

A PERSPECTIVE ON DATA-PARALLEL IMPLICIT SOLVERS FOR MECHANICS

David E. Keyes

Computer Science Department, Old Dominion University
Norfolk, VA 23529-0162
and
ICASE, NASA Langley Research Center
Hampton, VA 23681-0001

With the widespread acceptance of distributed memory multiprocessing as a cost-effective means of solving very highly resolved problems in computational fluid and structural mechanics, many engineers and scientists are encouraged with their initial ports of CFD or CSM codes for parallel execution, and are interested in learning whether applied mathematicians and computer scientists have anything to offer with the next step. This note is both a cautious affirmative and an attempt to define important roles that remain to be played by those familiar with the applications, themselves. We provide a high-level summary of the state of the art in domain decomposition methods, and a selective bibliography.

DATA AND TASK PARALLELISM

Our scope is limited here to data parallelism, that is, to the single program/multiple data (SPMD) approach in which each processor is responsible for applying essentially *all* of the operations to *part* of the data. This paradigm can be applied at almost any granularity, from a small collection of workstations on a network to tightly coupled machines costing millions of dollars and comprising hundreds of processors on a custom interconnect. Much can be said at an abstract level about the data-parallel solution of PDE-based problems. Though much of the future may lie under the complementary paradigm of task parallelism, in which the processors are programmed heterogeneously to apply *part* of the operations to *all* or *part* of the data, coarse-grained task parallelism is usually by nature problem-specific, so we flag it here only in passing. It is clearly of importance in multi-physics applications in which different parts of the domain are governed by different models (e.g., aeroelasticity, aeroacoustics, Navier-Stokes/Boltzmann) or in which multiple phenomena addressed with very different algorithms are supported over a common domain. We likewise pass over the fine-scale task parallelism that can be exploited automatically by compilers for processors with multiple functional units. Data parallelism can be employed, and therefore discussed, in a way that is nearly orthogonal to both of these outer and inner forms of task parallelism.

IMPLICIT VS. EXPLICIT METHODS

It is commonly understood that, apart from problems of partitioning the domain and mapping the resulting subdomains onto processors, explicit methods are triv-

ially parallelizable, whereas implicit methods appear to defy scalable parallelism. From a purely operational point of view, either type of algorithm maps an input vector into an output vector, generally through a sequence of intermediates, representing either an ordered set of nodal values or a set of coefficients for expression of a continuous function in terms of a given basis. The vectors may represent the actual unknowns of interest, or the successive corrections to those unknowns, or the residuals based on the current values of unknowns. The distinction between explicit and implicit methods relevant to parallelism comes in quantifying the discrete data dependencies — carrying both synchronization and data transmission overheads — between the components of successive vectors in the sequence. Thus, these traditional algorithmic classifications should be regarded not as opposites, but as different in degree.

We may define an implicit method as a mapping in which each component of the currently generated vector may depend upon all components of the previous, and an explicit method as a mapping in which each component of the currently generated vector depends only upon a small subset of the previous. While the particular sparsity structure of the matrix relating successive vectors is an artifact of the ordering, explicit methods have representations as stencil operations, with locally synchronized nearest neighbor dependence, whereas implicit methods are generally dense convolutions, implying globally synchronized all-to-all dependence. However, from the analogy between explicit time-stepping on a parabolic problem and relaxation applied to the corresponding steady elliptic problem, the inverse action required by the implicit algorithm must be approachable by a polynomial of the explicit operator. With this perspective, the distinction between explicit and implicit is simply a matter of semantics: we could elect to label multiple explicit steps as a single implicit step. Their distinction becomes less arbitrary in the context of an acceleration or multilevel process. Powerful implicit methods wrap multiple levels or acceleration, or both, around an inner method that has only local data dependencies. Computing the coarse level corrections or the parameters for the acceleration process generally requires nonlocal communication, and thereby sets up a trade-off between the faster convergence of implicit methods and their higher cost per iteration.

