Parallel Simulations of Dynamic Fracture Using Extrinsic Cohesive Elements

Isaac Dooley[*]
Sandhya Mangala[†]
Laxmikant Kale[‡]
Philippe Geubelle[§]

November 3, 2008

Abstract

In this paper, we present a novel parallel implementation of extrinsic initially rigid cohesive elements in an explicit finite element solver designed for the simulation of dynamic fracture events. The implementation is based on activating instead of inserting the cohesive elements and uses ParFUM, a parallel framework specifically developed for simulations involving unstructured meshes. Aspects of the parallel implementation are described, along with an analysis of its performance on 1 to 512 processors. Important cache effects and communication costs are included in this analysis. The implementation is validated by simulating the trapping of a crack along an inclined material interface. **Keywords:** Cohesive Finite Elements, Parallel Programming, Dynamic Fracture

1 Introduction

Due to its flexibility in capturing complex geometries, loading conditions, and material models, the cohesive finite element (CFE) scheme has been the method of choice for simulating a wide range of dynamic fracture events over the last decade [CO97, Nee97, GB98, PSV03]. In this finite element formulation, conventional (volumetric) elements are used to capture the

[*]Department of Computer Science, University of Illinois
[†]Department of Aerospace Engineering, University of Illinois
[‡]Department of Computer Science, University of Illinois
[§]Department of Aerospace Engineering, University of Illinois

bulk mechanical response of the material while interfacial (cohesive) elements are used to model the progressive failure of the material and the associated propagation of cracks in the discretized domain. Cohesive elements basically act as distributed non-linear springs, resisting the separation of volumetric elements, i.e., the introduction of displacement jumps in the domain, according to a prescribed traction-separation cohesive law. In two-dimensional (2D) problems, triangular volumetric elements are usually adopted to maximize the number of potential crack paths, while the cohesive elements are collapsed quadrilateral elements introduced at the interface between two adjacent volumetric elements, as shown in Figure 1(a). In that schematic, $\mathbf{\Delta}$ denotes the displacement jump across the cohesive element and Δ_n and Δ_t are the corresponding normal and tangential components.

Two types of cohesive constitutive laws have been used in the cohesive finite element modeling of dynamic fracture events. The first cohesive model, usually referred to as *intrinsic*, relates the cohesive traction to the displacement jump through a phenomenological relation that typically starts from the origin, reaches a maximum (corresponding to the failure strength) and then decays back to zero, at which point the failure process is completed (Figure 1(b)). The second model, referred to as *extrinsic*, typically assumes that the cohesive response is initially rigid and therefore only models the failure process through a monotonically decreasing relation between the failure strength and the displacement jump (Figure 1(c)). These two approaches thus differ in the way they capture the initial response of the cohesive element. In the intrinsic scheme, the cohesive elements are present in the finite element mesh from the start and, due to their finite initial stiffness, contribute to the deformation of the medium even in the absence of damage. In the extrinsic scheme, the cohesive elements are initially rigid and are only introduced in the finite element mesh based on an external traction-based criterion.

The key characteristics of the failure process are, however, identical for both models: in both cases, the failure process is captured with the aid of a phenomenological traction-separation law defined primarily by the failure strength (denoted by σ_{max} for the tensile failure case depicted in Figures 1(b) and (c)) and the critical value of the displacement jump (Δ_{nc}) beyond which complete failure is assumed. The area under the cohesive failure curve defines the fracture toughness (usually denoted by G_{Ic} in the tensile (mode I) case). Although various cohesive laws have been used in the past (linear, bilinear, exponential, polynomial, trapezoidal, etc.), the actual shape of the cohesive failure curve is considered to play only a secondary role on the failure process in many situations, especially in brittle materials for which the cohesive failure zone is very small. A discussion of the similarities and differences between the two cohesive failure models can

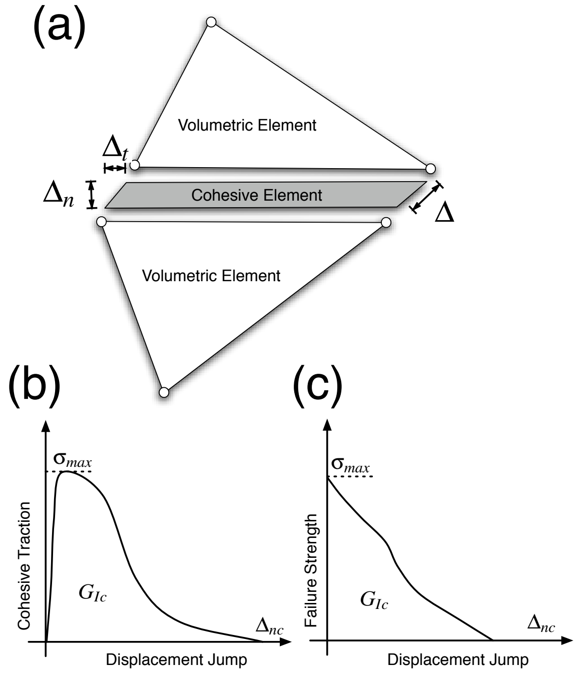


Figure 1: (a) Cohesive finite element concept, showing two 3-node triangular volumetric elements tied by a 3-node cohesive element shown in its deformed configuration. In its initial form, the cohesive element has no area and the adjacent nodes are superposed. (b) Schematic of an intrinsic cohesive failure law for the tensile failure case, for which the tangential displacement jump Δ_t is zero. (c) Generic extrinsic cohesive law.

be found in [KG03].

Due to its relative simplicity of implementation, the intrinsic cohesive finite element scheme has been more widely adopted than its extrinsic counterpart. However, as shown in [PSV03, FNR01], intrinsic elements suffer from convergence issues associated with the impact of the initial cohesive stiffness on the computed strain and stress fields, and thereby, on the fracture process. To achieve convergence, intrinsic cohesive elements should be used only to simulate dynamic fracture problems for which the crack path is prescribed (such as in interfacial fracture events). In the case of arbitrary crack motion and branching, a finite distance should be introduced between cohesive failure surfaces. The issues of spatial and temporal convergence for intrinsic and extrinsic cohesive elements are discussed in [PSV03, BRN06].

Dynamic fracture simulations need a very fine mesh near the failure zone to accurately capture the stress concentrations and the failure process, especially for brittle systems. Also, there is often a need for a large domain to capture the loading accurately. A large domain combined with fine mesh requirements make the problem computationally very challenging. Parallel simulations, where the problem domain can be partitioned into smaller domains and solved for on different processors, provide a powerful tool to solve these problems. Parallel computing can be used in conjunction with adaptive mesh refinement and coarsening [MWC$^+$07].

The objective of this paper is to develop and implement a parallel CFE scheme based on activated extrinsic cohesive elements. As mentioned earlier, extrinsic cohesive elements are chosen over intrinsic elements to prevent the effects of artificial compliance due to the initially elastic part of intrinsic cohesive elements. The parallel implementation of this scheme poses a set of challenges pertaining to the partitioning of the finite element mesh and to inter-processor communication due to the presence of cohesive elements at the interfaces. The complexities of communication are further increased with extrinsic cohesive elements because there are multiple types of elements in the discretization, each containing different fields.

To implement the CFE code in parallel, we use the Parallel Framework for Unstructured Meshing (ParFUM) [LCW$^+$06], a portable library for building parallel applications involving unstructured finite difference, finite element, or finite volume discretizations. The parallel framework is used in this work to partition the unstructured mesh, to distribute the partitions to processors and to setup the inter-processor communication lists. ParFUM is built on the Charm++/AMPI Adaptive Runtime System and thereby provides additional features such as dynamic load balancing between processors and adaptive overlapping of communication and computation [KK96].

