Parallel Computational Fluid Dynamics
Towards Teraflops, Optimization and Novel Formulations
D. Keyes, A. Ecer, J. Periaux, N. Satofuka and P. Fox (Editors)
© 2000 Elsevier Science B.V. All rights reserved.

Design of Large-Scale Parallel Simulations

Matthew G. Knepley
Ahmed H. Sameh
Vivek Sarin

Computer Science Department, Purdue University, West Lafayette, IN 47906-1398
{knepley, sameh, sarin}@cs.purdue.edu

We present an overview of the design of software packages called *Particle Movers* that have been developed to simulate the motion of particles in two and three dimensional domains. These simulations require the solution of nonlinear Navier-Stokes equations for fluids coupled with Newton's equations for particle dynamics. Furthermore, realistic simulations are extremely computationally intensive, and are feasible only with algorithms that can exploit parallelism effectively. We describe the computational structure of the simulation as well as the data objects required in these packages. We present a brief description of a particle mover code in this framework, concentrating on the following features: design modularity, portability, extensibility, and parallelism. Simulations on the SGI Origin2000 demonstrate very good speedup on a large number of processors.

1. OVERVIEW

The goal of our KDI effort is to develop high-performance, state-of-the-art software packages called *Particle Movers* that are capable of simulating the motion of thousands of particles in two dimensions and hundreds in three dimensions. Such large scale simulations will then be used to elucidate the fundamental dynamics of particulate flows and solve problems of engineering interest. The development methodology must encompass all aspects of the problem, from computational modeling for simulations in Newtonian fluids that are governed by the Navier-Stokes equations (as well as in several popular models of viscoelastic fluids), to incorporation of novel preconditioners and solvers for the nonlinear algebraic equations which ultimately result. The code must, on the one hand, be high-level, modular, and portable, while at the same time highly efficient and optimized for a target architecture.

We present a design model for large scale parallel CFD simulation, as well as the PM code developed for our Grand Challenge project that adheres to this model. It is a true distributed memory implementation of the prototype set forth, and demonstrates very good scalability and speedup on the Origin2000 to a maximum test problem size of over half a million unknowns. Its modular design is based upon the GVec package [14] for PETSc, which is also discussed, as it forms the basis for all abstractions in the code. PM has been ported to the Sun SPARC and Intel Pentium running Solaris, IBM SP2 running AIX, Origin2000 running IRIX, and Cray T3E running Unicos with no explicit code modification. The code has proved to be easily extensible, for example in the rapid development of a fluidized bed experiment from the original sedimentation code. The code for this project was written in Mathematica and C. The C development used the PETSc framework developed at Argonne National Laboratory, and the GNU toolset.

The goal of our interface development is mainly to support interoperability of scientific software. "Interoperability" is usually taken to mean portability across different architectures or between different languages. However, we propose to extend this term to cover what we call *algorithmic portability*, or the ability to express a mathematical algorithm in several different software paradigms. The interface must also respect the computational demands of the application to enable high performance, and provide the flexibility necessary to extend its capabilities beyond the original design. We first discuss existing abstractions present in most popular scientific software frameworks. New abstractions motivated by computational experience with PDEs arising from fluid dynamics applications are then introduced. Finally, the benefits of this new framework is illustrated using a novel preconditioner developed for the Navier-Stokes equations. Performance results on the Origin2000 are available elsewhere [11].

2. EXISTING ABSTRACTION

The existing abstractions in scientific software frameworks are centered around basic linear algebra operations necessary to support the solution of systems of linear equations. The concrete examples from this section are taken from the PETSc framework [18], developed at Argonne National Laboratory, but are fairly generic across related packages [10]. These abstractions differ from the viewpoint adopted for the BLAS libraries [4] in the accessibility of the underlying data structure. BLAS begins by assuming a data format and then formulates a set of useful operations which can be applied to the data itself. Moreover, the data structure itself is typically exposed in the calling sequence of the relevant function. In the data structure neutral model adopted by PETSc, the object is presented to the programmer as an interface to a certain set of operations. Data may be read in or out, but the internal data structure is hidden. This data structure independence allows many different implementations of the same mathematical operation.

The ability to express a set of mathematical operations in terms of generic interfaces, rather than specific operations on data, is what we refer to as *algorithmic portability*. This capability is crucial for code reuse and interoperability [5]. The operations of basic linear algebra have been successfully abstracted in to the Vector and Matrix interfaces present in PETSc. A user need not know anything about the internal representation of a vector to obtain its norm. Krylov solvers for linear systems are almost entirely based upon these same operations, and therefore have also been successfully abstracted from the underlying data structures. Even in the nonlinear regime, the Newton-Krylov iteration adds only further straightforward vector operations to the linear solve. These abstractions fail when we discretize a continuous problem, or utilize information inherent in that process to aid in solving the system. A richer set of abstractions is discussed in the next section.

