Parallel Simulations of Dynamic Fracture Using Extrinsic Cohesive Elements

Isaac Dooley, Graduate Student*
Sandhya Mangala, Graduate Student†
Laxmikant Kale, Professor‡
Philippe Geubelle, Professor§

July 12, 2007

Suggested Running Head:
Parallel Simulations of Dynamic Fracture Using Extrinsic Cohesive Elements

Submission For:
Journal of Scientific Computing (Springer)

Address(for office use):
Isaac Dooley
Seibel Center for Computer Science
201 North Goodwin Ave
Urbana, IL, 61801
phone: 217-244-4583
fax: 217-265-4035
idooley2@uiuc.edu

*Department of Computer Science, University of Illinois Urbana-Champaign, Urbana, Illinois
†Department of Aerospace Engineering, University of Illinois Urbana-Champaign, Urbana, Illinois
‡Department of Computer Science, University of Illinois Urbana-Champaign, Urbana, Illinois
§Department of Aerospace Engineering, University of Illinois Urbana-Champaign, Urbana, Illinois

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 activiating instead of inserting the cohesive elements and uses ParFUM, a parallel framework specifically developed for simulations involving unstructured meshes. Issues associated with the spatial and temporal convergence of the resulting scheme are discussed, together with aspects of the parallel implementation. We present scalability results obtained with the parallel cohesive finite element code which 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 [**?**, **?**, **?**, **?**]. In this finite element formulation, conventional (volumetric) elements are used to capture the 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

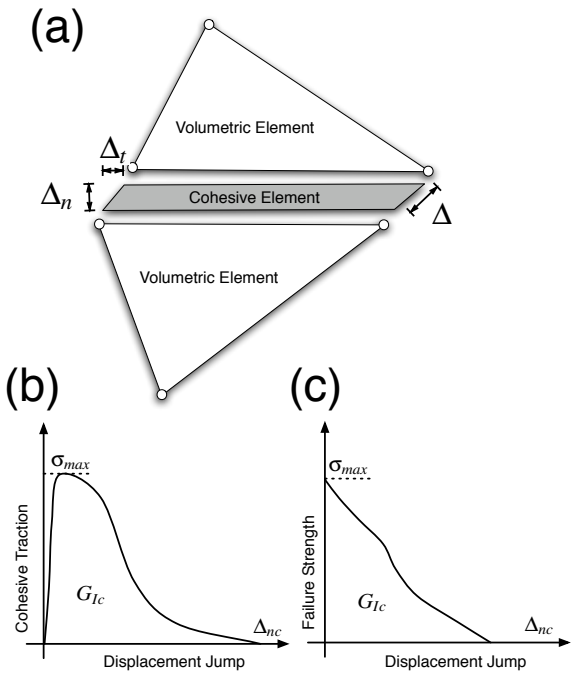

Figure 1: (a) Cohesive finite element concept, showing two 6-node triangular volumetric elements tied by a 6-node cohesive element shown in its deformed configuration. In its initial form, the cohesive element has no area and the adjacent nodes are superposed. (b) Schematic of an intrinsic cohesive failure law for the tensile failure case, for which the tangential displacement jump Δ_t is zero. (c) Generic extrinsic cohesive law.

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 be found in [**?**].

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 [**?**, **?**], 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 [**?**, **?**] and are revisited in Section 4.1 for the case of both straight and arbitrary crack paths.

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 a need for a large domain to capture the loading accurately and avoid premature wave reflections from the boundary. Very 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 [**?**].

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) [**?**], 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 [**?**].

This paper describes the use of the parallel framework ParFUM to implement our CFE scheme. Only a small number of libraries for handling parallel unstructured meshes exist. Other large parallel frameworks could have been used in place of ParFUM. Unfortunately some of the most fully featured production level frameworks are not available to the public and thus would not be suitable candidates for our application. Sandia National Laboratories' *SIERRA*[SE04] is one such unreleased framework. The University of Heidelberg's *UG*[**?**] is a large publicly available framework. Both *SIERRA* and *UG* support fully unstructured meshes on distributed memory supercomputers with a variety of compatible solvers. The *AOMD* framework[**?**] also provides a parallel mesh abstraction which works on distributed memory machines. *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[**?**]. *libMesh* can similarly use parallel solvers in these regards to *deal.ii* [**?**]. The extrinsic scheme proposed in this paper uses explicit timestepping and therefore does not require support for a solver library. The scheme requires just support for partitioning, distributing, and accessing an unstructured mesh on a distributed memory computer cluster.

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 ele-

ments 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 verify and 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 [**?**]. 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 [**?**], in which the cohesive traction $\mathbf{T}$ during the failure process is described by

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

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 $\boldsymbol{\Delta}_t$ are defined by

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

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 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 initated only for tensile loading), reaches the corresponding shear failure strength τ_{max}. A cohesive element completely fails when either of the displacement jump

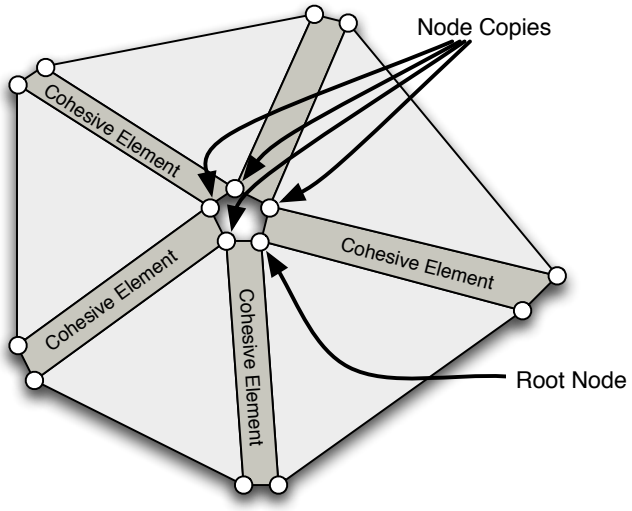

Figure 2: One root is chosen as a representative for the multiple node copies. When surrounding cohesive elements are inactive the node copies have identical displacements and velocities as the root node.

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 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:

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

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:

$$\mathbf{M}\,\mathbf{a} = \mathbf{R^{in}} + \mathbf{R^{ex}} \tag{6}$$

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 [**?**].

With the aid of the second-order central difference time stepping scheme [**?**], the nodal displacements, velocities and accelerations at every time step are computed as

$$\mathbf{d}_{n+1} = \mathbf{d}_n + \Delta t\, \mathbf{v}_n + \frac{1}{2}\Delta t^2 \mathbf{a}_n, \tag{7}$$

$$\mathbf{a}_{n+1} = \mathbf{M}^{-1}(\mathbf{R}^{\mathbf{in}}_{n+1} + \mathbf{R}^{\mathbf{ex}}_{n+1}), \tag{8}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \frac{1}{2}\Delta t(\mathbf{a}_n + \mathbf{a}_{n+1}), \tag{9}$$

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 [**?**]

$$\Delta t = \chi \frac{s_e}{C_d} \qquad \chi < 1, \tag{10}$$

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

$$C_d = \sqrt{\frac{E(1-\nu)}{(1+\nu)(1-2\nu)\rho_o}}, \tag{11}$$

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 [**?**].