This paper describes the use of ParFUM to implement the CFE scheme. Only a small number of libraries for handling parallel unstructured meshes exist. There are other large parallel mesh frameworks, but these are not options for various reasons. Unfortunately, some of the most fully featured production level frameworks are not currently available to the public and thus would not be suitable candidates for our application. Sandia National Laboratories' *SIERRA* [SE04] and the University of Heidelberg's *UG* [BBJ$^+$97] are two such publicly unavailable frameworks. Both *SIERRA* and *UG* support fully unstructured meshes on distributed memory supercomputers with a variety of compatible solvers. *deal.ii* is a common finite element framework, which unfortunately works in parallel only on shared memory machines. Although *deal.ii* cannot utilize a distributed memory computer system, it does interface with solver libraries such as *PETSc* which are parallelized for clusters [BGMS97]. Some frameworks do not support fortran codes. *AOMD* [RKFS01] and *libMesh* do not support fortran. The extrinsic scheme proposed in this paper requires just support for partitioning, distributing, ghost layer creation, and accessing an unstructured mesh on a distributed memory computer cluster. A framework supporting fortran was desired because the authors had an existing serial fortran CFE code. Another desired feature for the mesh framework is the ability for the existing fortran code to maintain control over its own existing data arrays. Some mesh frameworks unfortunately maintain all mesh data internally, and only expose it through a specified API. ParFUM supplies all these required and desired features.

The emphasis of this work is on the inter-process communication in parallel simulations performed with the extrinsic CFE scheme. Though mesh adaptivity would further improve the efficiency of the proposed parallel implementation, only the parallel implementation of extrinsic cohesive elements is discussed here. This paper provides in Section 2 the extrinsic constitutive law and the associated CFE formulations along with the stability conditions. Section 3 describes our parallel methodology and implementation for the extrinsic cohesive elements that overcomes various issues of partitioning and inter-processor communications. Section 4 presents a series of test simulations to validate the developed parallel extrinsic CFE scheme. The validation study involves the numerical simulation of dynamic fracture experiments performed on brittle specimens bonded along an inclined interface [XR03]. Finally, in Section 5, we present scaling results for the interface problems using the current parallel implementation.

2 Cohesive constitutive law and cohesive finite element formulations

The cohesive failure law adopted in this work is the linear extrinsic relation used by [RPO01], in which the cohesive traction **T** during the failure process is described by

$$\mathbf{T} = \frac{T}{\Delta}(\beta^2 \mathbf{\Delta}_t + \Delta_n \mathbf{n}), \tag{1}$$

where T denotes the effective cohesive traction defined by

$$T = \sqrt{\beta^{-2}|\mathbf{T}_t|^2 + T_n^2} \tag{2}$$

and T_n and $\mathbf{T}_t$ are the normal and tangential tractions, respectively. In (1), Δ, Δ_n and $\mathbf{\Delta}_t$ are defined by

$$\Delta = \sqrt{\beta^2 \Delta_n^2 + \Delta_t^2} \Delta_n = \mathbf{\Delta} \cdot \mathbf{n} \Delta_t = |\mathbf{\Delta}_t| = |\Delta - \Delta_n \mathbf{n}| \tag{3}$$

where $\mathbf{n}$ is the normal vector defining the undeformed orientation of the cohesive element. The parameter β in (1)-(3) assigns different weights to the sliding and normal opening displacements.

The traction vector $\mathbf{T}$ before the activation of a cohesive element is computed from the stress fields of the neighboring volumetric elements as

$$\mathbf{T} = \boldsymbol{\sigma}_{aver} \mathbf{n}, \tag{4}$$

where $\boldsymbol{\sigma}_{aver}$ is the average stress tensor of the two adajacent volumetric elements.

The failure process is initiated when either the normal traction T_n across the cohesive interface reaches the critical tensile failure strength σ_{max} or the tangential component T_t, while $T_n > 0$ (i.e, failure is initiated only for tensile loading), reaches the corresponding shear failure strength τ_{max}. A cohesive element completely fails when either of the displacement jump components Δ_n or Δ_t reaches the corresponding critical opening displacements Δ_{nc} or Δ_{tc}. The critical energy release rate of failure in mode I (G_{Ic}) and mode II (G_{IIc}) are related to the corresponding components of strength and critical displacement jump as follows:

$$G_{Ic} = \frac{\sigma_{max}\Delta_{nc}}{2}, \qquad G_{IIc} = \frac{\tau_{max}\Delta_{tc}}{2}. \tag{5}$$

When a cohesive element is not active, each pair of nodes across its width effectively represent a single regular finite element (FE) node thus introducing a discontinuity in the mesh representation. As every internal edge in the mesh is a cohesive element, inactive cohesive elements result in a situation where a single node of a regular FE mesh has multiple node copies in the CFE implementation as shown in Figure 2. To overcome this problem, a random node is chosen amongst the multiple node copies as the representative root node and it represents all nodes at the location for all

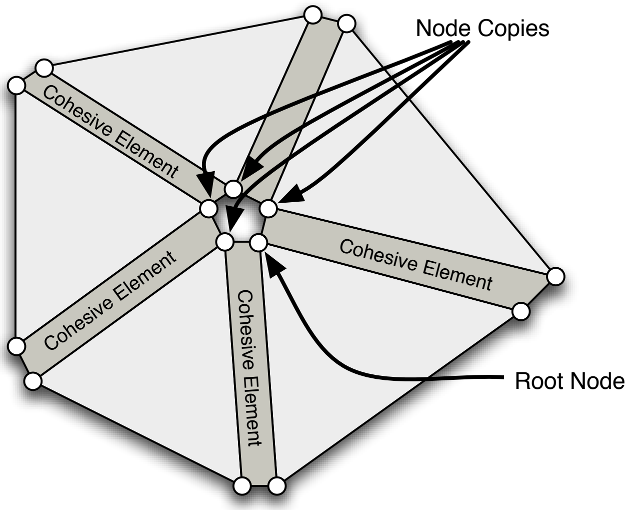


Figure 2: One root is chosen as a representative for the multiple node copies. When surrounding cohesive elements are inactive the node copies have identical displacements and velocities as the root node.

computational purposes. The masses and internal forces of all the nodes represented by this node are added together and the cumulative values are assigned to this representative node. Thus continuity of the mesh across inactive cohesive elements is ensured.

The basis of the finite element formulation is the following principle of virtual work defined over the deformable solid Ω:

$$\int_{\Omega} (\mathbf{S} : \delta\mathbf{E} - \rho_o \ddot{\mathbf{u}}.\delta\mathbf{u})\, d\,\Omega - \int_{\Gamma_{ex}} \mathbf{T}_{ex}.\delta\mathbf{u}\, d\,\Gamma - \int_{\Gamma_{in}} \mathbf{T}.\delta\boldsymbol{\Delta}\, d\,\Gamma = 0, \qquad (6)$$

where $\mathbf{T}_{ex}$ denotes the external tractions on the external boundary Γ_{ex} and $\mathbf{T}$ corresponds to the cohesive tractions acting along the internal boundary Γ_{in} across which the displacement jumps $\boldsymbol{\Delta}$ exist. In (6), ρ_o is the material density, $\mathbf{u}$ is the displacement field, a superposed dot denotes differentiation with time, $\mathbf{S}$ and $\mathbf{E}$ are the second Piola-Kirchoff stress tensor and the Lagrangian strain tensor, respectively. The principle of virtual work described by (6) is of standard form except for the presence of the last term, which is the contribution from cohesive tractions. The semi-discrete finite element formulation can be expressed in the following matrix form:

$$\mathbf{M}\,\mathbf{a} = \mathbf{R^{in}} + \mathbf{R^{ex}}, \qquad (7)$$

where $\mathbf{M}$ is the lumped mass matrix, $\mathbf{a}$ is the nodal acceleration vector and $\mathbf{R^{in}}$, $\mathbf{R^{ex}}$ respectively denote the internal and external force vectors [GB98].

With the aid of the second-order central difference time stepping scheme (see for instance [BCB76]), the nodal displacements, velocities and accelerations at every time step are computed as

$$\mathbf{d}_{n+1} = \mathbf{d}_n + \Delta t\, \mathbf{v}_n + \frac{1}{2}\Delta t^2 \mathbf{a}_n, \tag{8}$$

$$\mathbf{a}_{n+1} = \mathbf{M}^{-1}(\mathbf{R}^{\mathbf{in}}_{n+1} + \mathbf{R}^{\mathbf{ex}}_{n+1}), \tag{9}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \frac{1}{2}\Delta t(\mathbf{a}_n + \mathbf{a}_{n+1}), \tag{10}$$

where a subscript n denotes a quantity computed at time $t = n\Delta t$. The time step size Δt is chosen such that it satisfies the CFL stability condition (see for instance [CMP89])

$$\Delta t = \chi \frac{s_e}{C_d} \qquad \chi < 1, \tag{11}$$

where s_e is the smallest edge in the mesh and χ is Courant number. C_d is the dilatational wave speed, given in the plane strain isotropic case by

$$C_d = \sqrt{\frac{E(1-\nu)}{(1+\nu)(1-2\nu)\rho_o}}, \tag{12}$$

where E is the stiffness of the material and ν is the Poisson's ratio.

