CHAPTER 33

Distributed Computing and Adaptive Solution to Nonlinear PDES*

Jeffrey S. Scroggs†
Joel Saltz†

Abstract. A parallel algorithm for the efficient solution of nonlinear time dependent convection diffusion equations with small parameter on the diffusion term will be presented. Implementation aspects are emphasized. The method is based on a physically motivated domain decomposition that is dictated by singular perturbation analysis. The analysis is used to determine regions where certain reduced equations may be solved in place of the full equation. The analysis also motivates the adaptive aspects of the algorithm, dictating the appropriate scales for a coarse grid and a fine grid used to adaptively refine in a region where there is a singularity. Both the coarse grid and the fine grid are distributed across processors.

It is well known that very substantial amounts of computational time can be required to solve nonlinear time-dependent convection diffusion equations. The motivation consequently exists to solve such problems on high performance distributed memory multiprocessors such as the iPSC-2/860. In this paper we present methods and tools we have developed for implementing such adaptive codes on distributed memory multiprocessors.

1. Introduction. In this paper, an algorithm that is appropriate for solving nonlinear convection-diffusion equations in multiple dimensions is presented and demonstrated on a distributed memory parallel computer. Implementation utilized primitives that allow a single global coordinate system to access arrays defined in the local memories of a machine's processors. This method is a combination of the parallel processing techniques discussed in [15], [13] with a modification of the adaptive numerical method discussed in [16], and is an extension of the algorithm presented in [20],[19] to two dimensions.

*Supported by NASA Contract No. NAS1-18605.
†ICASE, MS 132C, NASA Langley Research Center, Hampton, VA 23665.

We will refer to the collection of subarrays indexed by this global coordinate system as a *distributed* array. For instance, we might partition a $n \times n$ distributed array into four $n/2 \times n/2$ subarrays. Each of these subarrays would be placed on one of the four processors. Global array reference $(n/2+1, n/2+1)$ would then actually refer to $(1,1)_{local}$ in the local memory of one of the processors. The PARTI (Parallel Automated Runtime Toolkit at ICASE) package of primitives allows users to define distributed arrays during program execution and then to use a global coordinate system to gather and scatter data elements to and from those distributed arrays.

There are two varieties of domain decomposition utilized here–a physical domain decomposition motivated by asymptotic analysis, and a computational domain decomposition used to distribute the work across processors. The computational domain decomposition increases the efficiency of implementation.

The physical domain decomposition is accomplished using a symbiosis of numerics and asymptotics. The asymptotic analysis identifies the regions where diffusion is negligible, where a reduced equation is solved in place of the full equation. Solving this reduced equation significantly reduces the work in the numerical method. The numerics provides a means of solution in the subdomains, and also a feedback mechanism. As a feedback mechanism, the numerical scheme can expose regions of unexpected behavior, confirming or correcting the asymptotics-induced subdomain boundaries. This decomposition permits the use of locally refined meshes, allowing the concentration of computational effort in the regions where it is needed most. Since the computational requirements are reduced, both the domain decomposition and the use of the reduced equation are preconditionings for this problem. Identification of subdomain boundaries is accomplished during the computation–no *a priori* knowledge of the shock location is assumed.

The physical domain decomposition is appropriate for equations that exhibit strong nonlinearities, such as the equations that arise when modeling fluids. The method is particularly well-suited in situations that require resolution of one or more of the small scales. Such situations arise, for example in hypersonic fluid dynamics and combustion problems where the chemistry depends on the viscous profiles.

2. Problem.

The example here will be taken from Computational Fluid Dynamics (CFD) in the transonic regime. The gasdynamic equations, including viscous effects, are used as a model in these settings. Except for very simple geometries and boundary conditions there is no analytic solution to these gasdynamic equations, and a numerical solution is difficult to obtain. For these reasons new algorithms are usually developed and tested on a more tractable canonical equation, such as

$$P[u] := u_t + uu_x + \sigma u_y - \epsilon \Delta u = 0. \tag{1}$$

This equation contains many of the properties that make the gasdynamic equations difficult to solve; namely, it is capable of modeling rapid variations such as shocks and boundary layers.

