Parallel Structured Grid Generation

Director : Geoffrey Fox
Project Leader : Nikos Chrisochoides
Research Assistants : Animesh Chatterjee and Rajani Vaidyanathan

Northeast Parallel Architectures Center
Syracuse University, 111, College Place
Syracuse, NY, 13244-4100

Contents

1 Abstract

The purpose of this project is to design and implement parallel algorithms and software modules for structured static and adaptive grids that are necessary for the scalability of the existing parallel Partial Differential Equation (PDE) solvers. In this report we present our progress on modules for parallel algebraic and elliptic grids, a parallel multi-block Euler discretization module, and parallel iterative solvers. Also we define the interfaces between : (1) parallel grid generation modules and multi-block Euler module, and (2) multi-block Euler module with parallel iterative solvers.

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 re-usability of existing well written and tested sequential structured multi-block methods for parallel CFD codes. Preliminary data indicate that this approach is ten times faster than the fastest traditional data-mapping method, for relatively small problems (i.e., tens of thousands of grid points) and approximately P times faster for very large problems (i.e., millions of grid points) that are processed on coarse-grain distributed address space MIMD machines with P processors.

The development phase of the parallel Algebraic and Elliptic grid generation modules, multi-block Euler discretization module and interface with Parallel Iterative Methods should be completed by the end of May. The interface of grid modules with NGP's[1] and EagleView's geometry modeler, and data mapper [8], as well as the extensive evaluation to various parallel platforms, and documentation will be completed by the end of this year.

2 Introduction

Numerical grid generation methods are used extensively for numerical solution of Partial Differential Equations on arbitrarily shaped regions. This procedure which is mainly used in computational fluid dynamics can also be applied to all physical problems that involve field solutions. The grid generation method here frees the computational simulation from any restriction to boundary shapes and allows general implementations with the boundary specified by the input, [18], [19], [20]. The computation is done on a fixed square grid in the computational space, which is rectangular by construction. Even though the correspondence between any physical region is a single rectangular block for three-dimensional configurations, the grid is likely to be skewed. Hence, the physical region is segmented into contiguous regions each of which transforms into a rectangular block in the computational region. The grid is generated within each subregion, [19], [21].

[1]National Grid Project at MSU

The generation procedures for curvilinear grids involve two general types : the numerical solution of partial differential equations and construction of algebraic interpolation. The algebraic generation system is faster, but the grid generated from solution of partial differential equations are smoother even though they are computationally intensive. The partial differential system may be elliptic, parabolic or hyperbolic. The hyperbolic and parabolic generation systems are faster than elliptic systems, but are more limited in the configurations that can be solved.

The partial differential system used here is elliptic. This system is most generally applicable with complicated boundary and is solved as a composite grid. For each block an independent curvilinear coordinate system is to be generated. The degree of continuity of grid lines across the interfaces of adjacent curvilinear coordinate systems requires either the specification of grid points at the same fixed locations on both the adjacent coordinate systems or the treatment of grid lines as a branch cut on which the generation system is solved just as it is in the interior of the blocks. Thus the interface locations are not fixed and are determined by the grid generation system, [21].

Since grids in the blocks cannot be generated totally independent, but only as an entire composite grid, an extra layer of points surrounding each block is provided. Here the grid points on an interface of one block is coincident in the physical space with those on another interface of the same or another block. Also, the grid points on the surrounding layer outside the first interface are coincident with those just inside the second interface and vice versa. This coincidence is maintained during the course of an iterative solution of an elliptic generation system by setting the values on the surrounding layers equal to those at the corresponding interior points after each iteration. Hence all the blocks are iterated to converge together.

The programming model for loosely synchronous computations is the Single-Program-Multiple-Data(SPMD) model, where parallelism is achieved by partitioning the geometric data of the partial differential equations and allocate the disjoint subproblems to the processors. Thus during each iteration, each processor performs three basic tasks:

- *Local communication*,
- *Computation (solution of partial differential equations)*, and
- *Global Communication.*

The local communication involves exchanging the inner layer of the adjacent blocks, to preserve the coincidence of the interface layers of each block. The computation involves solving the partial differential equations and updating the interior points of the grid, from the outer layer passed from the neighboring block. Finally, the global synchronization consists of reduction operations that are required for the convergence. The global synchronization also ensures the updating of interior points of each block before the next iteration, when these layers are communicated to the adjacent block.

In this implementation, we have focused on minimizing communication time, by overlapping communication with computation apart from using efficient implementations for massage passing . The other significant improvement has been with the reduction of the number of copy functions invoked at every iteration, which required changing the data structure of the blocks. The parallel implementation of the grid generation has been done using the p4 (*Portable Programs for parallel Processors*) environment. p4 is a subset of Message Passing Interface which allows the same code to be used on multiple machines with minimal changes. The current implementation can be run in IBM RS6000 and TMC CM-5 with changes in the processor group file, which configures the machine and spawns tasks to the processors. Section 3 describes the grid data structure, its mapping, and implementation in the p4 parallel programming environment. Communication, data dependencies, and the computation properties of the important components of a typical CFD code are analyzed and details associated with developing the code for DM-MIMD machines are presented in section 4. References are cited for more details wherever felt necessary. We conclude with some observations.

2.1 Current Status

The parallel grid generation system involves the following developments,

1. A C driver which would call on already tested fortran routines for computations, when required.
2. An efficient data structure for the grid, and incorporate this in the grid generation system to avoid multiple copying of data
3. A parallel implementation of this with communication provided for composite grid generation of generic structures.
4. Port this parallel implementation to other parallel machines
5. Exhaustive testing and performance analysis

A C driver, given in Item 1, has been debugged for a small data set and this needs to be expanded for a large data set which would include imperfections and hence require smoothing. The C data structure, Item 2 has been incorporated. Basic communication routines have been incorporated and this has to be made generic to accommodate generic partitions of complex structures, as required in item 3. Item 4 is trivial, since, communication is done over the p4 layer and hence porting requires very little changes in code. Item 5, exhaustive testing and performance analysis is yet to be done.

Parallel implementation of the multi-block CFD code required work in the following areas:

1. Developing a pre-processor for decomposing grid generated sequentially [17][2]

[2]The parallel grid generation module is at present limited in application to relatively simple geometries.

2. Developing a pre-processor to identify neighbouring elements and nodes in a composite grid with an arbitrary number of blocks for input to the data mapping program [8]
3. Code modifications to run on Sun workstations in sequential mode
4. Dynamic memory allocation to incorporate arbitrary sized blocks
5. C-driver program for efficient data communication between subdomains and between blocks
6. Incorporating message passing using MPI for the discretization module
7. Developing efficient parallel Matrix-vector product routines using MPI for the linear system generated in the discretization module [3]
8. Interfacing the discretization and PIM (Parallel Iterative Methods) solver modules
9. Porting code on a SP1, the TMC CM-5 and the Intel Paragon parallel platforms

A pre-processor has been developed for decomposing a simple C-grid around an isolated airfoil. The pre-processor in Item 2. has been developed for the same geometry. Further work is required to generalize these programs to more complex geometries (Two element airfoil, a SCRAMJET engine section and the Engine-airframe interaction problem in two dimensions). The goals under Items 3 and 4 have been achieved. Some modifications are underway in the C-driver program to incorporate MPI for data communication between processors. Current work is also focused on implementing message passing in the discretization module and developing an efficient parallel Matrix-vector product algorithm using MPI.

3 Parallel Grid Generation

The design challenge for large-scale multiprocessors is (i) to minimize communication overhead, and (ii) allow communication to overlap computation. Communication and computation overlap requires support of asynchronous communication. Blocking send/receive communication exacerbates the effects of network latency on communication latency. Also with synchronous communication it is not possible to overlap communication and computation. With asynchronous communication, tolerating latency becomes a programming problem, a communication must be initiated sufficiently in advance of the use of the result.

The computation involved in grid generation for each block is independent of the other blocks excepting when the interface layer is to be updated after every iteration. The local communication involves exchange of the interface layers (surface) between

[3]Sparse block banded matrix-vector product routines [10]

processors of the parallel machine, between neighboring sub-domain. The traversal required for the exchange depends on the efficiency of the partition and allocation phase. This local communication time for this implementation of parallel grid generation is minimized first by reducing the copying of buffers internally (by p4) and externally. The time spent in actual communication just involves the time to send the buffer and the communicator returns back to continue with the computation. Sufficient computation time is allowed before the receive to avoid a block during receive.

