Communication Primitives for
Unstructured Finite Element Simulations
on Data Parallel Architectures

Kapil K. Mathur
S. Lennart Johnsson

TR-23-92
November 1992

Parallel Computing Research Group
Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

This work has in part been supported by the Air Force Office of Scientific Research under contract AFOSR-89-0382, and in part by NSF and DARPA under contract CCR-8908285.
Communication Primitives for Unstructured Finite Element Simulations on Data Parallel Architectures

Kapil K. Mathur and S. Lennart Johnsson

Thinking Machines Corp.
245 First Street
Cambridge, MA 02142


Abstract

Efficient data motion is critical for high performance computing on distributed memory architectures. The value of some techniques for efficient data motion is illustrated by identifying generic communication primitives. Further, the efficiency of these primitives is demonstrated on three different applications using the finite element method for unstructured grids and sparse solvers with different communication requirements. For the applications presented, the techniques advocated reduced the communication times by a factor of between 1.5 – 3.

1 Introduction

The finite element method is a popular technique for solving boundary and initial value problems. Moderate sized engineering problems have been successfully simulated using this technique. The primary bottleneck for the simulation of large problems has been available computational resources. With the advent of massively parallel architectures, simulating significantly larger
problems is now possible. The inherent parallelism in the finite element method makes it very attractive for massively parallel architectures. However, the data structures that must be used to exploit the new computing architectures efficiently may be quite different from the data structures that have been used on traditional sequential architectures.

Two important issues that must be addressed on massively parallel architectures is load balance and the efficient use of the communication network that connects the processing nodes. A high overall efficiency can only be achieved with good use of the communication network. The data allocation and the algorithms must be chosen to preserve locality of reference, and to minimize contention in the network. These issues are particularly critical for unstructured discretizations.

The reader is referred to articles by Johnsson and Mathur [1989, 1990], Belytschko et al. [1990], Farhat et al. [1990 and 1992], Shapiro [1991], and Johan et al. [1992] for several different data parallel finite element algorithms. This article identifies a common set of communication primitives in the data parallel programming model which express most of the communication needs for unstructured finite element simulations. The need for such primitives is twofold. First, although most of these primitives can be expressed in high level data parallel programming languages, such as Fortran–90 [Metcalf, 90], Fortran–D [Fox, 91], and Fortran Y [Chen, 92], compilers are currently not capable of exploiting the network interconnecting the processing nodes optimally. High–level communication primitives allow communication operations to be implemented optimally as functions callable from the language of choice or by the compiler. The communication functions, if implemented well, provide optimal selection of paths for moving data, optimal schedules for moving the data along these paths, and highly–tuned implementations. Finally, it is shown that some of the primitives cannot be expressed by the available high level data parallel programming languages. Availability of the communication primitives suggested here, on different computing platforms, should significantly enhance portability of application codes both with respect to the programming effort and the achieved performance.

Three different finite element applications are used as examples illustrating the intended use of these communication functions. The Connection Machine systems, CM–2/200, have been used as model platforms for these applications. It is emphasized that although this article presents results on the Connection Machine systems, CM–2 and CM–200, these applications are
easily portable to any architecture with compilers that are based on a global address space (Fortran-90, Fortran-D, and Fortran-Y).

All three example applications are based on three-dimensional unstructured discretizations. The first example application is a three-dimensional, fully dynamic finite element formulation that has been used to study the rate dependent three-dimensional deformation in the Charpy V-notch test [Mathur et al., 1992a, Tvergaard and Needleman, 1986 and 1988]. This is an explicit formulation in which the desired solution phase is spread out over several thousand time increments. No solution of sparse linear systems is required. Since the mesh connectivity remains fixed for the entire simulation period, it is beneficial to preprocess the communication patterns and amortize the preprocessing cost over the several thousand time steps.