The method will be described and demonstrated by solving (1) on the spatial domain

$$\Pi := \{(x,y) \mid 0 \le x \le 1, 0 \le y \le 1\}, \tag{2}$$

where the temporal variable is restricted to $0 \le t < T$. Thus the entire computational domain is $D := [0,T] \times \Pi$. The solution satisfies

$$u(0,x,y) = \gamma(x,y), \quad \text{for} \quad (x,y) \in \Pi, \quad \text{and} \tag{3}$$

$$u(t,x,y) = \alpha(t,x,y) \tag{4}$$

for the inflow portion of ∂D. The boundary data are continuous and sufficiently smooth so that the solution to (1) is uniquely defined (for example, see [3]).

3. Physical Domain Decomposition. When the equation is nondimensionalized as in [5], the diffusion coefficient ϵ is inversely proportional to the Reynolds number. Based on free-stream conditions in transonic flow, the Reynolds number for this problem is large; thus, ϵ is a small parameter in this setting. Asymptotic analysis exploits the smallness of the positive parameter ϵ and involves study of the solution as ϵ tends to zero ($\epsilon \downarrow 0$).

Asymptotic analysis provides analytic tools to identify and utilize the various physical scales. The scales for Equation (1) are for the convection terms uu_x and σu_y, and for the diffusion term $\epsilon \Delta u$. Competition between convection and diffusion is crucial to the understanding of fluid flow, and the determining which of these is dominant can be made by examining the relationship between their scales. When modeling transonic flow, except in regions of rapid variation such as in shocks and boundary-layers, convection dominates diffusion. Thus, it is natural to first study the solution of the reduced equation

$$P_0[U] := U_t + UU_x + \sigma U_y = 0, \tag{5}$$

obtained by setting $\epsilon = 0$ in Equation (1). Weak solutions U are sought for (5) with data (3-4). In order that U be uniquely defined, we assume that an appropriate entropy condition is satisfied [11]. Suppose that U has a single shock. That is, suppose U is the solution to (5) subject to (3-4) that is discontinuous only along a curve $(t,x,y) = (t, \Gamma_1(t), \Gamma_2(t))$. For small ϵ, this curve lies in the shock-layer region of the solution to the full problem. The size of this region tends to zero as $\epsilon \downarrow 0$ [10].

Equation (5) is not a good model where the convection and diffusion are both important, such as in a shock-layer. In this region, the transformation defined by the scaling

$$\xi = x/\epsilon, \qquad \eta = y/\epsilon, \qquad \tau = t/\epsilon, \tag{6}$$

is applied and computations are performed in this new stretched coordinate system. The motivation of this transformation is to capture the smallest-scale behavior of the solution to (1). It may not be necessary to scale all of the variables. For example, if the profile of the shock is slowly varying, it may be possible to drop the temporal derivatives from the shock-layer solution [6].

The internal layer is the following neighborhood of Γ:

$$D_{IL} = \{(x,y,t) | (x,y,t) \in D, [(x - \Gamma_1^{-1})^2 + (y - \Gamma_2^{-1})^2]^{1/2} \leq \Delta(t)\}. \tag{7}$$

Here $\Delta(t)$ is the width of the internal layer at time t. In one dimension, the width of the shock-layer is $O(\epsilon|\ln(\epsilon)|)$ [18]. We can expect similar results to hold in two dimensions; hence, the number of points in D_{IL} should not be too large. The outer region is the complement of D_{IL} with respect to D, that is,

$$D_{OR} = \{(x,y,t) | (x,y,t) \in D, [(x - \Gamma_1^{-1})^2 + (y - \Gamma_2^{-1})^2]^{1/2} > \Delta(t)\}. \tag{8}$$

The solution in the outer region is used to provide boundary data for the problem in the internal layer.