Explicit methods, whether they accurately represent a system evolving in time or in iteration space, can readily be solved in a data-parallel manner. Balancing load, minimizing the number of neighbors (representing the number of distinct messages to be handled in each iteration), and minimizing the actual number of cut stencil edges (representing the number of words to be communicated) when partitioning an unstructured grid are the only challenges to practical explicit parallelism. These somewhat conflicting objectives have been reasonably well resolved by any of a variety of methods when the processors are considered to be dedicated to the parallel task, and when the partitioning itself is treated as a static and serially performed computational task. (For example, no fewer than seventeen papers on partitioning were delivered at the Seventh SIAM Conference on Parallel Processing [1] in February 1995.) In the case of dynamically changing work per grid point or dynamically changing processor availability, finding a good repartitioning strategy is still a matter for research. So is a partitioning strategy that adapts to the numerical values of the coefficients of the problem, rather than simple connectivity and/or mesh geometry.

Implicit methods for problems with elliptic terms seem challenging to parallelize from a data dependence perspective since every component of the output depends upon every input. Formally, the Green's function is global in the continuous formulation, or the inverse matrix is dense in the discrete formulation. The saving grace that permits the parallelization of implicitly discretized problems is the rapid decay with geometrical distance of the magnitude of the coupling between input and output data. This decay means that data dependencies arising between physically remote points can be coarse-grained, and it motivates the multilevel preconditioners discussed below. The key to implicit parallelism is the exploitation of knowledge of the coefficient magnitudes to overcome enslavement to the formal all-to-all communication requirements.

MEMORY IS NO LONGER "FLAT"

We have elsewhere [9] declared the parallel implicit principle to be: "Think globally; act locally." It is increasingly incumbent on computational scientists to respect the data access hierarchies that accompany the large memories required by applications programs. The departure from a flat memory model is imposed, ultimately, by the finite size of cache and local RAM storage and by the finite speed of light, but the presence of boundaries in the space of program data is often asserted more immediately by the hardware and software overheads of system protocols for its delivery. From the frame of reference of any given processor, an approximate cost function can be constructed for the minimum time required to access a memory element that is any given logical or physical distance away. Such cost functions can typically be approximated by plateaus separated by sharp discontinuities that correspond to software latencies where some boundary of the hierarchy, such as a cache size or a local memory size, is crossed.

The ratio of times required to access remote and local data varies from 10 to 10^5 in typical architectures, the latter being characteristic of network cluster computing. The fundamental justification for focusing on *parallel* algorithms (as distinct from parallel implementations of serial algorithms) is that these discontinuities should be respected by user applications. If users cannot afford to treat memory as "flat" in large problems, then neither can they afford to treat all nonzero data dependencies on an equal footing. Consequently, algorithms must adapt to architecture, guided by knowledge of the relative strengths of different couplings from the underlying physics. Ironically, such forced adaptation sometimes results not in compromise, but in the discovery of intrinsically better algorithms for flat memory environments, as well.

The parallel programming discipline of "acting locally" pays dividends both mathematically *and architecturally* when reflected back to the serial environment. Because of the widening performance gulf between cache-conscious and cache-oblivious code on contemporary processors, algorithms whose data access patterns are clustered can achieve up to an order of magnitude better uniprocessor performance. Reports abound of superlinear parallel speed-up on problems of fixed size, in which the advantage of having the node code fit within cache in the large-processor-number limit overcomes the disadvantage in the same limit of more internodal message traffic per byte of total storage.

TWO APPROACHES TO LOCALITY: SCHUR AND SCHWARZ

Steady-state natural and human-engineered systems are often zero-sum networks in which the overall distribution of a quantity to be determined is conserved. The conservation principle holds over any size control volume, from the smallest scales requiring resolution up to the global domain. Somewhere between these extremes are the scales at which the latencies of the memory hierarchy are asserted. This suggests a multilevel discretization of the conservation laws, with coarse-grained interactions between "basins" of fast memory ("thinking globally," but on a small problem) and with fine-grained interactions within them ("acting locally," on the scales of the resolution required). Algorithms exploiting multilevel discretization have evolved naturally and somewhat independently in a variety of applications, both continuous (e.g., conservation of energy in a conducting body) and discrete (e.g., conservation of current in a network of electronic components), but have a common algebraic structure. There is a history of applying both direct and iterative methods to such problems.