The second application is a three-dimensional deformation processing simulation where the material response is based on polycrystalline plasticity [Beaudoin et al., 1992, Mathur et al., 1992b and Dawson et al., 1992]. This application involves a solution of a nonlinear sparse system of equations at each time increment. Further the material behavior at each sampling point is based on a second level of discretization which models a material point on a continuum scale as a collection of crystals. This is an implicit-explicit formulation. The deformation field at a fixed state is computed implicitly. The mechanical state of the material and the geometry is then updated explicitly. There are several interesting levels of data interaction between the two discretizations representing two significantly different length scales.

Finally, the third application is from computational fluid dynamics where the steady state Euler flow on a complete airplane is computed for a mesh involving almost a million degrees of freedom [Johan et al., 1992]. This is a fully implicit formulation based on a preconditioned matrix free restarted GMRES algorithm [Saad and Schultz, 1986].

2 Data parallel finite element mappings

The finite element simulations that are described here make no assumptions about the topology of the mesh. Although the results presented here are based on meshes which are composed of only one type of elements, the set of communication functions described here can be used for meshes comprising of different types of elements. Other equally important issues such as effective
utilization of the processing nodes in the data parallel architecture must also be taken into account for meshes composed of several different types of elements.

The data parallel implementation for completely arbitrary finite element formulations is best based on two different “views” of the data set. In the first view, one unassembled finite element is mapped onto the processing node of the data parallel architecture. When there are more finite elements than processing nodes, each processing node is assigned a number of finite elements. Further, the mapping of a finite element onto the processing node is done randomly, that is, a finite element labeled \( i \) is mapped onto a virtual processing node labeled \( r(i) \) where \( r \) is a random permutation such that \( 1 \leq r \leq N_e \) and \( N_e \) is the number of finite elements in the mesh [Mathur, 1990]. The random mapping is quite effective in minimizing the contention for the communication channels that occur in the network connecting the processing nodes of any distributed memory architecture [Valiant and Brebner, 1981, Valiant, 1982, Ranade, 1987, Ranade et al., 1988]. Throughout this article, this view of the data set will be referred to as the “set of finite elements”. Finally, the second data representation maps a nodal point (and sometimes a nodal degree of freedom) onto the processing nodes of the architecture. As with the first mapping, a random mapping scheme is used to make the contention for the communication channels uniform. This view of the data set is referred to as a “set of nodal points”. The interaction between the two data sets is accomplished by the use of the mesh connectivity array.

3 Applications

This section describes the three applications that were used to identify the set of communication primitives described in this article. The reader is referred to the articles referenced in the appropriate sections for a detailed description of the formulation and their data parallel implementations on the model architecture.

An explicit finite element formulation

The first application involves a data parallel implementation of an unstructured three–dimensional finite element formulation for the rate dependent
deformation in the Charpy V–notch test [Mathur et al., 1992, Tvergaard and Needleman, 1986 and 1988]. This is a Lagrangian, fully dynamic and explicit implementation. The governing equations that result from substituting the finite element approximation of the momentum balance are of the form

\[ \mathbf{M} \mathbf{a} = \mathbf{F} \]  (1)

where \( \mathbf{M} \) is the mass matrix, \( \mathbf{a} \) is the nodal acceleration vector, and \( \mathbf{F} \) is the nodal force vector that is based on the externally applied traction boundary conditions and the internal forces generated as a consequence of the deformation. The formulation first computes the acceleration of each nodal point. Next, the equations of motion are integrated by an explicit numerical integration procedure to compute the nodal velocities and the nodal displacements. Finally, the deformation is driven forward in time and the whole procedure repeated until the solution is known over the entire time period of interest.

A lumped mass matrix based on the scheme proposed by Hinton, Rock, and Zienkiewicz [1976] is used instead of a consistent mass matrix. For explicit formulations, a lumped mass matrix scheme yields better accuracy [Hughes, 1987]. Further, a lumped mass matrix is also preferable from a computational efficiency point of view. For a lumped mass matrix, \( \mathbf{M} \) in Equation (1) is diagonal, and consequently a sparse linear system solution is no longer necessary to evaluate the nodal acceleration.

