A Data Parallel Finite Element Method for Computational Fluid Dynamics on the Connection Machine System

A Data Parallel Finite Element Method for Computational Fluid Dynamics on the Connection Machine System

Zdeněk Johan
Thomas J.R. Hughes
Kapil K. Mathur
S. Lennart Johnsson

TR-02-92

January 1992

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

A Data Parallel Finite Element Method for Computational Fluid Dynamics on the Connection Machine System

by

Zdeněk Johan

Thomas J.R. Hughes[*]

Division of Applied Mechanics
Durand Building
Stanford University
Stanford, CA 94305–4040, USA

and

Kapil K. Mathur

S. Lennart Johnsson[**]

Thinking Machines Corporation
245 First Street
Cambridge, MA 02142–1264, USA

Thinking Machines Technical Report TMC–212 1991

To appear in:

Computer Methods in Applied Mechanics and Engineering

September 8, 1994

[*] Also affiliated with CENTRIC Engineering Systems, Inc., 3801 East Bayshore Road, Palo Alto, CA 94303, USA.

[**] Also affiliated with the Division of Applied Sciences, Harvard University, Cambridge, MA 02138, USA.

Abstract

A finite element method for computational fluid dynamics has been implemented on the Connection Machine systems CM-2 and CM-200. An implicit iterative solution strategy, based on the preconditioned matrix-free GMRES algorithm, is employed. Parallel data structures built on both nodal and elemental sets are used to achieve maximum parallelization. Communication primitives provided through the Connection Machine Scientific Software Library substantially improved the overall performance of the program. Computations of three-dimensional compressible flows using unstructured meshes having close to one million elements, such as a complete airplane, demonstrate that the Connection Machine systems are suitable for these applications. Performance comparisons are also carried out with the vector computers Cray Y-MP and Convex C-1.

Contents

1. Introduction

Engineers have always been looking for techniques to improve the design of products. This has led to the development of complex models to represent more closely the physics of designs. For example, an aerodynamicist may consider viscous effects, turbulence and combustion, among others. At the same time, the engineer wants to improve the accuracy of calculations by refining the discretization of the computational domain and by modeling more complicated geometries. Unfortunately, all these efforts increase the need for memory and computation time required to obtain a solution. Classical vector supercomputers have been shown to be close to their performance limits, and it appears that they will not be able to keep up with the computing power required by the scientific community in the future. On the other hand, massively parallel computers have already shown great promise and are expected to be the ones which will solve the Grand Challenges of the 1990's [1].

The finite element method has taken the lead as an industrial numerical tool because of its ability to handle complex configurations through the use of unstructured meshes. However, there has been some scepticism in the community about how well finite element methodologies would perform on massively parallel computers. Our objective is to demonstrate that such computers are suitable for finite element techniques in large-scale computational fluid dynamics. We have chosen to work on the Connection Machine systems CM-2 and CM-200 built by Thinking Machines Corporation because they appeared as the most mature massively parallel computers both in terms of hardware and software. Finite element methods for structural analysis have been implemented on the Connection Machine system CM-2 by Johnsson and Mathur [2, 3], Belytschko, *et al.* [4] and Farhat, *et al.* [5], among others. Two-dimensional CFD codes using finite element (see [6]) and finite volume techniques (see [7] and references therein) have also been implemented on the Connection Machine system CM-2. These investigations demonstrated the potential of the CM-2 for finite element applications.

An outline of this paper follows: Brief descriptions of the Connection Machine systems CM-2 and CM-200 hardware and of Fortran 90 constructs are given in Section 2. The solution strategy is presented in Section 3. Implementation and communication issues are discussed in Sections 4 and 5, respectively. Numerical examples in Section 6 illustrate the techniques we have used on the Connection Machine systems. Finally, conclusions are drawn in Section 7.

2. The Connection Machine systems CM-2 and CM-200

2.1. Hardware description

The Connection Machine systems CM-2 and CM-200 are single instruction-multiple data (SIMD) massively parallel computers (see [8] and [9] for technical references). These systems can be viewed as supercomputers having up to 2,048 floating-point units (also called processing nodes) arranged in an eleven-dimensional binary cube topology (2,048 = 2^{11} since each axis is of length two in a binary cube.) Each processing node is composed of 32 one-bit processors and one 64-bit floating-point accelerator, a routing chip and some auxiliary hardware, and up to 4 Mbytes of memory. There is a 32-bit wide data path between each processing unit and its local memory. Each pair of neighboring processing nodes are connected by two bidirectional channels for data transfer. Figure 1 presents a 16-processing node configuration. The Connection Machine system CM-2 operates at a clock frequency of 7 MHz, and the system CM-200 at 10 MHz. There are some minor differences in the hardware to accommodate the difference in clock frequency. User programs can execute on either system without change.

The Connection Machine systems supports two modes of communication. Nearest-neighbor communication is supported by the "NEWS grid" (North-East-West-South). It consists of an array mapping procedure to the memory of the processing nodes such that adjacency is preserved [10], and of communication routines for accessing data in adjacent processing nodes. This mapping and communication strategy is very efficient for structured grid applications. The mapping is controlled by compiler directives. For Connection Machine Fortran, a dialect of Fortran 90 as described briefly below, "NEWS grid" is the default mapping. The alternative mapping use the standard binary encoding of array indices.

Nearest neighbor communication as defined by a "NEWS grid" is not feasible for unstructured meshes, like those used in many finite element applications. Communication between arbitrary memory locations in the distributed memory is required. On the Connection Machine systems, such communication is handled by the routing hardware. Communication issues for finite element methods are detailed in Section 4.

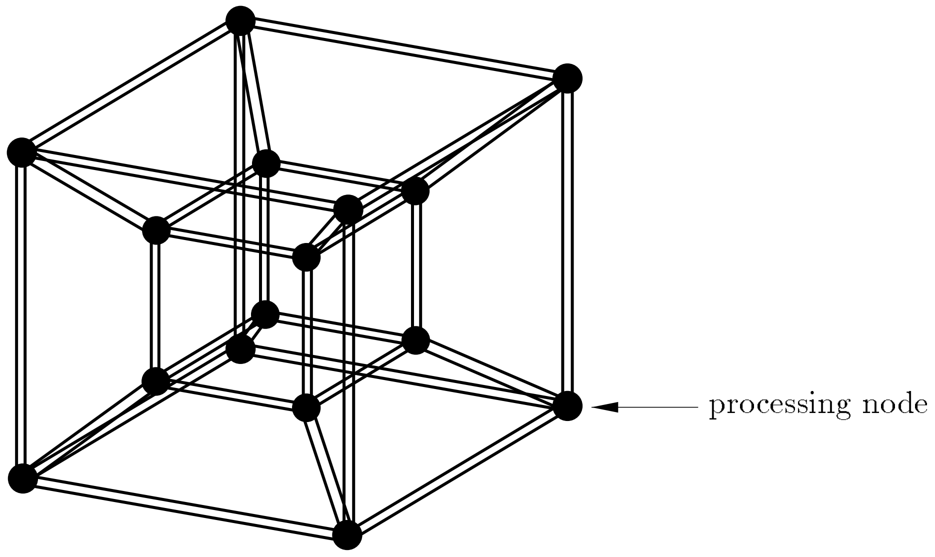

Figure 1. Connection Machine systems CM-2 and CM-200 binary cube topology (16 processing nodes).

The Connection Machine system CM-2 or CM-200 is connected to a front-end computer (a Sun SPARC workstation in our case) running an extended version of UNIX that allows for the control of the Connection Machine system. The user edits and compiles the code on the front-end computer before attaching to the Connection Machine system for execution. A high-speed input/output system, with up to eight 40 Mbytes/s channels and 100 Mbytes/s HIPPI channels, is available for parallel mass storage and graphics.

2.2 Fortran 90

The Fortran 90 programming language is a new standard derived from Fortran 77. It is a superset of the latter to ensure compatibility and includes a new set of programming features such as:

1. ***Array operations*** with array control statements. The following Fortran 90 example, with its Fortran 77 equivalent, illustrates these new capabilities. It represents a conditional summation of two arrays of size `N`:

   Fortran 90

   ```
   WHERE ( A > 0.0 )
     C = A + B
   END WHERE
   ```

   Fortran 77

   ```
      DO 10 I = 1, N
       IF ( A(I) .GT. 0.0 ) THEN
         C(I) = A(I) + B(I)
       END IF
   10 CONTINUE
   ```

   Note the conciseness of the Fortran 90 constructs. The programming style is therefore much closer to the formulas used in mathematical modeling of scientific and engineering problems.

2. Many new ***intrinsic procedures*** that include mathematical expressions and array manipulation functions. All these procedures make Fortran 90 a complete programming language for scientific applications.

3. ***Dynamic memory allocation***. This feature is absent in Fortran 77. The lack of dynamic memory allocation often lead to complex programming constructs for applications where storage is a crucial issue. Dynamic allocation now allows the programmer to

define arrays where needed, and storage to be discarded when the execution exits the subroutine.

4. First signs of an ***object-oriented style of programming*** with tools such as pointers, modules and new control constructs.

For an in-depth description of Fortran 90, the reader should consult Metcalf and Reid [11].

