A COMPUTATIONAL TOOLKIT FOR COLLIDING BLACK HOLES AND CFD*

N. P. Chrisochoides† G. C. Fox‡ T. Haupt§
Northeast Parallel Architectures Center ¶

Syracuse University, Syracuse, NY 13244-4100

Abstract

We present a framework for a high level toolkit for solving partial differential equations. The requirements for very large and complex PDE applications such as computational fluid dynamics and numerical relativity are examined in the framework of a modular toolkit approach based on visual programming. We address some of the principal non-numerical technical challenges : software integration, scheduling and distribution of the computation over a metacomputer. We also discus some of the challenges found in creating run-time support systems and parallel grid generation modules for future systems.

1 Introduction

In this paper, we discuss features of the computational solution of partial differential equations (PDEs) and their implications for the design and implementation of a toolkit supporting them on high performance computers. This article considers two areas, numerical relativity (NR) and computational fluid dynamics (CFD). These certainly do not exhibit all the features one needs to include in a computational toolkit for PDE solution. However, they are each internally rich in their computational structure but exhibit rather different needs. Thus designing a toolkit motivated by these two application areas is likely to lead to a system with broad capability. The toolkit should have an overall structure of general applicability but will probably lack specific modules needed by application areas not considered here.

Both NR and CFD are governed by a system of second order nonlinear partial differential equations. There are as usual four independent variables: space x and time t. The basic CFD equations have five dependent variables velocity field $\underline{u}$, density ϱ, and energy E. However, the number of simultaneous equations can increase in those applications such as reservoir simulation with several different constituents.

Numerical relativity, NR, is derived from Einstein's equations expressed elegantly in terms of index tensors where each index is four (space and time) dimensional. Tensors can have upto four indices. The number of distinct field variables depends on the formulation but is typically about 50! Here is a major complexity with the sheer number of dependent variables motivating a convenient symbolic front end to enable error free user friendly specification of the desired numerical formulation of the 50 differential equations.

An important feature of both NR and CFD is that the full solution procedure links several distinct sets of partial differential equations. In the case of NR, one typically first solves (at the initial time value) an elliptic differential equation set which solves the so-called constraint equations defining redundant field variables in terms of the non-redundant set chosen to represent the solution. This step depends on the so- called "gauge" used. The elliptic solution step is followed by the time evolution of hyperbolic partial differential equations. A remarkable feature of NR is the ability to make general coordinate transformations corresponding to choosing gauges and transforming between them. For one choice of gauge, the solution could be quite homogeneous and for others very irregular and adaptive. Correspondingly one can need regular meshes and finite difference methods for one gauge choice and adaptive meshes and finite element approaches for other choices. Thus, NR includes solution of multiple sets of PDE within each application and wide variety of equation characteristics between the different approaches.

One of the major uses of HPCC for CFD lies in the area of multidisciplinary analysis and design shown in Figure 1. This integration of simulation into design (and more ambitiously manufacturing, market and product support) requires the linkage of several distinct simulations e.g. those of manufacturing process, aircraft structure and radar (stealth) properties as well

*Work supported in part by NSF ASC 93 18152/ PHY 93 18152 (ARPA supplemented)

†Research Assistant Professor, Computer Science Department, and Research Scientist, NPAC, Syracuse University

‡Director; Professor, Computer Science and Physics Departments, Syracuse University.

§Research Scientist, NPAC, Syracuse University.

¶

as fluid flow around the plane. This is illustrated in Figure 2 which captures the problem which at the highest (computer science) level is the mapping of heterogeneous metaproblems onto heterogeneous metacomputers. NR and CFD are both metaproblems in that they are compound problems made up of several different linked modules. Correspondingly the target machine is a metacomputer made of several high performance computers of different architectures. A given problem module is most effectively implemented on particular computer architectures. Thus the mapping problem of Figure 2 involves first expressing the modules in scalable languages which will execute on today's and future machines. Then the modules need to be linked with a suitable coordination or integration language (we have most experience with AVS) and using performance estimates, each module is to mapped to part or all of a single or multiple components of the metacomputer.

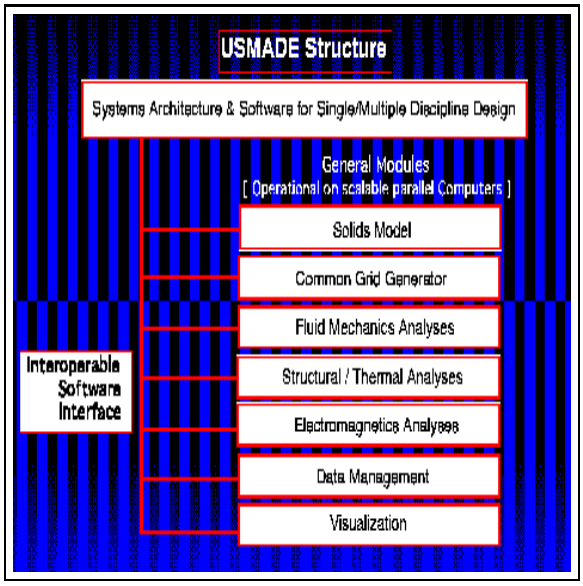

Figure 1: Multidisciplinary analysis and design.

A critical parameter in CFD is the visycosity η which is a multiplicative coefficient of the highest order derivative terms in the Navier Stokes equations typically these involve $\underline{u}$ where $\underline{u}$ is the vector valued velocity field. η is normally expressed in dimensionless form as the Reynold's number R proportional to $1/\eta$. The particularly interesting and challenging case of large Reynold's number corresponds to motion through air and other gases. Large R implies that there is a structure over distances of order $R^{-3/4}$ which can easily be a factor 10^{-3} of other dimensions in system. This rapidly varying turbulent behavior occurs at fluid/vehicle interfaces and can "separate" and travel into the fluid volume. This physical structure has important computational implications. In particular, one is essentially forced to use adaptive finite element methods to account for varying structure over differing length scales.