The mesh is composed of twenty node brick elements with selective integration to prevent locking of these elements due to the nearly incompressible deformations in the plastic range. The material is characterized by a viscoplastic version of an isotropically hardening Mises solid. The constitutive relation is integrated using a rate tangent time stepping procedure. For the class of problems to be analyzed here the element sizes are small enough that stability of the explicit integration of the momentum balance, rather than integration of the constitutive relation, controls the maximum permissible time step [Hughes, 1987]. The reader is referred to Mathur et al. [1992a] for a detailed description of the data parallel implementation and a detailed analysis of the three-dimensional simulation results.

The explicit nature of the implementation divides the solution procedure at each time increment into three distinct phases. First, based on the mesh connectivity, all finite elements in the mesh “gather” the nodal data values into local vectors. The nodal values of interest that need to be gathered,
are the nodal displacements. All other variables of interest already reside on the same processing node as the finite element. Next, the unassembled nodal forces are computed at an element level. The gather operation ensures that the computation of the unassembled nodal forces does not involve any interaction between the two different views of the data set. Consequently, the computation of the unassembled nodal forces can be performed independently for all elements (of the same type) in the mesh. Finally, the nodal forces are assembled into a global force vector by a “scatter” operation where each finite element sends the local unassembled force component to the corresponding nodal point. Since, many finite elements share a nodal point, data collision occurs at the destination. In the context of finite element simulations, the colliding values must be added together.

Only two communication primitives are necessary for most explicit finite element formulations – one which gathers the nodal quantities into local elemental vectors and the other which assembles the unassembled elemental vectors into a global vector. Both these primitives involve interaction between the set of finite elements and the set of nodal points. The gathering of the nodal displacements is a “one-to-many” mapping between the assembled nodal degrees of freedom and the unassembled nodal degrees of freedom. The assembly phase involves data motion in the reverse direction (“many-to-one” mapping).

Figure (1) shows the mesh used to simulate the three-dimensional Charpy V-notch test at the start of deformation and after the first 100 microseconds. The mesh consists of 3200 twenty node bricks. Tables (1) and (2) show the computation time per time increment for the three sections of the formulation described above, as a function of the number of quadrature points. The timings reported in these tables were measured on an 8K CM-200 configuration. The computation time per time increment per finite element per integration point is 11 μs for $2 \times 2 \times 2$ Gauss quadrature and 9μs for $3 \times 3 \times 3$ Gauss quadrature. If the finite element and the nodal points in the mesh are not mapped randomly onto the processing elements of the architecture, then for $2 \times 2 \times 2$ quadrature almost 45% of the total computation time is spent in data motion (gather and scatter). When the finite elements and the nodal points are mapped randomly, the reduction in the communication cost is significant. Only 28% of the total computation time is now spent in data motion.

6
<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (millisec.)</th>
<th>% of Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather</td>
<td>61</td>
<td>16</td>
</tr>
<tr>
<td>Scatter</td>
<td>111</td>
<td>29</td>
</tr>
<tr>
<td>Total</td>
<td>379</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (millisec.)</th>
<th>% of Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather</td>
<td>37</td>
<td>13</td>
</tr>
<tr>
<td>Scatter</td>
<td>42</td>
<td>15</td>
</tr>
<tr>
<td>Total</td>
<td>284</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 1: Computation time (milli−seconds) for one time increment for a mesh comprising of 3200 20−node brick elements (Figure 1). The integration of the constitutive equations was performed by a $2 \times 2 \times 2$ Gauss quadrature. The timing measurement was performed on a 8K CM−200 configuration.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (millisec.)</th>
<th>% of Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather</td>
<td>75</td>
<td>9</td>
</tr>
<tr>
<td>Scatter</td>
<td>124</td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td>873</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (millisec.)</th>
<th>% of Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather</td>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td>Scatter</td>
<td>55</td>
<td>7</td>
</tr>
<tr>
<td>Total</td>
<td>781</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 2: Computation time (milli−seconds) for one time increment for a mesh comprising of 3200 20−node brick elements (Figure 1). The integration of the constitutive equations was performed by a $3 \times 3 \times 3$ Gauss quadrature. The timing measurement was performed on a 8K CM−200 configuration.
Finite element modeling based on polycrystalline plasticity