To reduce the numerical oscillations inherent in the explicit scheme, artificial viscosity is also incorporated in the finite element formulation [MWC$^+$07].

3 Parallel implementation

The main goal for the parallel implementation of the CFE scheme is to guarantee excellent parallel performance on hundreds of processors while quickly parallelizing the initial serial Fortran code. The implementation adopted in this work uses ParFUM, a flexible framework for building parallel unstructured mesh based Finite Difference, Finite Volume, or Finite Element applications [LCW$^+$06]. This section summarizes the steps used to parallelize the extrinsic cohesive element scheme described in the previous section, while section 5 describes the efficiency and scalability of the resulting parallel implementation, which is portable across most major types of high performance systems including clusters, shared memory machines, and custom parallel machines.

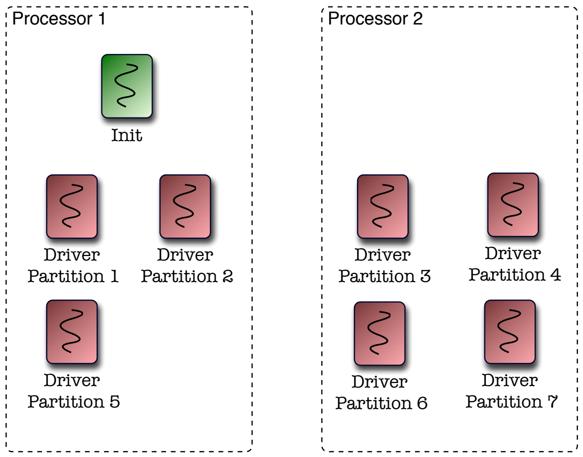


Figure 3: Task decomposition for a ParFUM application. `init` runs once, and `driver` runs once for each partition of the mesh. Each `driver` is associated with an MPI process, with potentially large and varying numbers of `driver` routines on each physical processor.

To port the serial CFE application to the parallel ParFUM framework, a key modification is to split the code into two parts as shown in Figure 3. The first part of the application is an `init` subroutine that loads an unpartitioned mesh on a single processor. The second part is a `driver` subroutine that performs the majority of the computation across multiple processors. After `init` has finished, ParFUM partitions the mesh, creates ghost layers, and builds communication lists. Then the user's `driver` routine runs in parallel, with an instance associated with each partition of the mesh. The `driver` routine creates some auxiliary data structures and then performs the explicit integrations for the associated mesh partition in a timestep loop. The `driver` routine also instructs ParFUM to synchronize values along the partition boundaries during each timestep. This synchronization is described later in more detail.

3.1 Parallelization issues

Parallelizing the serial CFE solver is complex because the scheme uses an unstructured heterogeneous mesh containing both triangular volumetric elements and quadrilateral cohesive elements. Furthermore, the mesh is non-manifold because it contains topological holes. These holes are a problem because a layer of ghost volumetric elements must be constructed along each partition's boundary. The ghost layer must be carefully constructed so that the non-manifold topology of the mesh does not preclude neighboring volumetric elements from being included in a ghost layer for a partition. To create appropriate ghost layers, the implementation registers only the homogeneous triangular mesh with ParFUM, while maintaining its own auxiliary data structures and connectivity tables for the cohesive elements. The ghost layers are then generated by ParFUM using this homogeneous manifold mesh.

In most parallel FE applications written for distributed memory machines, the mesh is partitioned and distributed across the nodes in the machine with one or more partitions belonging to each processor. In addition to each partition, a set of *ghost elements* and *nodes* is required. These *ghost elements* are essentially read-only copies of elements from a neighboring partition. Values, such as displacements and forces, associated with the elements in the *ghost layer* are updated from the original elements. In ParFUM there are *ghost elements* and *ghost nodes* which are read-only copies of elements and nodes from a neighboring partition. Collectively, the set of *ghost nodes* and *ghost elements* comprise a partition's *ghost layer*.

Different types of ghost layers need to be supported because FE applications have differing requirements depending upon the order of the integration scheme. Common schemes require a ghost layer one or two elements deep. ParFUM supports a generic specification for the ghost layers to generate when it partitions the initial mesh. In a triangular mesh, two common types of ghost layers are used. Figure 4(a) displays the first type which specifies that an element should be included in the ghost layer if it has at least one node in common with a local element. Figure 4(b) displays the second type of ghost layer which includes any element in the ghost layer if it shares an edge with a local element.

The problem with the extrinsic CFE parallel implementation is that after cohesive elements are added to the mesh, the heterogeneous mesh contains topological holes at vertices of the original triangular mesh, as illustrated in Figure 5. The presence of a hole causes problems when generating ghost layers. To solve this problem, only the manifold triangular mesh is registered with ParFUM initialization routine. The ghost layers are specified to be node-based as in Figure 4(a), so that any remote triangular bulk element sharing a node with a local element is included in the ghost layer. In the driver routine, auxiliary data structures are created to represent the cohesive elements in the mesh. Thus the application code maintains a heterogeneous mesh while the framework created ghost elements for the simpler homogeneous triangular mesh. The creation of the application's cohesive element data structures in the driver routine is identical to the initial mesh creation code used in the original serial version.

The nodal data associated with the ghost elements is synchronized at each timestep. The application synchronizes only nodal data, such as forces

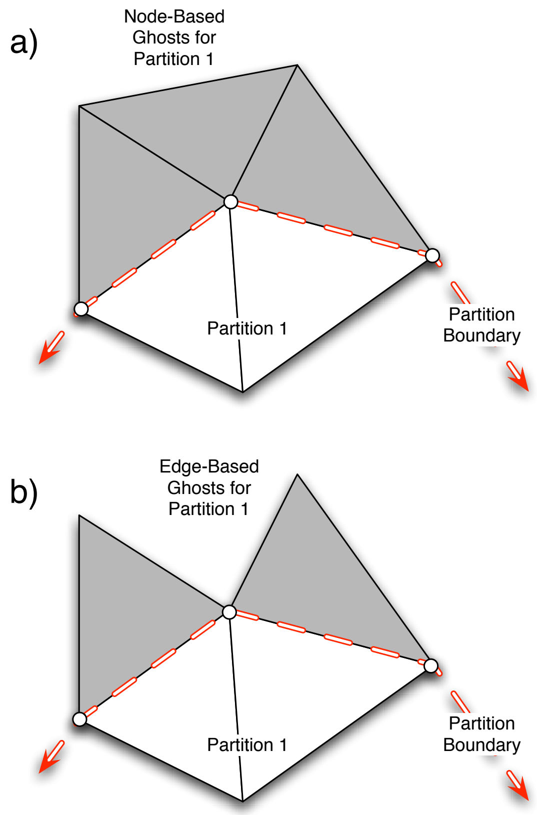

a)
Node-Based Ghosts for Partition 1
Partition 1
Partition Boundary
b)
Edge-Based Ghosts for Partition 1
Partition 1
Partition Boundary

Figure 4: Two types of ghost layers supported by ParFUM. a) Three elements included in the ghost layer for Partition 1 because they share at least one node with an element in Partition 1. b) Two elements included in the ghost layer for Partition 1 because these two share edges with triangles in Partition 1.