Asymptotics identified two subdomains and provided preconditioners for the problems within the subdomains. The preconditioner for the full equation in D_{IL} is the use of the local transformation (6). The preconditioning in the outer-region subdomain D_{OR} is to solve (5) in place of (1). Thus, the numerical method for D_{OR} may be chosen from the wide variety of methods designed for hyperbolic equations [7],[12],[21],[14]. Other asymptotic-induced preconditionings are possible. For examples of these, see [2],[8]. In the next section, the domain decomposition and preconditionings are combined with a functional iteration to form the computational method.

4. Boundary Detection. This boundary detection scheme in a neighborhood of a shock is based on the size of the second partials of the solution with respect to the spatial variables. The physical and analytic motivation were discussed in [17], and will not be discussed here.

The subdomains used in the numerical method are determined by comparing $|u_{xx}| + |u_{yy}|$ with the user-supplied quantity TOL. Heuristics, based on both accuracy and efficiency, can be used to choose TOL. Accuracy will suffer when TOL is too small. If TOL is too large, the internal-layer subdomain will be too large, and the computational mesh will be refined in regions where the solution is smooth, creating excess work.

It is important to be able to identify if the asymptotics has been done correctly, and to be able to recover from the errors. The iteration described in [16] provides just such a mechanism. The solution on the coarse mesh may be unreliable; thus, a refinement based on the coarse-mesh solution may result in errors in the location of the subdomain boundaries. The iteration coupled with the adaptive refinement would then allow the correction of the location of the refined region.

5. Computational Details. In this section the choice of the numerical schemes and some of the computer-science related issues are discussed. Both the numerical schemes and the choice of data structures allows the exploitation of parallelism.

The solution in D_{OR} is obtained using a strictly upwind explicit finite difference scheme on a tensor-product grid (e.g. equally spaced in x and y). Once a grid point has been identified as needing refinement by measuring the derivatives of the solution there, then each of the four grid rectangles adjacent to the grid point are included in the refined region surrounding the shock. These grid squares are refined based on the scaled coordinates; hence, the spacing within each of the coarse grid rectangles is $\Delta\xi = \Delta\eta = .1$. This spacing was chosen as a maximal size for accuracy. The temporal variable is also stretched, so $\Delta\tau = \epsilon\Delta t$.

The sub-problem in the internal-layer subdomain requires the solution of a parabolic PDE subject to boundary data provided by the solution in the outer region. The computational domain in D_{IL} has an irregular boundary, but is composed of many non-overlapping rectangular regions, each of which is composed of a tensor product grid. The mesh for these rectangles has been scaled; therefore, there are no large gradients in the solution on the refined mesh; hence, the computations are not sensitive to the particular difference scheme used to solve the partial differential equation. An explicit finite difference method was chosen to solve the equation. The scheme used was a combination of explicit strictly upwind discretization for the convection terms with a centered discretization for the diffusion in D_{IL}. Other methods could be employed to obtain the solution in the subdomains [4],[9].

The data structures for D_{IL} are a rectangle adjacency list plus the rectangular regions with some buffer regions. The domain changes as the solution is marched in time. Using a list of refined rectangles allows for simple creation and deletion of sections of the refined domain. Bi-linear interpolation is used to initialize refined regions. Values computed on the refined mesh are injected into the coarse mesh.

PARTI is utilized to implement the numerical method described in the previous sections. The primitives in PARTI combine the ability to control partitioning of data with the ability to reuse certain data. In addition, these primitives will support irregular distribution of data. PARTI is designed to be compiler-compatible; thus, all of the optimization techniques supported by PARTI can be implemented into a compiler.

6. Implementation. The PARTI primitives currently consist of two levels. The most fundamental of the primitives consist of routines to gather and/or scatter (read and write) values to elements of one dimensional arrays. These routines are described in [1], and will not be discussed here.

