interprocessor communication and disk I/O, visualization, filtering, compression, feature extraction, etc. Some of these "off the path" tasks share considerable data locality with critical path tasks, and can be performed during otherwise wasted cycles when a single-threaded code is data-starved. Thus, the scalability of "naked" sparse linear problems with limited arithmetic work between synchronizations, may be unrealistically pessimistic, relative to the richer "marketbasket" of work and synchronization in full simulations.

Other ways of reducing inefficiency attributable to synchronization include: reducing the penalty of each synchronization step by load balancing surface work phase simultaneously with dominant volume work phase; reducing the frequency of synchronizations by employing more speculative control using fewer norms, and/or less stable numerics in projection steps using fewer inner products; and

reducing the globality of most synchronization steps by means of the nonlinear Schwarz algorithm of the previous subsection.

5.3 Improvement in uniprocessor implementation efficiency

As alluded to above, the high-end superscalar pipelined RISC processors of contemporary parallel machines are typically utilized at a small fraction (e.g., 10%) of their theoretical peak performance by sparse PDE algorithms. Blocking of all of the data defined at each gridpoint improves considerably over other storage schemes, in part by reducing the memory traffic of integers describing the sparsity and in part by permitting unrolling of inner loops over components. This effect is illustrated in [6]. Additional illustrations of major improvements (ranging from a factor of 2.5 on the Intel Pentium II to a factor of 7.5 on the IBM Power2 SuperChip) in memory locality are presented in [8]. Beyond the intuitive data structure transformations described therein to enhance memory locality, we look to semi-automated data structure reorderings, by applying genetic optimization techniques to the problem of memory layout. Such optimization techniques require an objective function, which should be faster to evaluate than executing a large-scale program and measuring its execution time. The concept of "memtropy" is proposed in [11] as a surrogate objective, and a domain decomposition-like blocking for a sequential relaxation algorithm on a structure grid is shown to be an improvement on memtropy, relative to natural ordering. As mentioned in Section 2, improving local memory locality renders parallel computations less sensitive to workingset threshold effects, and thus makes fixed-problem-size scaling more meaningful.

6 Summary

Schwarz-style domain decomposition is not a truly scalable algorithm in the theoretical sense, in that its parallel efficiency degrades due to both algorithmic and implementation effects as $P \to \infty$ for elliptic problems. However, in practical large-scale nonlinear applications, Schwarz methods are often much better behaved than the theory for model problems predicts. Besides concurrency features, the memory locality of domain decomposition methods makes them highly suitable for advanced architectures, and many avenues remain open for conquering practical difficulties arising from complex applications and architecture, with correspondingly complex algorithmic adaptations.

It is difficult to define scalability metrics that are both practically useful and "fair" for nonlinear PDE-based problems, whether the candidate algorithm is domain decomposition, or anything else. Therefore, a multicriteria evaluation of any parallel PDE code is recommended, including fixed-size and fixed-storage-per-processor scaling, sustained per-processor computational rate as a percentage of peak, and optimality of tuning of the single subdomain code.

Acknowledgements

The author thanks W. Kyle Anderson of the NASA Langley Research Center for providing FUN3D, Dinesh Kaushik of Old Dominion University for his work in porting FUN3D to the parallel environment, and Satish Balay, Bill Gropp, Lois McInnes, and Barry Smith of Argonne National Laboratory who developed the PETSc software employed in obtaining the results of this paper. Computer time was supplied by DOE (through Argonne and NERSC).

References

[1] D. Bailey. How to fool the masses when reporting results on parallel computers. *Supercomputing Review*, pages 54–55, August 1991.

[2] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. The Portable, Extensible Toolkit for Scientific Computing, version 2.0.22. `http://www.mcs.anl.gov/petsc`, 1998.

[3] X.-C. Cai, W. D. Gropp, D. E. Keyes, R. G. Melvin, and D. P. Young. Parallel Newton-Krylov-Schwarz algorithms for the transonic full potential equation. *SIAM J. Sci. Comput.*, 19:246–265, 1998.

[4] X.-C. Cai and J. Zou. Some observations on the l^2 convergence of the additive Schwarz preconditioned GMRES method. Tech Report CU-CS-865-98, Univ. of Colorado at Boulder, 1998.

[5] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan-Kaufmann, 1998.

[6] W. D. Gropp, D. E. Keyes, L. C. McInnes, and M. D. Tidriri. Globalized Newton-Krylov-Schwarz algorithms and software for parallel implicit CFD. ICASE TR 98-24, 1998.

[7] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Univ. of Minn. CS Dept. TR 98-019, 1998.

[8] D. K. Kaushik, D. E. Keyes, and B. F. Smith. Newton-Krylov-Schwarz methods for aerodynamics problems: Compressible and incompressible flows on unstructured grids. Submitted to the Proceedings of the 11th Int. Conf. on Domain Decomposition Methods, 1998.

[9] D. K. Kaushik, D. E. Keyes, and B. F. Smith. On the interaction of architecture and algorithm in the domain-based parallelization of an unstructured grid incompressible flow code. In *Proceedings of the Tenth Int. Conf. on Domain Decomposition Methods*, pages 311–319. AMS, 1998.

[10] C. T. Kelley and D. E. Keyes. Convergence analysis of pseudo-transient continuation. *SIAM J. Num. Anal.*, 35:508–523, 1998.

[11] D. E. Keyes. Trends in algorithms for nonuniform applications on hierarchical distributed architectures. To appear in the Proceedings of a Workshop on Computational Aerosciences in the 21st Century (Salas, ed.), Kluwer, 1999.

[12] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Algorithms for Elliptic Partial Differential Equations*. Cambridge Univ. Press, 1996.