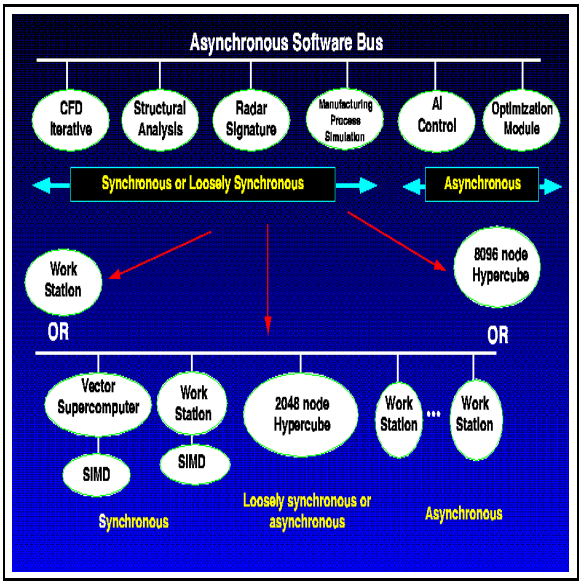

Figure 2: Mapping of heterogeneous metaproblems onto heterogeneous metacomputer systems.

In NR, one does not find the low dimension singularities such as shocks characteristic of CFD at large Reynold's number. The terms with highest order derivations are not multiplied by an unusually small coefficient. Rather, singularities are volume based. Thus, in NR, finite difference methods can be used with adaptive (multi grid) methods needed to describe areas where the fields are rapidly varying.

Both NR and CFD have challenging boundary value issues which would be modules in our toolkit. In CFD, one would need to match structural and fluid flow meshes and solutions at a vehicle surface. Further at large R, boundary layers are formed of thickness $R^{-1/2}$ over which (roughly) the tangential components of fluid velocity go from asymptotic (large) values outside the layer to zero (in frame where vehicle is at rest) at the vehicle surface. NR has key boundary conditions at the "event horizons" which surround each black hole. Information inside the event horizon is unphysical as information about it can never get out. The rectangular finite difference grid needs special treatment at the event horizons. NR also has critical boundary conditions at "infinity" for the result of the computation is the gravitational waves received at earth, i.e. traveling to infinity. Accurate "wave extraction" is a critical NR module. Note that as usual, treatment of boundaries involves careful interplay between physical and computational insight.

The multidisciplinary analysis embodiment of CFD, also implicitly involves wave equations in the electromagnetic simulation module. Currently, this module is usually implemented with the method of moments rather than discretizing Maxwell's equations on a grid.

This involves quite different computational issues (currently solution of 50,000 x 50,000 full matrix equations) from the sparse matrix systems found in the discretized grid approach to the same (wave) equations.

2 System Overview

As illustrated in Figure 2 we can naturally break problems and computer systems into modules and pose the general computational challenge as the mapping of metaproblems onto metacomputers. Computer science must provide a programming environment that allows :

A. User specification of the problem decomposition into modules and dynamic linkage,

B. High level languages to express individual modules,

C. Runtime support of the efficient execution of modules on appropriate components of the metacomputer,

D. Generic modules which can be used across many applications.

We first, illustrate this modular metacomputer framework with the example of a simple solver for an elliptic problem based on one of the first successful PDE toolkits, ELLPACK, developed at Purdue [27] and [8]. A high level description of a program that solves equation (1) with boundary conditions given by (2) is presented in Figure 3. It can be logically divided into three major phases of computation, namely the *man-machine interface*, corresponding to the problem definition, (ii) the *solution system*, corresponding to the creation and solution of the discrete problem that approximates the continuous definition of the PDE problem generated by the man-machine interface, and finally (iii) the *post-processing system*, corresponding to the visualization of the solution and performance of solution methods.

Elliptic Problem
Compute a function $u(x, y)$ that satisfy the PDE :

$$\frac{\vartheta u}{\vartheta^2 x} + \frac{\vartheta u}{\vartheta^2 y} + \frac{\vartheta u}{\vartheta x} - 4u = e^{x+y} sin(\pi x) \qquad (1)$$

in the unit square [0,1]×[0,1] and satisfying the boundary conditions :

$$\begin{array}{lll} \mathrm{u} = 0 & \mathrm{x} = 0\ ,\ 0 \le \mathrm{y} \le 1 \\ \mathrm{u} = sin(\pi x) - \frac{x}{2} & \mathrm{y} = 1\ ,\ 0 \le \mathrm{x} \le 1 \\ \mathrm{u} = \frac{y}{2} & \mathrm{x} = 1\ ,\ 0 \le \mathrm{y} \le 1 \\ \mathrm{u} = \mathrm{x} & \mathrm{y} = 1\ ,\ 0 \le \mathrm{x} \le 1 \end{array} \qquad (2)$$

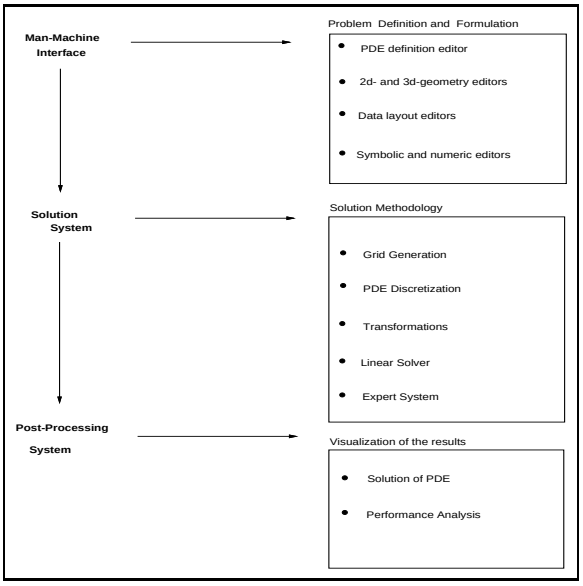

Figure 3: High level description of a PDE Toolkit.

Figure 4 illustrates an existing implementation of that idea, an ELLPACK program [27]. Phase I, "man-machine interface" is realized by modules such as EQUATION and BOUNDARY that define the elliptic problem and its boundary conditions. Phase II, "solution system" comprises the GRID module that defines the grid that tessellates the unit square, the DISCRETIZATION and SOLUTION segments that define the solution method, and actually make the computation. Finally, at the phase III, the OUTPUT module defines the way the solution $u = u(x, y)$ is going to be presented. For time dependent problems or systems of PDEs the ELLPACK description can be more complex and less intuitive.