The computation is staggered into three cycles in this code, to avoid bottle necks caused when more than one message is received by each processor. This is also used to preserve the order in which the messages are to be received. Thus in each cycle the processors will utmost receive or send only one message. The global synchronization consists of reduction operations that are required for checking the convergence criteria through out the composite block. This global synchronization also ensures the surface received do not lag in iteration values owing to unbalanced load in certain processors. Time to send includes time for packing and moving the message. Receive is blocking in most parallel machines and this is circumvented by using message available testing primitives like *mesg_test* or polling mechanisms. The communication is done through p4 (*Portable Programs for Parallel Processors*), [4], [5]. We chose p4 because of its impending compatibility to the Message Passing Interface Standards.

The packing time for messages in p4 is reduced by, avoiding overheads in internal formatted copying of the message buffer to be sent. This is done by allocating a pre-formatted memory to the buffer using the p4 function, *p4_msg_alloc*. Asynchronous message passing is used, the send and receive functions are locally blocking, that is they are blocking until the receive or send is completed. p4 does direct process-to-process communication as opposed to messages through daemons, which translates to efficient communication.

The Parallel grid generation system involves the following basic tasks: (i) Local communication, (ii) Computation (solution of partial differential equations), and (iii) Global Communication. Each of this is discussed in detailed in the following sub-sections.

3.1 Data Partitioning and Allocation

Data mapping is described in earlier work done by Chrisochoides [8],[9]. The optimization problem here, is split into two distinct phases corresponding to the *partition* and *allocation* of the grid. In the partition phase, the grid is decomposed into a specific number (usually equal to or multiples of the number of processors) of sub-domains or substructures with the following objectives: (i) the maximum difference in the number of active grid points of the sub-domain is minimum, (ii) the ratio of the number of interface points to the number of active interior points of the sub-domain is minimum, (iii) the number of adjacent sub-domains is minimal, (iv) each sub-domain is a connected domain. The allocation phase allocates the sub-domain to the processors, so that: geometric neighbor sub-domain are allocated to neighbor processors in the interconnection network of the parallel machine, [9].

3.2 Computation

In this section we discuss the grid structure, computational aspect of the grid generating system. The construction of computational fluid dynamics codes for complicated regions is greatly simplified by a composite block, the computations are done in the rectangular computational regions, [19], [20]. The entire three- dimensional physical region is filled with a set of interfacing hexahedrons with curved surfaces each of which corresponds to a rectangular computational block. Each of these blocks has its own set of right-handed curvilinear coordinates (independent of those in other blocks),[21]. Each computational block is surrounded by an extra layer of points, as can be seen from the data structure described later, in order to allow connections across the interfaces in the physical region to be formed, (Shown in figure 1). The interfaces between blocks are branch cuts, and the code establishes a correspondence across the interface using the surrounding layer of points outside the blocks. This allows points in the interface to be treated just as all other points, so that there is no loss of continuity.

The elliptical grid generation system is based on a system of Poisson-like equations,

$$\nabla_2 \, \xi_i = P_i \qquad \text{i=1,2,3}$$

A three dimensional elliptic grid generation system is constructed as a linear combination of the nine equations obtained by writing the basic three-dimensional elliptic system given above three times, in each case dropping the derivative with respect to one curvilinear coordinate. All control functions are dropped except for one of the diagonal elements. Thus, automatic evaluation of control functions are done from the initial algebraic grid (generated by transfinite interpolation) and then smooth it.The control functions can also be determined automatically to provide orthogonality at boundaries with specified normal spacing. Here, the iterative adjustments in the control functions are made by increments radiated from the boundary points where orthogonality has not been attained, [19]. The boundary orthogonality can be achieved through Neumann boundary conditions, which allow the boundary points to move over a surface spline, the boundary point locations being located by Newton iteration on the spline to be at the foot of normals to the adjacent field points.

3.2.1 Data Structures

The data structure used here has two main roles, the first is to keep track of the neighboring blocks and the processors they are residing in. The second comes from using separate pointers for the outer layers. This is done to exploit the C features for efficient copy. The main advantage of this data structure comes during message passing, where the inner layer is passed as it is without any internal (user to p4 memory) or external copying. Each block stores its size and hence the blocks do not have to be of a constant size, [3], [16], [22]. The blocks are characterized by the following pieces of information,

- Block id : the unique numerical id of this block.
- The number of points in the blocks in x, y, and z directions.

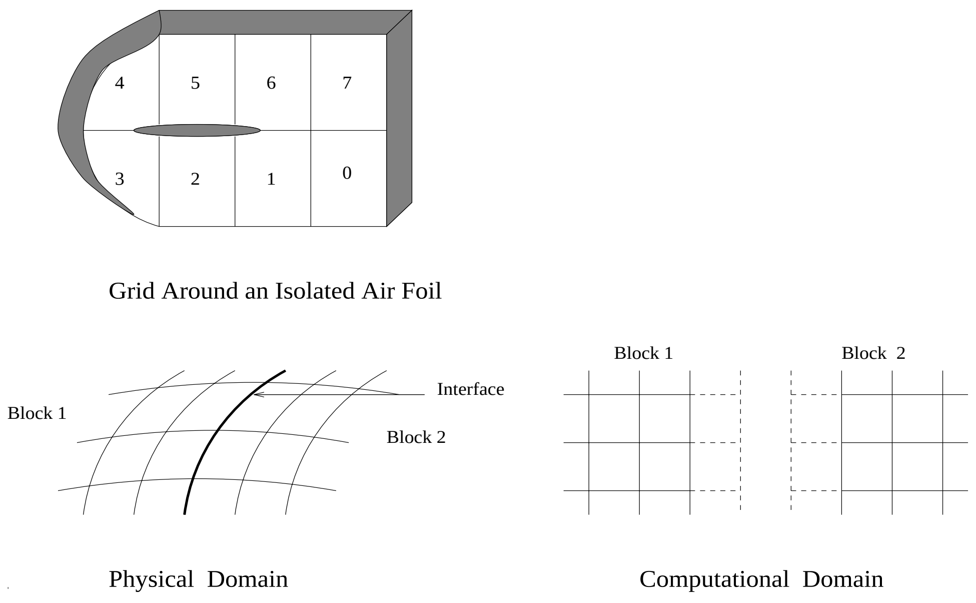

FIGURE 1:

- Pointers to surfaces x, y, and z.
- Pointer to an array of neighbor blocks, maximum six (3D).
- Pointer to an array of processor id's which hold the blocks.
- Total number of neighbors.
- Pointers to the outer top, bottom, right, left, front, and back surfaces in x, y, and z directions each.
- Pointers to the inner top, bottom, right, left, front, and back surfaces in x, y, and z directions each.

3.3 Message Passing

Message Passing Interface (MPI) defines a set of library interface standards for message passing in the various parallel machines, [4]. The goal of this interface, is to develop a widely used standard for writing message-passing programs. As such the interface establishes a practical, portable, efficient, and flexible standard for message passing.

Message passing is a paradigm used widely in parallel machines with distributed memory, [5]. The parallel grid generator is iterative and hence communication, both local and global are of much importance, for the overall efficiency of the implementation. Here the higher level routine and abstractions are built on lower level message passing routines. The standard is particularly beneficial, specially when it provides hardware support and thereby enhance scalability: global reduction operations. The iterations are to be checked for global convergence since this grid generation method is composite. This interface also facilitates quick implementation on different platforms with no significant changes in the underlying communication and system software.

The target machines for this interface are distributed memory multiprocessors, network of workstations, and combinations of all these. Shared memory implementations are also available. MPI provides with reliable message transmission. Check, for transmission errors and time outs to other error conditions, is not required. But MPI does not provide mechanisms for dealing with failures in the communication system or node failures. Resource errors may occur when a program exceeds amount of available system resources. This restriction of system resources, with respect to buffers should be taken care of since the messages sent and received are 3-dimensional surfaces and the data set is large. This is prevented from happening by splitting the send and receive, into three phases.

Some of the main features of MPI include mechanisms for : dynamically creating sets of locally named processes, and partitioning communication space, [4]. The group of dynamically created processes, is an ordered set, where the processes are identified by their ranks when communication occurs. Thus the logical name of the processor starts from zero to the net number of processors. The mechanism of partitioning space, *context* is an opaque object with additional attributes for inter-communication. For intra-communication, a context is a hyper-tag needed to make a communication safe for point-to-point and MPI defined collective communication. This hyper-tag also allows multiple messages to the same processor and ensure correct placement in the receive buffers. MPI communicators provide specific scope for the communications, it brings together group and context. Thus a communicator restricts the *spatialscope* of communication, and provides local process addressing. MPI also provides a *caching* facility that allows an application to attach attributes to context, group and communicator descriptors. Thus caching is the process which propagates implementation-defined data (and virtual topology data) in groups and communicators.