The Connection Machine Fortran [12] (abbreviated as CMF in the remainder of this paper) includes all the Fortran 90 syntax necessary to write data parallel finite element applications, though it does not yet contain all the features of the new Fortran standard. CMF also has a few non-Fortran 90 extensions, some of which were included in proposals for the new language but excluded from the final definition of the language. One of these extensions is the `FORALL` statement, used in Section 5.2, which acts like a parallel `DO` loop. We have refrained from using nonstandard features in our code to simplify its port to other platforms in the future. Compiler directives, known as `LAYOUT` directives, are available to control the allocation of array elements to the memory of the processing nodes. The "NEWS grid" allocation is enforced through the `:NEWS` directive, and the allocation through the conventional binary encoding through the `:SEND` directive. An array axis can also be made local to the memory of a processing node though the `:SERIAL` directive. In addition, the length of the segment of an array axis mapped to a processor can be changed through the use of axis ***weights***. These compiler directives affect the data allocation of a single data array. The layout of different arrays can be made to conform with each other through the `:ALIGN` directive (by default, the CMF compiler allocates all arrays of the same shape in the same way). The application of `LAYOUT` directives to the finite element data structures we use, is described in Section 4.1.

3. Implicit iterative finite element solver

The time-dependent compressible Navier-Stokes equations are discretized using a space-time Galerkin/least-squares variational formulation. This finite element method has been introduced and analyzed by Hughes and Johnson and their respective co-workers. The reader is referred to [13] and references therein for a description of the formulation implemented in our finite element program.

3.1. Implicit time-marching algorithm

At each discrete time t_n, finite element discretization of the compressible Navier-Stokes equations leads to the following nonlinear problem:

Given the solution vector $\widetilde{\boldsymbol{v}}_{(n-1)}$ at time t_{n-1}, and a time increment Δt, find the solution vector $\widetilde{\boldsymbol{v}}$ at time t_n, which satisfies the nonlinear system of equations

$$\widetilde{\boldsymbol{G}}(\widetilde{\boldsymbol{v}}; \widetilde{\boldsymbol{v}}_{(n-1)}, \Delta t) = \boldsymbol{0} \tag{1}$$

$\widetilde{\boldsymbol{G}}$ is a system of nonlinear functionals of $\widetilde{\boldsymbol{v}}$ and of parameters $\widetilde{\boldsymbol{v}}_{(n-1)}$ and Δt. This system is solved for $\widetilde{\boldsymbol{v}}$ by performing a succession of linearizations through a truncated Taylor series expansion of $\widetilde{\boldsymbol{G}}$. This leads to a set of linear systems of equations of the form

$$\widetilde{\boldsymbol{J}}^{(i)}\, \widetilde{\boldsymbol{p}}^{(i)} = -\widetilde{\boldsymbol{R}}^{(i)} \tag{2}$$

where

$$\widetilde{\boldsymbol{J}}^{(i)} = \frac{\partial \widetilde{\boldsymbol{G}}}{\partial \widetilde{\boldsymbol{v}}}(\widetilde{\boldsymbol{v}}^{(i)}; \widetilde{\boldsymbol{v}}_{(n-1)}, \Delta t) \tag{3}$$

$$\widetilde{\boldsymbol{p}}^{(i)} \overset{\text{def}}{=} \widetilde{\boldsymbol{v}}^{(i+1)} - \widetilde{\boldsymbol{v}}^{(i)} \tag{4}$$

$$\widetilde{\boldsymbol{R}}^{(i)} = \widetilde{\boldsymbol{G}}(\widetilde{\boldsymbol{v}}^{(i)}; \widetilde{\boldsymbol{v}}_{(n-1)}, \Delta t) \tag{5}$$

$\widetilde{\boldsymbol{v}}^{(i)}$ and $\widetilde{\boldsymbol{v}}^{(i+1)}$ being the approximations of $\widetilde{\boldsymbol{v}}$ at iterations i and $i+1$, respectively. $\widetilde{\boldsymbol{R}}$ is the residual of the nonlinear problem and $\widetilde{\boldsymbol{J}}$ is the consistent Jacobian associated with $\widetilde{\boldsymbol{R}}$. The consistent Jacobian is often replaced by a Jacobian-like matrix $\widetilde{\boldsymbol{\mathcal{J}}}$ leading to a more stable time-marching algorithm (see Johan, *et al.* [14]). A residual-like vector $\widetilde{\boldsymbol{\mathcal{R}}}$ associated with $\widetilde{\boldsymbol{\mathcal{J}}}$ can be defined as

$$\widetilde{\boldsymbol{\mathcal{J}}} \overset{\text{def}}{=} \frac{\partial \widetilde{\boldsymbol{\mathcal{R}}}}{\partial \widetilde{\boldsymbol{v}}} \tag{6}$$

The complete algorithm is summarized in Box 1. In this algorithm, $N_{\max}$ is the maximum number of time steps; $i_{\max}$ is the maximum number of Newton iterations; and $\widetilde{\boldsymbol{v}}_{(0)}$ is the initial guess of the problem. For steady-state computations, it is sufficient to set $i_{\max} = 1$. A detailed study of time-marching schemes for space-time finite element methods has been carried out by Shakib, *et al.* [13].

Box 1 - Implicit Time-marching Solution Algorithm.

Given $N_{\max}$, $i_{\max}$ and $\widetilde{\boldsymbol{v}}_{(0)}$, proceed as follows:

```
(Loop over time)
For  $n = 1, 2, \ldots, N_{\max}$
    (Newton-type algorithm)
    $\widetilde{\boldsymbol{v}}^{(0)} \leftarrow \widetilde{\boldsymbol{v}}_{(n-1)}$
    For  $i = 0, 1, \ldots, i_{\max} - 1$
        Solve  $\widetilde{\boldsymbol{\mathcal{J}}}^{(i)} \widetilde{\boldsymbol{p}}^{(i)} = -\widetilde{\boldsymbol{R}}^{(i)}$  for  $\widetilde{\boldsymbol{p}}^{(i)}$
        $\widetilde{\boldsymbol{v}}^{(i+1)} \leftarrow \widetilde{\boldsymbol{v}}^{(i)} + \widetilde{\boldsymbol{p}}^{(i)}$
    $\widetilde{\boldsymbol{v}}_{(n)} \leftarrow \widetilde{\boldsymbol{v}}^{(i_{\max})}$
```

3.2. Preconditioned matrix-free GMRES algorithm

A scaling (or preconditioning) transformation is first applied to the system of equations $\widetilde{\boldsymbol{\mathcal{J}}}\,\widetilde{\boldsymbol{p}} = -\widetilde{\boldsymbol{R}}$ to nondimensionalize it and improve its conditioning. We have used a block-diagonal preconditioner as it has been shown to be both inexpensive and efficient (see Shakib, *et al.* [15]). The size of the blocks equals the number of degrees of freedom per node. Let $\widetilde{\boldsymbol{W}}$ be the nodal diagonal blocks of the left-hand-side matrix $\widetilde{\boldsymbol{\mathcal{J}}}$. Since our finite element formulation generates symmetric positive-definite nodal diagonal blocks, $\widetilde{\boldsymbol{W}}$ accommodates a Cholesky factorization

$$\widetilde{\boldsymbol{W}} = \widetilde{\boldsymbol{U}}^T \widetilde{\boldsymbol{U}} \tag{7}$$

A two-sided preconditioning step is then applied to $\widetilde{\boldsymbol{\mathcal{J}}}\,\widetilde{\boldsymbol{p}} = -\widetilde{\boldsymbol{R}}$, leading to the scaled system of equations $\boldsymbol{\mathcal{J}}\,\boldsymbol{p} = -\boldsymbol{R}$ with

$$\boldsymbol{\mathcal{J}} = \widetilde{\boldsymbol{U}}^{-T}\widetilde{\boldsymbol{\mathcal{J}}}\,\widetilde{\boldsymbol{U}}^{-1} \tag{8}$$

$$\boldsymbol{p} = \widetilde{\boldsymbol{U}}\,\widetilde{\boldsymbol{p}} \tag{9}$$

$$\boldsymbol{R} = \widetilde{\boldsymbol{U}}^{-T}\widetilde{\boldsymbol{R}} \tag{10}$$

This preconditioned system of equations is solved using the Generalized Minimal RESidual (GMRES) algorithm. This algorithm was introduced by Saad and Schultz [16]. Its effectiveness for computational fluid dynamics problems has been demonstrated by several research groups (see, for example, [14, 15, 17, 18]). The GMRES algorithm computes an approximate solution $\boldsymbol{p}_0 + \boldsymbol{z}$, where $\boldsymbol{p}_0$ is an initial guess (usually taken to be $\boldsymbol{0}$) and $\boldsymbol{z}$ is in the Krylov space $\mathcal{K} \overset{\text{def}}{=} \{\boldsymbol{r}_0, \boldsymbol{\mathcal{J}}\,\boldsymbol{r}_0, \ldots, \boldsymbol{\mathcal{J}}^{k-1}\,\boldsymbol{r}_0\}$. $\boldsymbol{r}_0 = -\boldsymbol{R} - \boldsymbol{\mathcal{J}}\,\boldsymbol{p}_0$ is the residual and k is the dimension of $\mathcal{K}$. The vector $\boldsymbol{z}$ is solution of the least-squares problem

$$\min_{\boldsymbol{z}\in\mathcal{K}} \| -\boldsymbol{R} - \boldsymbol{\mathcal{J}}(\boldsymbol{p}_0 + \boldsymbol{z})\| \tag{11}$$

An orthonormal basis of $\mathcal{K}$, $\boldsymbol{U}_k = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k]$, is constructed using the modified Gram-Schmidt algorithm. In turn, a $(k+1)\times k$ upper Hessenberg matrix $\boldsymbol{H}_k$ satisfying

$$\boldsymbol{\mathcal{J}}\,\boldsymbol{U}_k = \boldsymbol{U}_{k+1}\,\boldsymbol{H}_k \tag{12}$$

is generated. Let $\boldsymbol{z} = \sum_{i=1}^{k} y_i \boldsymbol{u}_i$ and $\boldsymbol{e} = \{\|\boldsymbol{r}_0\|, 0, \ldots, 0\}^T$. The relation (12) is used to reduce the minimization problem (11) to

$$\min_{\boldsymbol{y}\in\mathcal{R}} \|\boldsymbol{e} - \boldsymbol{H}_k\,\boldsymbol{y}\| \tag{13}$$

The minimizer of (13) is obtained using the Q-R algorithm. The complete GMRES algorithm is presented in Box 2. ε_{tol} is the nondimensional convergence tolerance of the algorithm; and $l_{\max}$ is the maximum number of GMRES cycles. It is possible to perform the Q-R factorization during the generation of the basis $\boldsymbol{U}_k$, and to stop the formation of the vectors when the convergence check is satisfied. This more efficient GMRES algorithm is described by Shakib, *et al.* [15]. The implementation of the GMRES algorithm on the Connection Machine systems CM-2 and CM-200 is described in Section 4.2.