3 Parallel implementation

The main goal for the implementation of the CFE scheme was to guarantee excellent parallel performance on hundreds of processors while quickly parallelizing the initial serial Fortran code. This section describes the parallel implementation and some of the design considerations. The implementation uses the ParFUM framework because it is one of the best free, scalable, and portable frameworks that provides support for unstructured meshes.

The ParFUM (Parallel Framework for Unstructured Meshing) framework is a flexible framework for building unstructured mesh based Finite Difference, Finite Volume, or Finite Element applications [**?**]. It can also simplify porting of serial codes that utilize unstructured discretizations to parallel computing platforms. This section summarizes the steps used to parallelize the extrinsic cohesive element scheme. Section 5 describes the efficiency and scalability of the resulting parallel implementation, which is highly portable across most major types of high performance systems including clusters, shared memory machines, and custom parallel machines such as IBM BlueGene/L.

ParFUM is a framework is built upon Charm++ and AMPI[**?**, **?**]. Charm++ is a parallel language and adaptive runtime system that provides a robust, efficient, and portable system for writing parallel programs. The model of parallel programming used in Charm++ is that of migratable objects. A program written in Charm++ is a collection of C++ objects with remote asynchronous method invocations for communication between objects. The Charm++ runtime system determines the mapping of objects to processors and provides dynamic load balancing by migrating objects between processors. AMPI is an adaptive MPI implementation built upon Charm++ that supports multiple MPI processes on each physical processor. Charm++ and AMPI support a variety of dynamic instrumented load balancing schemes [**?**, **?**, **?**, **?**, **?**, **?**] as well as a number of advanced fault tolerance schemes [**?**, **?**, **?**]. ParFUM therefore provides a number of useful productivity enhancing features without requiring a user to be an expert parallel programmer.

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` function that loads a serial mesh on a single processor. The second part is a `driver` function that performs the majority of the computation across multiple processors. After `init` has finished, ParFUM partitions the mesh and builds communication lists. Then the user's `driver` function 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

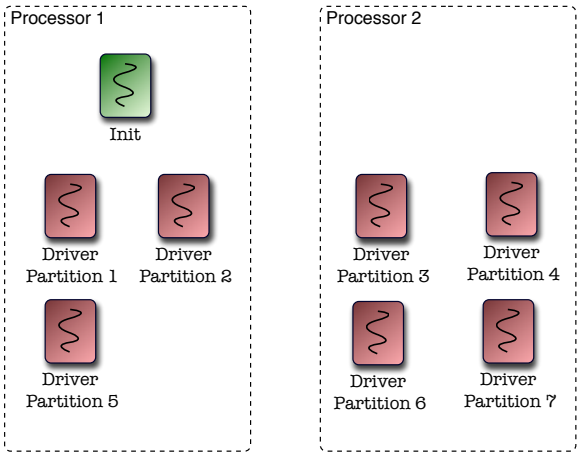

Figure 3: Task decomposition for a ParFUM application. `init` runs once, and `driver` runs once for each partition of the mesh. Each `driver` is associated with an MPI process, with potentially large and varying numbers of `driver` routines on each physical processor.

synchronizes 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 it uses an unstructured heterogeneous mesh containing both triangular volumetric elements and quadrilateral cohesive elements. Although the parallelization is relatively straightforward, planning is required to best use ParFUM's features with such a mesh. The implementer must determine how to form ghost layers which are used to synchronize values from elements on the boundary of one partition to the neighboring partitions. The ghost layer synchronization is closely tied to the topology of the mesh. ParFUM supports ghost layers for heterogeneous meshes, so the first implementation design used a heterogeneous mesh containing triangles and quadrilaterals. A number of difficulties were discovered with this seemingly intuitive design. The impediments were initially non-obvious, so this section describes them. The final implementation registers only the homogeneous triangular mesh with ParFUM, while maintaining its own auxiliary data structures and connectivity tables for the cohesive elements. This section discusses the use of ghost layers in ParFUM as well as some design decisions for our implementation.

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 *shared nodes*, which are nodes that belong to multiple partitions, as well as *ghost elements* and *ghost nodes*, which are read-only copies of elements and nodes from a neighboring partition. Collectively, the set of *ghost nodes*, *ghost elements*, and *shared nodes* for a partition is considered the partition's *ghost layer*.

Different types of ghost layers are supported by ParFUM because FE applications have differing requirements depending upon the order of the integration scheme. Often a layer of depth one or two elements is required. ParFUM supports a generic specification of what type of 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 shares an edge with a local element. Figure 4(b) displays the second type of ghost layer which includes any element in the ghost layer if it has at least one node in common with a local element. ParFUM applications specify the desired type of ghost layer using an abstract set of tuples that defines a neighboring relation. This relation is used to determine if an element is a neighbor of a local element and thus should be included in the ghost layer. One relation would be the set of 3 pairs representing each of the three edges in a triangle as in the case of $\{(0,1),(1,2),(2,0)\}$. The pair $(1,2)$ means that an edge of the element is defined by nodes 1 and 2 of the element. Alternatively three values could specify the nodes of a triangle as in the case $\{(0),(1),(2)\}$. When ParFUM partitions the mesh, it uses the relation specified by the sets of tuples provided by the user in `init` to determine which elements should be included in the ghost layer for a particular partition. The determination of whether to include an element uses a simple criterion. The tuples are a template applied to each element to produce sets of nodes for each element. If any of the resulting sets of nodes for a remote element intersects the set of nodes from some local element in a given partition, the former element is included in the partition's ghost layer. The same tuple relations can also be used in ParFUM to build topological adjacency tables.

The problem with the extrinsic CFE application 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 trying to create

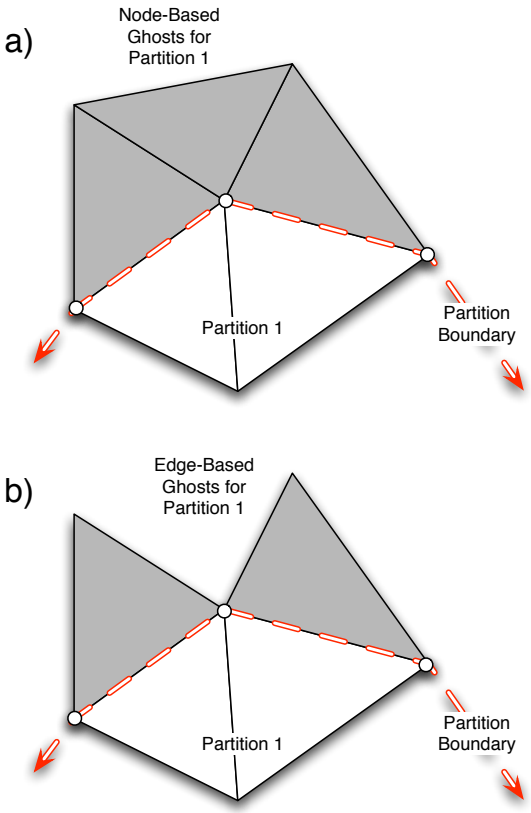

Figure 4: Two types of ghost layers supported by ParFUM. a) shows three elements included in the ghost layer for Partition 1 because they share at least one node with an element in Partition 1. b) shows two elements included in the ghost layer for Partition 1 because these two share edges with triangles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