Direct methods involve the construction, by explicit condensation, of lower-dimensional systems for degrees of freedom that act as separators between substructures. In the literature of differential equations, this is the Poincare-Steklov operator; in linear algebra, it is the Schur complement or capacitance matrix. Przemieniecki [14] provided an early formalization in mechanics, called "substructuring," and Kron [11] did the same

for electrical networks, coining the term “diakoptics” (meaning “method of tearing”). Przemieniecki was surprisingly prescient in writing “From past experiences [...], it is evident that some form of structural partitioning is usually necessary either because different methods of analysis are used on different structural components or because of the limitations imposed by the capacity of digital computers. Even when the next generation of faster and larger digital computers becomes a well-established tool [...], it seems rather doubtful, because of the large number of unknowns, that the substructure displacement method of analysis would be wholly superseded by an overall analysis carried out on the complete structure.” Because the Schur complement system itself is expensive to construct, though its action on individual vectors is relatively inexpensive, conjugate gradient iteration is preferred to direct Gaussian elimination, if a good preconditioner can be derived.

The simplest iterative methods involve cycling between the subdomains, whose unknown boundary data are updated through overlapping with neighbors. They may generically be called Schwarz methods after Hermann Schwarz of complex analysis, who employed them theoretically [16]. They were revisited in the age of computation by Miller [13]. An equivalence can be demonstrated between the Schur and Schwarz approaches, in the sense that the Schwarz iterates are produced when the subdomain interior solutions are reconstructed from the Schur iterates on the separator. This equivalence requires exact subdomain solves on each subdomain, a price which is usually relaxed when Schwarz is used in practice, in the interest of optimal complexity. As with the Schur method, Schwarz is usually used inside a preconditioned Krylov method. Schwarz and Schur methods can also be hybridized, as in Smith’s wirebasket method [17], which employs an overlapping Schwarz decomposition of the set of separators (the so-called “wirebasket”) onto which the subdomain problems are first condensed, as a preconditioner.

Both Schur and Schwarz are methods of divide-and-conquer type for solving the PDE BVP, $\mathcal{L}u = f$ in Ω , through solving a sequence of problems $\mathcal{L}'_k u'_k = f'_k$ in subdomains Ω_k covering Ω , and iteratively combining the partial solutions u'_k to form u . The two main advantages of divide-and-conquer are that the subdomain problems are individually smaller than the original, and they may also have simpler structure. The disadvantage to be overcome is the new task of their combination.

Schur methods are based on a partitioning of the grid by ordering the separator nodes last. Let the degrees of freedom associated with the separators be denoted u_B and the unknowns associated with the substructures thus created be denoted u_I . This partition induces a permutation of the discrete system $Au = f$, as follows:

$$\begin{bmatrix} A_I & A_{IB} \\ A_{BI} & A_B \end{bmatrix} \begin{pmatrix} u_I \\ u_B \end{pmatrix} = \begin{pmatrix} f_I \\ f_B \end{pmatrix}.$$

A_I is block diagonal, with a block for each substructure. Note that because the separator nodes correspond to a

physically lower dimensional operator, the algebraic dimension of A_B is relatively small. The Schur complement $S \equiv A_B - A_{BI}A_I^{-1}A_{IB}$, arises formally from the factorization:

$$A = \begin{bmatrix} A_I & A_{IB} \\ A_{BI} & A_B \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{BI}A_I^{-1} & I \end{bmatrix} \begin{bmatrix} A_I & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_I^{-1}A_{IB} \\ 0 & I \end{bmatrix}.$$

Its significance is that once $Su_B = \hat{f}_B \equiv f_B - A_{BI}A_I^{-1}f_I$ is solved for u_B , the block diagonal system $A_I u_I = \hat{f}_I \equiv f_I - A_{IB}u_B$ may be solved for u_I .