Box 2 - GMRES Algorithm.

Given $\boldsymbol{\mathcal{J}}$, $\boldsymbol{R}$, k, $\varepsilon_{\rm tol}$ and $l_{\max}$, proceed as follows:

```
(Initialization)
$\varepsilon \leftarrow \varepsilon_{\rm tol} \, \|\boldsymbol{R}\|$
$\boldsymbol{p} = \boldsymbol{0}$
(GMRES Cycles)
For  $n = 1, 2, \ldots, l_{\max}$
    $\boldsymbol{u}_1 \leftarrow -\boldsymbol{R} - \boldsymbol{\mathcal{J}}\, \boldsymbol{p}$
    $\boldsymbol{e} \leftarrow \{\|\boldsymbol{u}_1\|, 0, \ldots, 0\}^T$
    $\boldsymbol{u}_1 \leftarrow \dfrac{\boldsymbol{u}_1}{\|\boldsymbol{u}_1\|}$
    (GMRES Iteration)
    For  $i = 1, 2, \ldots, k$
        (Matrix-Vector Product)
        $\boldsymbol{u}_{i+1} \leftarrow \boldsymbol{\mathcal{J}}\, \boldsymbol{u}_i$
        (Modified Gram-Schmidt Procedure)
        For  $j = 1, \ldots, i$
            $\beta_{i+1,j} \leftarrow (\boldsymbol{u}_{i+1}, \boldsymbol{u}_j)$
            $\boldsymbol{u}_{i+1} \leftarrow \boldsymbol{u}_{i+1} - \beta_{i+1,j}\, \boldsymbol{u}_j$
        (End Modified Gram-Schmidt Procedure)
        $\boldsymbol{h}_i \leftarrow \{\beta_{i+1,1}, \ldots, \beta_{i+1,j}, \|\boldsymbol{u}_{i+1}\|\}^T$
        $\boldsymbol{u}_{i+1} \leftarrow \dfrac{\boldsymbol{u}_{i+1}}{\|\boldsymbol{u}_{i+1}\|}$
    (End GMRES Iteration)
    $\boldsymbol{H}_k \leftarrow [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_k]$
    Solve  $\min \|\boldsymbol{e} - \boldsymbol{H}_k\, \boldsymbol{y}\|$  for $\boldsymbol{y}$ using the Q-R algorithm
    (Solution Update)
    $\boldsymbol{p} \leftarrow \boldsymbol{p} + \sum_{j=1}^{k} y_j \boldsymbol{u}_j$
    (Convergence Check)
    If $\|\boldsymbol{e} - \boldsymbol{H}_k\, \boldsymbol{y}\| \leq \varepsilon$, Exit $l$ loop
(End GMRES Cycles)
Return
```

Since the matrix $\mathcal{J}$ is a Jacobian-like matrix, the matrix-vector products $\mathcal{J}\, \boldsymbol{u}_i$ of the GMRES algorithm can be replaced by the one-sided finite difference stencil

$$\mathcal{J}\, \boldsymbol{u}_i \approx \frac{\mathcal{R}(\boldsymbol{v} + \delta\, \boldsymbol{u}_i) - \mathcal{R}(\boldsymbol{v})}{\delta} \tag{14}$$

where $\boldsymbol{v}$ is the current solution and δ is a small scalar. This approximation circumvents the need for computing and storing the left-hand-side matrix $\mathcal{J}$, thus saving a substantial amount of storage. Matrix-free techniques for finite element applications have been analyzed by Johan, *et al.* [14].

Note that this implicit iterative scheme reduces to computing a succession of block-diagonal preconditioners $\widetilde{\widetilde{\boldsymbol{W}}}$ and residual vectors $\widetilde{\boldsymbol{R}}$, or residual-like vectors $\widetilde{\mathcal{R}}$. The classical technique for evaluating $\widetilde{\widetilde{\boldsymbol{W}}}$ and $\widetilde{\boldsymbol{R}}$ is first to compute the element arrays $\boldsymbol{w}^e$ and $\boldsymbol{r}^e$ and then obtain the global preconditioner and residual by performing an assembly operation, i.e.,

$$\widetilde{\widetilde{\boldsymbol{W}}} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \boldsymbol{w}^e \qquad \text{and} \qquad \widetilde{\boldsymbol{R}} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \boldsymbol{r}^e$$

where n_{el} is the number of elements. The basics of finite element programming can be found in [19]. A description of the parallel implementation of the above techniques are presented in the following section.

4. Implementational aspects

Our Fortran 90 finite element program for the Connection Machine systems was derived from a highly vectorized Fortran 77 program written by Shakib and Johan [20]. The conversion to the new Fortran standard was necessary to achieve parallel execution on the Connection Machine systems, since the CMF compiler does not recognize Fortran 77 constructs as parallel instructions. The parallel data structures chosen for the implementation and the work required for the conversion process are described in the following sections.

4.1. Parallel data structures

Appropriate data structures are essential to achieve good performance on a massively parallel computer. A description of possible data structures for finite element methods and a detailed analysis of their storage and arithmetic requirements can be found in [21]. Different data structures have also been analyzed by Farhat, *et al.* [7] for finite volume and finite element applications in computational fluid dynamics. A reduced number of data structures will limit the amount of communication required between the different data sets. Some authors have proposed a single data structure. However, having only one data set seemed cumbersome in our implementation, with a possible loss of finite element generality. Therefore, we have adopted the following two data structures:

1. At the element level, i.e., during the computation of the element arrays $\boldsymbol{w}^e$ and $\boldsymbol{r}^e$, the elements are assigned to the processing nodes of the Connection Machine systems. It is possible to have several elements per processing node, leading to the notion of virtual processing: Each processing node performs operations on a certain number of elements in a sequential fashion. The allocation of multiple elements to a node is handled by the CMF compiler, and the scheduling of the execution by the Connection Machine Run-Time System.

2. At the GMRES algorithm level, we assign the nodes of the mesh to the processing nodes with possible virtual processing. All the dot product and DAXPY operations [22] of the GMRES algorithm (as described in Box 2) are then executed in parallel over the nodes (with an additional global sum for the dot product operation).

Note that these data structures are both the most "natural" and the simplest to use in a general finite element program. Experience has shown that simplicity and efficiency are tightly coupled in data parallel programming.

All the arrays of the two data structures are allocated in the distributed memory of the Connection Machine systems through the dynamic allocation capability of Fortran 90, briefly described in Section 2.2. The mapping of array elements to the memory of the processing nodes is controlled by the `LAYOUT` directives in order to minimize communication needs. No communication is necessary in referencing array elements to the same node. As an example of the use of `LAYOUT` directives, consider the array `RES(NDOF,NUMNP)` containing the global residual. `NDOF` and `NUMNP` are the number of degrees of freedom per node and the number of nodes, respectively. The directive `RES(:SERIAL,:NEWS)` for the layout will ensure that the finite element nodes are spread as evenly as possible across the processing nodes (the `:NEWS` directive), while the degrees of freedom of each node are stored on the same processing node (the `:SERIAL` directive).

4.2. Fortran 77 to Fortran 90 conversion

The absence of dynamic memory allocation in Fortran 77 has led programmers to simulate their own dynamic allocation. This is usually done by defining a large one-dimensional array and then storing all the data in it. This feature had to be eliminated during the conversion to Fortran 90 to achieve the proper layout of the data structures described above. Each array is dimensioned in the routine where it is needed and then passed to subsequent subroutines.

The CMF compiler parallelizes only Fortran 90 array operations. Therefore, all the `DO` loops enumerating the nodes and the elements and the operations thereupon had to be replaced by array operations. This change involved merely editing work on the vectorized code. The simplicity of the conversion showed us that vectorization and data parallelism are actually two almost identical notions, both based on the concept of nonrecurrence in the operations. An operation which can be vectorized can also be parallelized.

The GMRES algorithm presented in Box 2 is implemented by distributing the workload between the Connection Machine system CM-2 or CM-200 and the front-end computer: All matrix-vector products and DAXPY operations are performed on the Connection Machine systems. However, the entries of the Hessenberg matrix, resulting from dot product operations, are stored in the memory of the front-end computer. The minimization problem solved using the Q-R algorithm is then performed on the front-end computer. This strategy is without any measurable loss in efficiency because the size of the least-squares problem is small (the dimension of the Krylov space has been in the range 5 to 15 for all the fluid flow problems we have solved). Timings have shown that the matrix-vector products account for almost all the computing cost of the GMRES algorithm, the other operations having a negligible impact.