The basic point to point communication requires an envelope which specifies the message destination and message to be packed in a buffer. Order of messages is preserved, within each context: if two messages are sent successively from the same source to the same destination, the message is received in the order they are sent, though this is not true with multi thread executions. In this implementation, the outer surfaces include points for x, y and z coordinates. These points for each surface is sent together and thus preservation of the order is important. Data type matching at both send and receive ends is important. The communication environment used in this portable implementation is p4 (*Portable Programs for parallel Processors*).

3.3.1 P4 : Portable Programs for Parallel Processors

The Message passing interface used in this parallel grid generation package is P4(*Portable Programs for Parallel Processors*), [10][11]. P4 is a library of routines designed to express a wide variety of parallel algorithms portably, efficiently and simply. It is currently portable to most parallel machines. IBM RS6000, TMC CM-5, IBM SP-1 (TCP/Ethernet, TCP/switch, EUI, and EUI-H), Ncube, Intel Paragon, Intel Touchstone Delta, to name a few. Although P4 tries to be completely portable, there are a small number of specific exceptions that has to be taken into account on a given machine. This allows comparison of different machine performances for this application. Some of the salient features of P4 include:

- Support for both message passing and explicit shared memory operations.
- xdr support for heterogeneous networks.
- automatic or user control of message-passing / buffer-management.
- An optional P4 server for quick startup on remote machines.

Efficiency requires the use of models of computations relatively close to those provided by the machines themselves and their system software. This reduces overheads caused by this extra layer of interpretation. Simplicity requires providing the programmers with a relatively small number of concepts, while providing a rich enough set to express the algorithms. These two are not consistent, and such inconsistencies are resolved by providing multiple ways of doing things. P4 provides completely automatic buffer management, but if the programmer prefers to deal with it, to avoid overhead of extra copy operation, p4 offers appropriate buffer management routines: *p4_msg_alloc*. This buffer management routine is used to allocate memory to the pointer which holds the inner layer of the interface used in communication: $itopx, itopy, itopz$, the top surface. This buffer is passed directly from processor to processor. Hence this buffer is used in both the send and receive end, and care is taken to ensure no overwriting of yet-to-be-used data is done.

Portability requires use of widely accepted models of computations, rather than specific implementations of those models. Considerable complexity has been absorbed into P4 itself in order to provide simplicity and portability. For the shared-memory MIMD model, it provides the monitor paradigm. for Distributed memory MIMD model, message passing functions and global operations are provided on all platforms which support this model. Explicit management of clusters for both shared- and distributed-memory MIMD models are implemented.

The precise configuration of processes (the machines they will run on and the executable) is specified in a $processgroupfile$. the name of the process group file is typically a command line argument or can be provided in a file with an extension *.pg*. This is the only change that has to be made to run the code provided in IBM RS6000 and CM5 and SUN clusters.

4 Application : Parallel Multi-block Euler Module

In this section some of the issues relating to parallel implementation of an implicit multi-block Euler flow solver on message passing MIMD architectures are discussed. Implicit schemes, due to their better stability properties, are usually the preferred numerical scheme. Such schemes however involve the simultaneous solution of large systems of coupled algebraic equations which inherently resist parallelism. Two common forms of parallelism are spatial and functional. In spatial parallelism, each subdomain interacts weakly with its neighbors, and can be assigned to a different processor for updating. Parallelism in the multi-block Euler solver is spatial and is obtained from domain decomposition. All blocks can be updated concurrently with exchange of boundary data at each time step. Typically for CFD applications, a significant part of the computation time is spent in setting up the stiffness matrix, constructing the RHS and evaluating boundary conditions. Efficient implementation of the discretization process along with the solution process is therefore necessary to obtain high performance from parallel implementation of the flow code.

Here, a version of REDCOON [1] is used as a model CFD application. In this code the 2-D Euler equations in general body fitted curvilinear coordinates are solved using a finite volume model. An implicit first or second order temporal and second order upwind spatial discretization of the resulting system is used. The left hand side of the system resulting from a time linearization of the flux terms is constructed using flux vector splitting. The fluxes through cell faces are evaluated using a high resolution approximate Riemann solver. A two pass scheme is used to solve the triangular systems resulting from an approximate factorization of the left hand side. Boundary conditions are implemented through phantom points using characteristic variable boundary conditions.

The original sequential code is written to solve the three dimensional Euler equations in general time dependent curvilinear coordinates. The total memory requirement of the code is about 220 times the total grid size. Further, at each time step $5 \times$ *imax-1* $\times$ *jmax-1* $\times$ *kmax-1* coupled equations are solved to obtain the solution vector for that time step. Here, *imax, jmax, kmax* are the maximum grid indices along the three coordinate directions. Further, significant computations on each cell are involved to set up the stiffness matrix, construct the RHS and implement BCs. For many real world applications, grid sizes can be very large ($\sim$ 500 000 points in 3D). For such applications both memory and computational demands require the use of supercomputers. The sequential code is therefore written for the CRAY line of supercomputers and makes use of some intrinsic vector functions and the extended memory on these machines. Substantial modifications had to be made to the code to implement it on a Sun workstation for the two dimensional case being studied here. Further modifications were made to make it easier to implement the code in parallel using message passing. These changes are detailed in later sections.

4.1 Numerical Algorithm

The governing equations of fluid motion are the Navier-Stokes equations; the physical laws that constitute these equations are (i)Conservation of mass (ii) Conservation of momentum and (iii) Conservation of energy. Mathematically, these equations represent a nonlinear second order hyperbolic-parabolic system of partial differential equations (PDEs). Under the assumption of inviscid flow these equations reduce to the Euler equations which now describe a nonlinear first order hyperbolic system of PDEs. This simplification of the governing equations by one order leads to substantial savings in computational expense. For many applications viscous effects are negligible, for such cases the Euler equations form a good model to study the flow phenomena.

The 2D Euler equations are solved in general time dependent curvilinear coordinates using a finite volume approach. The governing equations in the transformed coordinate system are presented in APPENDIX-A. The discretized form of the original PDEs is given in APPENDIX-B. The RHS of the equations is formulated using Roe's flux difference scheme. The development of the flux terms using this approach and certain features of the computation of these terms are detailed in APPENDIX-C. A time linearization of the original finite volume equation yields flux Jacobians that appear in the LHS of the linearized equations. Computation highlights in the evaluation of these terms is given in APPENDIX-D. The numerical scheme used to solve the linear algebraic system that arises from the discretization process is a two-step forward and backward sweep through the computational domain. The equations that arise using this method are presented in APPENDIX-E. Boundary conditions that arise in a multi-block Euler solver are outlined in the following section.

4.1.1 Implementation of Boundary Conditions

- The three different types of boundary conditions are (i)Solid boundaries (ii)Farfield boundaries(inflow/outflow) and (iii)Internal boundaries(block interfaces).
- Boundary conditions computed only at boundary cells.
- Boundary conditions for (i) and (ii) implemented by updating variables in phantom cells[3].
- Boundary conditions of type (iii) involve communication of q^n and dq^n values from 1 or 2 cell deep layer in adjacent block.

4.2 Parallel Implementation

The computations associated with data parallel PDE solvers that preserve the ordering of the corresponding sequential computations is loosely synchronous [13]. The programming model for loosely synchronous computations is SPMD (single program multiple data) where parallelism is achieved by partitioning the underlying geometric data of the PDE problem and allocating the smaller subsets of data to the processors. Efficient parallel implementation requires extraction of maximum parallelism with negligible communication overhead. This requires partitioning of the data across processors such that the load is evenly balanced and communication overhead is kept to a minimum. The data mapping algorithm used here minimizes the inter subdomain communication while maintaining an even load balance between processors.

New data structures are introduced at subdomain interfaces to obtain more efficient communication of boundary data between adjacent subdomains. Overlapping computation with communication is imperative to achieve high performance from a parallel implementation of the flow code. Asynchronous message passing is used to achieve this overlap; computation is buffered between the points in the execution where a send is initiated and the message is received, i.e. computation is overlapped with communication. The P4(*Portable Programs for Parallel Processors*)environment is used as the message passing interface to ensure portability across different platforms. In the following sections, some details on the parallel implementation of the CFD code are presented.

4.2.1 Code Modifications

As mentioned earlier the original Fortran sequential code is written for the CRAY vector supercomputers. For efficient implementation the code uses the CRAY extended memory - SSD (Secondary Storage Devices). Further, all data structures are declared global through the use of the COMMON directive. The program was originally developed for use with equal sized blocks. This imposes constraints on the range of geometries that can be modeled using this software. Modifications were therefore necessary to both implement the code on a Sun workstation and to develop a parallel version that would have the additional flexibility to handle arbitrary sized blocks.