Schwarz methods are a special case of “iterative subspace correction methods” [18]. A general iterative correction algorithm for $Au = f$ proceeds as follows:

1. Compute the residual: $r^l = f - Au^l$;
2. Solve $Ae = r^l$ approximately for an estimate of the error: $\tilde{e} = B^{-1}r^l$;
3. Update: $u^{l+1} = u^l + \tilde{e}$.

An iterative subspace correction method accomplishes the middle step above by:

- 2(a). Decomposing the space of the solution u : $U = \sum_k U_k$;
- 2(b). Finding the restriction of A to each U_k : $A_k = R_k A R_k^T$, for some restriction operators R_k and extension operators R_k^T ;
- 2(c). Forming B^{-1} from the A_k^{-1} , where the inverse is well defined within the k^{th} subspace.

Special cases include traditional Jacobi and Gauss-Seidel, in which the “subspaces” correspond to the nodal bases, multigrid, in which the subspaces have global support and are hierarchically nested by wavenumber, and Schwarz-style domain decomposition, which are subdomain-blocked Jacobi or Gauss-Seidel methods (with possibly overlapping blocks), possibly combined with solutions on one or more coarse grids. For parallelism, we will concentrate on the Jacobi-like (or “additive”) form of Schwarz, wherein

$$B^{-1} \equiv \sum_k R_k^T (\tilde{A}_k)^{-1} R_k,$$

where \tilde{A}_k^{-1} is a convenient approximation to $A_k = R_k A R_k^T$. Of course, we never actually assemble either A or B^{-1} globally. Rather, when their action on a vector is needed, a processor governing each subdomain executes local operations, possibly after receiving a thin buffer of data required from its neighbors to complete stencil operations on the boundary of the subdomain.

A mathematical signature of the data-parallel Schwarz and Schur methods is a triple product of matrix operators consisting of the inverse of a square operator in the middle, with “rectangular” operators on either side that map between spaces of different dimensions. A Schur complement contains such triple products in which the middle term may be of higher dimension than

the terms of the sum, itself. Roughly speaking, such a term is contributed for each subdomain adjacent to each separator segment, and the net action of all such terms on a vector defined over the wirebasket can be evaluated with subdomain-scale concurrency. A Schwarz preconditioner contains such triple products in which the middle term is of lower dimension than the terms of the sum. Again, the net action of all such terms on a vector defined over the total domain can be evaluated with subdomain-scale concurrency. The best methods include, in addition, a solution on a coarse grid (with as little as one grid point per subdomain, or even less). The coarse grid problem may be solved concurrently with the subdomain problems or sequentially, and on one processor, on a subset or processors, or redundantly on all processors. A key aspect of domain decomposition convergence theory is that, for many problem classes, the number of iterations required by the Krylov method (essentially, the number of times the preconditioned action has to be evaluated) depends only weakly on the granularity of the decomposition into subdomains. This leaves the parallel granularity at the disposal of architectural considerations.

CONVERGENCE AND COMPLEXITY

To illustrate what is meant by “weak” convergence dependence, and the prices that must be paid *per iteration* to obtain it, we quote from the literature some standard bounds on the condition number of various preconditioned Schwarz and Schur methods for the model problem of the two-dimensional Laplacian on a square grid with mesh size $h (\ll 1)$ and subdomain size $H (\leq 1, H \gg h)$. (Recall that when the conjugate gradient method is applied to a problem with condition number κ , $\mathcal{O}(\kappa^{1/2})$ iterations are required for a fixed residual reduction ratio, so $\mathcal{O}(1)$ is optimal.)