The overall structure of the program remained identical throughout the conversion process, implying that the initial phase of the port was a fairly trivial operation. However, rewriting some parts of the code to take advantage of the Fortran 90 constructs and to optimize memory usage and execution rate [23] was a more lengthy process. The final version of the parallel code has several advantages over the Fortran 77 version: it is shorter by about 30% and easier to read due to the array syntax, promising simplified maintenance of the program. Adding new features to the program is now an easier task. We hope other hardware vendors will provide Fortran 90 compilers, and C compilers possessing similar attributes, on their computers in the near future.

5. Communication issues

Inter-processing node communication on parallel computers is often viewed as the major difficulty for programmers. Communication can represent a substantial part of the total run-time, thus affecting the overall efficiency of the program. The current CMF compiler does not recognize special forms of communication to translate them into optimal code. Optimizing communication is indeed a nontrivial task, and the subject of leading-edge compiler research [24]. Meanwhile optimizing routing of data between processing nodes requires specialized routines written in a low-level language. But, many communication operations are generic, and communication libraries have emerged as means of providing both efficiency and portability.

The following sections describe the gather and scatter operations, which are the only two types of communications performed by our finite element program. The issue of mapping the data onto the processing nodes is also discussed. Finally, a performance comparison of the possible options for the gather/scatter operations on the Connection Machine system CM-2 is presented.

5.1. Gather operation

The computation of the element data $\boldsymbol{w}^e$ and $\boldsymbol{r}^e$ presented in Section 3.2 requires the knowledge of the current solution at the element nodes $\boldsymbol{v}^e$. The vector $\boldsymbol{v}^e$ is obtained by gathering the values from the nodal solution vector $\widetilde{\boldsymbol{v}}$. This simply consists of an indirect addressing via the mesh connectivity array. The gather operation is sometimes referred to as localization, or accumulation. In Fortran 90, this operation can be written

```
DO I = 1, NEN
  DO N = 1, NDOF
    VL(I,N,:)  = V(N,IEN(I,:))
  END DO
END DO
```

where `VL(NEN,NDOF,NUMEL)` is the array containing $\boldsymbol{v}^e$; `V(NDOF,NUMNP)` contains $\tilde{\boldsymbol{v}}$ and `IEN(NEN,NUMEL)` is the mesh connectivity array (see Hughes [19]). The scalar `NUMEL` is the number of elements; and `NEN` is the number of element nodes, e.g., `NEN` $= 4$ for a linear tetrahedron.

The above code fragment is recognized for parallel execution over the elements by the CMF compiler. However, the executable code generated by the compiler will call for the router to compute the addresses of the data to be gathered each time such an operation is required. The trace of the routing activity, i.e., the paths of all elements being moved, is the same for every gather operation as long as the connectivity and its layout do not change. Hence, the addresses and the routing information only need to be computed once for a given connectivity, and the information stored and reused for subsequent gather operations. The time expended in computing the routing information amounts to a couple of gather operations. Hence, saving the routing information also yields a performance improvement for slowly changing connectivities. The saving in communication time is achieved at the expense of some additional storage required to save the trace.

The CMSSL (Connection Machine Scientific Software Library) [25] primitives `sparse_util_gather_setup` and `sparse_util_gather` are used to compute and save the trace, and to perform the actual communication. The performance of these routines for general finite element applications is given in [26] along with a description of the methodology implemented in the primitives. The gather algorithm used in this implementation is a two-step process: First, the nodal data are duplicated as many times as there are elements connected to each node. Then, these duplicated data are sent through a one-to-one mapping to the corresponding elements. The preprocessing step computes this one-to-one communication pattern. In our finite element program we use a slightly modified version of the CMSSL `sparse_util_gather` routine. The modification facilitates the simultaneous handling of several degrees of freedom per node.

5.2. Scatter operation

Once the element data $\boldsymbol{w}^e$ and $\boldsymbol{r}^e$ are computed using a so-called "embarrassingly parallel algorithm" (i.e., one for which *no* communication is required), their components are scattered to the nodes (also said to be "assembled at the nodes") to evaluate the global preconditioner $\widetilde{\boldsymbol{W}}$ and residual $\widetilde{\boldsymbol{R}}$. The scatter operation is a send operation with addition of the colliding data at the nodes. The assembly of the residual can be written

```
      DO I = 1, NEN
        DO N = 1, NDOF
          FORALL (NEL = 1:NUMEL) RES(N,IEN(I,NEL)) = RES(N,IEN(I,NEL))
     &                                            + RL(I,N,NEL)
        END DO
      END DO
```

where `RL(NEN,NDOF,NUMEL)` is the element residual array.

The scatter operation presented above is not parallelized by the current CMF compiler. However, several alternatives are available to the programmer:

1. A coloring technique often used on vector computers to implement the scatter operation can also be used here. Such techniques are described in [15] and [27]. The idea is to decompose the mesh into blocks of disjoint elements. It can be easily shown that the number of blocks equals the maximum number of elements connected to a node. Consequently, this method is not suitable on the Connection Machine systems when the number of blocks becomes large (large tetrahedral meshes often require up to 100 blocks), because the routing activity corresponding to each block is not load-balanced. It will therefore not be analyzed in the remainder of this paper.

2. The CMF utility library provides a `CMF_send_add` routine. This routine uses the combining facility of the communication hardware, allowing all data to be scattered in parallel while the required additions are being performed in parallel. It is considered as the reference case for assembly operations in our performance benchmarks.

3. As with the gather operation, the routing activity corresponding to the nodal connectivity can be precomputed and the information used subsequently. The CMSSL routines `sparse_util_scatter_setup` and `sparse_util_scatter` [25] perform the preprocessing and the assembly, respectively. The scatter operation is done in a fashion similar to the gather operation: First, the element data are sent to the nodes using a one-to-one mapping, i.e., two or more data values arriving in the same node at the same time are stored in different memory locations on the same processing node. Then, all the values at a node are added up. For some finite element applications, this two-step procedure has been shown to be more efficient than the combining feature of the Connection Machine systems CM-2 and CM-200 router. The CMSSL `sparse_util_scatter` routine was modified in a way similar to the gather routine to simultaneously handle multiple degrees of freedom.

These gather/scatter communication procedures are very general and can be used for any finite element application. The efficiency of the different options described above are presented in Section 5.4.

5.3. Mapping

"Mapping" the elements and the nodes onto the distributed memory of the Connection Machine systems is the procedure that determines on which processing node the element- or node-based data will be stored. The mapping of the data can affect the performance of the communication routines presented in the previous sections by reducing communication channel and router contentions. Our objective is to find a suitable mapping procedure for unstructured finite element problems.

The `LAYOUT` directives provide some means for controlling the mapping at compile-time. The user can also control the data mapping by renumbering the nodes and the elements of the finite element mesh. Mapping techniques attempting to preserve locality for the Connection Machine systems, i.e, attempting to map interconnected nodes and elements to processing nodes close to each other, have been developed by Farhat, *et al.* [5] and Hammond and Schreiber [28]. These techniques are deterministic. Stochastic techniques are often used for very complex optimization problems. Simulated annealing is one such technique [29], which has been applied to data mapping on the Connection Machine systems CM-2 and CM-200 by Dahl [30]. Although these methods can generate very efficient mappings, the computation time required to compute a mapping may represent a substantial part of the total processing time. This fact makes them unsuitable for the types of applications we consider, since adaptive remeshing may be necessary.

Another approach to reduce the contention and minimize the communication time is to use randomized routing as proposed by Valiant, *et al.* [31, 32], or a random mapping as proposed by Ranade, *et al.* [33, 34]. In randomized routing, data is sent to a random location before being sent to the final destination. It can be shown that the risk of severe contention is extremely small for such a routing [31]. In [33], it is shown that a random allocation achieves the same goal for any deterministic, direct routing scheme between source and destination. The randomized allocation is an option in the CMSSL arbitrary sparse BLAS functions. Similar utility functions were used for the finite element application reported in [26]. The short preprocessing time for the randomized mapping makes it suitable for a wide range of computational fluid dynamics applications, from steady computations to arbitrary Lagrangian-Eulerian calculations. However, randomized mappings are not necessarily optimal for finite element problems, and the search for other mappings will be the subject of future research.

5.4. Comparison of various communication algorithms

A simple fluid flow example was chosen to illustrate the performance of the different communication strategies presented in the previous sections. It consists of a Mach 2 inviscid flow over a wedge. The problem is described by Figure 2. The computational domain (x, y, z) was discretized using $32 \times 2 \times 96$ trilinear bricks. A $2 \times 2 \times 2$ Gaussian integration rule was used on each element. Ten time steps at a CFL number of 10 were performed using the matrix-free GMRES algorithm to obtain convergence to steady-state. This test case was solved in double precision on a 128-processing node Connection Machine system CM-2 running CMSS version 6.0 (Connection Machine System Software) and CMF version 1.0.

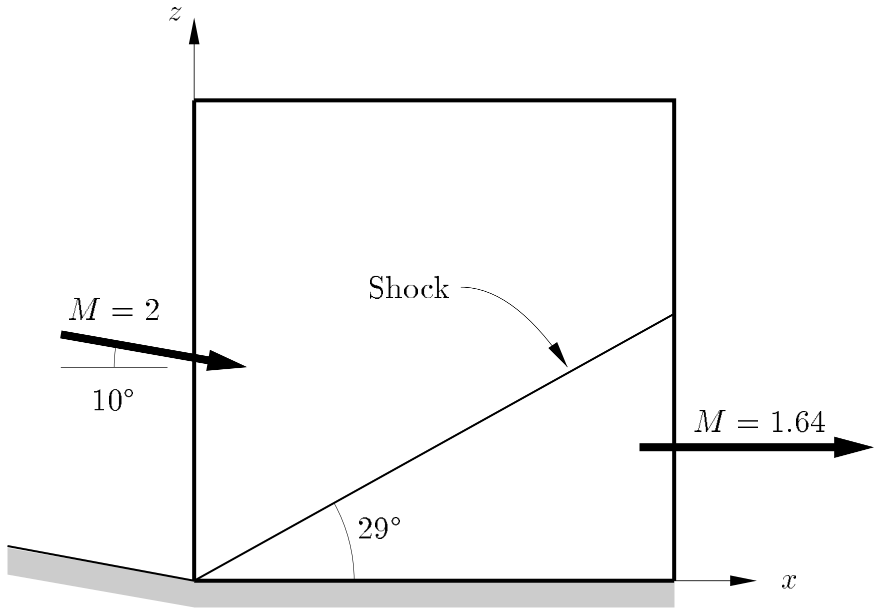

Figure 2. Mach 2 oblique shock. Problem schematics.

