Finite Elements in Analysis and Design 100 (2015) 41–46

Contents lists available at ScienceDirect

Finite Elements in Analysis and Design

journal homepage: www.elsevier.com/locate/finel

Implementation of a parallel finite-element library: Test case on a non-local continuum damage model

N. Richart *, J.F. Molinari

Civil Engineering Institute, Materials Science and Engineering Institute, École Polytechnique Fédérale de Lausanne (EPFL), Station 18, CH-1015 Lausanne, Switzerland

ARTICLE INFO

Article history:
Received 21 July 2014
Received in revised form
6 February 2015
Accepted 6 February 2015
Available online 13 March 2015

Keywords:
Finite element method
Parallel computing
Continuum damage
Non-local approach

ABSTRACT

This paper presents an efficient method to implement a damage law within an explicit time-integration scheme, in an open-source object-oriented finite-element framework. The hybrid object/vector design of the framework and implementation choices are detailed in the special case of non-local continuum damage constitutive laws. The computationally demanding aspect of such constitutive laws requires efficient algorithms, capable of using High Performance Computing (HPC) clusters. The performance of our approach is demonstrated on a numerically and physically challenging 3D dynamic brittle-fragmentation test case. An almost perfect scalability is achieved on parallel computations. The global dynamics and energy terms are in good agreement with classical cohesive models' predictions.

1. Introduction

Understanding the different mechanisms involved in the fracture of brittle materials under dynamic loading, such as crack branching, or the transition between different types of crack propagation (trans/intra-granular) is key in the design of new materials. A way to study these mechanisms is by the use of numerical methods. Many numerical approaches can be found in the literature, such as peridynamics [1,2], eigenerosion [3], or finite-element methods with additions to represent a discontinuity like cohesive models [4], the eXtended Finite-Element Method (X-FEM) [5], or continuum damage with localization limiters such as delayed damage [6], second gradient [7] or non-local integral [8].

In the context of this paper, the continuum damage with non-local integral type limiters will be used due to its ease of implementation and the fact that it is well-established for at least static-fracture mechanics problems [8,9]. This method however is computationally demanding, since in the constitutive law a neighborhood of each material point has to be taken into account. Therefore, the use of parallelism in the implementation of such a method can decrease the execution times considerably. To the author's knowledge there are only pure C++ object-oriented or pure Fortran implementations for the explicit non-local method implementation as shown in [10] or [11]. Contrasting such single programming paradigm implementations, this paper will present a new code developed by the authors and named Akantu [12], that aims to combine the advantages of both views: the genericity and extensibility of the object-oriented paradigm of C++, and the vectorial efficiency present in Fortran. In this code we included parallelism capabilities with the idea of combining performance with ease of implementation of complex algorithms.

This paper is organized in three major sections. First it presents some reminders on the finite-element method, particularly in the context of non-local continuum damage. Second, it explains the implementation details in the case of library Akantu. Third, it presents a 3D fragmentation test-case that validates the method.

2. Finite-element formulation

2.1. Work-flow

In this paper we will assume the case of continuum solid mechanics with an explicit time-integration scheme. If we consider a central difference scheme, the finite-element formulation can be reduced to the following equations that must be solved iteratively:

$$\boldsymbol{M}\ddot{\boldsymbol{u}}_{n+1} = \boldsymbol{f}_{\mathbf{ext}_{n+1}} - \boldsymbol{f}_{\mathbf{int}_{n+1}} \tag{1}$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t\dot{\boldsymbol{u}}_n + \tfrac{1}{2}\Delta t^2\ddot{\boldsymbol{u}}_n \tag{2}$$

$$\dot{\boldsymbol{u}}_{n+1} = \dot{\boldsymbol{u}}_n + \tfrac{1}{2}\Delta t\ddot{\boldsymbol{u}}_n + \tfrac{1}{2}\Delta t\ddot{\boldsymbol{u}}_{n+1} \tag{3}$$

* Corresponding author.
E-mail address: nicolas.richart@epfl.ch (N. Richart).