Unpreconditioned

$$\kappa = \mathcal{O}(1/h^2)$$

Block-Jacobi (Additive Schwarz with no overlap)

$$\kappa = \mathcal{O}(1/Hh)$$

Additive Schwarz (no coarse grid, generous overlap)

$$\kappa = \mathcal{O}(1/H^2)$$

Additive Schwarz (coarse grid, modest overlap of δ)

$$\kappa = \mathcal{O}(1 + (H/\delta))$$

Additive Schwarz (coarse grid, generous overlap)

$$\kappa = \mathcal{O}(1)$$

Bramble-Pasciak-Schatz (coarse grid, non-overlapped wirebasket)

$$\kappa = \mathcal{O}((1 + \log(H/h))^2)$$

Smith Vertex-Space (coarse grid, overlapped wirebasket)

$$\kappa = \mathcal{O}(1)$$

A simple argument for domain decomposition methods is as follows: Given a problem of size $N (= 1/h^2$ in the example above) and a solver with arithmetic complexity $c \cdot N^\alpha$, partition the problem into $P (= 1/H^2)$ subproblems of size N/P . The complexity of applying the solver once to the entire set of subproblems is $P \cdot c \cdot (N/P)^\alpha$. Even in serial, therefore, $I = P^{\alpha-1}$ iterations can be afforded to coordinate the solutions of the subproblems, while breaking even. If $\alpha = 1$, there is no “headroom” for extra iterations in serial on the basis of arithmetic complexity alone. However, there may still be headroom even if $\alpha = 1$ and: (1) the $\alpha = 1$ method involves too much communication to parallelize efficiently, (2) a hierarchical or multiblock data structure is “natural” for modeling or implementation reasons, or (3) memory limitations, cache thrashing, or I/O costs demand decomposition anyway.

Given one of the order estimates above for the number of iterations, one can construct a cost function for the total amount of arithmetic and the total amount of data exchange between subdomain-oriented processes that are required to bring a Krylov-Schwarz or Krylov-Schur method to convergence for a decomposition of given granularity. This complexity information can be combined with a model of parallel computer architecture and a mapping of subdomains to processors to estimate the total running time (computation plus communication) of a parallel implementation, as a function of the of the number of grid points and the total number of processors. By taking the partial derivative of this cost function with respect to the number of processors and setting it to zero, one can derive an optimal order processor granularity, P_{opt} , and hence a optimal order running time. For three-dimensional problems on a hypercube, it may be shown that P_{opt} scales like $1/h^3$, i.e., that a number of processors proportional to the number of grid points may be effectively used. This is for zero-overlap, coarse-grid-free Schwarz preconditioning and assumes that the subdomain problems are solved exactly. However, it is known experimentally that the subdomain problems may be solved inexpensively and approximately, e.g., with calls to a *local* ILU subroutine or with a local multigrid V-cycle, with small deterioration in convergence rate. This observation encourages the employment of *other* favored global solvers as subdomain preconditioners, without any attempt to parallelize the solvers themselves, which is generally difficult.

PRESENT CHALLENGES

Theoretical and experimental results on Krylov-Schwarz and Krylov-Schur methods have been reported on extensively since 1987 in the proceedings of the nearly annual International Conference on Domain Decomposition Methods for Partial Differential Equations [3,4,6,7,8,10,15], together with applications and parallel implementations. Browsers will find a considerable breadth of topics investigated, including PDEs with non-smooth coefficients, PDEs with large first-order terms, saddle-point systems, eigensystems, high order discretizations (including spectral multidomain), non-

conforming discretizations, as well as the relationships of domain decomposition to sparse direct methods and to multigrid methods, tools for partitioning and parallel environments, and applications to many regimes of structural mechanics, fluid mechanics, electricity and magnetism, and geophysics.

Domain decomposition methods have been extended to nonlinear problems primarily through the Newton-Krylov-Schwarz (NKS) paradigm, in which the methodologies for linear problems described above are used inside of a Newton iteration loop, usually in a matrix-free manner. Tuning the inner iteration convergence tolerances to outer nonlinear convergence and to continuation parameter progress is an active and important research area, as is tuning preconditioner strength to computer architecture and memory limits.

The crux of the matter in complex applications that lie outside of the scope of existing convergence theory is, not surprisingly, the preconditioning. Since retaining all data dependencies *in the preconditioner* is ruled out by communication complexity considerations, physical intuition must be employed to choose partitions and preconditioners which preserve or approximate well the strongest coupling, while sacrificing or coarse-graining the weakest. (Of course, all coupling, strong or weak, is retained in the system actually being solved.) Knowledge of asymptotics or approximate eigenmodes can usually be put to advantage in preconditioning. Interdisciplinary, problem-specific collaborations between applications experts, computer scientists, and applied mathematicians are required to identify and capture in some low-dimensional, low-complexity way the key determinants to fast convergence, and to implement the best algorithm for the architecture, with the ultimate metric being accuracy per wall-clock time unit, subject to memory constraints.