Computation and communication times are reported in Table 1. The first row corresponds to a calculation using the Fortran 90 code segment presented in Section 5.1 for the gather operation and the `CMF_send_add` utility routine for the scatter operation. The CMSSL communication routines were used in the computation whose timings are presented in the second row of the table. In row 3, a randomized mapping was used together with the CMSSL routines. Speedups of factors of 2 for the gather operation and 5 for the scatter operation were obtained using the CMSSL routines without randomization of the data allocation, increasing substantially the overall efficiency of the finite element program. Applying the random mapping to the data improved the communication times by an additional 20 to 25%. The good performance of the CMSSL primitives led us to solve the more complex flows presented in the next sections.

Table 1. Mach 2 oblique shock. Computation and communication elapsed times on a 128-processing node Connection Machine system CM-2 using an 8-point integration rule.

	Gather	Computation	Scatter
indirect addr. / `CMF_send_add`	127 s	237 s	209 s
CMSSL (no randomization)	62 s	237 s	42 s
CMSSL (random mapping)	46 s	237 s	33 s

6. Numerical examples and benchmarks

Three-dimensional compressible flow problems were solved using the techniques presented in the previous sections to further evaluate their performance. A random mapping was used for all cases, as well as a local time-stepping strategy associated with the matrix-free GMRES algorithm. The dimension of the Krylov space was set to 5 and the tolerance ε_{tol} equaled 0.1. All examples were computed in double precision. A comparison with the Fortran 77 version of the code running on vector computers was made when possible. All computations were initiated with the free stream flow. No attempt was made to project coarse mesh solutions on refined meshes. It is believed that this approach would have considerably shortened the solution times. In addition, it is believed that multigrid methods are capable of dramatically reducing solution times. These will be studied in future work.

6.1. Three-dimensional blunt body

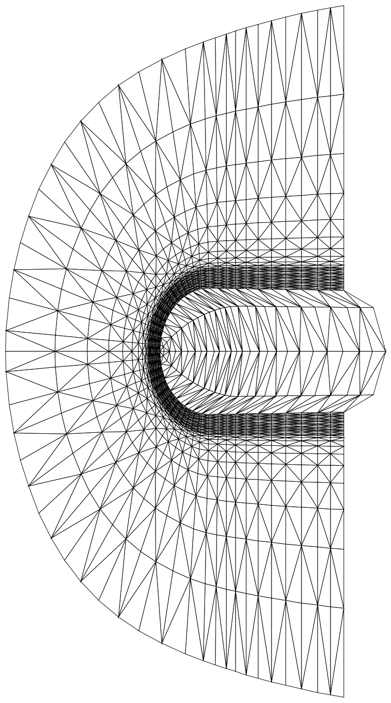

Figure 3. 3-D blunt body.
Surface mesh of body
and plane of symmetry.

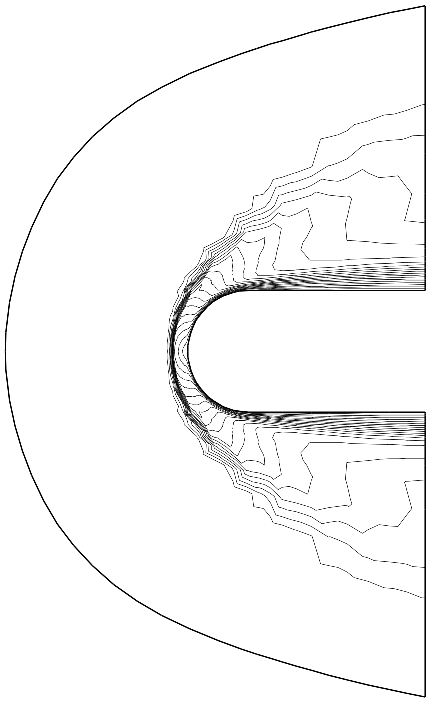

Figure 4. 3-D blunt body.
Mach number contours
in the plane of symmetry.

This example consists of a Mach 3 viscous flow around a blunt body made of a half-sphere extended by a cylinder. The angle of attack is 0 degree and the Reynolds number is 1,000 based on the radius of the sphere. The computation was only performed for half the body since the flow is symmetric. The mesh contains 3,566 nodes and 13,280 tetrahedra. A 4-point integration rule was used in the elements. A view of the symmetry plane and half-body is shown in Figure 3. This problem was solved on a 128-processing node Connection Machine system CM-2 running CMSS version 6.0 and CMF version 1.0. The Mach number contours in the symmetry plane after 250 time steps are depicted in Figure 4. One can note the bow shock and the development of the boundary layer on the body. Timings for the first 20 time steps are presented in Table 2. The same problem was also solved on a Convex C-1 using the vectorized version of the finite element program. No timings are reported for the gather/scatter on the Convex C-1, because their vectorization has made them a negligible part of the total time. Several remarks can be made:

1. The 128-processing node Connection Machine system CM-2 is about 12 times faster than the Convex C-1, bringing this Connection Machine system configuration to the performance level of a one-CPU Cray 2.
2. The computation time accounts for two-thirds of the total time. The ratio between computation and communication times is a function of the number of integration points in the elements. If 1-point element quadrature had been used, the computation time would have been substantially smaller. However, the communication time would have remained the same since it is only a function of the number of nodes and elements. This fact is evident in some of the following performance results.

Table 2. 3-D blunt body. Computation, communication and total elapsed times on a 128-processing node Connection Machine system CM-2 and a Convex C-1 using a 4-point integration rule.

	Gather	Computation	Scatter	Total
CM-2 (CMSS 6.0)	39 s	170 s	44 s	253 s
Convex C-1	—	—	—	3003 s

6.2. Falcon Jet at level cruise

A transonic inviscid flow at Mach 0.85 was computed around a generic Falcon Jet at a 1 degree angle of attack. The Falcon Jet airplanes are designed and built by Dassault Aviation. The calculation was only done on half the airplane since the flow is symmetric.

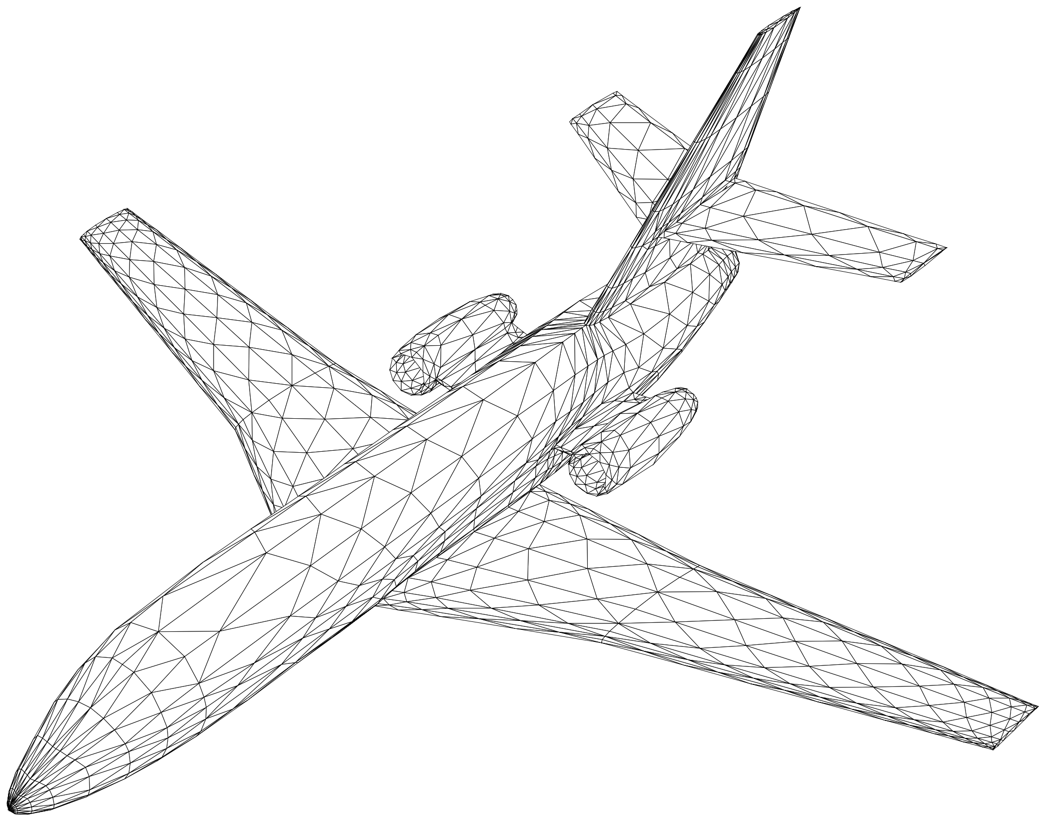

Figure 5. Falcon Jet. Coarse surface mesh.