3. HIGHER LEVEL ABSTRACTIONS

To obtain a generic expression of more modern algorithms, we must supplement our set of basic operations and raise the level of abstraction at which we specify algorithmic components [3,2]. For instance, in most multilevel algorithms there is some concept of a hierarchy of spaces, and in finite element based code this hierarchy must be generated by a corresponding hierarchy of meshes [7,1,17,19]. Thus the formation of this hierarchy should in some sense be a basic operation. The detailed nature of the meshes affects the performance of an algorithm but not its conceptual basis; however, it might be necessary to specify nested meshes or similar restrictions. Raising the level of abstraction, by making coarsening a basic operation, allows the specification of these algorithms independently of the detailed operations on the computational mesh. We propose a set of higher level abstractions to encapsulate solution algorithms for mesh-based PDEs that respects the current PETSc-like interface for objects such as vectors and matrices.

A *Mesh* abstraction should allow the programmer to interrogate the structure of the discrete coordinate system without exposing the details of the storage or manipulation. Thus it should allow low level queries about the coordinates of nodes, elements containing a given node or generic point, and also support higher level operations such as iteration over a given boundary, automatic partitioning and automatic coarsening or refinement. In the GVec libraries, a *Partition* interface is also employed for encapsulating the layout of a given mesh over multiple domains.

A mesh combined with a discretization prescription on each cell provides a description of the discrete space which we encapsulate in the *Grid* interface. The Grid provides access to both the Mesh and a *Discretization*. The Discretization for each field which understands how to form the weak form of a function or operator on a given cell of the mesh. In GVec, *Operator* abstractions are used to encapsulate the weak form of continuous operators on the mesh, and provide an interface for registration of user-defined operators. This capability is absolutely essential as the default set of operators could never incorporate enough for all users. Finally, the Grid must maintain a total ordering of the variables for any given discretized operator or set of equations. In GVec, the *VarOrdering* interface encapsulates the global ordering and ties it to the nodes in the mesh, while the *LocalVarOrdering* interface describes the ordering of variables on any given node.

The Grid must also encapsulate a given mathematical problem defined over the mesh. Thus it should possess a database of the fields defined on the mesh, including the number of components and discretization of each. Furthermore, each problem should include some set of operators and functions defining it which should be available to the programmer through structured queries. This allows runtime management of the problem which can be very useful. For example, in the Backward-Euler integrator written for GVec, the identity operator is added automatically with the correct scaling to a steady-state problem to create the correct time-dependence. The only information necessary from the programmer is a specification of the the fields which have explicit time derivatives in the equation. This permits more complicated time-stepping schemes to be used without any change to the original code. The Grid should also maintain information about the boundary conditions and constraints. In GVec, these are specified as functions defined over a subset of mesh nodes identified by a boundary marker, which is given to each node during mesh generation. This allows automatic implementation of these constraints with the programmer only specifying the original continuous representation of the boundary values.

3.1. Node Classes

In order to facilitate more complex distributions of variables over the mesh, GVec employs a simple scheme requiring one extra integer per mesh node. We define a *class* as a subset of fields defined on some part of the mesh. Each node is assigned a class, meaning that the fields contained in that class are defined on that node. For example, in a P2/P1 discretization of the Stokes equation on a triangular mesh, the vertices might have class 0, which includes velocity and pressure, whereas the midnodes on edges would have class 1, including only velocity. Thus this scheme easily handles mixed discretizations. However, it is much more flexible. Boundary conditions can be implemented merely by creating a new class for the affected nodes that excludes the constrained fields, and the approach for constraints is analogous. This information also permits these constraints to be implemented at the element level so that explicit elimination of constrained variables is possible, as well as construction of operators over only constrained variables. This method has proven to be extremely flexible while incurring minimal overhead or processing cost.

4. MULTILEVEL PRECONDITIONING FOR NAVIER-STOKES

As an example of the utility of higher level abstractions in CFD code, we present a particulate flow problem [9,11] that incorporates a novel multilevel preconditioner [17] for the Navier-Stokes equations augmented by constraints at the surface of particles [13]. The fluid obeys the Navier-Stokes equations and the particles each obey Newton's equations. However, the interior forces need not be explicitly calculated due to the special nature of our finite element spaces [8,9,15]. These equations are coupled through a no-slip boundary condition at the surface of each particle. The multilevel preconditioner is employed to accelerate the solution of the discrete nonlinear system generated by the discretization procedure.