The second set of applications described here involves two levels of discretization. The first level of discretization is on the macroscopic scale. On the continuum scale, the discretization is similar to any other finite element simulation. The domain is discretized into a collection of finite elements which model the geometry. The second level of discretization models the microstructure of the material. Underlying each finite element is an aggregate of crystals. The properties of the material at a sampling point within the element are a function of the deformation of the individual crystals within the element. In addition to the two views (a set of nodes and a set of finite elements) described above, there is a third data representation in which a crystal in the aggregate is mapped on to the processing nodes of the model architecture. The reader is referred to [Beaudoin et al., 1992, Mathur et al., 1992b and Dawson et al., 1992] for a detailed description of these simulations. Briefly, such simulations can be viewed as having three distinct sections. First, the macroscopic deformation field is computed at fixed state. In the context of polycrystalline models, the state of the material is characterized by the orientation of the crystals comprising the material and the strength of the slip systems of each crystal. The evaluation of the macroscopic deformation field involves data interaction between all three data representations. The macroscopic deformation field, at fixed state, is computed from the solution of the discretized form of the macroscopic balance of linear momentum

\[ \nabla \cdot \sigma = 0 \]  \hspace{1cm} (2)

and the conservation of mass

\[ \nabla \cdot \mathbf{u} = 0 \]  \hspace{1cm} (3)

where \( \sigma \) is the macroscopic Cauchy stress tensor and \( \mathbf{u} \) is the velocity. Note that the expression for the conservation of mass assumes that the elastic effects are small compared to the plastic deformation. The nonlinear viscoplastic material behavior results in a nonlinear sparse system of equations, which is solved by a method of successive approximations. Iteration \( i \) of the nonlinear scheme results in a sparse linear system of the form [Zienkiewicz
\[
\left[ K_D(U^i) + \lambda \epsilon GM^{-1}G^T \right] \{U^{i+1}\} = \{f\} + [G] \{P^i\}
\]

(4)

where \([K_D]\) is the material stiffness matrix based on the fixed state and the velocity field at the \(i\)-th iterate, \([G]\) is the matrix that enforces the conservation of mass, and \([M]\) is the pressure mass matrix of the consistent penalty method [Engleman et al., 1982]. The sparse linear system described by Equation (4) is solved by a conjugate gradient method with diagonal scaling. The nonlinear iterations are assumed to have converged when

\[
\|\{U^{i+1}\} - \{U^i\}\|_2 < \epsilon \|\{U^i\}\|_2
\]

(5)

where \(\epsilon\) is a prescribed tolerance.

The evaluation of the material stiffness matrix, \([K_D]\), involves interactions between the three data representations. First, the nodal coordinates and the nodal velocities must be “gathered” from the set of nodal points to the set of finite elements. The macroscopic rate of deformation and the Jacobian at a quadrature point within a finite element, are computed in the set of finite elements. Next, the macroscopic rate of deformation at the quadrature point is broadcast to all the crystals underlying the quadrature point. This is a segmented “all-to-all” broadcast [Johnsson and Ho, 1989, Brunet and Johnsson, 1991, and Mathur and Johnsson, 1992]. It is assumed that all the crystals within the aggregate experience the macroscopic deformation [Taylor, 1938]. After the broadcast stage, data is readily available in the data set representing the crystals. A Newton-type method with line search is necessary to solve for the microscopic stress deviator that is required to achieve the prescribed rate of deformation. The quasi-Newton scheme is required for all crystals in all aggregates. This nonlinear solution for each crystal is independent of all other crystals. Once the microscopic stress deviator is known, the microscopic crystal stiffness matrices are computed and the macroscopic stiffness matrix at the quadrature point is computed by averaging the crystal stiffness matrices. This averaging procedure requires a segmented “all-to-all” reduction [Mathur and Johnsson, 1992].