The higher level routines in PARTI are the user interface between an application code and the more fundamental primitives. These are the Level 1 PARTI Primitives (we will refer to these as PARTI or as the primitives for the remainder of the paper). PARTI allows the dynamic allocation of distributed multi-dimensional arrays and supports data transfer between these arrays. The primitives consist of declaration procedures and of communication procedures. The declaration procedures allow declaration of a dynamically allocated distributed array such that the partitioning of the array across processors is specified. Coupled to these declarations are the level gather and scatter procedures. These procedures are designed to allow users to fetch or store array elements from the distributed memory in a way that does not require the user to keep track of where array elements are stored. This makes it relatively straightforward to write codes that allow data structures to be repartitioned during program execution. Application programs will consist of

1. code written to execute on individual processors
2. communications calls that consist of gathers or scatters to distributed arrays
3. communication calls that consist of lower level 0 gathers or scatters
4. send and receive message passing calls

The Level 1 `Gather Exchanger` and `Scatter Exchanger` routines allow communication of user data based on the global index set. The Scatter Exchanger inputs lists of distributed array indices and values. It places the values in the distributed memory locations specified by the indices (and the initially supplied array mapping). The Gather Exchanger inputs lists of distributed array indices along with a pointer to a memory buffer in the calling processor. Data values from the appropriate distributed memory locations are obtained and placed in the calling processor's buffer. An initialization or Scheduler procedure call is required for Gather exchanger or Scatter exchanger. The initialization procedure precomputes the locations of the data that will to be sent and received by each processor. This initialization is needed only once–it may be reused any number of times.

6.1. Adaptive Mesh Partial Differential Equation Solver. As described already, the structure of this computation changes with time and there is a complicated (potentially non-uniform) communication pattern required by the sharing of data between grids.

The method initially computes the solution on a coarse mesh. An error estimator is then applied to determine the regions that will be covered by a refined mesh. An example mesh from this two-level refinement is shown in Figure 6.1.

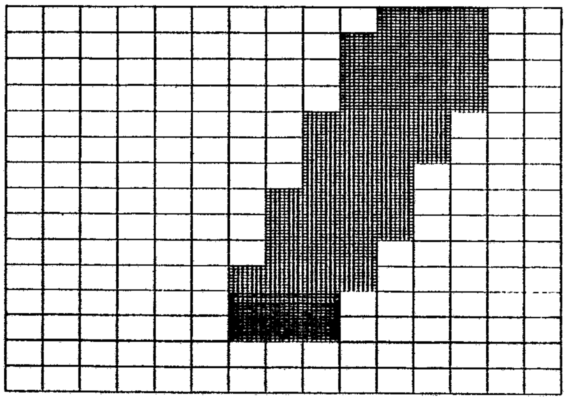

FIG. 1. *Two-mesh refinement.*

The solution is time-dependent. Time-marching on the refined mesh is performed by taking many (e.g. 100) time steps on the refined mesh for a single coarse-grid time step. Let U represent the solution, k be the temporal step-counter, and (i, j) represent the discrete location on a spatial grid. The subscripts c and r are used to refer to the coarse and refined meshes, respectively. The data structure used for the coarse mesh is a two-dimensional array. The solution on the refined mesh is represented by a three-dimensional data structure in which the third index represents a block of the refined mesh (each block corresponds to a single coarse grid square), and first two indices represent the spatial location within the block. The general structure of the kernel is outlined in Algorithm 1. In general, a shock moves and changes shape. Thus, the refined mesh will be dynamic – its location, shape, and size all change. This means that both the communication pattern within a distributed mesh and the relationship of the two meshes will change during the execution of the program. Classes of computations that require inter-processor communication in a distributed computing environment are; (1) coarse mesh sweeps (Step I.A.), (2) fine mesh sweeps (Step II.C.1.), (3) sharing of values between the coarse and fine meshes (Steps II.B. and II.D.), and (4) modification of the

```
For $k_c = 1$ to $K$
  I. Sweep over the coarse mesh
     A. Compute $U_c$.
     B. Flag region that should be refined.
  II. If flagged region is not empty.
     A. Modify shape of refined region
     B. Interpolate boundary values for $U_r$ from $U_c$.
     C. For $k_r = 1$ to $K_r$
        1. Sweep over the refined mesh
        2. Share values between blocks in $U_r$
     D. Inject values of refined region into coarse grid
```

ALGORITHM 1 *Two-mesh algorithm.*

shape of the refined region (Step II.A.). In our implementation of Algorithm 1, the PARTI primitives are used in all but the fourth set of communications.