The first mesh used had 10,202 nodes and 54,957 tetrahedra. The surface mesh for the whole jet can be seen in Figure 5. A 4-point integration rule was used on the elements. A converged solution was obtained after 50 time steps at a CFL number of 10. This problem was solved on a 512-processing node Connection Machine system CM-2 as well as on a one-CPU Cray Y-MP and a Convex C-1. The timings are presented in Table 3. First, a comparison was made between versions 6.0 and 6.1 of CMSS, showing a 15% speedup of the total execution time when upgrading the software. Most of the gain is due to faster communication. CMSS 6.1 yielded a 19-minute run time on the Connection Machine system CM-2 versus 39 minutes on the one-CPU Cray Y-MP. The execution rate on the

Cray Y-MP, measured using the hardware performance monitor, is 178 MFlops/s. Hence, an effective performance of 370 MFlops/s is achieved on a 512-processing node Connection Machine system CM-2 running CMSS version 6.1. A mini-supercomputer like the Convex C-1 is not an option for these types of computations.

Table 3. Falcon Jet at level cruise (coarse mesh). Computation, communication and total elapsed times on a 512-processing node Connection Machine system CM-2, a one-CPU Cray Y-MP and a Convex C-1 using a 4-point integration rule.

	Gather	Computation	Scatter	Total
CM-2 (CMSS 6.0)	236 s	897 s	234 s	22 min 47 s
CM-2 (CMSS 6.1)	148 s	833 s	150 s	18 min 51 s
Cray Y-MP	—	—	—	39 min 26 s
Convex C-1	—	—	—	20 h 42 min

The same problem was solved on a 2,048-processing node Connection Machine system CM-2 running CMSS 6.0 using a finer mesh. It has about 8 times more data than the previous mesh, with 77,279 nodes and 439,272 elements (see the surface mesh for the complete airplane in Figure 6). The number of equations equals 386,395. The Mach number contours are shown in Figures 7 and 8. Note the supersonic pockets on the outward part of the wings followed by recovery shocks (adaptive mesh refinement is necessary to better resolve the latter). The quality of the calculation can also be deduced from the Mach number contours at the top of the cockpit. For well-designed airplanes cruising at transonic speeds (the Falcon Jet family of airplanes falls in that category), the Mach number on the cockpit remains just below the sonic point during cruise. Supersonic pockets would be followed by recovery shock waves, generating additional unwanted drag. The computation, done under the same conditions as the one on the coarse mesh, took 47 minutes and 50 s, indicating that scalability is achieved since the ratio (elapsed time)$\times$(number of processing nodes)/(number of elements) is very close to that for the previous case.

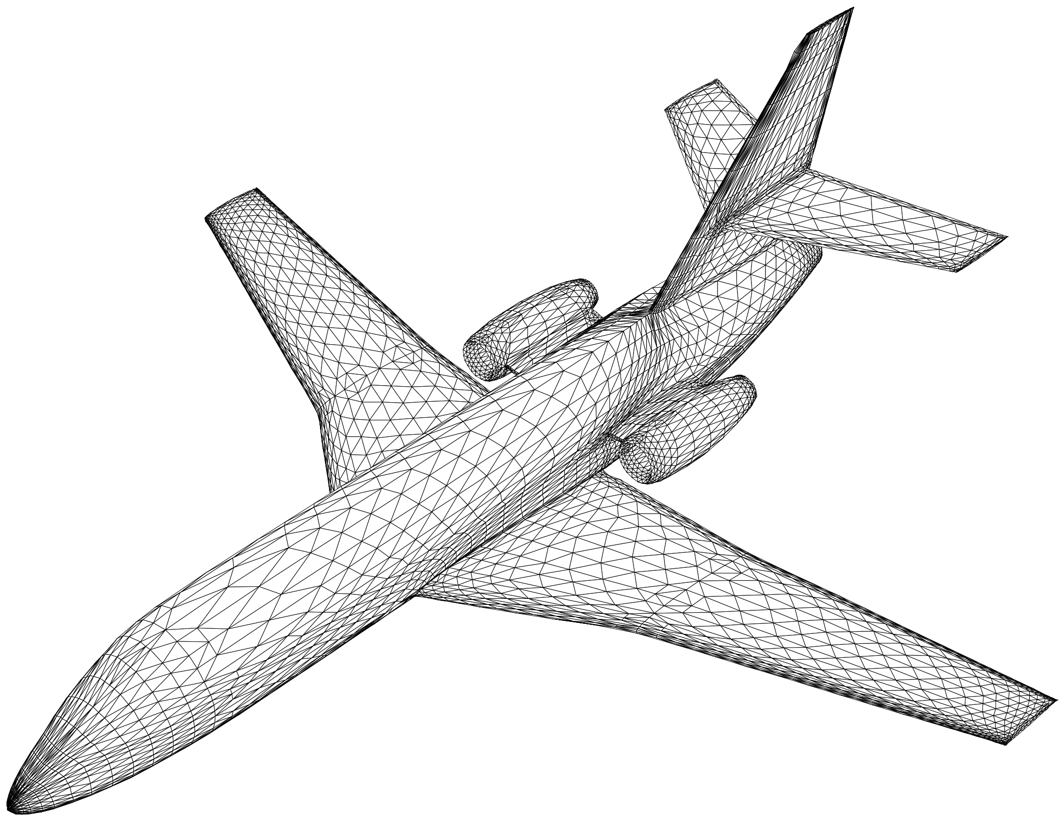

Figure 6. Falcon Jet. Fine surface mesh.

Finally, the fine mesh was used to compare the performances of a 2,048-processing node Connection Machine system CM-2 running CMSS version 6.0 and a system CM-200 running CMSS version 6.1. A 1-point element integration rule was used and 60 time steps at a CFL number of 20 were calculated. The timings are reported in Table 4. The CM-200 running CMSS version 6.1 offered a performance about twice that of the CM-2 running CMSS version 6.0. Note that the ratio between computation and communication has decreased compared to previous computations. This decrease is due to the reduction in the number of integration points per element, as described in Section 6.1.

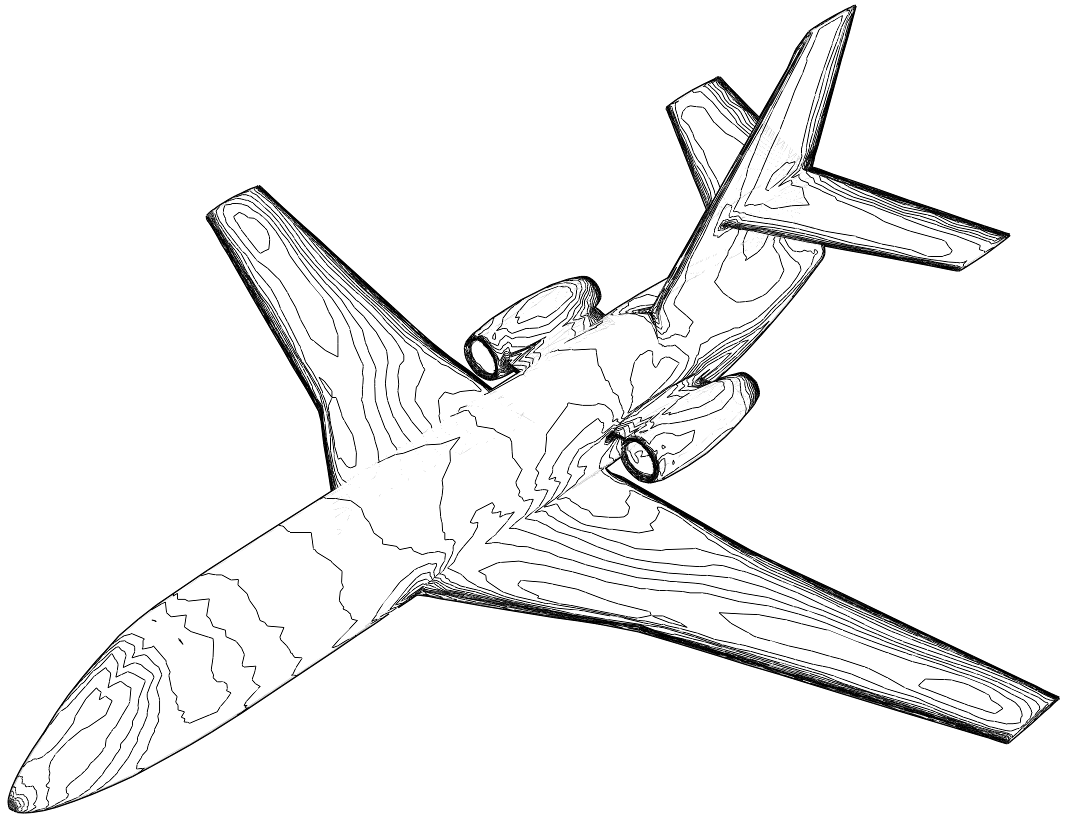

Figure 7. Falcon Jet at level cruise. Mach number contours.