The performance of the conjugate gradient solver is dominated by the sparse matrix vector multiply. The implementation details of the conjugate gradient method are described in Golub and vanLoan [1989]. After the evaluation of the elemental stiffness matrices, the sparse matrix vector multiply
can be expressed as an “element–by–element” operation. This avoids the explicit assembly of the global stiffness matrix. Instead, the assembly is performed implicitly during the sparse matrix vector multiply operation. First, the nodal quantities are “gathered” from the set of nodal points. After the gather operation, all relevant data is in place for a local matrix vector multiplication of the unassembled stiffness matrices and the elemental vectors. This operation takes place in the set of finite elements. The matrix vector multiplication produces the un assembled product vector which is then “scattered” back to the set of nodal points. Since many elements share the same nodal points, data collision occurs at the destination. These colliding values must be added to ensure that the contributions from all the elements sharing a nodal point are accounted for. The boundary conditions are applied in the set of nodal points.

Once a converged deformation field corresponding to a fixed mechanical state has been computed, the mechanical state of the material and the geometry of the workpiece must be updated. An explicit Euler integration scheme is used here. The update of the microstructural quantities (the crystal orientations and the strength of the slip systems) must be done in the data representation of the crystals. The geometry of the workpiece is updated in the nodal data representation. Again, data interaction between all the three data sets is necessary. The reader is referred to Mathur and Dawson [Mathur and Dawson, 1989] for a more detailed description of the update procedure. The communication primitives that are necessary for an efficient implementation of the update are identical to the communication primitives that are necessary for computing the macroscopic deformation field at a fixed state.

Figure (2) shows the deformed mesh after the first time increment and the deformed mesh after significant deformation during a hydroforming simulation of an initially textured aluminum sheet [Dawson et al., 1992]. The initial crystal orientation underlying each material point corresponds to a cube texture. The sheet properties were assumed to be initially uniform. As a result, all elements are initialized with identical initial textures. The color contours in Figure (2) represent the magnitude of the radial velocity in the plane of the flat portion of the sheet. The deviation from an axisymmetric solution due to the initial anisotropy is readily apparent. Material is moving inwards faster at 45° relative to 90°. Tables (3) and (4) show representative timing data for a mesh with 864 eight node brick elements and 1407 nodal points (5221 degrees of freedom). Underlying every material point is an ag-
Table 3: Computation time (milli–seconds) and computation rate for the sparse matrix vector multiply in the conjugate gradient solver. All timings and floating point rates correspond to an 8K CM–200. The finite element mesh had 864 eight node brick elements and 1407 nodal points. The total time per conjugate gradient iteration was 14.4 milli–seconds for no random mapping and 8.5 milli–seconds for random mapping.

<table>
<thead>
<tr>
<th>Operation</th>
<th>No random mapping</th>
<th>Random mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (millisecond)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gather</td>
<td>5.6</td>
<td>3.7</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>Scatter</td>
<td>7.2</td>
<td>3.4</td>
</tr>
<tr>
<td>Total</td>
<td>13.6</td>
<td>7.8</td>
</tr>
<tr>
<td>Mflops (8K CM–200)</td>
<td>73</td>
<td>127</td>
</tr>
</tbody>
</table>

Table 4: Time for the computation intensive portions of the micromechanical finite element method. All times correspond to an 8K CM–200.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stiffness (/grain/quad. point/element/iteration step)</td>
<td>8.2</td>
</tr>
<tr>
<td>Update (/grain/element/time step)</td>
<td>66</td>
</tr>
</tbody>
</table>
aggregate of 256 crystals. The timing data reported here was measured on an 8K CM-200 system. As with the first application, randomizing the mapping of the finite elements and the nodal points to the processing nodes of the model architecture reduces the time spent in data motion significantly.

**Data parallel computational fluid dynamics**

The third application used in identifying an appropriate set of high level communication primitives for unstructured finite element simulations involves a computational fluid dynamics application. Three–dimensional compressible flows on unstructured meshes having almost one million degrees of freedom have been simulated using an implicit iterative solution method [Johan et al., 1990 and 1992]. The solution strategy involves an implicit time marching algorithm to converge to a steady state solution. A preconditioned matrix–free generalized minimal residual algorithm is used to solve the linear system. The reader is referred to Johan et al. [1991] for a detailed description of the solution strategy. The data parallel implementation is described in detail in Johan et al. [1992].