7. Experiments. Unless otherwise noted, the experiments described in this paper used either the 32 processor iPSC-2/860 machine located at ICASE, NASA Langley Research Center or the 128 processor iPSC-2/860 machine located at Oak Ridge National Laboratories. Each processor had 8 megabytes of memory.

7.1. Primitives Benchmark Timings. We first measure the time required for the higher level Scheduler, Gather Exchanger and Scatter Exchanger procedure calls. We use the Level 1 initialization primitive to declare a 128×128 element distributed array of single precision numbers. We allocate four processors configured in a 2×2 grid G and allocate an array block to each processor.

We repeatedly exchange information between two processors in the grid by first scattering and then gathering $n \times n$ sub-blocks of array elements between opposite corners of the grid. In Table 1 we present the results of this experiment. We first

TABLE 1
Overheads for Level 0 and Level 1 Primitives

Number of Data Elements	Send Receive Time(ms)	Level 1 Gather (ratio)	Level 1 Scheduler (ratio)
100	0.5	1.2	7.0
400	1.0	1.3	9.2
900	1.8	1.5	10.7
1600	2.9	1.6	11.2
2500	4.3	1.6	11.1
3600	6.0	1.6	11.2

note that the cost of a Level 1 Gather or Scatter Exchange takes 1.6 as much time as

TABLE 2
Adaptive Mesh Solver- Timings (seconds)

Number of Processors	Total Time	Total Inspector Time	Gather/Scatter Time
	Time(ms)	(seconds)	seconds
8	794	4	65
16	417	6	40
32	248	10	27

it would take for two processors to exchange data using send/receive procedure calls. This ratio increases gradually as the size of the messages send increase; the ratio is only 1.3 when n is equal to one. The cost of performing the Level 1 Scheduler is roughly 7 times the costs of a Level 1 Gather or Scatter Exchange.

7.2. Computational Results for Domain Decomposition Algorithm. We present computational results for the parallelized domain decomposition algorithm in Table 2. This table depicts the computation time in seconds required to solve the fine-grid problem. The coarse-grid problem required no more than an additional 3% of execution time. The example problem that we ran had the tolerance for the second derivative test set to $TOL = 21$, and was run until $t = .88$, for a total of 440 coarse-grid time steps. Due to memory constraints, we we unable to run this problem on fewer than eight processors. The 16 and 32 processor runs had speedups of 1.8 and 3.0 over the 8 processor run. This is a good speedup, especially considering locality was not considered in the distribution of fine-grid work.

We also measure the total time required by all Level 1 schedule procedure calls. The scheduling took very little time; the overhead for scheduling ranged from 0.5 % of the total time to 4.0 % of the total time. Finally we measured the time required for gather/scatter procedure calls. The time required for the gather/scatter procedure calls ranged from 8 % to 11 % of the execution time.

8. Concluding Remarks. Asymptotics and numerics have been blended to form a new computational method suitable for a variety of difficult simulations arising in fluid dynamics and chemistry. The method has potential to exploit a large amount of parallelism and provides high accuracy. Asymptotic analysis provided a theoretical basis for the domain decomposition, identifying two types of subdomains: smooth outer regions, and an internal-layer subdomain with a shock. In the context of this driving problem, we present methods and tools we have developed for implementing adaptive codes on distributed memory multiprocessors.

REFERENCES

[1] H. S. Berryman, J. Saltz, and J. S. Scroggs. exectuion time support for adaptive scientific algorithms on distributed memory machines. ICASE Report 90-41, ICASE, NASA Langley Research Center, Hampton, Virginia 23665-5225, 1990.