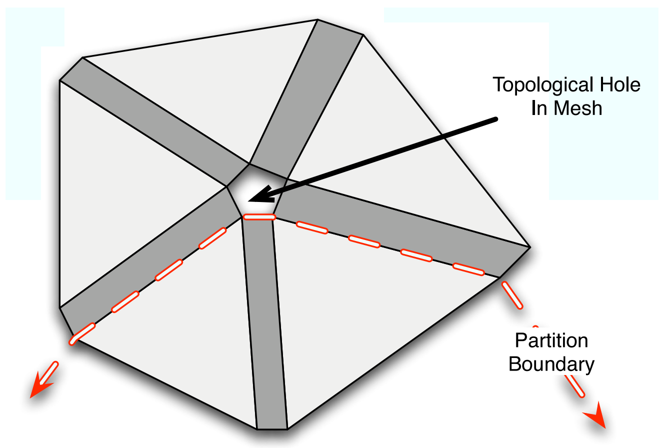


Figure 5: A topological hole is present in the heterogeneous mesh wherever a single node is present in the original triangular mesh.

and masses. The synchronization copies the data from the nodes where the data is computed to any corresponding ghost copies on adjacent partitions. Specifically this is done by treating the nodal data as element data because of the non-manifold topology of the CFE mesh. There are multiple copies of a node for each original node, but each with its own nodal data. Each of these copies of a node is associated with one triangular element. The data is copied to the elements prior to synchronization and then copied back to the nodes after synchronization. This process is shown schematically in Figure 6. Overall, only a small number of lines of code is required for the whole ghost value synchronization process because the framework handles the significant work of building all required send and receive lists as well as the data packing and sending.

3.2 Load Balancing

Maintaining a uniform load balance across processors is critical to obtaining good performance and scalability to large numbers of processors. ParFUM uses METIS[KK98] to partition the mesh. The result of the partitioning technique is that each partition contains approximately the same number of elements, and the sum of the lengths of the partition boundaries is approximately minimized. Because the size of the partition boundaries is approximately minimized, the associated communication volume occurring across the partition boundaries is also approximately minimized. Because each partition contains approximately the same number of elements, the

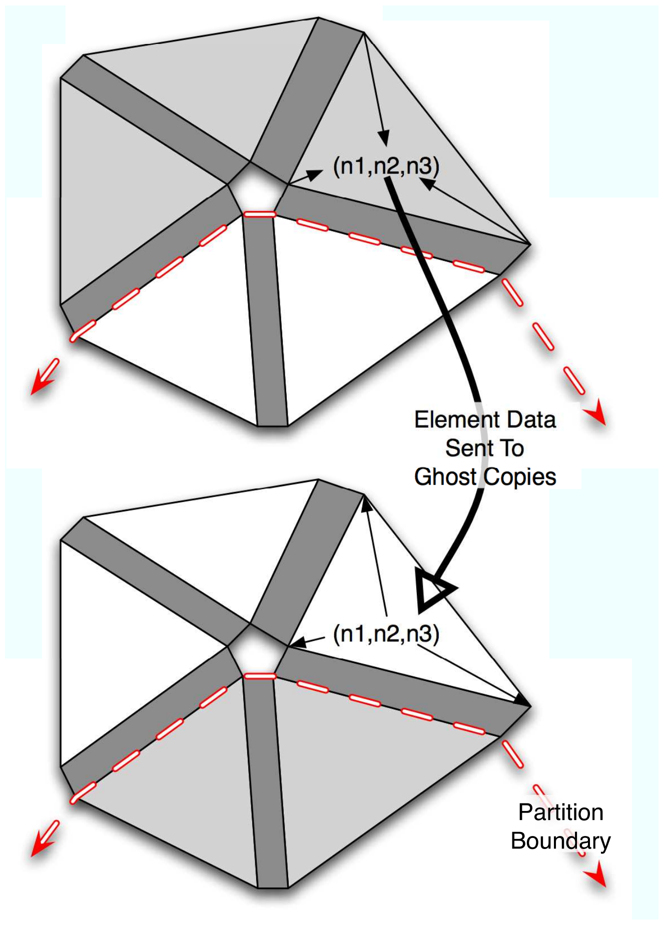


Figure 6: Nodal data is copied to the elements and the elements synchronize their tuples of nodal data to their ghost copies. The synchronized data is propagated to nodes from the recipient ghost elements.

per-timestep computational costs for each partition are almost identical. The n partitions are distributed to the p processors after partitioning. The resulting distribution of partitions places n/p partitions on each processor. Because all processors contain the same number of partitions, each processor will have a similar computational load. As expected, no significant load imbalance was detected for executions of the parallel CFE implementation. The parallel efficiency of the implementation is 95% on 256 processors. This necessarily means that the load is balanced well across the processors. Further discussion regarding the scalability is in section 5.

3.3 Floating-point accuracy and stability

Dynamic fracture problems are unstable physically. This inherent instability translates into a numerical instability where tiny differences in computed forces can cause inconsistent results between different implementations of the method. The inconsistent results are divergences in the crack paths and times during which cohesive elements are activated. Both the floating point precision, and the ordering of nodal force vector summations can impact the results. The implementers of any similar dynamic fracture method in serial or parallel should carefully consider the ramifications and costs associated with their numerical methods in the context of floating point math.

Floating point addition is neither associative nor commutative, [RBSF93]. The classic solution to numerical problems that arise when summing values, such as force contributions, is to sort all the values then sum them from smallest to largest. Sorting however is expensive and significantly complicates an application's source code. FE codes commonly iterate over all elements adjacent to a given node, accumulating a sum of forces. In a parallel implementation, the elements in the mesh are partitioned, and their relative orderings around a node may differ between two partitions. When the relative orderings are different, the sum of the forces could also be different.

To address these problems with floating-point arithmetic, two versions of the CFE method were created. The first version simply sums contributions as it iterates over the elements. 64-bit double precision is used for all values. This version does not achieve identical results when the mesh is partitioned into different numbers of partitions. The second version retrieves all force contributions, sorts these force contributions, then sums them, while using 128-bit precision for some operations. The code for this second version is more complex and the resulting compiled program uses more memory while executing slower. However, this slower version produces consistent results when the original serial mesh is partitioned differently. The first version executes 4.3 times faster than the slower version

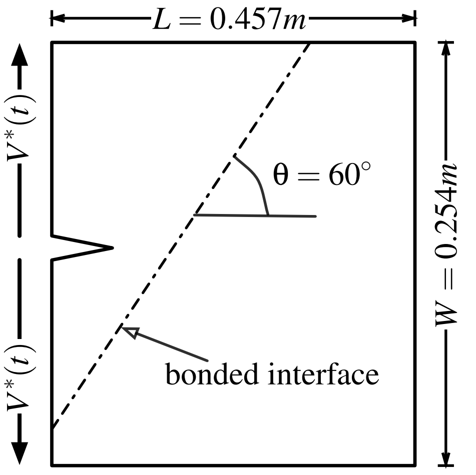


Figure 7: Schematic of the inclined interface fracture problem (not to scale). The initial crack length is $a_0 = 29.58mm$ and the inclined interface is located $46.82mm$ ahead of the initial crack tip.

when both operate on a single partition of 1554 volumetric elements. The changes required to produce the consistent but slower version include sorting the cohesive element force contributions before summing to the nodes, sorting the volumetric element force contributions before adding to nodes, sorting the mass contributions before summing to the associated nodes, and computing the stresses for the volumetric elements in 128-bit precision. Although the faster version may be less accurate, section 4 shows that it produces physically meaningful results, and therefore this version is used instead of the slower version.

4 Validation study

The inherent instability of dynamic fracture processes tend to cast some doubts on the ability of the extrinsic CFE scheme to provide accurate predictions of actual fracture events in which the crack path is not prescribed *a priori*.

To address this issue, we now turn our attention to a validation exercise in which we use the parallel extrinsic CFE code to solve the dynamic fracture problem depicted in Figure 7. This problem, which has been investigated experimentally [XR03], consists of a pre-notched compact tension specimen made of Homalite 100, with length $L = 0.457m$, width

$W = 0.254m$, initial crack length $a_0 = 0.02958m$. Dividing the domain almost diagonally, an inclined bonded interface has been introduced in the specimen at an angle $\theta = 60$ degrees, creating a straight material interface that interferes with the propagation of the rapidly propagating crack. The bonded interface has failure properties that are different from those of the bulk material. The bulk material properties of Homalite-100 are defined by the Young's modulus $E = 3.45GPa$, Poisson's ratio $\nu = 0.35$ and material density $\rho_0 = 1230kg/m^3$ [XR03]. Two interface strengths have been investigated: a strong interface obtained with a Weldon-10 adhesive, and a weaker one for which Loctite-384 was used. The failure properties of the various constituents (bulk material and interface) have been extracted experimentally and are listed in Table 1.