This data parallel implementation assigns two views to the data set. In the first view, the unassembled finite elements are mapped on to the processing nodes of the model architecture. The second view maps the assembled nodal points to the processing nodes. In this application, the bulk of the simulation time is spent in the evaluation of the sparse matrix vector multiply used in the GMRES algorithm. First, the nodal values are “gathered” from the set of nodal points to the set of finite elements. All processing nodes perform local matrix vector operations concurrently to evaluate the unassembled elemental product vectors. These vectors are then assembled to a global vector by a “scatter” operation. Thus, the communication primitives that are needed in the computation–intensive part of this application are identical to the primitives needed for the other two applications described here.

Figures (3) and (4) show the surface mesh and the surface pressure distribution for a generic Falcon Jet at an effective angle of attack of six degrees. The complete finite element mesh consists of 150,752 nodal points and 878,544 tetrahedra elements, yielding 753,620 equations. Table (5) shows the overall performance of the application for a one–point integration rule and 250 time steps at a CFL number of five. Similar computations with a four–point integration rule indicate that the fraction of time spent in data motion
Table 5: Overall time (s) for simulating Euler flow over a complete Falcon Jet during approach. The timings reported here were measured on a fully configured Connection Machine system CM–200 with 2048 processing nodes. The timings correspond to 250 time steps at a CFL of five with a one–point integration rule.

is significantly less. The time spent in communication is proportional to the number of degrees of freedom per unassembled finite element. The computation time increases linearly with the number of integration points used for numerical quadrature.

4 Unstructured communication primitives

The three applications described above illustrate the use of the following communication primitives:

- Global gather
- Global scatter
- All–to–all broadcast
- All–to–all reduce

The first set of communication primitives that are needed by almost all unstructured (arbitrary sparse) applications are the global gather and scatter functions. During the global gather operation, every destination array element accumulates data values from a source array element based on a set of pointer (index) arrays. Since many destination array elements may want to fetch the same source data value, this is a “one–to–many” mapping. For the finite element simulations described above, the gather operation can be expressed as a vector–valued Fortran–90 statement [Metcalf and Reid, 1990] of the form

$$D = S(P)$$  \hspace{1cm} (6)
where $D$ is the destination array, $P$ the pointer array identifying the source element that needs to be accumulated and $S$ is the source array. The extent of array $P$ must be identical to the extent of the destination array $D$. The Fortran–90 expression in (6) is equivalent to an element-wise gather operation. A similar expression can be written for a “vector–gather” operation which is very useful for applications involving multiple degrees of freedom per nodal point.

The scatter operation is the reverse of the gather operation. Many source array elements send their data to the destination array elements. This is a “many–to–one” mapping, which can be formally expressed as

$$D(P|+) = S.$$  \hspace{1cm} (7)

where the pointer array $P$ has the same extent as the source array $S$. Expression (7) is not a legal Fortran–90 expression. Data collision occurs at the destination. The data collision can be handled in a variety of different ways. For finite element simulations, data collision at the destination represents the contributions from different finite elements sharing a nodal point (or a nodal degree of freedom). Therefore, the colliding data values must be added. There is no legal way of expressing a scatter operation with addition in the subset of Fortran–90 implemented on the Connection Machine systems. This functionality is achieved by calling a utility library function. The reader is referred to the CM Fortran reference manual [1991] for details.