4.1. Preconditioning

The nonlinear system may be formulated as a saddle-point problem, where the upper left block A is a nonlinear operator, but the constraint matrix B is still linear:

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}. \tag{1}$$

Thus we may approach the problem exactly as with Stokes, by constructing a divergenceless basis using a factorization of B. The ML algorithm constructs the factorization

$$P^T B V = \begin{pmatrix} D \\ 0 \end{pmatrix}, \tag{2}$$

$$V^T V = I, \tag{3}$$

where V is unitary, D is diagonal, but P is merely full rank. If P were also unitary we would have the SVD, however this is prohibitively expensive to construct. Using this factorization, we may project the problem onto the space of divergenceless functions spanned by P_2. Thus we need only solve the reduced nonlinear problem (4) in the projected space

$$P_2^T A P_2 \hat{u}_2 = P_2^T \left(f - A P_1 D^{-T} V^T g \right) \tag{4}$$

The efficiency issues for this scheme are the costs of computing, storing, and applying the factors, as well as the conditioning of the resulting basis. The ML algorithm can store the factors in $O(N)$ space and apply them in $O(N)$ time. The conditioning of the basis can be guaranteed for structured Stokes problems, and good performance is seen for two and three dimensional unstructured meshes. The basis for the range and null space of B are both formed as a product of a logarithmic number of sparse matrices. The algorithm proceeds recursively, coarsening the mesh, computing the gradient at the coarse level $\tilde{B}$, and forming one factor at each level. The algorithm terminates when the mesh is coarsened to a single node, or at some level when an explicit QR decomposition of $\tilde{B}$ can be accomplished. In a parallel setting, the processor domains are coarsened to a single node and then a QR decomposition is carried out along the interface.

4.1.1. Software Issues

The ML algorithm must decompose the initial mesh into subdomains, and in each domain form the local gradient operator. Thus if ML is to be algorithmically portable, we must have basic operations expressing these actions. The ability of the Mesh interface to automatically generate this hierarchy allows the programmer to specify the algorithm independently of a particular implementation, such as a structured or unstructured mesh. In fact, the convergence of ML is insensitive to the particular partition of the domain so that a user may select among mesh coarsening algorithms to maximum other factors in the performance of the code. Using the Grid interface to form local gradients on each subdomain frees the programmer from the details of handling complex boundary conditions or constraints, such as those arising in the particulate flow problem. For example, the gradient in the particulate flow problem actually appears as

$$B = \begin{pmatrix} B_I \\ \tilde{P}^T B_\Gamma \end{pmatrix}, \tag{5}$$

where B_I denotes the gradient on the interior and outer boundary of the domain, B_Γ is the gradient operator at the surface of the particles, and $\tilde{P}$ is the projector from particle unknowns to fluid velocities at the particle surface implementing the no-slip condition. This new gradient operator has more connectivity and more complicated analytic properties. However, the code required no change in order to run this problem since the abstractions used were powerful enough to accommodate it.

4.1.2. Single Level Reduction

We begin by grouping adjacent nodes into partitions and dividing the edges into two groups: interior edges connecting nodes in one partition, and boundary edges connecting partitions. The gradient matrix may be reordered to represent this division, $\begin{pmatrix} B_I \\ B_\Gamma \end{pmatrix}$. The upper matrix B_I is block diagonal, with one block for each partition. Each block represents the restriction of the gradient operator to that cluster. Furthermore, we may factor each block independently using the SVD, so that if $U_i B_i V_i^T = \begin{pmatrix} D_i \\ 0 \end{pmatrix}$ we may factor each domain independently. We now use these diagonal matrices to reduce the columns in $B_\Gamma V^T$ by block row reduction. A more complete exposition may be found in [13].

4.1.3. Recursive Framework

We may now recursively apply this decomposition to each domain instead of performing an exact SVD. The factorization process may be terminated at any level with an explicit QR decomposition, or be continued until the coarsest mesh consists of only a single node. The basis P is better conditioned with earlier termination, but this must be weighed against the relatively high cost of QR factorization. Thus, we have the high level algorithm

```
1. Until $numNodes \leq threshold$ do:
    (a) Partition mesh
    (b) Factor local operator
    (c) Block reduce interface
    (d) Coarsen mesh
2. QR factor remaining global operator
```