Table 1: Failure properties of the bulk material and of the weak and strong interfaces used in the dynamic failure study of the inclined interface.

		Homalite-100	Locite-384	Weldon-10
			(weak)	(strong)
		[KM78]	[XR03]	[XR03]
σ_{max}	(MPa)	11.0	6.75	7.74
τ_{max}	(MPa)	25.0	7.45	22.0
G_{Ic}	(J/m^2)	250.0	41.9	46.4
G_{IIc}	(J/m^2)	568.0	199.7	568.0

To model the wedge-induced loading used in the experimental study, we adopt in this work a time-dependent vertical velocity $V^*(t)$ applied upward (downward) along the upper (lower) left edge of the domain, as illustrated in Figure 7. As earlier, the applied velocity follows a linear ramp from 0 at $t = 0$ to $V_0 = 0.8m/s$ at $t = t_{ramp} = 0.0093L/C_d$ and then remains constant. Due to the tensile nature of the applied load, the crack propagates primarily in mode I, and hence travels along a straight line before meeting the interface. Then, depending on the strength of the interface, the crack either deflects into the interface and propagates under mixed-mode condition, or penetrates through the interface. It should be noted at this point that this problem constitutes an excellent testbed for the extrinsic CFE scheme since all the quantities entering the description of the geometry, loading and material properties have been determined experimentally, leaving absolutely no fitting parameters.

Due the relatively large size of the fracture specimen (and in the absence of mesh refinement), the domain is discretized into $1,200,414$ 3-node triangular constant strain elements with $1,801,909$ interfacial 4-noded co-

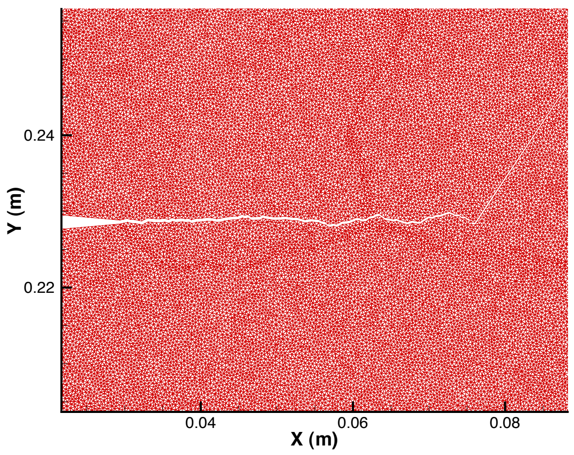


Figure 8: Details of the deformed mesh in the vicinity of the initial crack tip and the inclined interface for the domain in Figure 7.

hesive elements. A detail of the mesh in the vicinity of the initial crack tip is shown in Figure 8. The average number of elements in the cohesive zone for this discretization are 6.5 for Homalite-100, 3 for Weldon-10 and 10.5 for Loctite-384. The simulations are performed on 16-processors with about $75,000$ elements per processor. A Courant number $\chi = 0.05$ is used.

Figure 9 presents snapshot of the S_{22} stress contours at time $t = 1.857L/C_d$ for the weak and strong interface cases, clearly illustrating the existence of a sharp stress concentration in the vicinity of the propagating crack tip. Figure 10 shows the close-up view of Figure 9 showing the details of crack trajectory near the inclined interface. At that moment in the simulation, the crack has completed its motion along the interface and has resumed its propagation in the right half of the Homalite specimen. The difference in the resulting crack path, already apparent in Figure 9 is further visualized in Figure 11, which presents a direct comparison between the two crack trajectories. As anticipated, the weaker interface traps the crack for a much longer time than its stronger counterpart, as its lower fracture toughness makes it energetically more favorable for the crack to propagate under mixed-mode conditions. In both cases, the mode I crack propagation eventually prevails and the crack kinks out of the interface. This predicted behavior is in good qualitative and quantitative agreement with the observations by [XR03], especially for the strong interface case for which the predicted crack length along the interface ($5.0mm$) compares very favorably with the observed value ($4.3mm$).

The interaction of the crack with the interface is further illustrated in

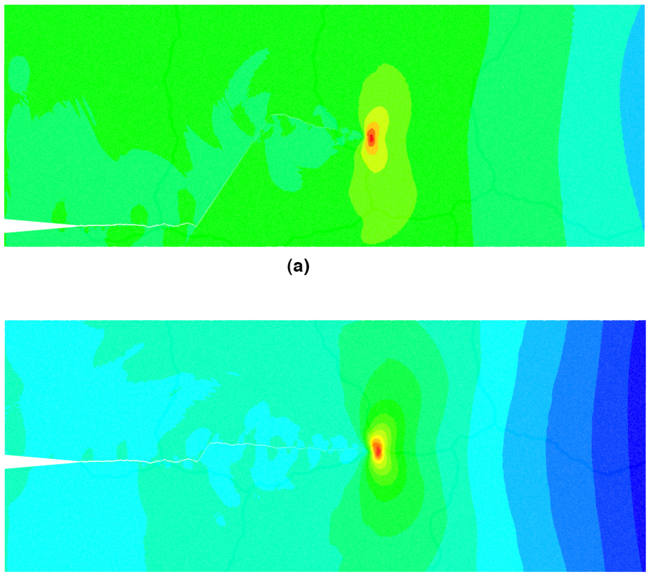
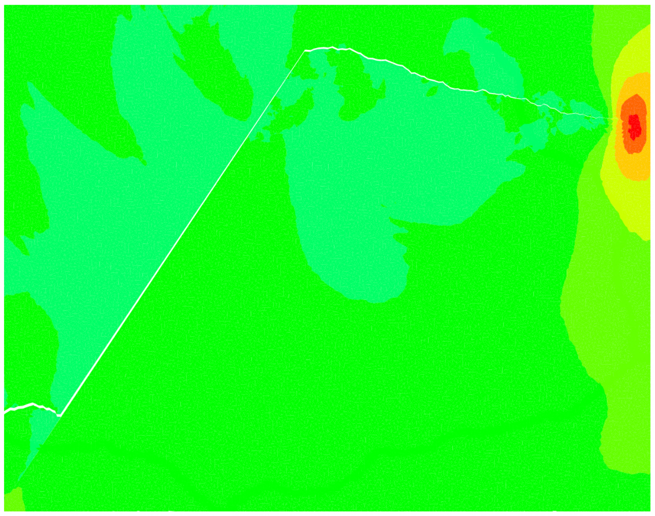
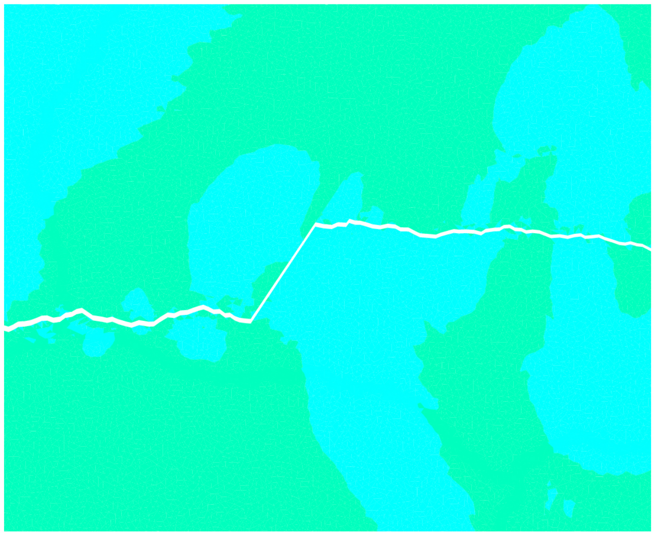


Figure 9: σ_{22} stress contour plot at time $t = 1.857L/C_d$ for (a) weak interface strength, showing the trajectory of the crack trapped momentarily along the inclined interface and (b) for the strong interface case.



(a)



(b)

Figure 10: Close-up view of the two cases from Figure 9, showing details of crack path near the vicinity of the inclined interface: (a) weak interface, (b) strong interface.