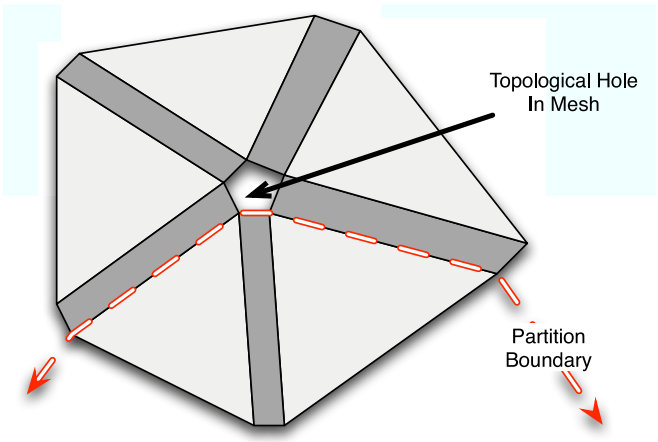

Figure 5: A topological hole is present in the heterogeneous mesh wherever a single node is present in the original triangular mesh.

ghost layers. In order to use ParFUM's automatic ghost layer creation, the user registers a simple triangular mesh with ParFUM in the initialization routine. We specify the ghost layers to be node-based as in Figure 4(a), so that any remote element sharing a node with a local element is included in the ghost layer. In the driver routine, we then create auxiliary data structures to represent the cohesive elements in the mesh. Thus the application code maintains a heterogeneous mesh while the framework only knows about a simple homogeneous triangular mesh. This allows the code to take advantage of the mesh refinement features of the framework in the future, since the framework can dynamically refine triangular meshes. The creation of the application's secondary heterogeneous mesh on each processor in the driver routine is identical to the initial mesh creation code used in the original serial version.

After ghost layers are created, the element and nodal data associated with the ghost elements and nodes must be synchronized each step. The ParFUM framework provides a simple mechanism for performing this synchronization. The application synchronizes only nodal data, such as displacements, velocities, and accelerations. The synchronization copies the data from the nodes where the data is computed to any corresponding ghost copies on adjacent partitions. We do this by treating the nodal data as element data because of the unusual topology of the CFE mesh as shown in Figure 5. There are multiple copies of a node for each original node, each with slightly different nodal data. Each of these copies of a node is

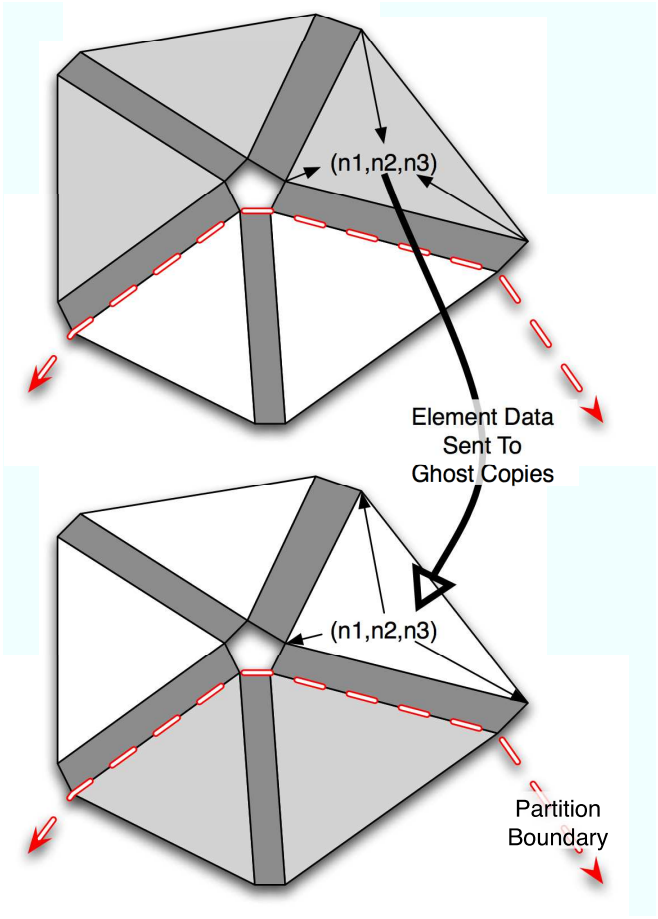

Figure 6: Nodal data is copied to the elements, The elements sync with ghost copies, and the updated data is copied from ghost elements to the nodes.

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. In total, only a small number of lines of code is required for the whole ghost value synchronization process because the ParFUM framework handles the significant work involved in building all required send and receive lists as well as the data packing and sending.

The ParFUM framework disallows global variables in any applications built upon it. Global variables in FORTRAN programs are those declared with `COMMON` or `SAVE`.Therefore the parallel implementaiton of the CFE scheme wraps all `COMMON` or `SAVE` variables in a module. Using a module to wrap these variables is sufficient to meet this requirement of ParFUM.

3.2 Floating-point stability

Dynamic fracture problems are unstable physically. This inherent instability translates into a numerical instability as the limitations of floating-point arithmetic can cause the solution to be affected by the number of partitions into which the mesh is partitioned. This numerical instability is caused by differing orders in which floating point operations are applied when summing nodal force vectors. Since the parallel code adds values from local elements to a node before adding on the ghost element values, the order of these additions for a single logical shared node differs between the different processors sharing that node. This difference in the order of the additions affects the solution because addition is neither associative nor commutative in floating point arithmetic, i.e., $a + b + c \neq c + a + b$, especially if the values have widely differing exponents in their binary representation. Due to the inherently unstable nature of dynamic fracture simulations, these small errors tend to accumulate over time and can affect the numerical results, including the predicted crack path.

The classic solution to numerical problems caused by non-commutativity or non-associativity of floating-point additions 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. Additionally, in FE codes, the common practice is to simply iterate over all elements adjacent to a given node, accumulating the sum. The code chooses this standard practice. Unfortunately, this method of iterating over elements without regard to the values being summed is inherently flawed when maximal accuracy is required. We have not yet implemented the more complicated method for summing the nodal force values using sorted lists. It should be noted that this problem does not just occur in parallel. The order in which the forces are added at each node produces slightly different results for the activation time (and ultimately the failure time) of extrinsic cohesive ele-

ments. Since the specific order in which the elements are added matters, even in the serial case, attention should be paid to correctly add the forces in a sorted order, whether in parallel or serial.

3.3 Load Balancing

Maintaining a uniform load balance across processors is critical to obtaining good performance and scalability to large numbers of processors. In the parallel implementation, a good load balance is achieved without any explicit load balancing. Section 5 shows that the parallel efficiency of the implementation exceeded 99% in many cases without applying any dynamic load balancing. Since the parallel efficiency is high, the load is necessarily well distributed among almost all processors. For this reason the support for load balancing provided by ParFUM is not used. However, other dynamic fracture applications may require load balancing to achieve optimal performance [**?**].