Using the ELLPACK approach one can describe in generic form an organization of PDE modules as shown in the diagram of Figure 5. In this way (also illustrated in Figures 1 and 2) we indeed divide PDEs into modules which can each be mapped into parts of individual or combinations of computers linked by high speed networks.

In fact ELLPACK is an example of a toolkit that addresses all the issues A) and D) defined above. We need to extend their ideas to include the more general differential equations needed by CFD and NR; to support adaptive three dimensional grids shared by different solution stages, and to allow flexible mapping of PDE-metaproblems onto metacomputers - these span a single sequential computer, a homogeneous SIMD/MIMD parallel machine; a network of workstations and a general collection of modern architectures as shown in Figure 2.

```
OPTIONS.   TIME $    MEMORY

EQUATIONS.  UXX + UYY + 3.0 * UX - 4.0 *U = EXP(X+Y) *SIN(PI*X)

BOUNDARY.      U = 0                    ON  X  = 0.0
               U = SIN(PI*X)-X/2.0      ON   Y = 1.0
               U = Y/2.0                ON   X = 1.0
               U = X                    ON   Y = 0.0

GRID.          100   X POINTS
               100   Y POINTS

DISCRETIZATION.   5  POINT STAR

SOLUTION.         LINPACK  BAND

OUTPUT.           TABLE (U)
                  PLOT (U)

END.
```

Figure 4: ELLPACK program for the elliptic PDE problem.

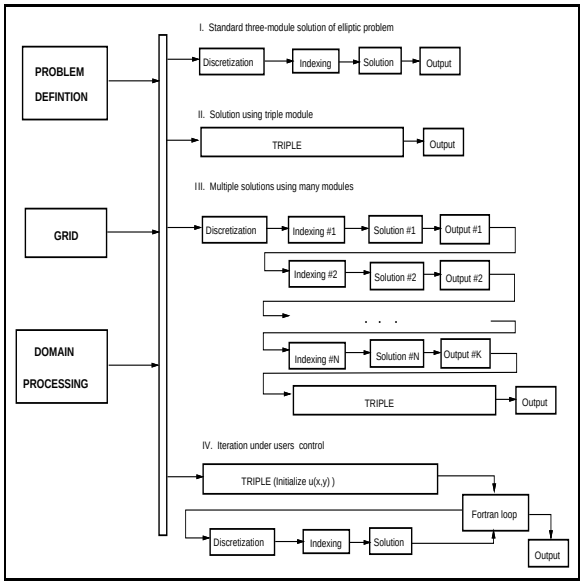

Figure 5: Organization of an ELLyPACK computation. The user specifies the modules to be used and more than one combination may be used on a single ELLPACK execution [27].

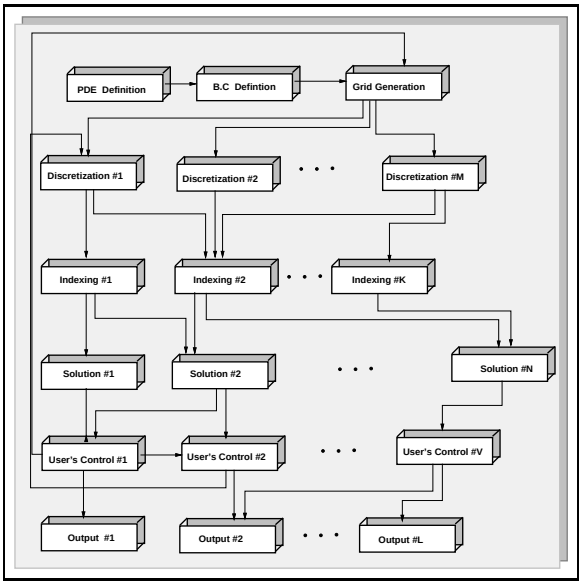

Figure 6: Organization of a Visual program using a network editor [1]. The user specifies the modules to be used and the component of the metacomputer to be executed. The final result is a PDE network of modules.

We address here computer science issues underlying a toolkit like ELLPACK [27], to specify and execute the modules. Here we will not address the full specification of the needed toolkit but rather illustrate the key features. These are divided into the areas A, B, C, and D defined at the start of this section and discussed briefly in sections 3, 4, 5 and 6 respectively.

Any programming environment must make explicit or implicit trade offs between user convenience (productivity) efficiency and portability. Extreme approaches would be represented by a) fully optimized machine language for a given machine, b) visual BASIC or some other high level object oriented system.

Approach a) is non portable as it runs on only one machine; it is highly efficient on that machine; is highly time consuming for user. Approach b) is highly convenient for user and portable but currently object oriented systems are not able to generate efficient code for metacomputers. The new high performance HPC++ system [18], [3] will address portable parallelism for C++ users. Nevertheless C++ will always be less efficient than less capable but easier to compile languages such as Fortran.

We will adopt an intermediate approach which stresses portability but is neither the most productive environment for users and will not generate the most efficient execution. User productivity is hard to quantify but we aim at producing execution times which of a factor 2 (typically) or 4 (at worst) lower than optimal implementations.

The use of true and de facto standards is essential both for portability and reasonable user productivity. Relevant HPCC software standards include AVS (to control and specify modules - see Section 3), High Performance Fortran [17] (HPF-the language for individual modules), Message Passing Interface [26] (MPI - efficient and portable runtime support - see Section 4). We also, advocate using portable HPCC libraries such as LAPACK to provide reasonable efficient runtime AVS modules for matrix algebra (Section 5). We have successfully used AVS and LAPACK for electromagnetic simulations (Section 3).

Note that current MPP vendors do not have the resources to develop their own parallel programming environments. Thus our standards are the basis of the best MPP programming environments which can make use of the standards to implement better support for debuggers, performance visualization, compilers and runtime systems.

3 Software Integration of the components of a metaproblem

The software integration problem is exemplified by Figures 1, 2, 5 and 6 where modules need to be linked together to solve the full problem. This linkage can be static as the problem of Figure 7 where matrix element generation tasks feed data (the calculated elements) on fixed links to matrix solve and visualization modules. In multi-disciplinary analysis and design (Figure 2) the links are dynamic, where for instance, information flows back and forth and forth between structural and fluid flow modules under control of the optimizer.