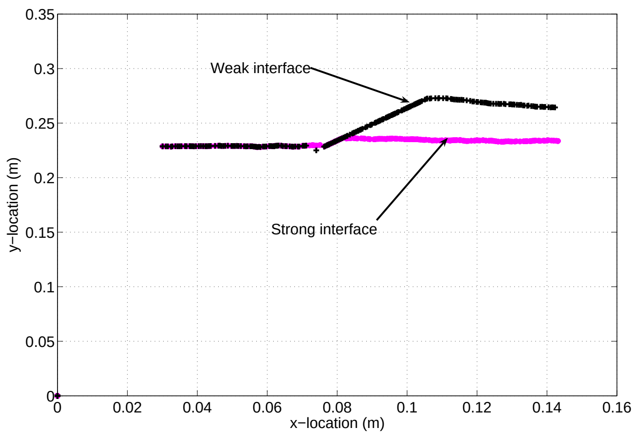


Figure 11: Comparison between the crack trajectory obtained for the strong and weak interfaces.

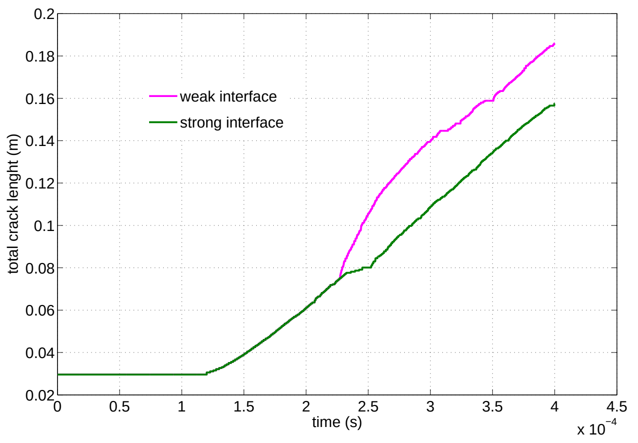


Figure 12: Total crack length *vs*. time for the strong and weak interface cases.

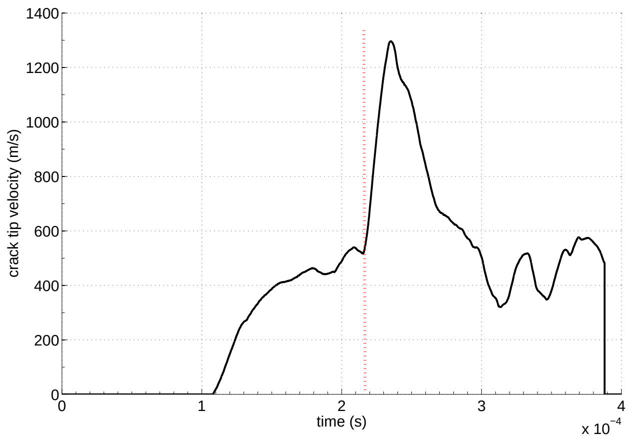


Figure 13: Evolution of crack tip velocity for the weak interface. The dashed vertical line shows the instant the crack hits the interface.

Figure 12, which shows the evolution of the total crack length *vs* time obtained for the strong and weak interfaces. As expected, the initial stage of the crack motion is identical for the two cases. As the crack reaches the interface, however, the curve corresponding to the weak interface shows a marked change in its slope, indicating a sudden acceleration of the crack. This transient crack motion is also illustrated in Figure 13, which shows the evolution of the crack speed for the weak interface case. After an incubation period associated with the creation of the transient stress concentration in the vicinity of the initial crack tip, the initially mode I crack quickly accelerates to reach the experimentally observed speed of about $400m/s$. As it reaches the interface (at the time indicated by the dashed line), the crack accelerates rapidly before decelerating to the observed value of about 650 to $700m/s$. After kinking out of the interface, the crack velocity drops back to its initial value of about $400m/s$. It should be noted, however, that the peak in the crack tip speed obtained during the initial stages of the interfacial failure (about $1200m/s$) exceeds substantially that observed by [XR03]. This might be due to the inaccuracy in the description of the loading conditions, and, in particular, with the absence of compressive (lateral) component of the applied velocity. This error in peak velocity also explains the discrepency of the calculated crack length along the weak interface from the experimental value.

5 Analysis of parallel performance

The parallel implementation of the CFE scheme exhibits excellent scaling to hundreds of processors. This section describes the measured performance and its dependence upon an interesting parameter called virtualization. First the performance evaluation methodology is described. Then the performance results and their relationships to degree of virtualization are characterized. Specifically, cache effects are discussed in relationship to the degree of virtualization. Finally this section shows that the parallel implementation scales almost perfectly to 512 processors. The communication costs that can limit scalability are quantified to explain the good scaling results.

The parallel scaling runs described in this section were performed on the Turing cluster at the University of Illinois. Each node in the cluster is an Apple Xserve with dual 2.0 *GHz* G5 processors. The nodes are connected via a Myrinet network that provides an MPI latency of 3.5 μs between pairs of nodes. The data bandwidth rates to and from each node's network interface are 250 *GB/s*. All performance results in this paper were obtained by using both processors on a node for computation. For example, the 256 processor timings were performed on 128 dual-processor nodes. The test problem used for the performance results presented in this section was the same cohesive finite element problem described in Section 4 on a mesh containing 1.2 million elements.

5.1 Benefits due to virtualization

Since the implementation uses the ParFUM framework to implement the CFE method, the user can configure a runtime parameter called virtualization. Virtualization in ParFUM is defined to be the average number of mesh partitions per physical processor. The term virtualization comes from AMPI [HZKK06] where multiple MPI processes, or virtual processors, are run inside a single physical processor. ParFUM is built partially upon AMPI, and accordingly it inherits this terminology.

Using multiple mesh partitions per processor decreases the runtime of the implementation. On 8 physical processors, the execution time of 1000 timestep loop iterations is 145 seconds when using 1 partition per processor. The time decreases by 20% to 116 seconds when using 32 partitions per processor. Beyond 32 partitions per processor, the time increases due to extra overhead. Figure 14 shows that the best number of partitions for 8, 16, and 32 processors are 256, 512, and 1024 respectively. This optimum performance point or sweet spot can be determined experimentally, but approximate rules of thumb can also be used for a particular application. To maximize the performance of this implementation, each partition should

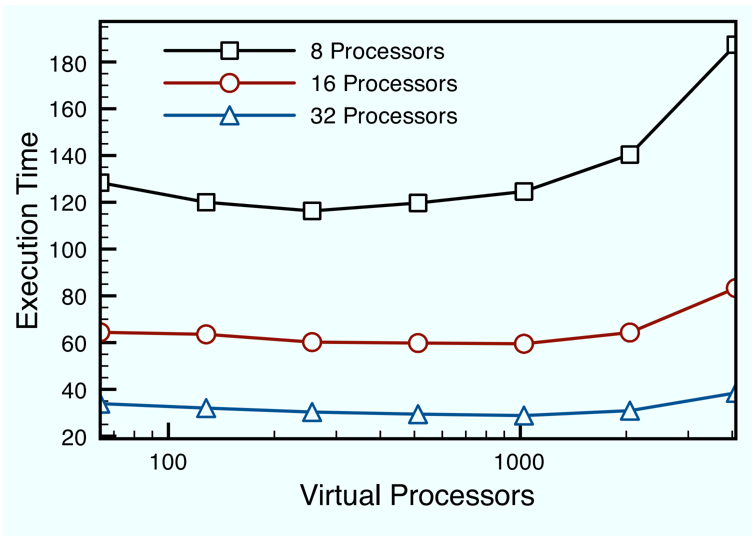


Figure 14: Execution time (in seconds) with varying numbers of mesh partitions (Virtual Processors).

contain between one thousand and four thousand elements. The two main reasons that performance can improve when the number of mesh partitions increases are that communication can overlap computation and that the memory hierarchy access patterns change.