3.4 Implementation Summary

The main parallelization steps are as follows:

- Restructure the serial code into two main subroutines `driver` and `init`
- Load an unpartitioned mesh in `init`
- Add ghost layer synchronization calls in `driver`
- Modify the output subroutine to create filenames based on processor id
- Eliminate global variables (`COMMON` or `SAVE`) by wrapping them in a module

4 Verification and validation study

4.1 Convergence study

As indicated in the introduction, the spatial and temporal convergence of the CFE scheme needs to be addressed. To that effect, we investigate in this section two fracture problems involving the dynamic mode I failure of a thick pre-notched PMMA specimen of length $L = 0.05m$ and width $W = 0.03m$, with an initial crack length $a_0 = 0.005m$. The bulk material properties are defined by the Young's modulus $E = 3.45GPa$, Poisson's

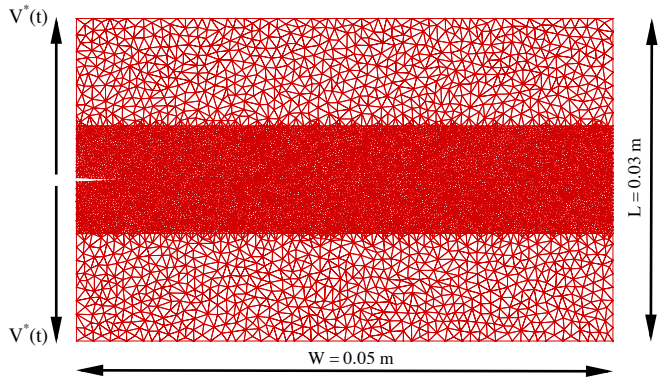

Figure 7: Mesh and boundary conditions used for crack propagation along a pre-defined straight line problem. Only the elements along this line are allowed to be activated in this simulation. A larger view of the notch tip is in figure 8.

ratio $\nu = 0.35$ and material density $\rho_0 = 1190kg/m^3$. The cohesive failure properties adopted in this work are defined by the tensile and shear strength values $\sigma_{max} = \tau_{max} = 20MPa$ and by the mode I and mode II fracture toughness values $G_{Ic} = G_{IIc} = 352J/m^2$, which corresponds to a mode mixity parameter $\beta = 1$.

In both fracture problems, the loading is symmetric with respect to the initial crack plane, so the crack is expected to travel straight, unless it reaches a sufficiently high speed at which crack branching might occur. Two discretizations are considered here. In the first one (Figure 7), the crack is confined to propagate along its original plane, which acts like a straight material interface. In the second one, the crack path is not pre-defined and a random mesh is used in the region ahead of the notch tip (Figure 12). As alluded to in the introduction, convergence issues are expected to arise only in the second case. However, the first problem is studied for completeness and to compare the intrinsic and extrinsic CFE schemes. In both cases, the analysis is performed in plane strain, and the applied loading consists of an applied vertical velocity $V^*(t)$ that increases linearly from 0 to peak value V_0 for $0 < t \leq t_{ramp}$ and remains constant afterwards.

The introduction of a cohesive model introduces a new length scale in the problem, the length of the cohesive failure zone, the small region in the vicinity of the advancing crack front where the cohesive failure process

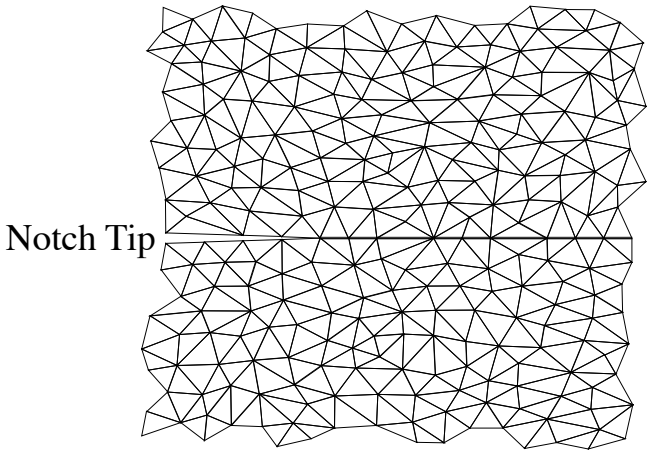

Figure 8: Zoomed view of notch tip of mesh in figure 7. The mesh contains a pre-defined horizontal straight line path that the crack can follow.

takes place. A static estimate of the mode I cohesive zone size is

$$R = \frac{\pi}{8} \frac{E}{1-\nu^2} \frac{G_{Ic}}{\sigma_{max}^2}. \tag{12}$$

Although R is expected to decrease as the crack speed increases, (12) is used to compare various levels of mesh refinement. As illustrated in Figures 7 and 12, since the crack motion is expected to remain in the vicinity of the mid-plane, only a limited region around the mid-plane is meshed with a fine mesh, while the remainder of the domain is meshed more coarsely to reduce the computational cost. At all times, the time step size Δt is kept below the CFL stability conditions.

In the first problem, for which a straight crack path is prescribed (Figure 7), the imposed vertical velocity is applied along the left edge of the domain, while the remaining boundary is left traction-free. The peak value of velocity is $V_0 = 2.5m/s$ and the ramp time is $t_{ramp} = 0.058L/C_d$. Figure 9 present the time evolution of the crack length obtained with various meshes, with the time step size kept at a constant value of the Courant number $\chi = 0.05$. The mesh density is characterized by the average number of elements in the static cohesive zone size R. As apparent there, the CFE scheme is spatially convergent and about five cohesive elements are needed in the active cohesive zone.

Figure 10 addresses the issue of temporal convergence of the CFE simulation, showing the time evolution of the straight crack length for four

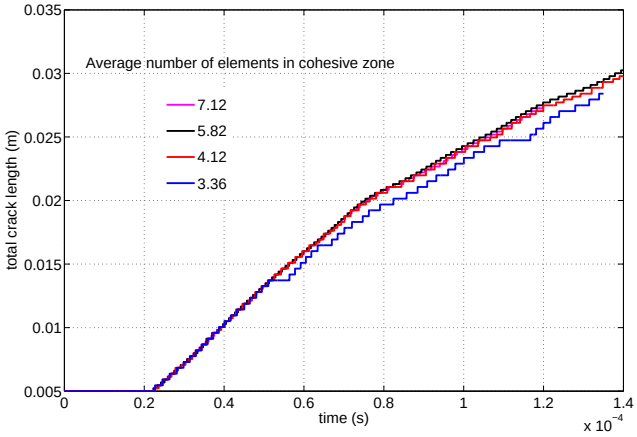

Figure 9: Spatial convergence of the crack motion for the case of a prescribed straight crack path, for four mesh densities defined as the ratio of the static cohesive zone size R given by equation (12) and the average cohesive element size.

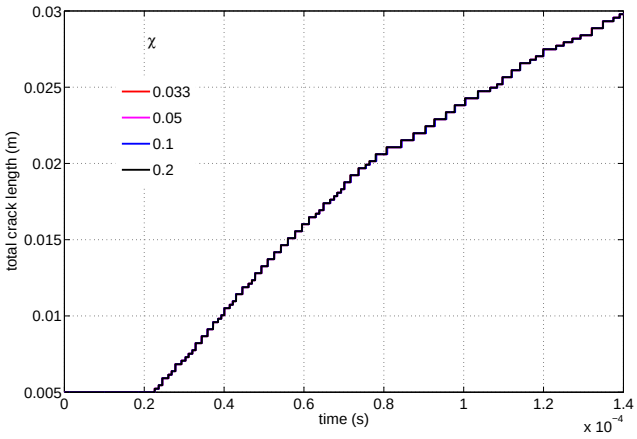

Figure 10: Temporal convergence of evolution of the total crack length, for the straight line crack propagation problem for 4 different Courant numbers χ. The mesh density is fixed at about 4 cohesive elements in the cohesive zone.