For this purpose a C-driver program was written and the Fortran routines are called from within this program. Dynamic memory allocation for the arrays is used in the driver program to allow for variable sized blocks. All variables, block dimensions and pointers to arrays are passed as arguments to subroutine calls. The arrays are dimensioned within the subroutine; all arrays are therefore now local and are passed wherever necessary to other modules called from within the Fortran routines.

Application of the code will be initially performed on an isolated airfoil, followed by more complex geometries involving a 2-element airfoil configuration, a SCRAMJET engine section and the airframe-propulsion interference geometry in two dimensions.

4.2.2 Data Partitioning

The methodology used for solving the discretized PDEs in parallel on DM-MIMD machines is based on decomposing the computational domain into non-overlapping subdomains. This decomposition is carried out such that local communication between subdomains is minimized and an even load balance is maintained between processors. The PDEs described on each subdomain are now solved with continuity of the solution across subdomain interfaces ensured by communicating boundary data from adjacent subdomains.

The data partitioning heuristics can be based either on the algebraic data structures (coefficient matrix describing the *discrete Euler operator*) or on geometric data structures (geometric domain on which the Euler equation is defined) [8]. Heuristics based on the former approach require : (a) the sequential generation of the computational grid (b) the sequential discretization of the Euler operator (c) the data mapping of a large set of data (d) the loading of large amounts of data, (e) and finally the parallel solution of large linear system. In the geometric data partitioning approach the corresponding steps are (a) the data mapping of smaller sets of data (blocks) (b) the loading of smaller amounts of data (c) the parallel generation of the computational grid (d) the parallel discretization and (e) parallel solution to the linear system. [7]

The advantage offered by the algebraic data partitioning method is that the linear system can be generated sequentially and therefore requires no extra work in this direction. However, for large problems (three-dimensional viscous solutions) memory constraints can be inhibitive and this approach does not solve this problem. Further, it offers only a limited amount of freedom in data mapping methods, namely (i) row/column partitioning and (ii) cyclic block (i.e., block round robin) technique. For adaptive methods scalability cannot be achieved with this approach. The geometric data mapping approach offers several advantages over the algebraic data partitioning method; memory requirements can be largely cut down by suitable decomposition of the domain. It also offers greater flexibility in mapping. For applications requiring updating of the mesh during the solution process (as in adaptive methods) this approach offers an advantage since the grid is generated in parallel and can therefore lead to some savings in total time required for the solution. It is also easier to optimize certain criteria related to communication costs using this approach [8]. Further, scalable adaptive methods can be developed with this technique.

Here the partitioning approach used is geometric data partitioning; the algorithm used in this approach is described in [8]. For the case of structured grids, however, any geometric partitioning of the grid can be mapped into the algebra. A software has been developed for the case of a C-grid around an isolated airfoil that decomposes the domain into an arbitrary number of blocks along the ξ- axis. The code then generates appropriate boundary conditions on the subdomain boundaries based on the boundary conditions on the original grid defined over the complete domain. For each grid element and node, neighboring nodes and elements are identified. This data is then used by the partitioning algorithm to generate an optimal partitioning of the domain.

4.2.3 Data Structures

Data dependencies in evaluating the RHS are different than those that arise during the solution to the system of equations. Constructing higher order flux terms at subdomain interfaces requires q^n data from the two neighboring cells in the adjacent block (refer to section on evaluation of R^n). Solution to the linear system requires dq^n data from neighboring cells in adjacent blocks(refer to section on solution scheme). Here, q^n is the flow variable vector at time step n and dq^n is change in q^n between time step n and $n+1$.

The high performance of the flow solver on message passing MIMD machines depends on the minimization of the local and global communication time. New data structures are therefore introduced at subdomain interfaces to obtain more efficient communication of boundary data between adjacent subdomains. The original data structure for q^n was split into one for the interior region of the subdomain and two smaller data structures 4 cells deep along the ξ and the η-coordinate directions for the boundary regions. The data structure for dq^n was similarly decomposed into one for the interior cells and two smaller data structures 2 cells deep along the ξ and the η-coordinate directions for the boundary regions. The interface data can now be communicated to neighboring processors without any copy at the sending node. The pointer to the interface data structure is passed to the processor holding the adjacent subdomain. At the receiving node, use is made of the efficient C block copy function to load the data into the interface buffer for that subdomain.

4.2.4 Communication

The basic steps of the parallel multi-block Euler solver at each time step can be expressed as :

```
for n=1, maxblock
    1. Parallel discretization
      1.1 Local communication
      1.2 Block Interface Continuity /* Local Communication (inter-subdomain)*/
    2. Parallel iterative solution to the system of equations :
      for i=1, maxiter
        2.1 Matrix-vector product /* Local Communication (inter-subdomain) */
        2.2 Vector-vector product /* Global Communication */
        2.3 Convergence tests /* Global Communication */
      end for
    3. Block Interface Continuity /* Local Communication (inter-block) */
end for
```

The linear system is generated in the discretization module; an efficient parallel implementation of both the discretization and the solver modules is required. This is because the coefficient matrix, the residual vector and boundary conditions are updated during the solution process and hence a new system is solved at every time step. (refer to sections on evaluation of R^n, evaluation of flux jacobians and solution scheme). Communication of data across block interface boundaries is required during both the discretization and the solver steps. As previously mentioned in preceding sections, in the discretization process communication of q^n is required to construct the fluxes at block boundaries and to compute the flux jacobians at these faces (refer to earlier sections on evaluation of R^n and evaluation of flux jacobians). In the solver module communication of dq^n is required since the computational stencil at the boundary involves cells on the boundary layer of the adjacent block.

With the new interface data structures local communication now involves sending and receiving the interface data structures for q^n and dq^n. Overlap of communication with computation is obtained by buffering computation between the point where a send is initiated and data is received. Asynchronous send and receive is used to allow for computations to overlap communication. The points in the algorithm where data is sent and received is shown more clearly in the following section.

4.3 Computation Algorithm

The basic steps in the solution algorithm are as follows:

```
1. Read input parameters

  • Startup solution process

for n=1, maxblock
    2. Allocate memory for each array
    2. Read grid for subdomain into processor
    3. Set initial conditions for $q^n$
    4. Send interface $q^n$ data to neighboring processors
    5. Compute grid metrics
    6. Implement boundary conditions
end for

  • Enter time iteration loop

for time=1, maxtime
  for n=1, maxblock
    7. Receive interface $q^n$ data from neighboring processors
    8. Construct the residual vector $R^n_{i,j}$
      6.1 Compute the eigensystem for $\overline{A}_{i,j}$ and $\overline{B}_{i,j}$
      6.2 Evaluate the fluxes $F_{i\pm 1/2,j}$, $G_{i,j\pm 1/2}$ and store in $dq^n$
    9. Send interface $dq^n$ data to neighboring processors
    10. Compute the time step $\overline{\Delta\tau}_{i,j}$ using local time stepping
    10. Convergence test. Calculate the l2-norm of the residual vector $R^n_{i,j}$
    11. Evaluate the Flux jacobians $A^{\pm}_{i,j}$, $B^{\pm}_{i,j}$ and $C^{\pm}_{i,j}$
    12. Construct $D_{i,j}$
    13. Receive interface layer $dq^n$ data from neighboring processors
    14. Solve the triangular linear systems for the forward and backward passes
    15. Update $q^n$
    16. Send interface $q^n$ data to neighboring processors
    17. Update boundary conditions.
  end for
end for
```

4.4 Parallel Iterative Solver

A significant amount of computation time in a typical CFD code is spent in the solver module. An efficient implementation of these routines is therefore imperative to achieve good performance from the parallel code. The PIM (Parallel Iterative Methods) routines were surveyed for a suitable solver. Routines based on the CG(Conjugate Gradient) method are not suitable here since they require that the coefficient matrix be positive definite. This condition is not satisfied by the coefficient matrix in a CFD application. The PIM routine GMRES is therefore chosen as the solver for the linear systems for the forward and backward substitution steps given. Details about this method are presented in a later section.

PIM offers the iterative methods, *Conjugate-Gradients (CG), Bi-Conjugate-Gradients (Bi_CG), Generalized minimal residual (GMRES), Generalized conjugate residual (GCR)*, to name a few.

PIM was developed with two main goals: (i) To allow user complete freedom with respect to matrix storage, access and partitioning. (ii) To achieve portability across a variety of parallel architectures and programming environments.