5. CONCLUSIONS

The slow adoption of modern solution methods in large CFD codes highlights the close ties between interoperability and abstraction. If sufficiently powerful abstractions are not present in a software environment, algorithms making use of these abstractions are effectively not portable to that system. Implementations are, of course, possible using lower level operations, but these are prone to error, inflexible, and very time-consuming to construct. The rapid integration of the ML preconditioner into an existing particulate flow code[16] demonstrates the advantages of these more powerful abstractions in a practical piece of software. Furthermore, in the development and especially in the implementation of these higher level abstractions, the architecture must be taken into account. It has become increasingly clear that some popular computational kernels, such as the sparse matrix-vector product, may be unsuitable for modern RISC cache-based architectures[17]. Algorithms such as ML which possess kernels that perform much more work on data before it is ejected from the cache should be explored as an alternative or supplement to current solvers and preconditioners.

REFERENCES

1. Achi Brandt, Multi-Level Adaptive Solutions to Boundary-Value Problems, *Mathematics of Computation* **31** (1977) 333—390.
2. David L. Brown, William D. Henshaw, and Daniel J. Quinlan, Overture: An Object Oriented Framework for Solving Partial Differential Equations, in: *Scientific Computing in Object-Oriented Parallel Environments*, Lecture Notes in Computer Science 1343 (Springer, 1997). Overture is located at `http://www.llnl.gov/casc/Overture`.
3. H.P. Langtangen, *Computational Partial Differential Equations—Numerical Methods and Diffpack Programming* (Springer-Verlag, 1999). Diffpack is located at `http://www.nobjects.com/Products/Diffpack`.
4. Jack Dongarra, J. DuCroz, S. Hammarling, and R. Hanson, A proposal for an extended set of Fortran basic linear algebra subprograms, Technical Memo 41, Mathematics and Computer Science Division, Argonne National Laboratory, December, 1984.

5. The ESI Forum is located at `http://z.ca.sandia.gov/esi`.
6. William D. Gropp, Dinesh K. Kaushik, David E. Keyes, and Barry F. Smith, Cache Optimization in Multicomponent Unstructured-Grid Implicit CFD Codes, in: *Proceedings of the Parallel Computational Fluid Dynamics Conference* (Elsevier, 1999).
7. Ami Harten, Multiresolution representation of data: General framework, *SIAM Journal on Numerical Analysis* **33** (1996) 1205–1256.
8. Todd Hesla, A Combined Fluid-Particle Formulation, Presented at a Grand Challenge Group Meeting (1995).
9. Howard Hu, Direct simulation of flows of solid-liquid mixtures, *International Journal of Multiphase Flow* **22** (1996).
10. Scott A. Hutchinson, John N. Shadid, and Ray S. Tuminaro, Aztec User's Guide Version 1.0, Sandia National Laboratories, TR Sand95–1559, (1995).
11. Matthew G. Knepley, Vivek Sarin, and Ahmed H. Sameh, Parallel Simulation of Particulate Flows, in: *Solving Irregularly Structured Problems in Parallel*, Lecture Notes in Computer Science 1457 (Springer, 1998).
12. Denis Vanderstraeten and Matthew G. Knepley, Paralell building blocks for finite element simulations: Application to solid-liquid mixture flows, in: *Proceedings of the Parallel Computational Fluid Dynamics Conference* (Manchester, England, 1997).
13. Matthew G. Knepley and Vivek Sarin, Algorithm Development for Large Scale Computing: A Case Study, in: *Object-Oriented Methods for Interoperable Scientific and Engineering Computing* (Springer, 1999).
14. Matthew G. Knepley, GVec Beta Release Documentation, available at `http://www.cs.purdue.edu/homes/knepley/comp_fluid/gvec.nb.ps`.
15. Matthew G. Knepley, Masters Thesis, University of Minnesota, available at `http://www.cs.purdue.edu/homes/knepley/iter_meth`.
The *Mathematica* software and notebook version of this paper may be obtained at `http://www.cs.purdue.edu/homes/knepley/iter_meth`.
16. Vivek Sarin, *An efficient iterative method for Saddle Point problems*, PhD thesis, University of Illinois, 1997.
17. Vivek Sarin and Ahmed H. Sameh, An efficient iterative method for the generalized Stokes problem, *SIAM Journal on Scientific Computing*, **19** (1998) 206–226.
18. Barry F. Smith, William D. Gropp, Lois Curman McInnes, and Satish Balay, PETSc 2.0 Users Manual, Argonne National Laboratory, TR ANL–95/11, 1995, available via `ftp://www.mcs.anl/pub/petsc/manual.ps`.
19. Shang-Hua Teng, *Coarsening, Sampling, and Smoothing: Elements of the Multilevel Method*, Unpublished, 1999.