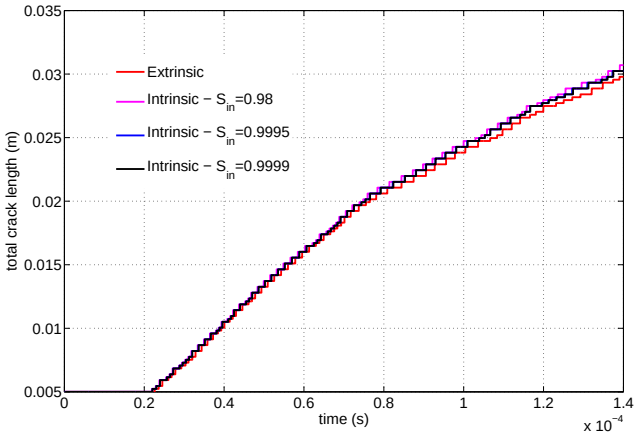

Figure 11: Extrinsic *vs*. intrinsic CFE predictions of the straight crack propagation history for a constant mesh density of about 4 elements in the cohesive zone and a constant time step size $\chi = 0.05$.

values of the time step size (described through the corresponding value of the Courant number χ defined in (10). It should be noted that the use of extrinsic cohesive modeling allows us to use time steps much larger than those commonly adopted in the intrinsic case, at least for problems involving prescribed crack paths. In intrinsic CFE simulations, values of χ are typically an order of magnitude smaller.

The evolution of the crack length is also used to compare the intrinsic and extrinsic FE solutions. In the intrinsic case, a bilinear cohesive failure law is adopted [**?**] that uses the same values of failure strength σ_{max} and fracture toughness G_{Ic}. The initial cohesive stiffness K_{c0}, i.e., the slope of the rising part of the cohesive traction-separation curve, is defined by the parameter S_{init} as in

$$K_{c0} = \frac{\sigma_{max}}{(1 - S_{init})\Delta_{nc}}. \tag{13}$$

The closer S_{init} is to unity, the stiffer the cohesive element is, with the case $S_{init} = 1$ corresponding to the limiting case of an initially rigid cohesive element. A direct comparison between extrinsic and intrinsic CFE predictions of the evolution of the crack length is presented in Figure 11. In the intrinsic case, three values of the non-dimensional parameter S_{init} are used. As expected, as the initial stiffness of the intrinsic cohesive elements tends to infinity, the extrinsic CFE solution is recovered, verifying the extrinsic implementation of the cohesive finite element solver.

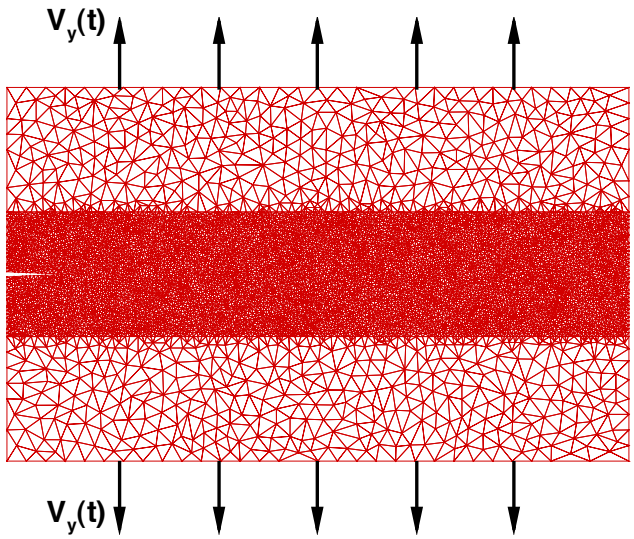

Figure 12: Mesh and boundary conditions used for the crack propagation problem with arbitrary crack path, for which a random discretization is used in the central refined zone. The notch tip is enlarged in figure 13

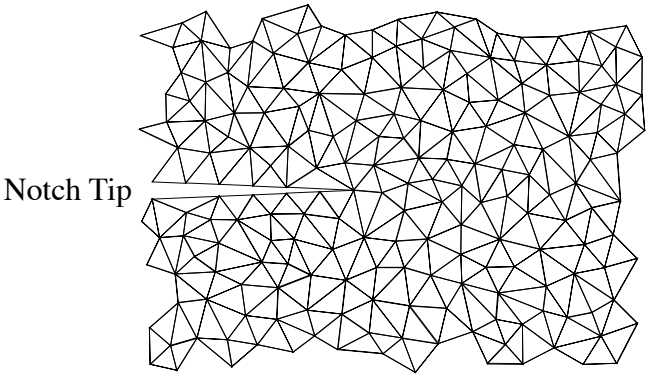

Figure 13: Zoomed view of notch tip of mesh in figure 12. The mesh does not contain a horizontal straight path for a crack to follow.

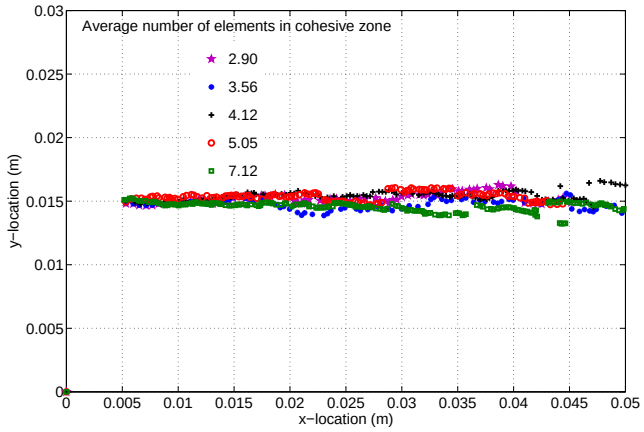

Figure 14: Spatial convergence of the crack path for crack propagation problem shown in Figure 12, obtained for 5 different mesh densities. In all cases, the time step size is determined by $\chi = 0.05$.

We now turn our attention to the second fracture problem shown in Figure 12, for which the crack path is not prescribed and the transient vertical velocity is applied along the top and bottom edges, with a maximum amplitude of $V_0 = 0.5m/s$ and a ramp duration $t_{ramp} = 0.058L/C_d$. This loading results in two stress waves ($\sigma_0 = \rho_o C_d V_0$) traveling at the dilatational wave speed C_d, downward and upward from the top and bottom edges, respectively. Upon reaching the crack tip, these waves create a highly transient stress concentration, which leads to the initiation and subsequent propagation of the crack.