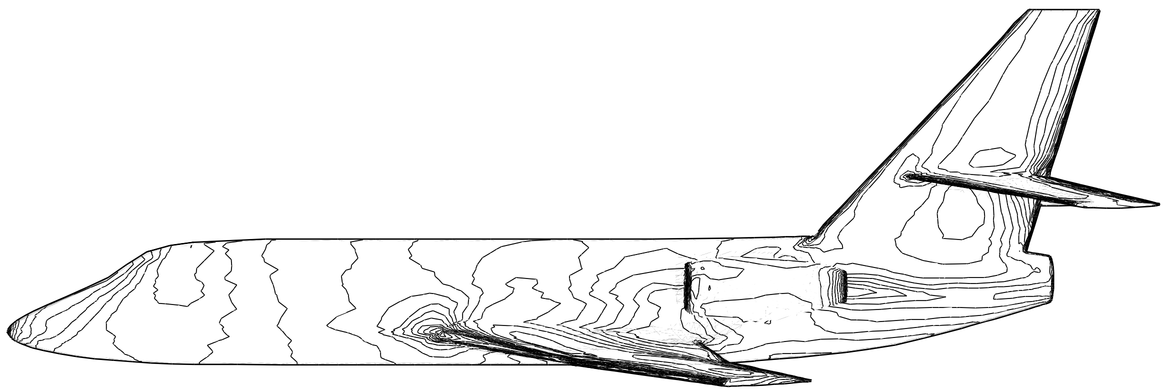

Figure 8. Falcon Jet at level cruise. Mach number contours.

Table 4. Falcon Jet at level cruise (fine mesh). Computation, communication and total elapsed times on 2,048-processing node Connection Machine system CM-2 and CM-200 using a 1-point integration rule.

	Gather	Computation	Scatter	Total
CM-2 (CMSS 6.0)	1051 s	1326 s	967 s	55 min 44 s
CM-200 (CMSS 6.1)	453 s	800 s	415 s	27 min 48 s

6.3. Falcon Jet in a crosswind

This last example is an excellent illustration of how the Connection Machine systems can be used as a design tool by the aerodynamicist. The complex flow around the Falcon Jet in a 120-knot approach configuration with a 15-knot crosswind component was solved on a 2,048-processing node Connection Machine system CM-200 running CMSS version 6.1 and CMF version 1.1. The airplane is approaching on a 3-degree glide slope with a 3-degree nose-up attitude, generating an angle of attack of 6 degrees. A 7.2-degree yaw angle is applied to allow the airplane to remain on the localizer. The outside temperature is 15 °C, yielding a Mach number of 0.18. The descent profile and the airplane attitude are shown in Figure 9. Note that the wind sock in Figure 9 indicates a right crosswind. Therefore, the flow comes from the left in a reference frame attached to the airplane. The mesh generated for this calculation has 150,724 nodes and 878,544 tetrahedra (see Figure 6 for the surface mesh). The number of equations equals 753,620. The flow field had to be computed around the complete airplane since it is nonsymmetric. A 1-point integration rule was used and 250 time steps were performed at a CFL number of 5. The timings can be found in Table 5. The Mach number contours on the jet are presented in Figures 10 and 11.

Valuable information can be deduced from such a computation. For example, an

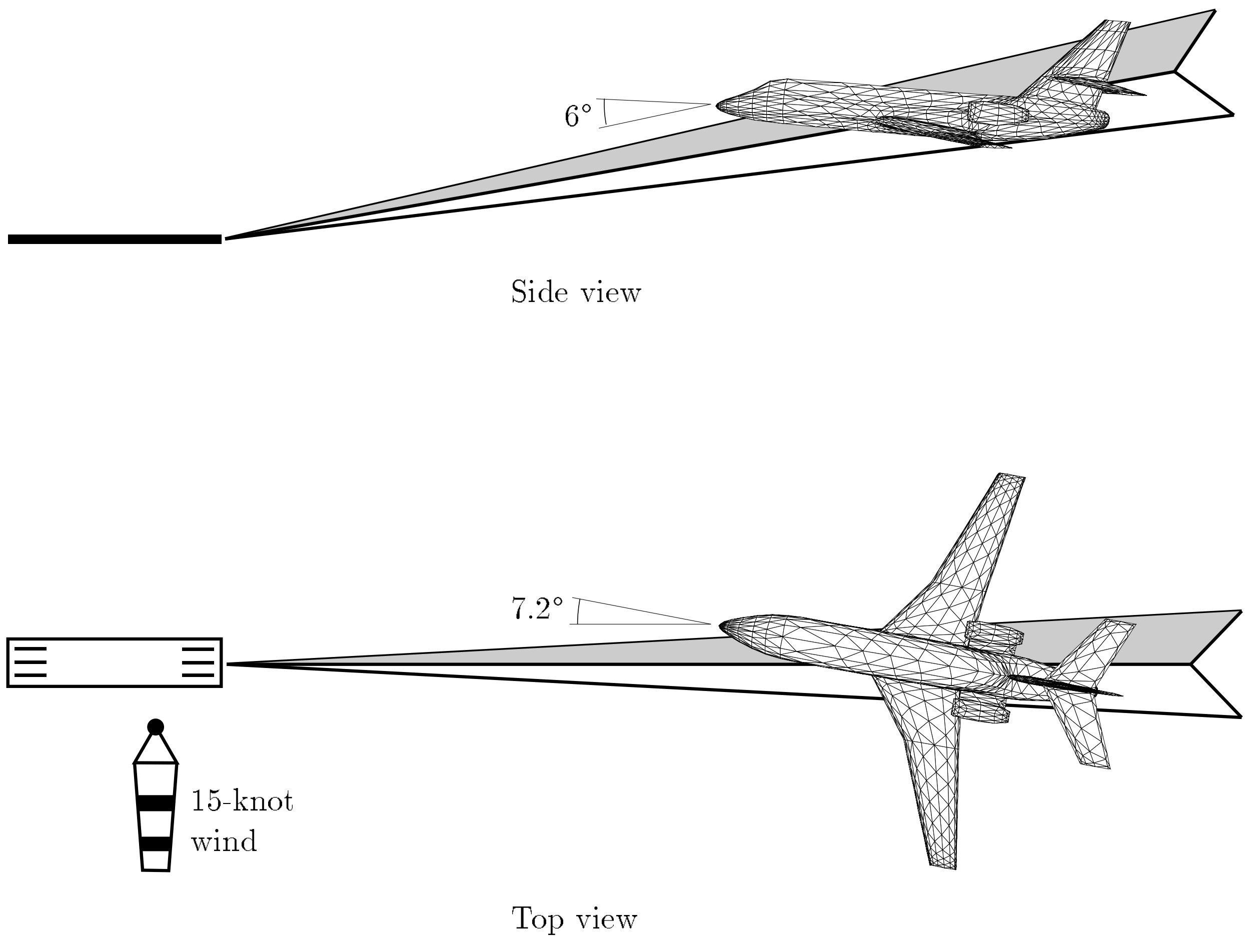

Figure 9. Descent profile of the Falcon Jet in a crosswind. The wind direction is indicated by the wind sock on the top view.

approximation of the aerodynamic forces and moments can be calculated for a study of the flight stability in that configuration. The air mass flux in the engine intakes can also be computed, allowing the aerodynamicist to determine how the engines will perform in a strong crosswind, especially the leeward side engine. Most important of all is that such computations can be done in 1 to 2 hours on a 2,048-processing node Connection Machine system CM-200. Several configurations can therefore be evaluated per day, accelerating considerably the design of the aircraft. The next generation of massively parallel super-

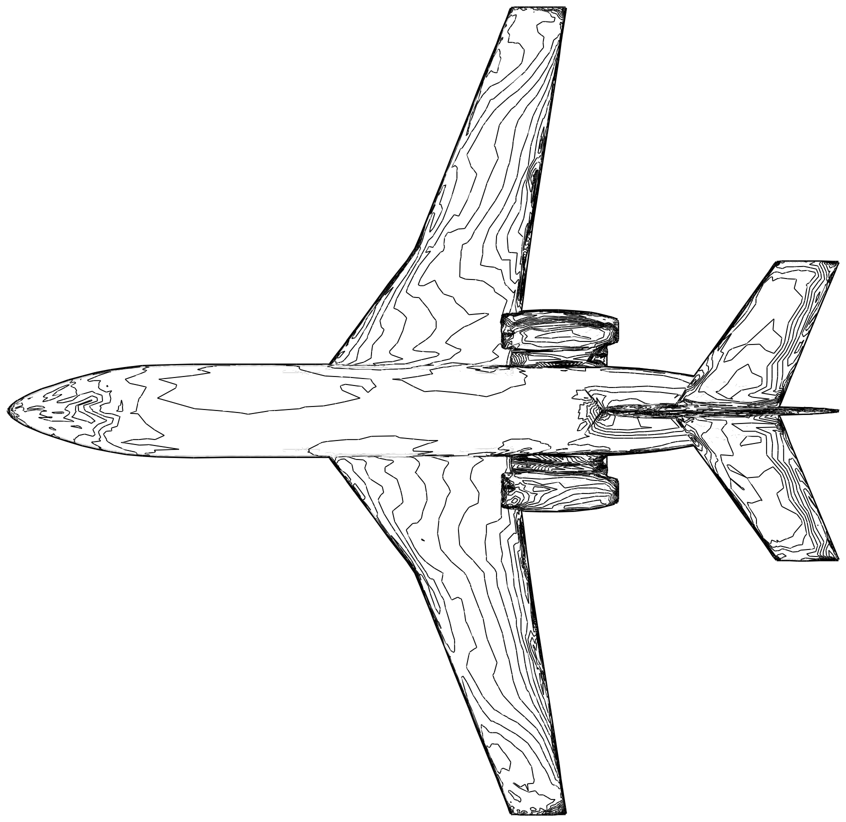

Figure 10. Falcon Jet during a crosswind approach. Mach number contours.