In the context of the finite element method, the pointer array $P$ represents the mesh connectivity. The three applications described above make use of the same mesh connectivity for the entire duration of the simulation. Note that in the second application, the geometry of the workpiece changes significantly. However, the mesh connectivity remains fixed. For such situations, it is highly desirable to preprocess the mesh connectivity and reuse the preprocessed information repeatedly during the course of the simulation. This preprocessing can be performed in several ways. Additional memory is required to store the information that is generated during the preprocessing step. The amount of memory needed is a function of the number of processing nodes on the architecture and the mesh connectivity. The implementations used for the applications reported here transform the one–to–many mapping of the gather operation and the many–to–one mapping of the scatter operation to one–to–one mappings. The data motion related to the one to one
mapping is achieved by the use of the general purpose communications hardware on the model architecture. The gather operation requires a local copy operation before the permutation operation, while the scatter operation requires an addition step after the permutation. The one-to-one mapping is constructed in such a way that the addition step may be performed locally on each processing node. The algorithms used for the gather and scatter operations are described in detail in Mathur [1991]. The preprocessed gather and scatter functions perform significantly better than the simple Fortran-90 expression of Equation (6). These gather and scatter functions are a part of the Scientific Software Library on the model architecture. The reader is referred to the CMSSL reference manual [1992] for more detail.

The all-to-all broadcast and reduce functions that are necessary for the second application are part of a general class of communication primitives that are useful for several dense linear algebra algorithms [Mathur and Johnson, 1992] and applications based on regular grids. A segmented spread is expressed straightforwardly in Fortran-90 as

\[ D = \text{spread}(d, \text{dim} = 1, \text{ncopies} = N_g) \]  

where \( D \) is a two-dimensional matrix of shape \((N_g \times N_e)\), \( d \) is a vector \( N_e \) long. In the context of the micromechanical model, \( N_g \) represents the number of crystals underlying each finite element and \( N_e \) is the number of finite elements in the mesh representing the continuum length-scale discretization. The parameters \( N_e \) and \( N_g \) represent the resolution of the macroscopic and the microscopic scale discretization respectively. The vector \( d \) represents a component of the macroscopic rate of deformation and the matrix \( D \) is the equivalent component of the microscopic rate of deformation of a crystal in the aggregate.

The Fortran-90 expression of Equation (8) may, in fact, define an all-to-all broadcast. To illustrate this fact, consider the allocation of arrays on the model architecture in some detail. The Connection Machine Run Time System allocates array elements to the processing nodes based on the shape of the array. The run time system attempts to configure the processing nodes as an array with the same number of axes as the data array. The extent of each processing node axis is chosen such that the number of data array elements along each array axis assigned to a processing node is approximately equal. Thus, for a one-dimensional data array the processing nodes are also

\[ (8) \]
configured as a one-dimensional array, with the data elements divided as evenly as possible among the processing nodes. For a two-dimensional data array, the processing nodes are configured as a two-dimensional array. The subarray assigned to a processing node is approximately square. Figure (5) shows a simple example which illustrates that the segmented spread of Equation (8) indeed implies an all-to-all broadcast in a column major ordering of the processing nodes.

A spread of a one-dimensional vector, \( d \), into a two-dimensional matrix, \( D \), as shown in Figure (5), requires that each instance of the vector \( d \) instead of being distributed among all processing nodes (labeled 0 \( \cdots \) 7) is now distributed among processing nodes labeled 0, 2, 4 and 6, and also processing nodes labeled 1, 3, 5, and 7. After the spread operation, the two segments, \( d_0 \) and \( d_1 \) of vector \( d \), initially allocated to processing nodes labeled 0 and 1 respectively, must now reside in both processing nodes 0 and 1. Similarly, the two segments, \( d_2 \) and \( d_3 \), of vector \( d \), initially allocated to processing nodes labeled 2 and 3 respectively, must now reside in processing nodes 2 and 3. Thus, for the column major ordering of the processing nodes, the spread operation implies an all-to-all broadcast within the columns of the two-dimensional processing node configuration. For a row major ordering of the processing nodes, a transposition corresponding to a conversion between the column and the row major ordering must precede the all-to-all broadcast.