Generally Integration Software must support coarse grain task parallelism where each task or module is :

- Itself a complex structure implemented in a parallel language such as HPF, HPC++ (Section 4) or message passing (PVM, MPI, etc.)
- linked either statically or dynamically to other modules.

A visual interface is attractive for this problem as typically one does not have a large number of modules. These correspond to the finite tasks defining the problem - in contrast the data parallelism, measured for instance by the number of grid points, can be arbitrarily large in our problem class. Thus graphical specification of the integration problem is quite natural even though it may not be appropriate for data parallelism.

In Figures 7 and 8 we illustrate our very successful work using AVS for software integration with the examples of computational electromagnetics and data assimilation (weather forecast optimally combining model and data) Similar software environments are illustrated by CODE [2], and HENCE [19]. In this class of programming environments, one clearly separates the paradigm used for integration (e.g. AVS) with the language used for each module (e.g. HPF, Fortran plus PVM or MPI calls). Alternatively one can link the full support of tasks and data parallelism into a single system - this approach is illustrated by HPC++ and the experimental linkage of Fortran-M with Fortran 90D [33].

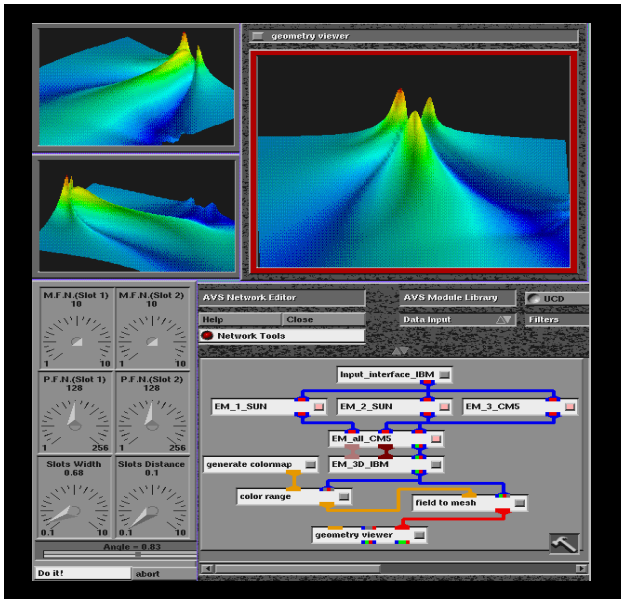

Figure 7: AVS as integrating software to facilitate both networking and scientific visualization for electromagnetic scattering [5].

Currently no system offers all the features one needs. For instance AVS can support parallel or sequential modules [4] but this is not part of the basic design and requires additional system support. Further AVS currently does not support parallel I/O between modules. For instance AVS will support task communication over a single (ATM) high speed link. However suppose one ahs say, one module using 8 nodes on an IBM SP-2 connected to another such 8 node module on the same machine. Currently AVS cannot use the high speed parallel network on SP-2 for inter-module I/O. One will use this network for intra-module communication. In many cases, intermodule communication does not need high bandwidth and current version of AVS is sufficient. AVS also does not elegantly support dynamic links although these can be implemented clamsily.

AVS was originally designed for visualization applications and for this reason is available on a broad range of platforms. As well as the many workstations, we have used AVS to support integration with modules

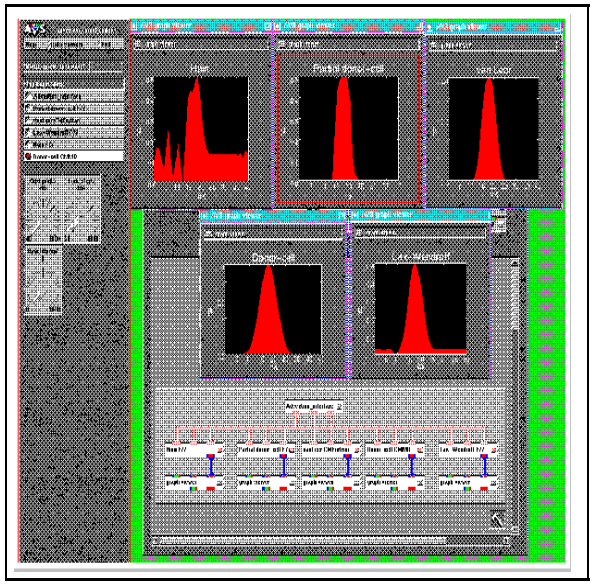

Figure 8: AVS as the visualization for a four-dimensional data assimilation project [4].

on the TMC CM-5, Maspar MP-1, IBM SP-1 and DEC alpha-farm. Further one can of course easily include the visualization module as part of the full system as these are a full set of AVS support modules for visualization.

4 Exploiting Parallelism With Modules

4.1 Programming Paradigms

Here we focus on the computationally intense modules for which parallel computing is required. There are three natural parallel programming paradigms one could use for individual modules. Each has its advantages and disadvantages and one would mix them in a given problem with different modules using different programming paradigms. This complex software environment requires a carefully crafted runtime environment to support multiple paradigms used at the same time.

Figures 3 and 4 illustrate that ELLPACK supports symbolic analysis so that the differential equations can be expressed in "mathematics" - or more precisely in domain (here the PDE solution domain) specific language. SINAPSE is a new interesting system of this type build on top of Mathematica. The NCSA group is evaluating its use for the Black Hole problem.

The second major paradigm is data parallel languages where High Performance Fortran (HPF) is an agreed standard and many commercial compilers are being build. The Convex and Portland Group compilers are directly based on the work Rice [22] and Syracuse [33]. HPF is described below but we emphasize that its parallel extensions are largely language independent. The corresponding parallel extensions of C++, HPC++ is based directly on those in Fortran. We are constructing with ARPA support a runtime system that will support multiple languages (C++, Fortran, ADA).

The final lowest practical level, is Fortran(or C) plus message passing. Here PVM has become a de facto public domain standard. We expect PVM to continue, but a new more formal standard MPI to become an important portable message passing platform.