To assess the spatial convergence for this case, the crack path ($x-y$ plot of the failed cohesive elements) and the evolution of the total crack length are computed for various central zone mesh densities. The time step size Δt is chosen such that $\chi = 0.05$ for each mesh. Figure 14 shows the crack path for five different mesh densities. As apparent there, although the overall predictions of the crack path are relatively similar for these four meshes, no clear convergence is achieved. This is especially true for the later part of the crack motion, for which larger deviations are observed as the crack approaches the right edge of the domain. This might be due to the very high crack speed achieved in that region, which decreases the active cohesive zone size and tends to lead to an increased level of physical instability, i.e., of crack branching. This onset of branching is apparent for the finest resolution (square symbols). This lack of apparent convergence in the crack

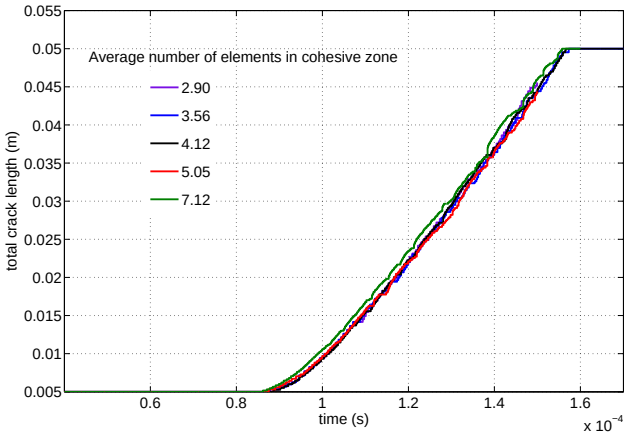

Figure 15: Spatial convergence of the evolution of the crack length for the five simulations shown in Figure 14.

path prediction obtained with unstructured meshes has been alluded to elsewhere [**?**]. This apparent lack of spatial convergence translates, but to a smaller degree, to the evolution of the total crack length (i.e., the sum of the length of all failed cohesive elements), as shown in Figure 15. This type of simulation provides an indication of the scatter of the numerical predictions of the macroscopic crack path.

The issue of temporal convergence is addressed in Figures 16 and 17, which respectively present the crack path and crack length evolution results obtained for a fixed mesh (with an average of 4.12 cohesive elements in the cohesive zone) and for four values of the time step size. Although the temporal convergence for the arbitrary crack path is less conclusive than for the prescribed path case, the results obtained for the four values of χ are very similar especially during the initial stage of the crack motion. After a while, however, the numerical results start to deviate from each other, first slightly, then in a more pronounced fashion. Note once again, however, that a stable numerical solution is obtained with the extrinsic CFE scheme for values of χ larger than 0.2, while a value as small as $\chi = 0.033$ has to be used in the intrinsic case to achieved stability [**?**].

As the final step of this convergence study, we compare the results for the serial and parallel implementations of the extrinsic CFE scheme. The parallel results are obtained on a four-processor platform. The mesh and time step sizes are kept constant, an average of 4.12 elements in the cohesive zone and a Courant number $\chi = 0.05$. It can be inferred from the Figure

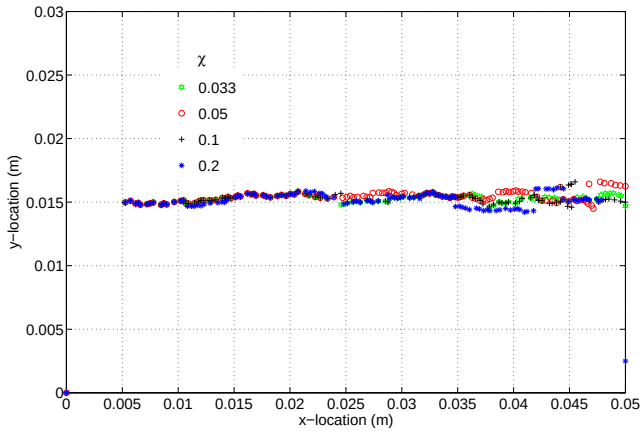

Figure 16: Temporal convergence of the extrinsic CFE simulations for arbitrary crack path, for a fixed mesh density of about 4 elements in the cohesive zone size.

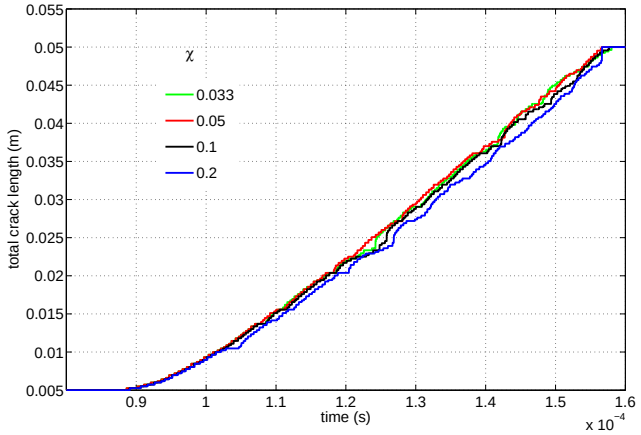

Figure 17: Temporal convergence of the crack length *vs*. time curves for the simulations shown in Figure 16.

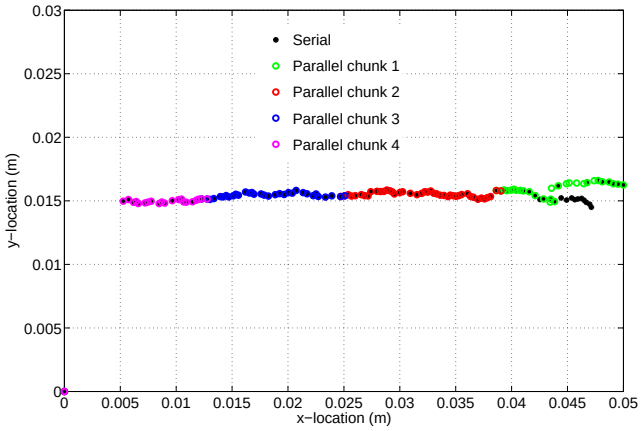

Figure 18: Serial *vs.* parallel simulations of the extrinsic CFE dynamic fracture modeling showing the crack propagating across the four partitions and the slight deviations obtained towards the end of the simulation.

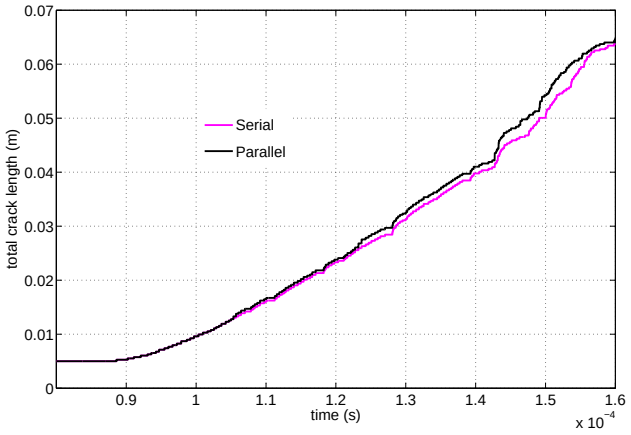

Figure 19: Crack length *vs.* time curves obtained for the serial and parallel runs shown in Figure 18.

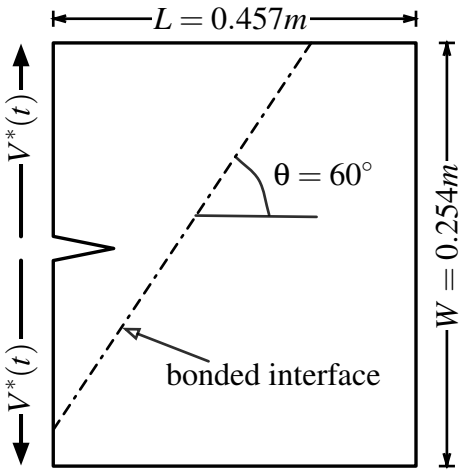

Figure 20: Schematic of the inclined interface fracture problem (not to scale). The initial crack length is $a_0 = 29.58mm$ and the inclined interface is located $46.82mm$ ahead of the initial crack tip.