These are achieved by hiding from the PIM routines the specific details concerning the computation of the following three linear algebra operations, that is, these routines are provided by the user,

1. Matrix-vector (and transport-matrix-vector) product.
2. pre-conditioning step.
3. Inner-products and vector norm.

The solution scheme adopted for the discretized PDE leads to sparse banded block triangular systems for the forward and backward substitution steps. An important feature of the coefficient matrix in typical computational fluid dynamics (CFD) applications is that it is nonsymmetric. The generalized minimal residue method has been chosen due to its property of being a very robust method to solve nonsymmetric systems. The following sections deal with, details about the Generalized minimal residue method, the structure of the matrix, the enumeration sequence used for the equations, the matrix structure resulting from it and the partitioning of the matrix resulting from the partitioning in the geometry.

4.4.1 Generalized Minimal Residue

This method is used to solve a non-singular system of n linear equations of the form,

$$Q_1 A Q_2 x = Q_1 b$$

where Q_1 and Q_2 are the preconditioning matrices.

The Generalized Minimal Residue method is very robust for non-symmetric systems. The method uses Arnoldi process to compute an orthonormal basis $v_1, v_2, ..., v_k$ of the Krylov subspace $K(A, v_1)$. The solution of the system is taken as $x_0 + V_k y_K$ where V_k is a matrix whose columns are the orthonormal vectors v_i, and y_k is the solution of the least squares problem $H_k y_k = \|r_0\|_2 e_1$, where the upper Hessenberg matrix H_k is generated during the Arnoldi process and $e_1 = (1, 0, 0, ..., 0)^T$. This least-squares problem can be solved using a QR factorization of H_k.

A problem that arises in connection with generalized minimal residue is that the number of vectors of order n that need to be stored grows linearly with k and the number of multiplications grow quadratically. This may be avoided by using a *restarted* version of generalized minimal residue.

4.4.2 Matrix Structure

The method used for enumerating the equations reflects in the bandwith of the block matrix for a domain. The efficiency of the algorithm for banded matrix-vector multiplication (Parallel BLAS routine) is a function of the bandwith of the coefficient matrix; higher performance from this algorithm is realized for smaller bandwith systems. The equations are therefore enumerated along the smaller index (i or j in 2D) for a domain. Further the enumeration is carried out for the interior cells first, followed by the interface cells. This enumeration is illustrated in Figure 2. for one domain in a simple rectangular domain. The structure of the global coefficient matrix resulting from this enumeration is shown in Figure 3. In Figure 4, the structure of the matrix for the interior of a domain for the forward substitution step is shown. The regular structure of the matrices is exploited to store only the non-zero elements. Each of the elements $A^{\pm}, B^{\pm}$ and D is a 4×4 matrix.

4.4.3 Programming Model of PIM

PIM uses the *Single Program, Multiple data (SPMD)* programming model. The main implication of using this model is that certain scalar values are needed in each processor. With PIM, the iterative method does not have access to the user mode of data storage. The assumption made is that each processor knows the number of elements of each vector stored in it and that all vector variables in a processor have the same number of elements. This allows different data partition schemes, including contiguous, cyclic and scattered partitioning. Owing to this freedom of storage by the user, operations involving matrices and vectors which require individual indices of vectors are handled by the user: matrix vector product and norm or vector being some. The Generalized minimal residue method uses its own stopping criterion which is equivalent to the 2-norm of the residual. So user supplied stopping criterion is not required.

The Global sum and vector norm routines supplied by the p4 environment is used. The main routine we have supplied here is with regard to data storage and access of matrices, apart from matrix-vector and vector-vector product.

In the current implementation, The coefficient matrix is partitioned by columns (by rows in version 2) among P processors, which are considered to be logically connected on a grid. Each processor stores at most $[N/P]$ columns of the matrix A. Each processor computes a vector with the same number of elements as that of the target processor which holds the partial sums for each element. This vector is sent across the network to be

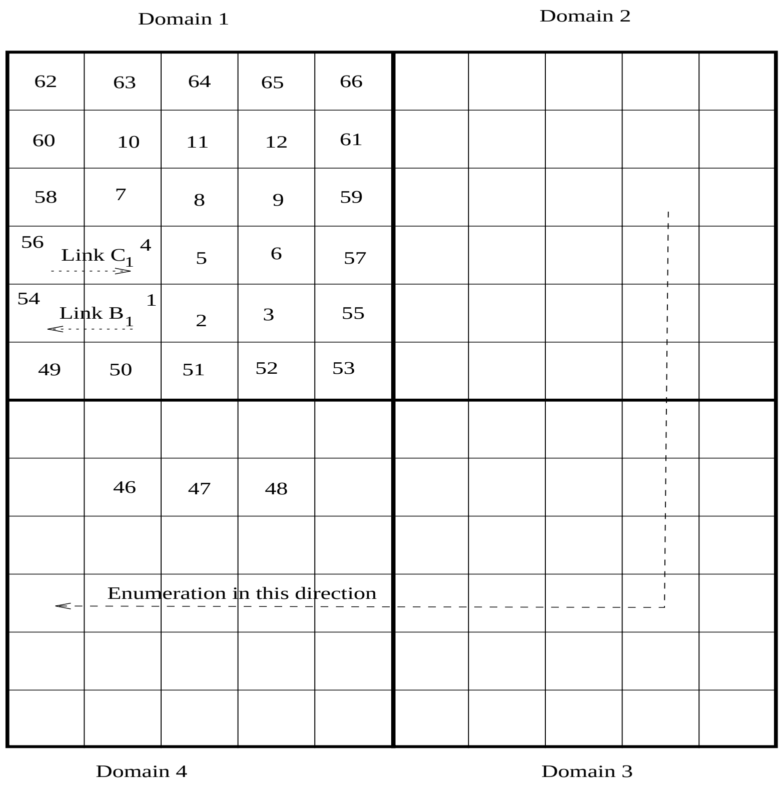

Figure 2:

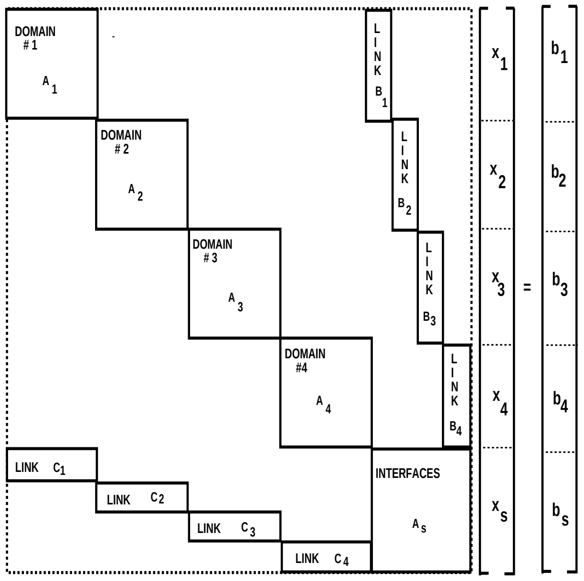

Figure 3: Decomposed Global Matrix

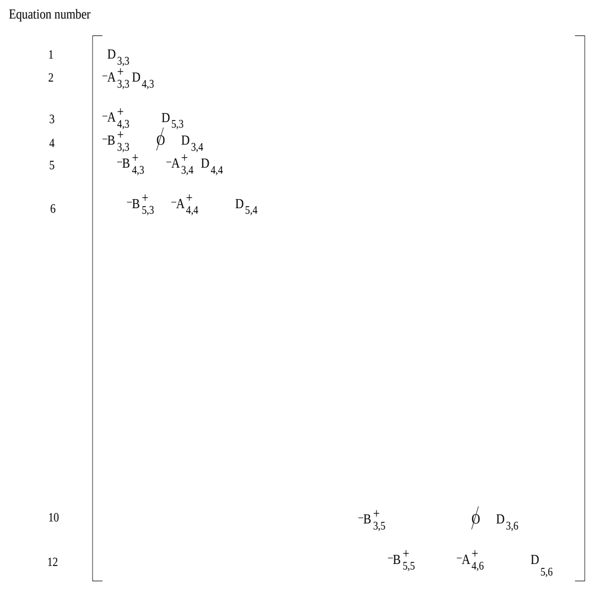
Equation number
1 $D_{3,3}$
2 $-A^{+}_{3,3}$ $D_{4,3}$
3 $-A^{+}_{4,3}$ $D_{5,3}$
4 $-B^{+}_{3,3}$ ϕ $D_{3,4}$
5 $-B^{+}_{4,3}$ $-A^{+}_{3,4}$ $D_{4,4}$
6 $-B^{+}_{5,3}$ $-A^{+}_{4,4}$ $D_{5,4}$
10 $-B^{+}_{3,5}$ ϕ $D_{3,6}$
12 $-B^{+}_{5,5}$ $-A^{+}_{4,6}$ $D_{5,6}$

FIGURE 4: Matrix structure for domain interior for the forward substitution step

summed in a recursive-doubling fashion until the accumulated vectors are then summed together with the partial sum vector computed locally in the target processor, yielding the elements of the vector resulting from the matrix-vector product. This process is repeated for all processors. The storage of the vectors(banded matrix) is also optimized by having a vector to specify range, and only few zero elements are stored and they are never operated on.