Practically HPF or Fortran + MPI are good standards which can be recommended for new applications for the next few years. In our application fields, HPF looks fully suitable for NR which is based on rectangular grids. However CFD needs HPF extensions to handle irregular unstructured adaptive meshes. These extensions have been prototyped using work of Saltz's group [29]. We can expect them to be incorporated in future compilers.

4.2 High Performance Fortran

HPF is an extension of Fortran 90 to support data parallel programming model, defined as single threaded, global name space, loosely synchronous parallel computation. The idea behind HPF is to provide means to produce scalable, portable, and top performance codes for MIMD and SIMD computers with non-uniform memory access cost. The portability of the HPF codes means that the efficiency of the code is preserved for different machines with comparable number of processors.

The HPF extensions to the Fortran 90 standard fall into three categories: compiler directives, new language features, and new library routines. The HPF compiler directives are structured comments that suggest implementation strategies or assert fact about a program to the compiler. They may affect the efficiency of the computation performed, but they do not change the value computed by the program. The most important assertions suggest the data decomposition needed to get good performance.

The new language features are FORALL statement and construct as well as minor modifications and additions to the library of Fortran 90 intrinsic functions. In addition, HPF introduces new functions that may be used to express parallelism, like new array reduction functions, array combining scatter functions, arrays suffix and prefix functions, array sorting functions and others. These functions are collected in a separate library,

the HPF library. Since it was anticipated that not all algorithms can be easily expressed in HPF syntax, an escape mechanism, the extrinsic functions, has been introduced. The extrinsic functions may be written in languages other than HPF and may support a different computational model. Finally, HPF imposes some restrictions to Fortran 90 definition of storage and sequence associations. A lot of the excellent performance of HPF comes from the explicitly parallel runtime library called from the data parallel level. This library is typically written an optimal (lower level) message passing such as Fortran + MPI.

The HPF approach is based on two key observations. First, the overall efficiency of the program can be increased, if many operations are performed concurrently by different processors, and secondly, the efficiency of a single processor is likely be the highest, if the processor performs computations on data elements stored in its local memory. Therefore, the HPF extensions provide means for explicit expression of parallelism and data mapping. It follows that an HPF programmer expresses parallelism explicitly, and the data distribution is tuned accordingly to control the load balance and minimize communication. On the other hand, given a data distribution, an HPF compiler may be able to identify operations that can be executed concurrently, and thus generate even more efficient code.

5 Runtime support

Parallel computing and more specifically high performance software for scientific computing will be made more attractive to scientists and engineers if these systems will provide unified runtime support for all aspects of a physical simulation : PDE discretization and indexing, grid generation, adaptive refinement and load balancing, PDE solution, parallel I/O and visualization. While a fair amount of work has been carried out in building libraries for the individual components - mainly for field solvers - limited or none effort made towards the integration of the individual libraries. In this section we address the issues related to the development of a common runtime support system for both task and data parallelism specific to PDE applications.

A typical example of PDE solution involves a mixture of data and task parallelism. For instance, Figure 6 indicates that a discretization module can be linked with a number of different indexing modules (i.e., different methods in re-ordering the rows and columns of the linear systems of equations that approximates the continuous PDE) and different solution modules. In this case one can explore task parallelism in the level of the PDE network of the modules and data parallelism in the level of individual modules. Unfortunately there is no single language available with sufficient power and flexibility to express both data and task parallelism in coarse and/or fine grain levels. Thus the application scientist or engineer stands alone in the archipelagos of many specialized languages and libraries - with little or none common interfaces - that have been developed for different computational paradigms and purposes.

Examples of such languages and libraries are HPF [17] for data parallelism, HPC++ [18] for task parallelism, PVM [20] and MPI [26] for message passing, CMSSL [24] and SCALAPACK [15] for linear algebra, PITPACK [7] and PIM [28] for iterative solvers, P++ [25], DIME [34], and MENUS-PGG [9] for parallel grids, DecTool [8] and Chaco [21] for automatic partitioning and load balancing.

These systems are supported by lower level operating system capabilities - especially message passing. Currently PVM and in the future MPI will be critical standards on which to build coexisting tools. We also need runtime communication support for special data structures such as those provided in PARTI [31]. In several approaches a good light weight thread runtime library is critical is used in several methods of local balancing and many shared memory programming environments.

The objective of the runtime support system for PDE toolkits is to collect most of these libraries and provide a complete integration framework which is portable across all HPCC platforms with : (a) common interface for high level machine independent debuggers and performance analysis tools, (b) scalable and efficient data movement from module to module of the PDE network, (c) parallel I/O from logical component to logical component of the metacomputer, (d) scalable partitioning and load balancing capabilities, and finally (e) a facility for generating knowledge base rules from a database of performance data.

There is no single debugging or performance analysis tool that provides all needed functionality to debug and optimize all kinds of software. Existing debuggers and analysis tools can be used for individual modules, but can not be used in analyzing a complex PDE network of modules. Typical available software for debugging and performance systems have been developed for very fine grain level analysis; for example in the case of debugging they can trace the value of a variable or pointer within the same address space. They did not meant to trace problems over a PDE network of modules that are distributed over a LAN or WAN. Moreover in PDE software systems a problem might appear far a way form the module on which the problem has been generated. For example a problem in grid generation might show up in the visualization of the solution where a singularity or jump appears in a place where

the solution is supposed to be smooth. Our effort will focus in developing high level visualization debugging and analysis systems custom made for PDE networks of modules.

In Section 3 we mentioned a limitation of AVS in taking advantage of high speed networks to handle parallel data movement between modules that assigned on the same component of the metacomputer. The runtime support system will address some special cases of this problem that occur in PDE computations. A more general instance of this problem is the case where different modules are placed on different components of the metacomputer. The distribution of modules over powerful different supercomputers is meaningless if the support for parallel I/O is absent. For example consider a parallel adaptive solver that runs on nCUBE II and a parallel grid generation that runs over an SP-2; we have seen in [8] that the sequential loading of millions of grid points from SP-2 to nCUBE II through a workstation (host) is very costly. According to Amdahl's law this limitation will deteriorate the overall speed up of the system.