18 that the crack paths derived from the serial and parallel simulations do not match exactly, especially during the later stage of the dynamic failure process. This difference between serial and parallel solutions is also apparent in Figure 19, which presents the evolution of the total crack path. This difference is related to the numerical inaccuracies induced by finite arithmetic, as alluded to in Section 3.

4.2 Validation study

The apparent lack of convergence and 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 20. This problem, investigated experimentally by [**?**], 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 $\rho_0 = 1230kg/m^3$ [**?**]. 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 (weak) [**?**]	Weldon-10 (strong) [**?**]
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 20. 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 cohesive elements. A detail of the mesh in the vicinity of the initial crack tip is shown in Figure 21. 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.

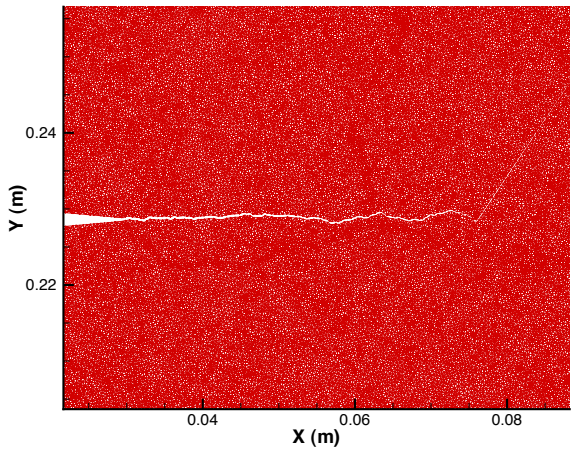

Figure 21: Details of the deformed mesh in the vicinity of the initial crack tip and the inclined interface for the domain in Figure 20.

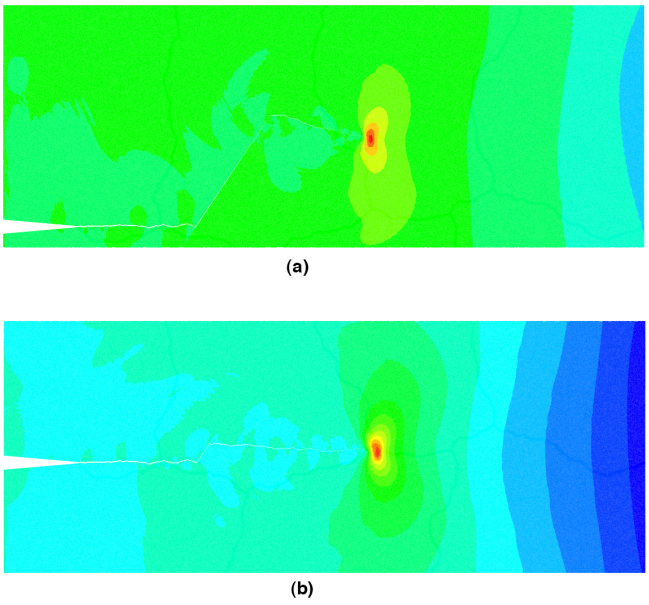

Figure 22: σ_{22} stress contour plot at time $t = 1.857L/C_d$ for (a) weak interface strength, showing the trajectory of the crack trapped momentarily along the inclined interface and (b) for the strong interface case.

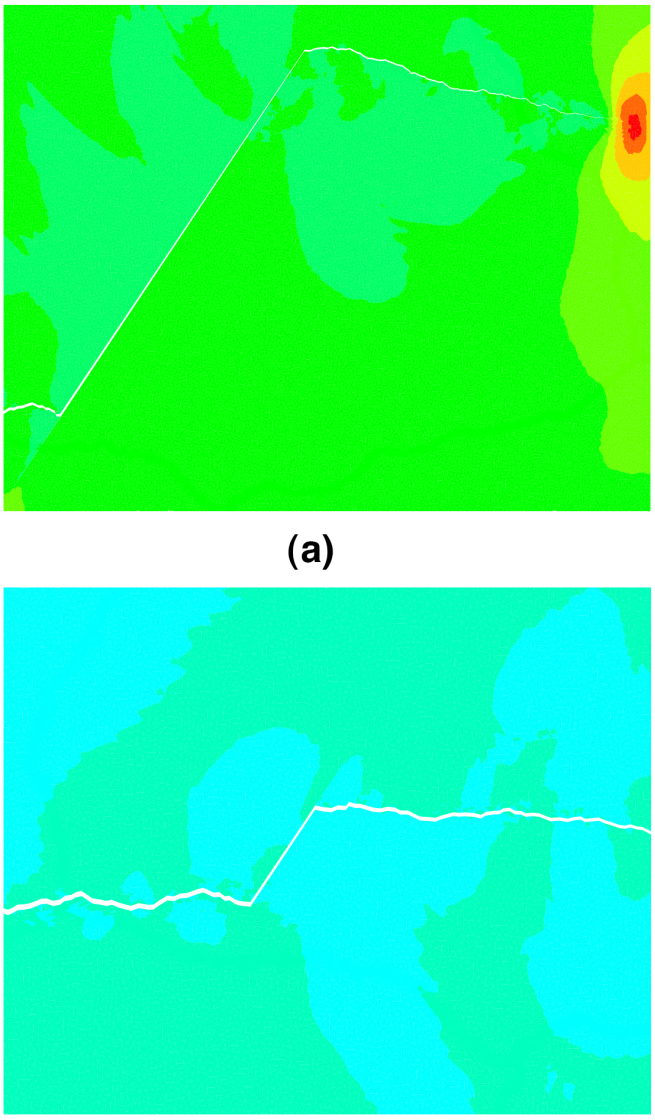

Figure 23: Close-up view of the two cases from Figure 22, showing details of crack path near the vicinity of the inclined interface. (a) is the weak interface case while (b) is the strong interface case.

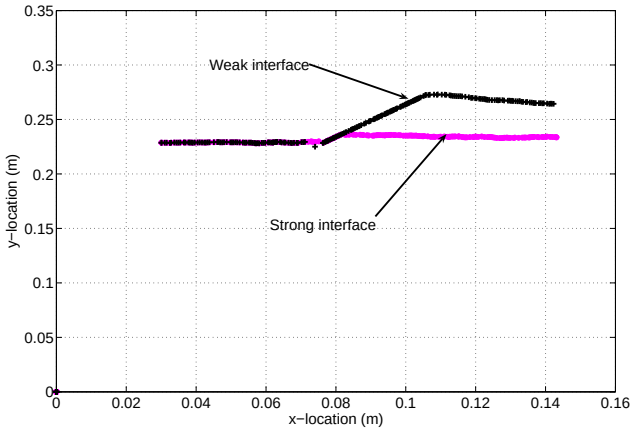

Figure 24: Comparison between the crack trajectory obtained for the strong and weak interfaces.

Figure 22 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 23 shows the close-up view of Figure 22 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 22 is further visualized in Figure 24, 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 [**?**], 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 25, 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.

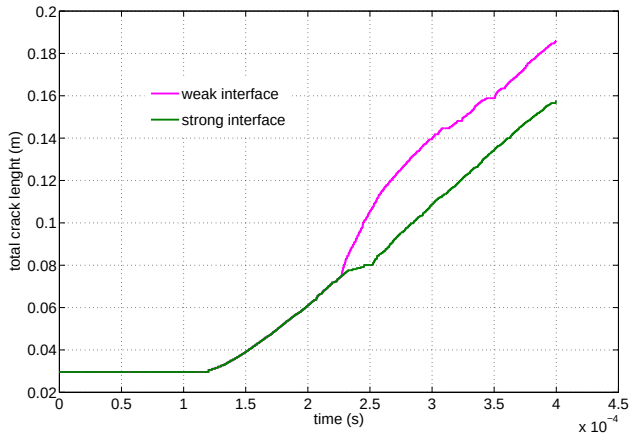

Figure 25: Total crack length *vs*. time for the strong and weak interface cases.