REFERENCES

1. D. H. Bailey, P. E. Bjørstad, J. R. Gilbert, M. V. Mascagni, R. S. Schreiber, H. D. Simon, V. J. Torczon & L. T. Watson, eds., *Proc. Seventh SIAM Conf. on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1994.
2. T. F. Chan and T. P. Mathew, *Domain Decomposition Algorithms*, Acta Numerica, 1994, pp. 61–143.
3. T. F. Chan, R. Glowinski, J. Périaux and O. B. Widlund, eds., *Proc. Second Int. Symp. on Domain Decomposition Methods for Partial Differential Equations* (Los Angeles, 1988), SIAM, Philadelphia, 1989.
4. T. F. Chan, R. Glowinski, J. Périaux, O. B. Widlund, eds., *Proc. Third Int. Symp. on Domain Decomposition Methods for Partial Differential Equations* (Houston, 1989), SIAM, Philadelphia, 1990.
5. C. Farhat and F.-X. Roux, *Implicit Parallel Processing in Structural Mechanics*, Computational Mechanics Advances **2**, 1994, pp. 1–124.
6. R. Glowinski, G. H. Golub, G. A. Meurant and J. Périaux, eds., *Proc. First Int. Symp. on Domain Decomposition Methods for Partial Differential Equations* (Paris, 1987), SIAM, Philadelphia, 1988.
7. R. Glowinski, Yu. A. Kuznetsov, G. A. Meurant, J. Périaux and O. B. Widlund, eds., *Proc. Fourth Int. Symp. on Domain Decomposition Methods for Partial Differential Equations* (Moscow, 1990), SIAM, Philadelphia, 1991.
8. D. E. Keyes, T. F. Chan, G. A. Meurant, J. S. Scroggs and R. G. Voigt, *Proc. Fifth Int. Conf. on Domain Decomposition Methods for Partial Differential Equations* (Norfolk, 1991), SIAM, Philadelphia, 1992.
9. D. E. Keyes, Y. Saad and D. G. Truhlar, eds., *Domain-based Parallelism and Problem Decomposition Methods in Science and Engineering*, SIAM, Philadelphia, 1995.
10. D. E. Keyes and J. Xu, eds., *Proc. Seventh Int. Conf. on Domain Decomposition Methods in Science and Engineering* (Penn State, 1993), AMS, Providence, 1995.
11. G. Kron, *A set of Principles to Interconnect the Solutions of Physical Systems*, J. Appl. Phys., **24**, 1953, pp. 965–980.
12. P. Le Tallec, *Domain Decomposition Methods in Computational Mechanics*, Computational Mechanics Advances **2**, 1994, pp. 121–220.
13. K. Miller, *Numerical Analogs for the Schwarz Alternating Procedure*, Numer. Math., **7**, 1965, pp. 91–103.
14. J. S. Przemieniecki, *Matrix Structural Analysis of Substructures*, AIAA J., **1**, 1963, pp. 138–147.
15. A. Quarteroni, J. Périaux, Yu. A. Kuznetsov and O. B. Widlund, eds., *Proc. Sixth Int. Conf. on Domain Decomposition Methods in Science and Engineering* (Como, 1992), AMS, Providence, 1994.
16. H. A. Schwarz, *Gesammelte Mathematische Abhandlungen*, **2**, pp. 133–134, Springer, Berlin, 1890.
17. B. F. Smith, P. E. Bjørstad and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Algorithms for Elliptic Partial Differential Equations*, Cambridge Univ. Press, Cambridge, 1995.
18. J. Xu, *Iterative Methods by Space Decomposition and Subspace Correction*, SIAM Review **34**, 1991, pp. 581–613.