Scalable dynamic load balancing is another import difficult optimization problem whose solution is essential for the efficient execution of parallel PDE solvers. There are however many effective static and dynamic data distribution and re-distribution methods which are developed [16]. So we do not need to develop fresh approaches. Rather our objective is to unify the existing libraries of algorithms and provide common interfaces for discretization modules based on finite-difference, finite-element, finite-volume and spectral methods. Finally, the PDE toolkit will be a "self-learning" system. A knowledge base [23] of performance data will be created for each run and configuration of the pair (PDE-problem, PDE-network). This knowledge will be encapsulated in the form of rules and will be used to solve the following problem : for a given pair (PDE-problem, computing environment) find the "closests" muching point in the the knowledge space.

We have already mentioned that some PDE solvers use matrix solvers - in particular the method of moments for computational electromagnetics : such approaches would use matrix library runtime support such as the portable LAPACK system.

6 Parallel Grid Generation

In previous sections we described the non-numerical technical challenges one phases in designing and implementing high level PDE solving systems for complex applications such as NR and CFD. In this section we address the problems related to the parallel grid generation on one or multiple components of the metacomputer. In addition, we present our progress in this direction as well as preliminary results that justify the effectiveness of our approach.

We present a brief overview of our effort in developing parallel grid generation modules for static and adaptive structured and unstructured grids [9]. The parallel modules for structured and unstructured grids (see Figures 10, 11 and 12) include : (1) a module for parallel 2D and 3D static and adaptive Algebraic and Elliptic grids for general domains, (2) a module for parallel 2D and 3D multi-level multi-component curvilinear grid refinement for general domains and (3) a parallel 2-dimensional adaptive Delaunay triangulation .

The fact that curvilinear grids (see Figure 9) can be considered to be logically (i.e., computationally) rectangular is very attractive to explore data locality and parallelism using data parallel languages like FORTRAN 90D or FORTRAN 90 plus message passing. The data-parallel grid generation method [9] for a given domain Ω is described by the following five steps : (1) Decompose the physical domain, Ω into a small number of contiguous hexahedron subregions, Ω_i, that can be mapped into rectangular computational blocks B_i which form an initial composite block structure $C_0(\Omega) = \{B_i\}_{i=1}^{N}$ [32]. In this step we also assign the size of the grid that we want to generate on each of the blocks B_i. (2) Generate sequentially a coarse algebraic grid that provides an explicit control of the physical grid shape with a minimal number of grid points. This grid is used as a background for the partitioning of the finer composite block structure, $C_f(\Omega) = \{B_i^{'}\}_{i=1}^{N^{'}}$ that satisfies the following three properties : (i) $\mid C_0(\Omega) \mid < \mid C_f(\Omega) \mid$, (ii) $\forall B_i \in C_0(\Omega)\ \exists I_i \subset \aleph\ \ni: B_i = \cup_{j \in I_i} B_j^{'},\ B_j^{'} \in C_f(\Omega)$, and (iii) $||B_i^{'}| - |B_j^{'}|| < T_A\ \forall B_i^{'}, B_j^{'} \in C_f(\Omega)$, where T_A is the workload machine-dependent tolerance. (3) Map the composite block structure $C_f(\Omega)$ onto distributed memory MIMD machine so that the workload of the processors is balanced and the required communication and synchronization among the processors is minimum. (4) Generate in parallel a finer algebraic grid G_A, using $C_f(\Omega)$, that provides an explicit control of the physical grid spacing that is required by the application. (5) Generate in parallel the final Elliptic grid using the G_A grid as an initial solution for the Elliptic solver.

The data-mapping problem represented in step 3 is formulated on the computational space by a rectilinear graph $G(V, E)$. The vertices V of the graph G correspond to the blocks $B_i^{'} \in C_f(\Omega)$ and the edges E indicate the connectivity of the blocks $B_i^{'}$ with their neighbor blocks. On each vertex $v_i \in V$ of the graph G we assign various attributes like Cartesian coordinates of the mass center of the block $B_i^{'}$ and curvilinear co-

ordinates that correspond on the six nodes of the computational block $B_i^{'}$. On each edge $e_i \in E$ we assign weights that reflect the number of surface grid points of the block $B_i^{'}$.

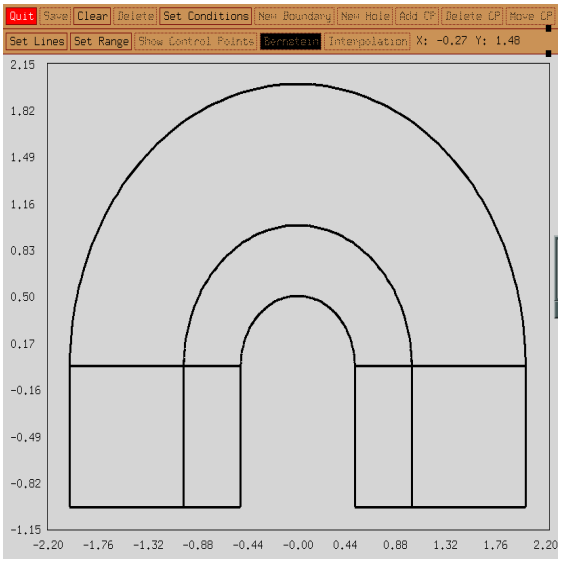

Figure 9: 2D-geometry editor.

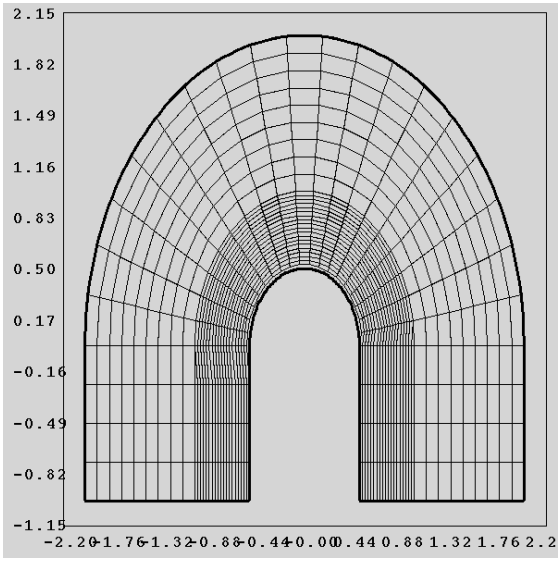

Figure 10: Curvilinear grids.