A segmented reduce function such as

\[
s = \text{sum}(S, \text{dim} = 1)
\]

where, as before, \( S \) is a two-dimensional matrix of shape \((N_g \times N_e)\) and \( s \) is a vector \( N_e \) long implies an all-to-all reduce operation. In the micromechanical modeling effort, \( S \) represents a component of the microscopic Cauchy stress deviator and \( s \) is the corresponding component from the macroscopic Cauchy stress deviator (which is formed by averaging the microscopic stress tensor). Reduction along the columns in Figure (5) performed such that the result from the reduction on the first half of the columns assigned to a column pair of processors is assigned to the top processor, and the result of the reduction on the second half of the columns is assigned to the bottom processing node, defines an all-to-all reduce within the columns of the processing node array. The all-to-all reduction is easily generalized to more than two processing nodes per column.
As with the gather function, although the all-to-all broadcast and reduce functions can be expressed in Fortran–90, it is very beneficial to identify them as communication primitives. These primitives can then be optimized for the network interconnecting the processing nodes of the architecture. Optimal algorithms for the all-to-all broadcast and the all-to-all reduce for the network of the model architecture are described in [Johnsson and Ho, 1989, Brunet et al., 1990, Brunet and Johnsson, 1991, and Mathur and Johnsson, 1992]. Table (6) compares an optimized all-to-all reduce operation with the standard reduce and send instruction on the Connection Machine system, CM–200. Note that the standard reduce and send instruction itself is also optimized.

### 5 Summary

This article introduces a set of communication primitives that are very useful for unstructured finite element simulations. The intended use of these
primitives is illustrated by three applications that have been implemented on the Connection Machine systems, CM-2 and CM-200. The choice of the applications is such that both fully explicit and fully implicit formulations have been investigated.

The communication primitives are a part of the scientific software library on the model architecture. They have been optimized for the model architecture. With the use of the optimized communication primitives, the three applications presented here run very efficiently on the model architecture. These communication functions support the notion of scalable computing. The physical configuration of the computing platform is completely transparent to the application. The intended goal of portability between different architecture supporting compilers with a global address space is achieved. This has been demonstrated by simply compiling these codes for the next generation Connection machine system, CM-5. The initial performance numbers on the Connection Machine system, CM-5, are very promising. Simulations with significantly finer resolutions are now possible in a reasonable time.

Acknowledgements

The first application described here was a collaborative effort with Alan Needleman of Brown University and Viggo Tvergaard of the Technical University of Denmark. The second application was a collaborative effort with Paul Dawson and Armand Beaudoin of Cornell University. The CFD simulations were implemented in collaboration with Zdenek Johan and Tom Hughes of Stanford University. The authors would like to thank these collaborators for making the simulation data available for this article. The mesh for the generic Falcon Jet was provided by Dassault Aviation, France. The first two applications were run on the Connection Machine system at the Pittsburgh Supercomputing Center. The third application was run on the Connection Machine system at the Advanced Computing Laboratory at Los Alamos National Laboratory. The authors would also like to thank Susan Arnott for her help in preparing the color images presented here.
References


Figure 1: Three dimensional Charpy V-notch simulation [Mathur, et al., 1992]. The middle image shows the $x$ component of the displacement field after the first 100$\mu$s. The image on the right shows the magnitude of the displacement field after the first 100$\mu$s. The finite element mesh comprises of 3,200 twenty node bricks with 15,083 nodal points and 45,249 nodal degrees of freedom.
Figure 2: Hydroform simulation of an initially textured aluminum sheet [Dawson, et al., 1992]. The magnitude of the radial velocity in the plane of the flat portion of the sheet is shown. The deformed mesh corresponds to a time increment after significant deformation has occurred. The finite element mesh is comprised of 864 eight node bricks with 1407 nodal points and 5221 nodal degrees of freedom. Underlying each finite element is an aggregate of 256 crystals.
Figure 3: Surface mesh for Euler flow simulation on a generic Falcon jet [Johan, et al., 1992]. The complete mesh consists of 878,544 tetrahedra with 150,752 nodal points and 753,620 nodal degrees of freedom.
Figure 4: Surface pressure distribution for a generic Falcon jet at an effective angle of attack of six degrees.
Figure 5: One and two dimensional column major ordered configurations of eight processing nodes.