The two other methods for mapping the data to the processors are algebraic data partitioning and geometric data partitioning (see section on Data partitioning). In the former method the global coefficient matrix is partitioned either by rows or by columns and a set of these rows/columns is allocated to a processor to perform the computations. For the case of a $N \times N$ matrix and P processors, each processor in the simplest case would be allocated N/P rows or columns. Here, geometric data partitioning is used; in this approach a set of domains is allocated to a processor for local operations involved in the discretization and the solution steps. The mapping of the global matrix resulting from this approach is illustrated by the shading in Figure 5. In a later paper, a comparison of the two data mapping methods will be done.

5 Summary

The parallel grid generation system has been implemented through the p4 environment and is currently portable to most of the parallel machines supported by p4. This code can be run in IBM RS6000, TMC-CM5, and SUN clusters, by just changing the processor

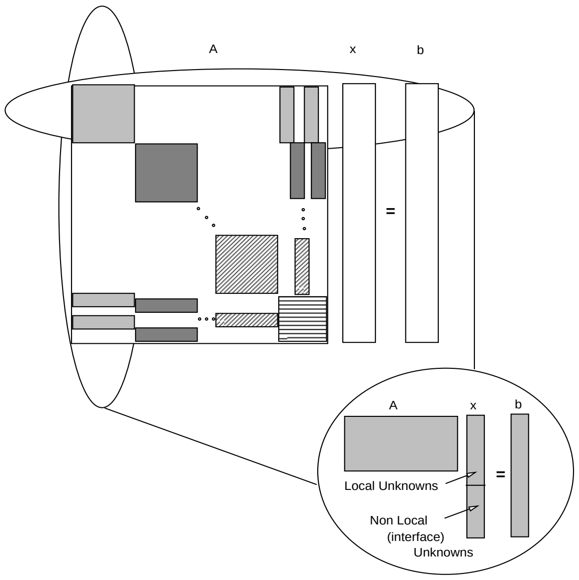

FIGURE 5: Mapping of Global Matrix

group file to access the machine required. The Local communication time has been reduced considerably, first by reducing the packing time and secondly by overlapping computation with communication. This has been possible due to the data structure introduced, active messages and asynchronous message passing protocols.

The block Euler solver has been implemented in sequential mode using the C-driver program and block interface data structures. Currently message passing routines are being included using the P4 message passing interface to implement the code on the CM-5 and the SP1 platforms.

References

[1] D.M. Belk, J.M. Janus and D.L. Whitfield, *Three-Dimensional Unsteady Euler Equations Solutions on Dynamic Grids* Air Force Armament Laboratory Report, AFATL-TR-86-21, April 1986.

[2] D.M. Belk, *Unsteady Three-Dimensional Euler Equations Solutions on Dynamic Blocked Grids* Ph.D. Dissertation, Mississippi State University, August 1986.

[3] Marsha J. Berger, *Data Structures for adaptive mesh refinement*

[4] Ralph M. Butler, and Ewing L. Lusk, *User's Guide to p4 Parallel Programming System* Oct 1992, Mathematics and Computer Science division, Argonne National Laboratory.

[5] Ralph M. Butler, and Ewing L. Lusk, *Monitors, Messages, and Clusters: the p4 Parallel Programming System.* DRAFT, Document for a standard Message Passing Interface, Message Passing interface Forum, August 1993.

[6] Animesh Chatterjee, *Numerical Prediction of Induced Drag for Wings and Wing-Body Combinations* M.S. Thesis, Mississippi State University, August 1993.

[7] N. P. Chrisochoides, Animesh Chatterjee, Rajani Vaidyanathan, and Geoffrey Fox, *An Evaluation of Data Mapping Approaches for Parallel Multi-Block Euler Solvers* To appear in the proceedings of Parallel CFD '94 at Kyoto, Japan. May16-19 1994.

[8] N. P. Chrisochoides, Elias Houstis and John Rice. *Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers* To appear in the Special Issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming, Vol 21, No 1, April 1994.

[9] Nikos Chrisochoides, Nashat Masour, and Geoffrey Fox, *Performance evaluation of data mapping algorithms for parallel single-phase iterative PDE solvers.* To appear in the proceedings of the Scalable High Performance Computing Conference, May, 1994, Knoxville, TN.

[10] Nikos Chrisochoides, Aboelaze Mokhtar, E. N. Houstis, and C. E. Houstis. The parallelization of level 2 and 3 *BLAS* operations on distributed memory machines In *Advances in Computer Methods for Partial Differential Equations VII*, (R. Vichnevetsky. D. Knight and G. Richter, eds) IMACS, New Brunswick, NJ, pages 119-126, 1992.

[11] Rudnei Dias da Cunha and Tom Hopkins, *PIM 1.0, The parallel iterative methods package for systems of linear equations: User's guide*

[12] Thorsten von Eicken, Davin E. Culler, Seth Cooper Goldstein, and Klaus Erik Schauser, *Active Messages: a mechanism for integrated communication and computation*

[13] G.C. Fox, *The Architecture of Problems and Portable Parallel Software Systems* Technical Report SCCS-134, NPAC, Syracuse University, 1991.

[14] Gene H. Golub and Charles F. Van Loan, *Matrix Computations* John Hopkins University Press.

[15] J.M. Janus, *Advanced 3-D CFD Algorithm for Turbomachinery* Ph.D. Dissertation, Mississippi State University, August 1989

[16] Hyun Jin Kim, and Joe F. Thompson, *Three-Dimensional Adaptive grid generation on a composite block grid* AIAA Journal, March 1990, vol 28.

[17] Joe F. Thompson, *Composite Grid Generation Code for General 3D Regions - EAGLE Code*, AIAA Journal, Vol 26, No 3, March 1988.

[18] Joe F. Thompson, *A general three-dimensional Elliptic Grid generation system on a composite block structure* Computer methods in applied mechanics and Engineering, 1987.

[19] Joe F. Thompson, *Grid Generation techniques in computational fluid dynamics* AIAA Journal, Nov 1984, vol 22.

[20] Joe F. Thompson, *A survey of composite grid generation for general three-dimensional regions*

[21] Joe F. Thompson *A composite grid generation code for general 3-D regions*

[22] Joe F. Thompson *A survey of Dynamically Adaptive grids in the numerical solution of Partial Differential Equations* Applied Numerical Mathematics, 1985.

6 APPENDIX-A: Governing Equations

The two dimensional Euler equations in Cartesian coordinates, neglecting body forces, can be written in the vector form:

$$\frac{\partial q}{\partial t} + \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y} = 0 \tag{1}$$

where

$$q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix} \qquad f = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(e+p) \end{bmatrix} \tag{2}$$

$$g = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(e+p) \end{bmatrix} \tag{3}$$

and all variables are non-dimensionalized with free-stream parameters as shown below:

$$\rho = \frac{\overline{\rho}}{\overline{\rho}_\infty}, \quad u = \frac{\overline{u}}{\overline{a}_\infty}, \quad v = \frac{\overline{v}}{\overline{a}_\infty}, \quad x = \frac{\overline{x}}{\mathcal{L}}, \quad y = \frac{\overline{y}}{\mathcal{L}}, \quad a = \frac{\overline{a}}{\overline{a}_\infty},$$

$$p = \frac{\overline{p}}{\overline{\rho}_\infty \overline{a}_\infty^2}, \quad e = \frac{\overline{e}}{\overline{\rho}_\infty \overline{a}_\infty^2}, \quad t = \frac{\overline{a}_\infty \overline{t}}{\mathcal{L}}$$

Here ρ is the density, u and v are the velocities along the x and y directions, a is the speed of sound in the medium, p is the pressure, e is the energy, t denotes time and $\mathcal{L}$ represents a characteristic length of the problem.