Tables 1 and 2 indicate that our approach for the solution of the data-mapping problem reduces the employment of sequential data pre-processing required for the data-parallel PDE solvers and at the same time exploits the reusability of existing well written and tested sequential structured and unstructured multi-block methods for parallel CFD codes.

A challenge for parallel multi-block multi-zone solvers is to distribute the grids and thus the computation in such a way so that the inter-block and intra-

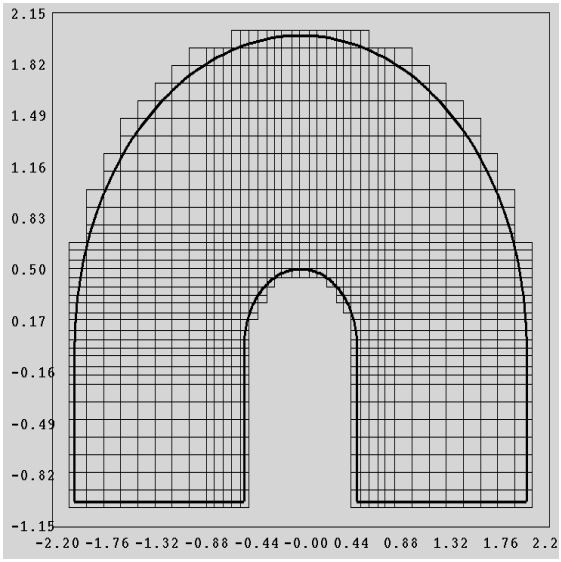

Figure 11: Non-uniform orthogonal grids.

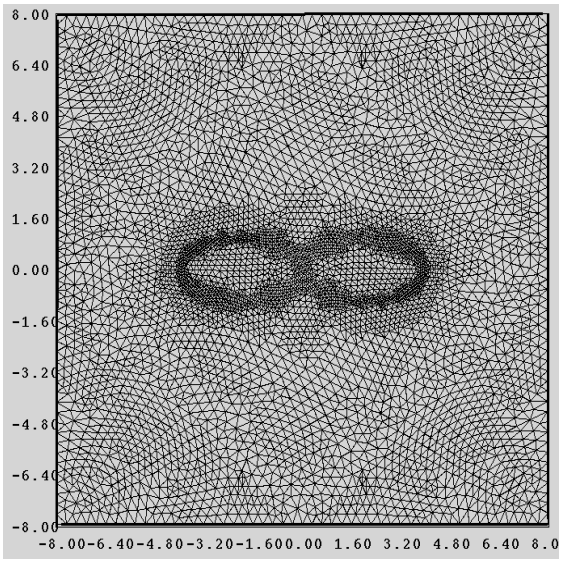

Figure 12: Unstructured adaptive grids.

Table 1: The time (in sec) required to generate (Grid-Gen.) structured grids on a SPARC workstation, the time to partition (P×Q) [3] (Mapping) the subgrids onto the memory of the 64 processors of the nCUBE 2 and the sum (Total) of the grid generation, partition and storing times

Grid Points	Grid-Gen.	Mapping	Total
2.5×10^3	0.38	17.81	18.19
10×10^3	1.61	46.45	48.06
22.5×10^3	3.61	100.15	103.76
40×10^3	6.45	195.13	201.58

Table 2: The time (in sec) to generate $C_f(\Omega)$, map and store the data on the memory of the 64 processors of the nCUBE 2(Pre-Proc.) and the time (in sec) to generate on a 64 node nCUBE 2 an algebraic grid using $C_f(\Omega)$.

Grid Points	Pre-Proc.	(//) Par. Grid-Gen.	Total
2.5×10^3	3.44	0.08	3.52
10×10^3	4.38	0.35	4.73
22.5×10^3	8.48	0.71	9.19
40×10^3	16.06	1.25	17.31

block communication are minimized. In [10] present two ab-initio load balancing methods for multi-block multi-zone solvers that are based on composite block structures. These methods eliminate the inter-block communication that keep intra-block communication to a minimum.

Parallel adaptive Delaunay triangulations are very important for computations with strong variations in the solution over scattered regions (eg. in CFD flow over a multi-element airfoil wing) and computations with moving boundary whose parametric or exact representation is unknown (eg. in NR collision of two black holes). The challenge here is the dynamic load balancing of parallel a computations with irregular dependencies. We address this problem with two different approaches. The first [11] is based on an extension of the load balancing method presented in [6] that minimizes local synchronization and reduces data migration by taking into account the data distribution before the adaptation of the grid. The approach provides data partitions of the same quality as Recursive Spectral Bisection presented in [30]. The second approach is based on based on a priority based multilist multithread system [12] and [13]. The most important advantages of the proposed approach are: (1) minimization of processors idle time, because idle processors immediately schedule threads created on busy processors without waiting to reach a global barrier; and (2) reduction of the overhead to recognize the processors state, because we check only the counter of the "ready" and "wait" queues of some small subsets of processors.

7 Conclusions

Essential features of a toolkit are proper break up of the problem into modules linked by open interfaces and well defined requirements that will enable commercial or academic sources to develop independently tools and implementations for each module. It is unrealistic for a single vendor to supply all the software for a future agile manufacturing system or a single university to solve for ever the dynamics of black holes. Rather virtual organizations must be formed with each member contributing part of the system. Here we have analyzed a few of the issues underlying a toolkit for PDE solutions. This is a rich problem with both individual modules and their integration very challenging to implement. However the toolkit approach is the only practical one and hope to refine our ideas in future research and papers. Comments and collaborators will be welcome.

Acknowledgements

This work was support by Black Hole Binaries: Coalescence and Gravitational Radiation NSF Grant #ASC-9318152 and CRPC. Chrisochoides is also supported by the Alex G. Nason Prize Award.

References

[1] Advanced visual Systems Inc., AVS 4.0 Developer's Guide and User's Guide, 1992. *Journal of Computational Physics*, Vol. 53 pp 484-512, 1984.

[2] Browne, J., M. Asam, and S. Sobek, CODE a unified approach to parallel programming, IEEE Software, 6(4):10-18, July 1989.

[3] Chandy, M. and C. Kesselman, Compositional parallel programming in CC++, Technical Report, Caltech, 1992.