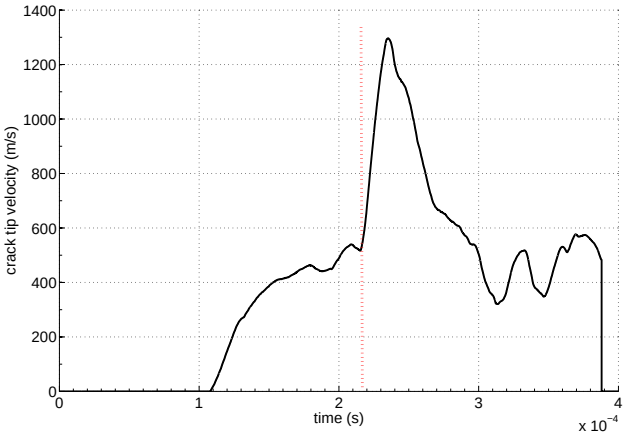

Figure 26: Evolution of crack tip velocity for the weak interface. The dashed vertical line shows the instant the crack hits the interface.

This transient crack motion is also illustrated in Figure 26, 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 [**?**]. 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 Parallel performance analysis

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*. All parallel runs described in this section were performed on the Turing cluster at the University of Illinois Urbana-Champaign. Each node in the cluster is an Apple Xserve with dual 2.0 GHz G5 processors. The nodes are connected via a Myrinet network. The performance results 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 implementation uses no hand-optimized code tailored to any particular platform or machine; It only uses some standard compiler optimization flags.

The performance analysis presented in this section was performed on the same cohesive finite element problem described in Section 4.2 with a mesh containing 1.2 million elements. 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. Increasing the number of mesh partitions per processor can improve performance by overlapping communication and computation and by improving cache performance. The cache effects in ParFUM applications occur because smaller partitions contain fewer elements and thus may fit inside a smaller faster level of cache[**?**]. The term virtualization comes from AMPI

[**?**] where multiple MPI processes, or virtual processors, are run inside a single processor. ParFUM is built partially upon AMPI, and thus it inherits this terminology. When multiple mesh partitions reside on a single processor, computation for one partition can overlap the latency of communication for a different partition.

Using multiple mesh partitions per processor benefits performance for our implementation. On 8 physical processors, we found the execution time of 1000 timestep loop iterations to be 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 27 shows that 256 to 1024 partitions gives the best runtimes when using 8 to 32 processors. This range or *sweet spot* was determined experimentally, but approximate rules of thumb can also be used for a particular application. To maximize the performance of this implementation on each partition should contain between one thousand and four thousand elements.

Scalability to a large number of processors is crucial for large FE codes. 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 divided by the sequential runtime. When the *speedup* is close to the number of processors, an application is scaling well. The CFE application scales well to a large number of processors. Figure 28 displays the speedup of the application for up to 512 processors while using 512 partitions in all cases. For the baseline serial version, we run the same application, but just run it with one partition on one processor, thus there is no communication overhead. Table 2 shows the same speedup data along with execution times and parallel efficiency. The parallel efficiency is a ratio of the speedup to the number of processors used. A parallel efficiency of 1.0 on 128 processors means that the parallel version was 128 times faster than the serial version. Parallel efficiencies of greater than 1.0 are rarely seen, but can occur for a number of reasons including cache effects, suboptimal serial performance, or poor algorithm choices. The application scales almost perfectly to 256 processors. 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 for overlapping communication and computation from different partitions on a single processor.

6 Conclusion

Initially rigid (extrinsic) cohesive elements are better suited for simulation of dynamic fracture events when the crack path is not pre-defined in

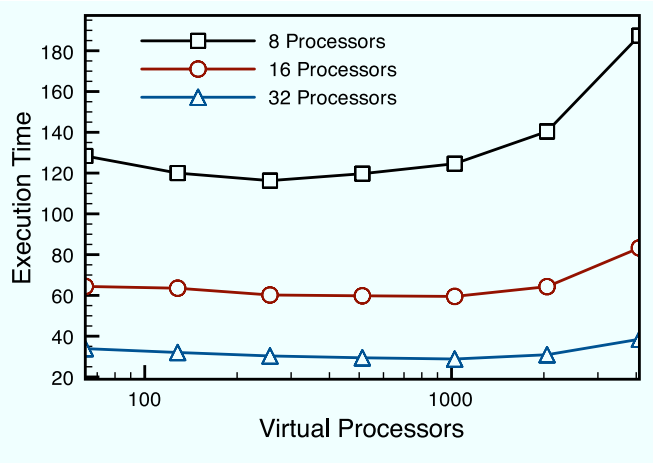

Figure 27: Execution time (in seconds) with varying numbers of partitions (Virtual Processors).

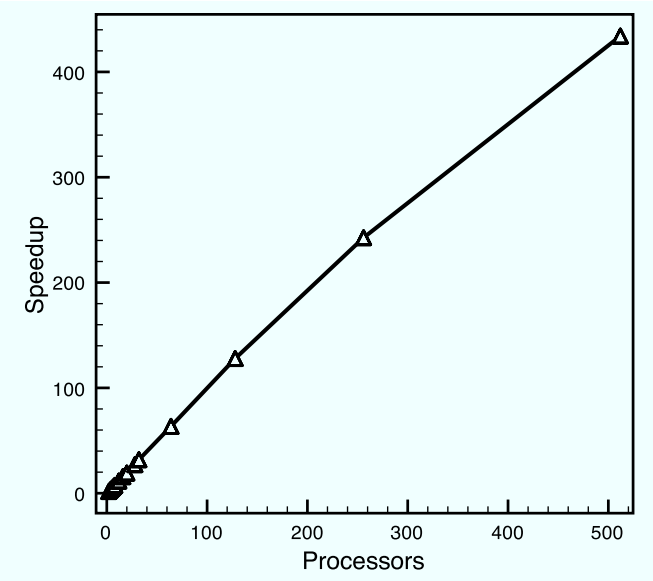

Figure 28: Speedup results for up to 512 processors for a fixed number of 512 partitions (Virtual Processors). The same input mesh is used in all cases.

Table 2: Speedup and parallel efficiency with a 1.2M element mesh, 512 Virtual Processors.

Processors	Execution time (s)	Speedup	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

comparison to their intrinsic counterparts. A novel methodology for parallel implementation of extrinsic cohesive elements based on activation of elements, and implemented with the aid of the Parallel framework for Unstructured Meshes (ParFUM). The implementation was tested for spatial and temporal convergence, and though the crack behavior was captured well, spatial convergence was not clearly observed. The developed parallel CFE scheme was validated against the experimental studies performed on the dynamic crack deflection-penetration behavior in inhomogeneous specimens by [**?**]. Simulated results are in very good agreement with experimental observations. A detailed scalability study performed on up to 512 processors shows excellent speedup for the parallel cohesive finite element solver.

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, and of the U.S. Department of Energy HPCS Fellowship Program.

References

[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.