To preserve generality, Eqs.1 are transformed to the 2-D time-dependent curvilinear coordinate system defined as:

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

$$\eta = \eta(x, y, t) \tag{5}$$

$$\tau = t \tag{6}$$

The resulting equations are:

$$\frac{\partial Q}{\partial \tau} + \frac{\partial F}{\partial \xi} + \frac{\partial G}{\partial \eta} = 0 \tag{7}$$

where

$$Q = J \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix} \qquad F = J \begin{bmatrix} \rho U \\ \rho u U + \xi_x p \\ \rho v U + \xi_y p \\ U(e+p) - \xi_t p \end{bmatrix} \tag{8}$$

$$G = J \begin{bmatrix} \rho V \\ \rho u V + \eta_x p \\ \rho v V + \eta_y p \\ V(e+p) - \eta_t p \end{bmatrix} \tag{9}$$

$$\begin{aligned} \xi_x &= J^{-1}(y_\eta z_\zeta - z_\eta y_\zeta) & \eta_x &= J^{-1}(z_\xi y_\zeta - y_\xi z_\zeta) \\ \xi_y &= J^{-1}(z_\eta x_\zeta - x_\eta z_\zeta) & \eta_y &= J^{-1}(x_\xi z_\zeta - z_\xi x_\zeta) \\ \xi_t &= -x_\tau \xi_x - y_\tau \xi_y & \eta_t &= -x_\tau \eta_x - y_\tau \eta_y \end{aligned}$$

are the metrics of the transformation and the contravariant velocities are

$$\begin{aligned} U &= \xi_x u + \xi_y v + \xi_t \\ V &= \eta_x u + \eta_y v + \eta_t \end{aligned}$$

with the Jacobian of the transformation given by

$$J = x_\xi(y_\eta z_\zeta - z_\eta y_\zeta) - y_\xi(x_\eta z_\zeta - z_\eta x_\zeta) + z_\zeta(x_\eta y_\zeta - y_\eta x_\zeta) \tag{10}$$

7 APPENDIX-B: Discretization

An implicit first order temporal and second order spatial finite volume discretization of Eq.7 yields the following difference expression for the original PDE [6]:

$$[I + \Delta\tau(\delta_i A. + \delta_j B.)]\Delta Q^n_{i,j} = -\Delta\tau R^n_{i,j} \tag{11}$$

where the flux jacobians

$$A = (\tfrac{\partial F}{\partial Q})^n$$

$$B = (\tfrac{\partial G}{\partial Q})^n$$

R^n is the residual vector

$$R^n_{i,j} = \delta_i F^n + \delta_j G^n$$

$Q^{n+1}_{i,j}$ is the required solution vector and is obtained from

$$\Delta Q^n_{i,j} = Q^{n+1}_{i,j} - Q^n_{i,j}$$

and the fluxes, F^{n+1} and G^{n+1}, have been linearized about time level n.

There are several ways to solve Eq.11. The method followed in the code under study uses flux vector splitting to construct the left hand side of the equation and the right hand side is evaluated using Roe's approximate Riemann solver. One form of flux vector splitting is described in Appendix-B of [6]; application of this technique leads to the following form for Eq.11:

$$[I + \Delta\tau(\delta_i A^+. + \delta_i A^-. + \delta_j B^+. + \delta_j B^-.)]\Delta Q^n_{i,j} = -\Delta\tau R^n_{i,j} \tag{12}$$

The interested reader is referred to Appendix-B of [2] for a development of the flux jacobians $A^\pm$, $B^\pm$ and $C^\pm$. Eq.12 can now be solved for the Cartesian variable vector Δq^n as:

$$[I + \overline{\Delta\tau}(\delta_i A^+. + \delta_i A^-. + \delta_j B^+. + \delta_j B^-.)]\Delta q^n_{i,j} = -\overline{\Delta\tau} R^n_{i,j} \tag{13}$$

where

$$\overline{\Delta\tau} = \tfrac{\Delta\tau}{J}$$

8 APPENDIX-C: Evaluation of R^n

Rewriting Eq. 7

$$R^n_{i,j} = (F^n_{i+1/2} - F^n_{i-1/2}) + (G^n_{j+1/2} - G^n_{j-1/2}) \tag{14}$$

where $(i \pm 1/2)$ and $(j \pm 1/2)$ indicate cell face indices for cell (i, j).

8.1 Formulation

The evaluation of the fluxes in Eq. 14 requires the solution to a Riemann problem at each cell interface that arises in the finite volume context from the assumption of uniform properties within each computational cell. The approach followed here is the approximate Riemann solver due to Roe and its extension to second and third order spatial accuracy. Details about this technique are given in Appendix A of [15] and will not be repeated here.

Application of this method leads to the following formulations for the fluxes:

$$F^{(1)}_{i+1/2} = F_i + \Sigma^m_{j=1} \alpha_{j,i+1/2} \lambda^{(-)j}_{i+1/2} r^j_{i+1/2} \tag{15}$$

$$F^{(2)}_{i+1/2} = F^{(1)}_{i+1/2} + F^{(c)}_{i+1/2} \tag{16}$$

where $F^{(c)}_{i+1/2}$ is the higher order correction

$$F^{(c)}_{i+1/2} = \Sigma^m_{j=1} (\frac{1-\psi}{4}[L^{(+)}_j(-1,1) - L^{(-)}_j(3,1)] + \Sigma^m_{j=1} \frac{1+\psi}{4}[L^{(+)}_j(1,-1) - L^{(-)}_j(1,3)]) r^j_{i+1/2} \tag{17}$$

The various terms appearing above are defined as

$$L^{(\pm)}_j(l,n) = minmod(\sigma^{(\pm)}_{j,i+1/2}, b\sigma^{(\pm)}_{j,i+n/2})$$

$$minmod(x,y) = sign(x)max(0, min[|x|, ysign(x)])$$

$$\sigma^{(\pm)}_{j,i+p/2} = \lambda^{(\pm)j}_{i+1/2} \alpha_{j,i+p/2}$$

$$\alpha_j = l^j.dQ$$

$$\alpha_{j,i-1/2} = l^j_{i+1/2}.(Q_i - Q_{i-1}) \tag{18}$$

$$\alpha_{j,i+1/2} = l^j_{i+1/2}.(Q_{i+1} - Q_i) \tag{19}$$

$$\alpha_{j,i+3/2} = l^j_{i+1/2}.(Q_{i+2} - Q_{i+1}) \tag{20}$$

$$b = \frac{3-\psi}{1-\psi}$$

and $\psi = 1/3$ yields a third order scheme whereas $\psi = -1$ is used to obtain a second order formulation for the fluxes. Here, the variables $l^j(\in R^{1\times4})$, $r^j(\in R^{4\times1})$ are the left and right eigenvectors of Roe's matrix $\overline{A}$, $\lambda^{(\pm)j}$ are the eigenvalues of $\overline{A}$ and m for the 2-D Euler equations equals four. The superscripts (1) and (2) above indicate first and second order fluxes respectively and the subscripts $i \pm 1/2$ indicate metrics are evaluated at these cell faces. Equations similar to Eq.15 and Eq.16 can be written for the first and second order fluxes, $G^{(1)}_{j+1/2}$ and $G^{(2)}_{j+1/2}$ along the η -direction.

8.2 Computation Methodology

- The computational stencil associated with cell (i,j) can be identified from Eq.16; evaluation of the flux through cell (i,j) requires q-variables at time level n, from two-neighboring cells along the ξ and η coordinate directions. This is more clearly illustrated in Figure 6 which shows the computational stencil used in constructing the flux through cell (i,j).
- At block boundaries, q-variables at the previous time level n, from a two-cell deep

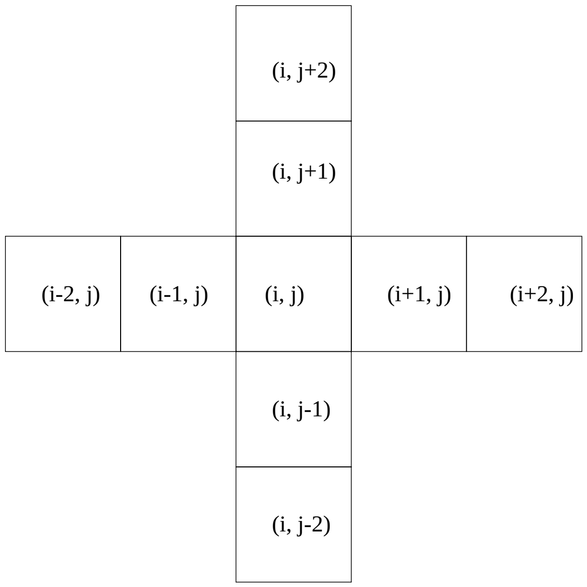

FIGURE 6: Computational stencil for evaluating flux through cell (i,j)

layer in adjacent blocks will be required to calculate the fluxes at the boundary cells. This is illustrated in Figure 7 for an arbitrary arrangement of blocks. This figure shows the outermost layer of cells for block 2 and the shaded area represents the two cell thick layer in block 1 required to construct the flux in this outermost layer of cells.