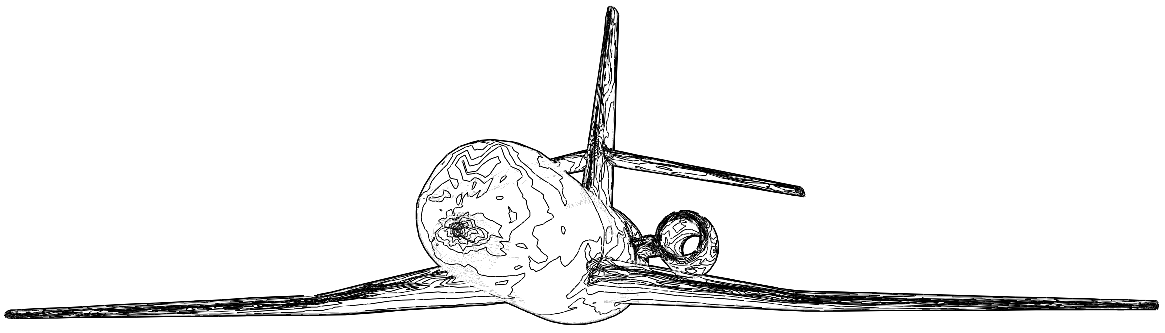

Figure 11. Falcon Jet during a crosswind approach. Mach number contours viewed from the free stream direction.

computers should accommodate data parallel computational aerodynamics codes capable of automated optimal design, as computation time for full airplane configurations promises to be reduced to at most a few minutes.

Table 5. Falcon Jet during approach. Computation, communication and total elapsed times on a 2,048-processing node Connection Machine system CM-200 using a 1-point integration rule.

	Gather	Computation	Scatter	Total
CM-200 (CMSS 6.1)	1331 s	2055 s	1490 s	1 h 21 min

7. Conclusions

An implicit iterative finite element solver for computational fluid dynamics has been ported to the Connection Machine systems CM-2 and CM-200 using a data parallel style of programming. The use of Fortran 90 constructs simplified the implementation process, leading to a clearer and better structured program. Communication routines provided through the Connection Machine Scientific Software Library improved substantially the overall efficiency. Performance benchmarks on industrial examples having close to one million degrees of freedom are the final proof that data parallel programming on the Connection Machine systems is suitable for solving CFD problems on unstructured meshes.

All techniques presented in this paper are open for considerable improvement. During the course of the implementation and benchmarking of our finite element code, the performance of some of the system software functions improved by 50% or more. Moreover, improved CMF compilers will appear in the near future, generating faster code at the processing node level. Improvements in mapping and communication procedures are also possible. It is the current thrust of our research. Last, but not least, is the evolution in hardware technology. A recent example is the announcement of the massively parallel Connection Machine system CM-5, which is expected to give an order of magnitude improvement in performance over the CM-2. Initial work on this system has shown great promise for our methodology. In the coming years, we anticipate developments in massively parallel supercomputing to completely revolutionize computational engineering analysis and design.

Acknowledgments

The authors would like to express their appreciation to Steve Daly, John Kennedy and Arthur Raefsky for their interest in this work and their helpful comments. This research was supported by the NASA Langley Research Center under Grant NASA-NAG-1-361 and Dassault Aviation, Saint Cloud, France. The blunt body and Falcon Jet meshes were generated by Dassault Aviation. Access to Connection Machine systems was provided by Thinking Machines Corporation, the Los Alamos Advanced Computing Laboratory and the Department of Geophysics at Stanford University. Access to a Cray Y-MP was provided by the San Diego Supercomputer Center.

References

[1] "Grand Challenges: High performance computing and communications. The FY 1992 US research and development program," Report by the Committee on Physical, Mathematical, and Engineering Sciences; Federal Coordinating Council for Science, Engineering and Technology; Office of Science and Technology Policy.

[2] K.K. Mathur and S.L. Johnsson, "The finite element method on a data parallel computing system," *International Journal of High Speed Computing*, **1** (1989) 29–44.

[3] S.L. Johnsson and K.K. Mathur, "Experience with the conjugate gradient method for stress analysis on a data parallel supercomputer," *International Journal for Numerical Methods in Engineering*, **27** (1989) 523–546.

[4] T. Belytschko, E.J. Plaskacz, J.M. Kennedy and D.L. Greenwell, "Finite element analysis on the Connection Machine," *Computer Methods in Applied Mechanics and Engineering*, **81** (1990) 229–254.

[5] C. Farhat, N. Sobh and K.C. Park, "Transient finite element computations on 65,536 processors: The Connection Machine," *International Journal for Numerical Methods in Engineering*, **30** (1990) 27–55.

[6] R.A. Shapiro, "Implementation of an Euler/Navier-Stokes finite element algorithm on the Connection Machine," *AIAA 29th Aerospace Sciences Meeting*, AIAA-91-0438, 1991.

[7] C. Farhat, L. Fézoui and S. Lantéri, "Computational fluid dynamics with irregular grids on the Connection Machine," *Computer Methods in Applied Mechanics and Engineering*, to appear.

[8] *Connection Machine model CM-2 technical summary, Version 6.0*, Thinking Machines Corporation, Cambridge, MA, 1990.

[9] *The Connection Machine CM-200 series technical summary*, Thinking Machines Corporation, Cambridge, MA, 1991.

[10] S.L. Johnsson, "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures," *Journal of Parallel and Distributed Computing*, **4** (1987) 133–172.

[11] M. Metcalf and J. Reid, *Fortran 90 explained*, Oxford University Press, Oxford, Great Britain, 1990.

[12] *CM Fortran reference manual, Versions 1.0 and 1.1*, Thinking Machines Corporation, Cambridge, MA, 1991.

[13] F. Shakib, T.J.R. Hughes and Z. Johan, "A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations," *Computer Methods in Applied Mechanics and Engineering*, **89** (1991) 141–219.

[14] Z. Johan, T.J.R. Hughes and F. Shakib, "A globally convergent matrix-free algorithm for implicit time-marching schemes arising in finite element analysis in fluids," *Computer Methods in Applied Mechanics and Engineering*, **87** (1991) 281–304.

[15] F. Shakib, T.J.R. Hughes and Z. Johan, "A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis," *Computer Methods in Applied Mechanics and Engineering*, **75** (1989) 415–456.

[16] Y. Saad and M.H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal of Scientific and Statistical Computing*, **7** (1986) 856–869.

[17] L.B. Wigton, N.J. Yu and D.P. Young, "GMRES acceleration of computational fluid dynamics codes," *AIAA 7th Computational Fluid Dynamics Conference*, AIAA CP854, 1985.

[18] M. Mallet, J. Périaux and B. Stoufflet, "Convergence acceleration of finite element methods for the solution of the Euler and Navier-Stokes equations of compressible flow," *Notes on Numerical Fluid Mechanics*, Vieweg, **20** (1988) 199–210.

[19] T.J.R. Hughes, *The finite element method: Linear static and dynamic finite element analysis*, Prentice-Hall, Englewoods Cliffs, NJ, 1987.

[20] F. Shakib, Z. Johan and T.J.R. Hughes, "ENSA-3C: A space-time Galerkin/least-squares finite element program to analyze the compressible Euler and Navier-Stokes equations for general divariant gases," *User's Manual*, Stanford University, Stanford, CA, 1990.

[21] S.L. Johnsson and K.K. Mathur, "Data structures and algorithms for the finite element method on a data parallel supercomputer," *International Journal for Numerical Methods in Engineering*, **29** (1990) 881–908.

[22] C.L. Lawson, R.J. Hanson, D.R. Kincaid and F.T. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM TOMS*, **5** (1979) 308–323.

[23] G. Sabot, J. Marantz and D. Gingold, "CM Fortran optimization notes: Slicewise model," ***Technical Report***, TMC-166, Thinking Machines Corporation, Cambridge, MA, 1991.

[24] M. Chen, Y.-I. Choo and S.L. Johnsson, ***Compiler Technology for Massively Parallel Architectures***, DARPA Contract to Yale University and Thinking Machines Corporation, 1991.

[25] ***CMSSL for CM Fortran, Version 2.2***, Thinking Machines Corporation, Cambridge, MA, 1991.

[26] K.K. Mathur, "On the use of randomized address maps in unstructured three-dimensional finite element simulations," ***Technical Report***, TMC-37/CS90-4, Thinking Machines Corporation, Cambridge, MA, 1990.

[27] R.M. Ferencz, "Element-by-element preconditioning techniques for large-scale, vectorized finite element analysis in nonlinear solid and structural mechanics," ***Ph.D. Thesis***, Stanford University, Stanford, CA, 1989.

[28] R. Schreiber and S. Hammond, "Mapping unstructured grid problems to the Connection Machine," ***RIACS Technical Report***, 90.22, 1990.

[29] S. Kirkpatrick, C.D. Gellatt, Jr., and M.P. Vecchi, "Optimization by simulated annealing," ***Science***, **220** (1983) 671–680.

[30] E.D. Dahl, "Mapping and compiled communication on the Connection Machine System," ***Proceedings of the 5th Distributed Memory Computing Conference***, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[31] L. Valiant, "A scheme for fast parallel communication," ***SIAM Journal of Computing***, **11** (1982) 350–361.

[32] L. Valiant and G.J. Brebner, "Universal schemes for parallel communication," ***Proceedings of the 13th ACM Symposium on the Theory of Computation***, ACM, (1981) 263–277.

[33] A.G. Ranade, "How to emulate shared memory," ***Proceedings of the 28th Annual Symposium on the Foundations of Computer Science***, IEEE Computer Society Press, Los Alamitos, CA, (1987) 185–194.

[34] A.G. Ranade, S.N. Bhatt and S.L. Johnsson, "The fluent abstract machine," Advanced Research in VLSI, ***Proceedings of the 5th MIT VLSI Conference***, MIT Press, Cambridge, MA, (1988) 71–93.