[2] A. Bourgeat and M. Garbey. Computation of viscous (or nonviscous) conservation law by domain decomposition based on asymptotic analysis. Preprint 86, equipe d' analysie numerique, LYON-St Etienne, September 1989.

[3] J. Rozier Cannon. *The One-Dimensional Heat Equation*, volume 23. Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[4] D. B. Gannon. Self adaptive methods for parabolic partial differential equations. Technical Report PhD. Thesis, University of Illinois, Urbana-Champaign, 1980.

[5] R. C. Y. Chin, G. W. Hedstrom, J. R. McGraw, and F. A. Howes. Parallel computation of multiple-scale problems. In A. Wouk, editor, *New Computing Environments: Parallel, Vector, and Systolic*, pages 136–153. SIAM, Philadelphia, 1986.

[6] Marc Garbey and Jeffrey S. Scroggs. Asymptotic-induced method for conservation laws. In H. Kaper and M. Garbey, editors, *Proceedings for the Workshop on Asymptotic Analysis and Numerical Solution of Partial Differential Equations*, 1990.

[7] A. Harten, P. D. Lax, and B. van Leer. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, 25(1):35–61, 1983.

[8] G. W. Hedstrom and F. A. Howes. A domain decomposition method for a convection diffusion equation with turning points. In T. F. Chan, R. Glowinski, J. Periaux, and O. B. Widlund, editors, *Domain Decomposition Methods*, pages 38–46, Philadelphia, 1989. SIAM.

[9] D. Hoff and Joel Smoller. Error bounds for finite-difference approximations for a class of nonlinear parabolic systems. *Math. Comp.*, 45(171):35–49, 1985.

[10] F. A. Howes. Multi-dimensional initial-boundary value problems with strong nonlinearities. *Arch. for Rat. Mech. Anal.*, 91:153–168, 1986.

[11] P. D. Lax. *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves.* SIAM, Philadelphia, 1973.

[12] B. van Leer. On the relation between the upwind-differencing schemes of Godunov, Engquist-Osher and Roe. *SIAM J. Sci. Stat. Comp.*, 5(1):1–20, 1984.

[13] R. Mirchandaney, J. H. Saltz, R. M. Smith, D. M. Nicol, and Kay Crowley. Principles of runtime support for parallel processors. In *Proceedings of the 1988 ACM International Conference on Supercomputing , St. Malo France*, pages 140–152, July 1988.

[14] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial-Value Problems.* Interscience Publischer, John Wiley and Sons, New York, NY, 1967.

[15] J. Saltz, K. Crowley, R. Mirchandaney, and Harry Berryman. Run-time scheduling and execution of loops on message passing machines, (to appear in Journal Parallel and Distributed Computing, April 1990). Report 89-7, ICASE, January 1989.

[16] J. S. Scroggs. An iterative method for systems of nonlinear hyperbolic equations. *Computers and Mathematics with Applications*, to appear, 1989.

[17] J. S. Scroggs. A parallel algorithm for nonlinear convection-diffusion equations. In *Proceedings for SIAM Conference on Domain Decomposition.* SIAM, 1989.

[18] J. S. Scroggs. Shock-layer bounds for a singularly perturbed equation. ICASE Report 90-52, ICASE, NASA Langley Research Center, Hampton, Virginia 23665-5225, August 1990.

[19] J. S. Scroggs. A physically motivated domain decomposition for singularly perturbed equations. *SIAM Journal on Numerical Analysis*, to appear, 1990.

[20] J. S. Scroggs and D. C. Sorensen. An asymptotic induced numerical method for the convection-diffusion-reaction equation. In Julio Dias, editor, *Mathematics for Large Scale Computing*, pages 81–114. Marcel Decker, 1989.

[21] H. C. Yee and A. Harten. Implicit TVD schemes for hyperbolic conservation laws in curvilinear coordinates. *AIAA Journal*, 25(2):266–274, 1987.