One reason that performance increases when using more than 1 partition per processor is that the working set of data for smaller partitions is smaller and hence can better fit in the smaller faster cache memory in the processor [Kal04]. Figure 15 shows the cache performance of the CFE implementation on 8 processor cores within a single SMP compute node of the NCSA Abe cluster. This cluster was used to test cache performance because it provides support for PAPI performance counters [BDG+00], whereas Turing did not provide such support. The PAPI performance counter library was used to measure the total number of L2 cache misses, `PAPI_L2_TCM`, for the driver calls on all 8 processor cores. The application execution time was also measured with the PAPI counters disabled to minimize any perturbations to these reported times. As expected, the L2 cache misses decrease when the mesh is divided into increasing numbers of partitions. The cache performance correlates well with the application's speed on this system. As the number of partitions increases, other overheads eventually become larger and the overall application performance

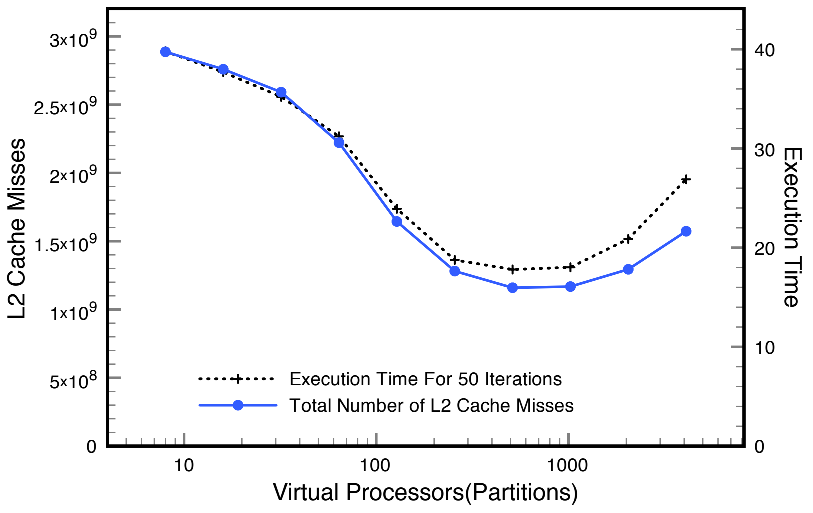


Figure 15: Over-decomposing the mesh into multiple partitions per processor increases application performance by using the smaller faster levels of cache memory more effectively. With too much overdecomposition, other overheads reduce performance.

degrades.

5.2 Scalability

The speedup gained by using virtualization is only one performance concern. The most important performance concern is scalability to a large number of processors. One common measure of the performance and scalability of a parallel application is speedup which is defined to be the ratio of the parallel runtime to the serial one processor runtime. For the results presented in this section, the baseline serial one processor version is the CFE application program run with only one partition on one processor. Thus the serial version has no parallel communication overhead. Figure 16 displays the speedup of the CFE implementation for up to 512 processors while using 512 partitions in all cases. Table 2 lists the same speedup data along with corresponding execution times and parallel efficiencies. The parallel efficiency is a ratio of the speedup to the number of processors used. A parallel efficiency of 1.0 on p processors means that the parallel version perfectly scaled to be p times faster than the serial version.

The parallel CFE implementation scales almost perfectly to 256 pro-

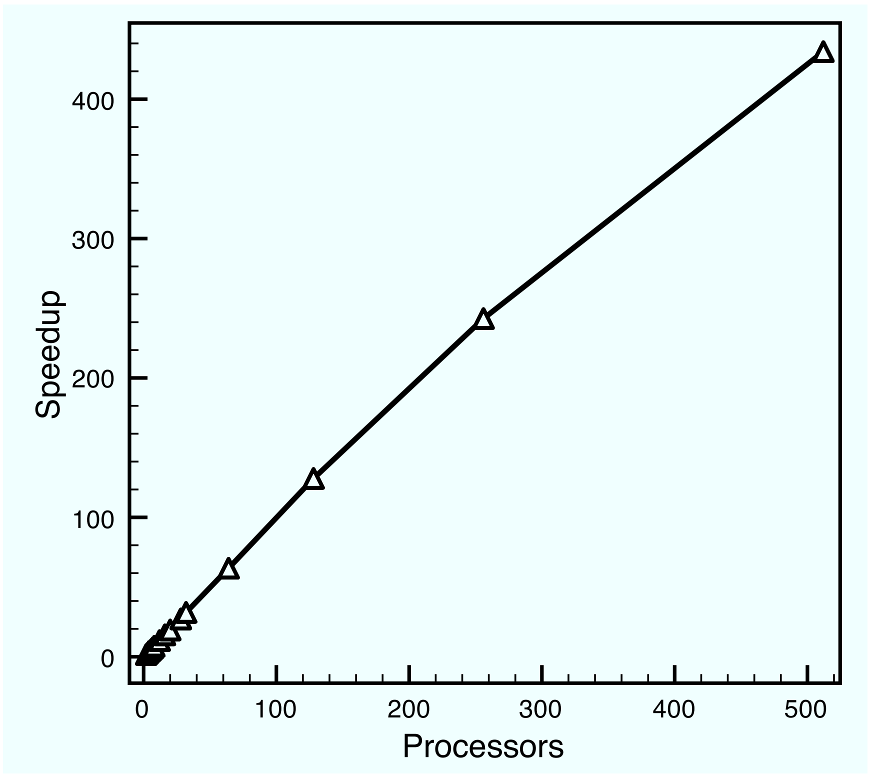


Figure 16: Speedup for CFE implementation up to 512 processors for dynamic fracture problem using a 1.2 M element mesh divided into 512 mesh partitions (Virtual Processors). A detailed listing of these speedups along with execution times and parallel efficiencies is found in Table 2.

Table 2: Execution time, speedup, and parallel efficiency for CFE application on up to 512 processors. This problem uses a 1.2 M element mesh divided into 512 mesh partitions (Virtual Processors). The execution times are for 1000 timestep iterations.

Processors	Execution time (s)	Speedup	Parallel Efficiency
1	946.3		
2	581.7	1.63	0.81
3	366.4	2.58	0.86
4	255.6	3.7	0.93
5	209.8	4.51	0.9
6	171.5	5.52	0.92
7	145.7	6.49	0.93
8	125.9	7.52	0.94
12	81.3	11.64	0.97
16	60.2	15.72	0.98
20	49.2	19.23	0.96
28	34.9	27.11	0.97
32	29.8	31.76	0.99
64	14.9	63.51	0.99
128	7.4	127.88	1.00
256	3.9	242.64	0.95
512	2.18	434.08	0.85

cessors. At 512 physical processors, there is no benefit from virtualization because the 512 partitions are mapped one-to-one onto the physical processors. Thus there is one virtual processor on each physical processor and hence no opportunity to gain the benefits of virtualization. The sub-optimal parallel efficiencies for the small number of processors occur in the CFE program because messages are sent between processors, unlike the serial version which has no communication overhead. This extra communication cost incurred in the parallel implementation hurts performance for small numbers of processors, but the cost is eventually amortized across processors as the number of processors increases. The overall near perfect speedups observed are expected, because there is enough work performed for each mesh partition relative to the amount of communication required for each mesh partition. The scalabilities of nearest-neighbor communicating parallel programs are expected to be high in such cases where enough work is performed for each partition.

Because the parallel implementation scales well to 512 processors, the communication costs of our program are not bottlenecks when scaling the application with a 1.2 M element mesh. Figure 17 shows the number of messages received and the total bytes received in a timestep by each of 32 processors. The maximum size of the messages destined for one of the processors is 209 KB spread across 33 messages. In this case the communication volume per node is significantly lower than the maximum bandwidth supported by the interconnection network. The network interface on each node is able to receive 7.5 MB in the duration of a single timestep, 0.03s. Thus with two processors per node, each processor has an available effective bandwidth of 3.75 MB. Then the maximum incoming communication volume for any of the 32 processors, 209 KB, uses only 6% of the available effective bandwidth. The communication costs become a bottleneck if they become a large portion of the timestep duration. At 512 processors, 20% of the effective bandwidth during a timestep is used. This higher communication cost is starting to reduce parallel efficiency. Even though the communication is starting to become important at 512 processors, the communication volume is small enough that it does not dominate the overall time of the parallel execution time.