- The left and right eigenvectors, l^j and r^j respectively, and the eigenvalues $\lambda^{(\pm)j}$ are evaluated using Roe's averaged variables at each cell interface $i+1/2$ and $j+1/2$. Calculation of these averaged properties requires q-data at time step n, to the left and right of the interface.

- The residual vector R^n is constructed *at each time step*

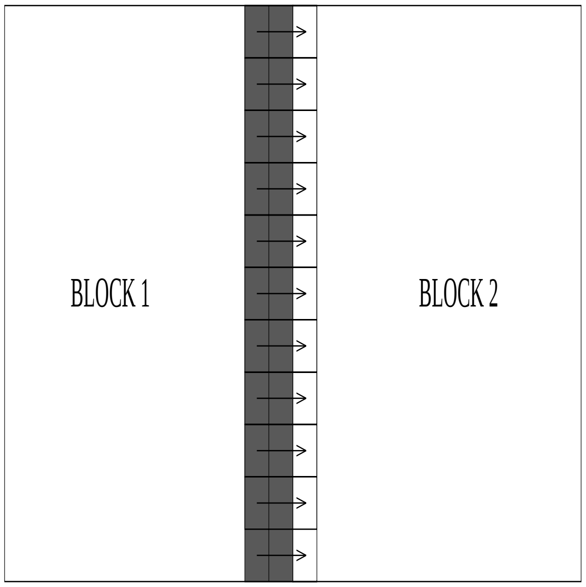

Figure 7: Block interface communication

9 APPENDIX-D: Evaluation of flux jacobians

- Flux jacobians computed using analytical expressions for all cells from i=1 to *imax* and j=1 to *jmax* for $A^{\pm}$ and $B^{\pm}$ respectively.
- Calculations require q^n-values in cell for the flux jacobians associated with the positive eigenvalues.
- Calculations require q^n-values in cell *(i+1,j)* for the flux Jacobian, $A^{-}_{i,j}$ associated with the negative eigenvalues. Similarly q^n-values in cell *(i,j+1)* are required for the flux Jacobians $B^{-}_{i,j}$.
- All scalar operations.
- For steady problems, infrequent flux jacobian updating used. Typically updating may be done every 10 cycles.
- $A^{\pm}$, $B^{\pm} \in R^{4\times4}$ in 2D.

10 APPENDIX-E: Solution scheme

10.1 Formulating the Linear System

Expanding the difference operators in Eq. 12 and approximately factorizing the resulting expression leads to the following two-pass scheme [6]:

Forward Pass:

This is a forward substitution sweep through the computational domain

$$D_{i,j,k}X^1_{i,j,k} - [A^+_{i-1,j,k}X^1_{i-1,j,k} + B^+_{i,j-1,k}X^1_{i,j-1,k} + C^+_{i,j,k-1}X^1_{i,j,k-1}] = -R^n_{i,j,k} \tag{21}$$

Backward Pass:

This is a backward substitution sweep through the computational domain

$$D_{i,j,k}X^3_{i,j,k} + [A^-_{i+1,j,k}X^2_{i+1,j,k} + B^-_{i,j+1,k}X^2_{i,j+1,k} + C^-_{i,j,k+1}X^2_{i,j,k+1}] = 0 \tag{22}$$

$$X^2_{i,j,k} = X^3_{i,j,k} + X^1_{i,j,k} \tag{23}$$

where

$$D_{i,j,k} = A^\pm_{i,j,k} + B^\pm_{i,j,k} + C^\pm_{i,j,k} \tag{24}$$

and

$$\Delta q^n_{i,j} = X^2_{i,j,k}$$

This yields block sparse banded matrices for the forward and backward sweeps respectively.

10.2 Computational Highlights

- $4 \times (imax - 1) \times (jmax - 1)$ nonlinear coupled equations solved at every time step.
- Iterative techniques used to solve the system of equations
- The partitioning of the system of equations with a multi-block approach can be represented as in Figure 8. The off-diagonal blocks involve communication with contiguous blocks. Figure 9 shows the sequence of operations in obtaining a solution for the entire field using domain decomposition.

Forward pass:

$$\left[\begin{array}{c:c} I + L_{I,I} & 0 \\ \hdashline L_{II,I} & I + L_{II,II} \end{array}\right] \left[\begin{array}{c} X_{I} \\ \hdashline X_{II} \end{array}\right] = \left[\begin{array}{c} R_{I} \\ \hdashline R_{II} \end{array}\right]$$

Backward pass:

$$\left[\begin{array}{c:c} I + U_{I,I} & U_{I,II} \\ \hdashline 0 & I + U_{II,II} \end{array}\right] \left[\begin{array}{c} Q_{I} \\ \hdashline Q_{II} \end{array}\right] = \left[\begin{array}{c} X_{I} \\ \hdashline X_{II} \end{array}\right]$$

Figure 8: Matrix partitioning with domain decomposition

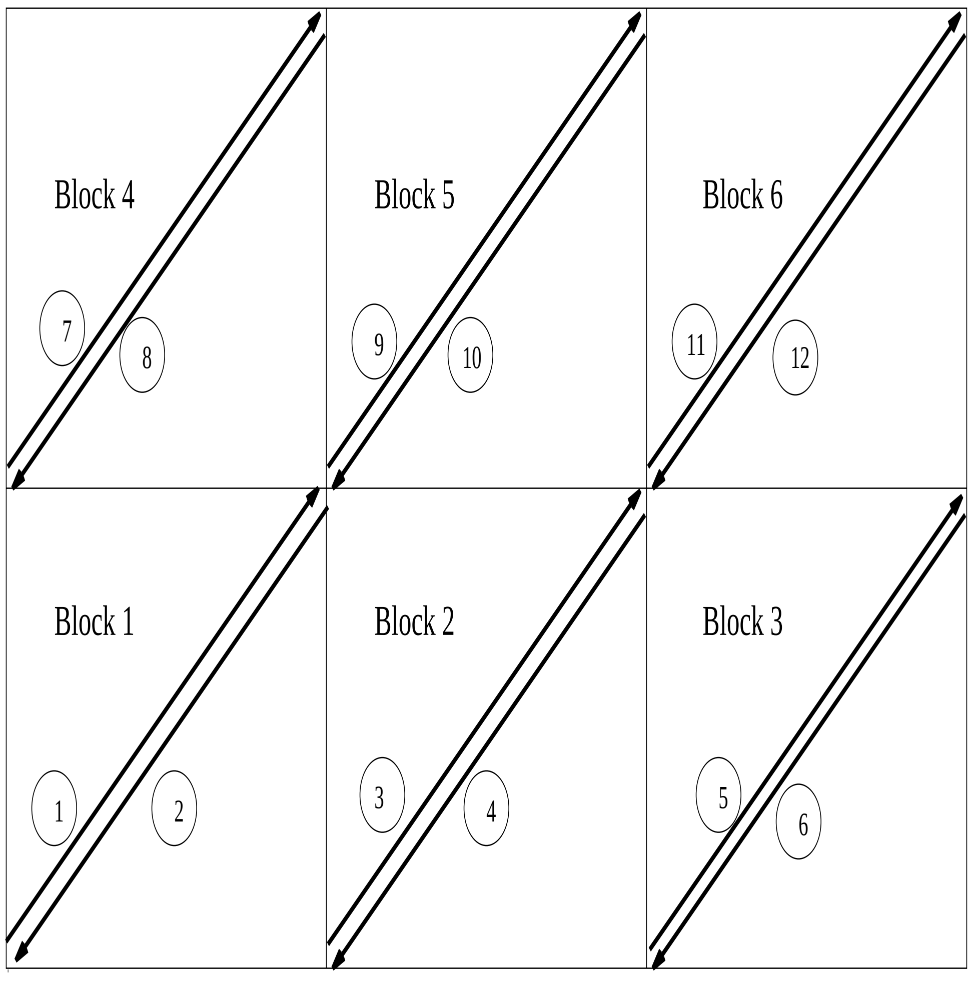

FIGURE 9: Sequence of operations for entire field using domain decomposition

11 APPENDIX-F: C-driver for the Grid Modules

12 APPENDIX-G: C-driver and code for the Euler Module