[4] Cheng, G., G. C. Fox, G. C. and K. Mills, Integrating Multiple Programming Paradigms on Connection Machine CM5 in a Dataflow-based Software Environment, Technical Report, Syracuse Center for Computational Science, August, 1993.

[5] Cheng, G., Y. Lu, G. Fox, K. Mills and T. Haupt, An Interactive Remote Visualization Environment for an Electromagnetic Scattering Simulation on a High Performance Computing System, Proceedings of Supercomputing '93, Portland, Oregon, November 15-19, 1993.

[6] Chrisochoides, N., C. Houstis, and E. Houstis, Geometry based mapping strategies for PDE computation. In E. N. Houstis and D. Gannon, editors, *Proceedings of International Conference on Supercomputing*, pages 115-127. ACM Press,1991.

[7] Chrisochoides, N., E. Houstis, S. Kim, M. Samartzis, and J. Rice, Parallel iterative methods. In *Advances in Computer Methods for Partial Differential Equations VII*, (R. Vichnevetsky. D. Knight and G. Richter, eds) IMACS, New Brunswick, NJ, pages 134-141, 1992.

[8] Chrisochoides, N., E. Houstis and J. Rice, Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers, *Special Issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming,* Vol. 21, No 1, pp 75–95, 1993.

[9] Chrisochoides, N., G. Fox and J. Thompson, MENUS-PGG : Mapping Environment for Numerical Unstructured & Structured - Parallel Grid Generation. In the Proceedings of the *Seventh International Conference on Domain Decomposition Methods in Scientific and engineering computing,* 1993.

[10] Chrisochoides, N. and G. Fox, Data layout methods for parallel multi-block multi-zone solvers. (In preparation, to be submitted in Frontiers'95).

[11] Chrisochoides, N. and G. Fox, Dynamic load balancing based on a generalized spectral bisection. (In preparation, to be submitted in 9th International Parallel Processing Symposium).

[12] Chrisochoides, N., Multithread approach for parallel adaptive Delaunay triangulations, Unpublished manuscript submitted to ICASE/NASA.

[13] Chrisochoides, N., Multithread PDE solving systems for distributed address space parallel machines. To appear in the Proceedings of the IMACS World Congress on Computational and Applied Mathematics, Atlanta, July 11-15, 1994.

[14] Chrisochoides, N., An Alternative to Data Mapping for Parallel PDE Solvers : Parallel Grid Generation, *Proceedings of Scalable Parallel Libraries Conference*, pp 36–44, October, 1993.

[15] Choi, J., J. Dongarra, D. Walker, R. Whaley Scalapack Reference manual, Version 1.0 Beta, 1993 ORNL/TM-12470. *In Numerical Grid Generation* J.F. Thompson, ed. Elsevier Science Publishing, Inc. 317-338, 1982.

[16] Fox, G., R. Williams and P. Messina *Parallel Computing Works!* Morgan Kaufmann Publishers, Inc. San Francisco, California, 1994.

[17] High Performance Fortran Forum, High Performance Fortran Language Specification, Scientific Programming, vol.2 no.1, July 1993. Also available by anonymous ftp from ftp.npac.syr.edu.

[18] Gannon, D. S. Yang, and P. Beckman. User Guide for a portable Parallel C++ Programming System pC++. Department of Computer Science and CICA, Indiana University, January 1994.

[19] Belguelin, A., J. Dongarra, Geist, A., R. Manchek, K. Moore HeNCE:A Heterogeneous Networking Computing Environment, cs-93-205, Computer Science Department, University of Tennessee at Knoxville, 1993.

[20] Geist, A., A., Belguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam PVM 3 User's guide and reference Manual ORNL/TM-12187.

[21] Hendrickson B. and R. Leland The Chaco User's Guide, Version 1.0 SAND93-2339.

[22] Hiranandani, S., K., Kennedy, C., Tseng, Compiler optimization for Fortran D and MIMD distributed-memory machines, *Proc. Supercomputing'91, Nov., 1991.*

[23] Houstis, E., C. Houstis, M. Katzouraki, T. Papatheodorou, J. Rice and P. Varodoglu, Athena: A Knowledge based system for //Ellpack. CSD-TR-950, 1990.

[24] Johnsosn, L. CMSSL : A Scalable scientific software library, *Proceedings of Scalable Parallel Libraries Conference*, pp 57–66, October, 1993.

[25] Lemke M., D., Quinlan P++, A Parallel C++ array class library for architecture-independent development of structured grid applications. ACM SIGPLAN Notices, 28(1):pp 21-23, September 1992.

[26] MPI Forum, Message-Passing Interface Standard, April 15, 1993.

[27] Rice, J. and R. Boisvert, Solving Elliptic Problems Using ELLPACK, Springer–Verlag, New York, 1985.

[28] Rudnei Dias da Cunha and Tom Hopkins, *PIM 1.0, The parallel iterative methods package for systems of linear equations: User's guide,* Universidade Federal do Rio Grande do Sul, Brasil, 1994.

[29] Ponnusamy, R., J., Saltz, A., Choudhary, Y., Hwang, G., Fox Runtime Support and Compilation Methods for User-Specified Data Distributions, Submitted to IEEE Transactions on Parallel and Distributed Systems in Nov 1993.

[30] Simon, H., Partitioning of Unstructured Problems for Parallel Processing, RNR-91-008, NASA Ames Research Center, Moffet Field, CA, 94035.

[31] Sussman, A., J. Saltz, R. Das, S. Gupta. D. Mavriplis and R. Ponnusamy. PARTI Primitives for Unstructured and Block Structured Problems, Published in Computing Systems in Engineering in 1992.

[32] Thompson, J., Z. U. A. Warsi and C. Wayne Mastin, *Numerical Grid Generation.* North-Holland, New York, 1985.

[33] Bozkus, Z. A. Choudhary, G. Fox, T. Haupt, S. Ranka and M. Wu Compiling Fortran 90D/HPF for Distributed Memory MIN Computers, *Journal oc Parallel and Distributed Computing*, 21, 15-26, 1994.

[34] Williams, R. DIME: A programming environment for unstructured triangular meshes on a distributed-memory parallel processor, Caltech Report, C3P-502.