6 Conclusion

This paper describes a novel methodology for parallel implementation of extrinsic cohesive elements based on activation of elements, implemented using the Parallel framework for Unstructured Meshes (ParFUM). The developed parallel CFE scheme was validated against the experimental studies performed on the dynamic crack deflection-penetration behavior in inho-

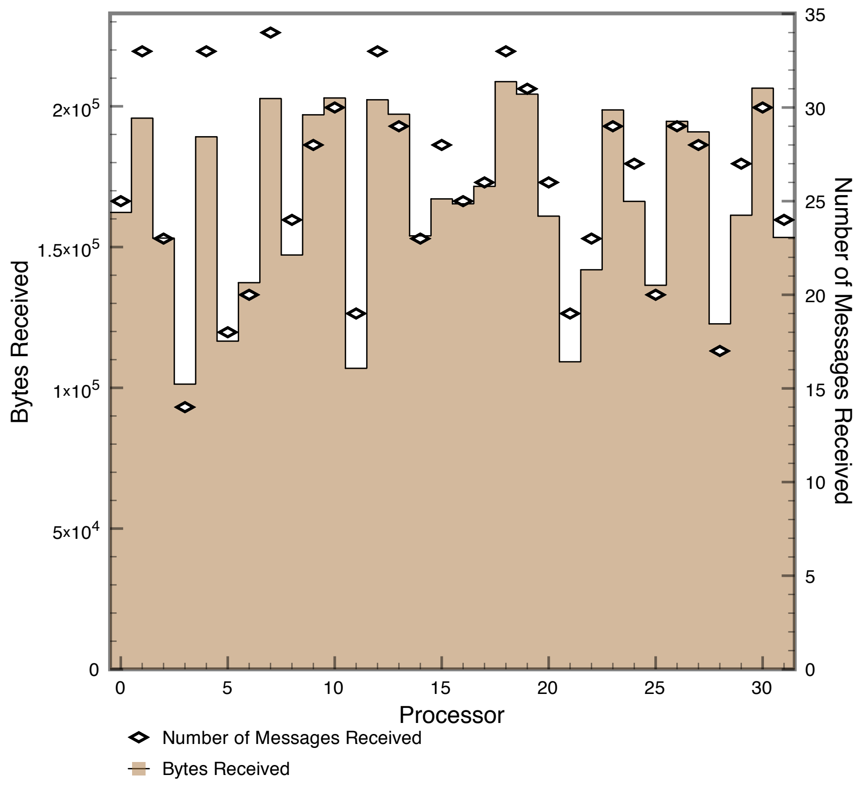


Figure 17: Number of messages and total bytes received by each of 32 processors for a single timestep. Communication between multiple partitions within a single processor is not included in this plot. The 1.2 M element mesh is decomposed into 512 partitions (Virtual Processors).

mogeneous specimens by [XR03]. Simulated results are in very good agreement with experimental observations both in terms of predicted fracture path and crack speed. A detailed scalability study performed on up to 512 processors shows excellent speedup for the parallel cohesive finite element solver. The scalability is expected because the quantified communication volume does not dominate the time required for a timestep. The measured cache effects for the code explain the observed speedup when increasing the number of partitions on each processor. This paper also describes how floating point arithmetic can produce different crack path results if forces are summed in differing orders, as occurs when a mesh is partitioned into differing numbers of partitions. A solution based on increasing floating point precision and sorting forces is provided.

7 Acknowledgements

The authors gratefully acknowledge the support of NSF through grant EIA 01-03645, and of the Center for the Simulation of Advanced Rockets under contract number B341494 by the U.S. Department of Energy, the U.S. Department of Energy HPCS Fellowship Program, the Computational Science and Engineering Program at the University of Illinois for use of their Turing cluster, and the National Center for Supercomputing Applications for use of their Abe cluster.

References

[BBJ$^+$97] P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. RentzReichert, and C. Wieners. UG – a flexible software toolbox for solving partial differential equations, 1997.

[BCB76] T. Belytschko, R. L. Chiapetta, and H. D. Bartel. Efficient large scale non-linear transient analysis by finite elements. *International Journal for Numerical Methods in Engineering*, 10:579–596, 1976.

[BDG$^+$00] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A Portable Programming Interface for Performance Evaluation on Modern Processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, 2000.

[BGMS97] Satish Balay, William Gropp, Lois Curfman McInnes, and Barry Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[BRN06] R. Borst, J.J.C Remmers, and A. Needleman. Mesh-independent discrete numerical representations of cohesive-zone models. *EFM*, 73:160–177, 2006.

[CMP89] R. D. Cook, D. S. Malkus, and M. E. Plesha. *Concepts and Applications of Finite Element Analysis.* John Wiley & Sons, fifth edition, 1989.

[CO97] G. T. Camacho and M. Ortiz. Adaptive Lagrangian modelling of ballistic penetration of metallic targets. *Computer Methods in Applied Mechanics and Engineering*, 142:269–301, 1997.

[FNR01] M. L. Falk, A. Needleman, and J. R. Rice. A critical evaluation of dynamic fracture simulations using cohesive surfaces. *Journal de Physique IV*, 11:43–52, 2001.

[GB98] P. H. Geubelle and J. S. Baylor. Impact-induced delamination of composites: a 2D simulation. *Composites Part B*, 29B:589–602, 1998.

[HZKK06] Chao Huang, Gengbin Zheng, Sameer Kumar, and Laxmikant V. Kalé. Performance evaluation of adaptive MPI. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006*, March 2006.

[Kal04] Laxmikant V. Kalé. Performance and productivity in parallel programming via processor virtualization. In *Proc. of the First Intl. Workshop on Productivity and Performance in High-End Computing (at HPCA 10)*, Madrid, Spain, February 2004.

[KG03] D. V. Kubair and P. H. Geubelle. Comparative analysis of extrinsic and intrinsic cohesive models of dynamic fracture. *International Journal of Solids and Structures*, 40:3853–3868, 2003.

[KK96] L. V. Kale and Sanjeev Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In Gregory V. Wilson and Paul Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.

[KK98] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96 – 129, 1998.

[KM78] A. S. Kobayashi and S. Mall. Dynamic fracture toughness of homalite-100. *Experimental Mechanics*, 18:11–18, 1978.

[LCW$^+$06] Orion Lawlor, Sayantan Chakravorty, Terry Wilmarth, Nilesh Choudhury, Isaac Dooley, Gengbin Zheng, and Laxmikant Kale. Parfum: A parallel framework for unstructured meshes for scalable dynamic physics applications. *Engineering with Computers*, 22(3-4):215–235, 2006.

[MWC$^+$07] Sandhya Mangala, Terry Wilmarth, Sayantan Chakravorty, Nilesh Choudhury, Laxmikant V. Kale, and Philippe H. Geubelle. Parallel adaptive simulations of dynamic fracture events. *Engineering with Computers (accepted for publication)*, 2007.

[Nee97] A. Needleman. Numerical modeling of crack growth under dynamic loading conditions. *Computational Mechanics*, 19:463–469, 1997.

[PSV03] K. D. Papoulia, C-H. Sam, and S. A. Vavasis. Time continuity in cohesive finite element modelling. *International Journal for Numerical Methods in Engineering*, 58:679–701, 2003.

[RBSF93] Fred Ris, Ed Barkmeyer, Craig Schaffert, and Peter Farkas. When floating-point addition isn't commutative. *SIGNUM Newsl.*, 28(1):8–13, 1993.

[RKFS01] Jean-François Remacle, Ottmar Klaas, Joseph Flaherty, and Mark Shephard. Parallel algorithm oriented mesh database. In *Proceedings of the 10th International Meshing Roundtable*, pages 197–208, October 2001.

[RPO01] G. Ruiz, A. Pandolfi, and M. Ortiz. Three-dimensional cohesive modeling of dynamic mixed-mode fracture. *International Journal for Numerical Methods in Engineering*, 52:97–120, 2001.

[SE04] James R. Stewart and H. Carter Edwards. A framework approach for developing parallel adaptive multiphysics applications. *Finite Elements in Analysis and Design*, 40:1599–1617, 2004.

[XR03] Y. Y. Xu, L. R.and Huang and A. J. Rosakis. Dynamic crack deflection and penetration at interfaces in homogeneous materials: experimental studies and model predictions. *Journal of the Mechanics and Physics of Solids*, 51:461–486, 2003.