http://dx.doi.org/10.1016/j.finel.2015.02.003
0168-874X/

In these equations, $\boldsymbol{u}_n$, $\dot{\boldsymbol{u}}_n$ and $\ddot{\boldsymbol{u}}_n$ represent the approximations of the displacement, velocity and acceleration at a time t_n, $\boldsymbol{M}$ is the mass matrix, and $\boldsymbol{f}_{\mathbf{ext}_n}$, $\boldsymbol{f}_{\mathbf{int}_n}$ are the external and internal forces respectively. Finally, Δt is the time step defined such that $t_{n+1} = t_n + \Delta t$. In order to have a stable explicit time-integration, Δt is submitted to the Courant–Friedrichs–Lewy condition [13].

Eqs. (2) and (3) are usually rewritten in a predictor/corrector way, which leads to a possible separation of a time iteration in four stages: prediction of the kinematic variables, computation of the new internal forces, resolution of the acceleration and correction of the kinematic variables.

In this formulation, the material's behavior is taken into account in the computation of the internal forces.

2.2. Constitutive model

The forces are computed from the stress at each material point. Therefore, for all such points, the constitutive law has to be defined.

In the simplified case of a linear elastic and brittle material, isotropic damage can be represented by a scalar variable d, which varies from 0 to 1 for no damage to fully broken material respectively. The stress–strain relationship then becomes

$$\boldsymbol{\sigma} = (1-d)\boldsymbol{C} : \boldsymbol{\varepsilon}$$

where $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ are the Cauchy stress and strain tensors, and $\boldsymbol{C}$ is the elastic stiffness tensor. This formulation has been proven to lead to a localization of the strain-softening region [14].

As previously mentioned, there exists many localization limiters. In this paper, we will only consider the integral non-local approach. This approach consists in replacing a variable v in the constitutive law with its average v^{nl} on the direct neighborhood (4). In continuum damage models this is usually done on a scalar variable that is used as a criterion for damage evolution.

$$v^{\mathrm{nl}}(x) = \int_N \alpha(x,\omega) v(\omega)\, \mathrm{d}\omega \tag{4}$$

where N is the neighborhood of interest and $\alpha(x,\omega)$ is the weight function defined as

$$\alpha(x,\omega) = \frac{\alpha_o(\|x-\omega\|)}{\int_N \alpha_o(\|x-\eta\|)\, \mathrm{d}\eta} \tag{5}$$

The non-normalized actual weight function α_o can have different expressions [15]. One that is commonly used is the bell-shaped function (6).

$$\alpha_o(r) = \begin{cases} \left(1-\dfrac{r^2}{R^2}\right)^2 & \text{for } r \le R \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

In this weight function, R defines the radius of the sphere N. In some cases it can be related to a material parameter, but in most cases R is an internal parameter that has to be chosen carefully to get meaningful results.

To completely define the damage model, we still have to define an evolution law for the damage d. Many evolution laws can be found in the literature. For the purposes of this paper, we will use a simple isotropic damage evolution based on an energy criterion [16,17].

$$Y = \tfrac{1}{2}\boldsymbol{\varepsilon} : \boldsymbol{C} : \boldsymbol{\varepsilon} \tag{7}$$

$$F = Y - Y_d - Sd \tag{8}$$

$$d = \begin{cases} \min\left(\dfrac{Y-Y_d}{S},\ 1\right) & \text{if } F > 0 \\ \text{unchanged} & \text{otherwise} \end{cases} \tag{9}$$

In this formulation Y is the strain energy release rate, Y_d is the rupture criterion and S is the damage energy. The non-local version of this damage evolution law is constructed by averaging the energy Y.

3. Effective implementation

3.1. The philosophy behind Akantu

We implemented an efficient general purpose finite-element library called Akantu [12]. As a demonstration of the possibilities of this library, we implemented in it the model presented in the previous section. This open-source object-oriented library distinguishes itself from other finite-element codes by its hybrid object/vector architecture. It uses the object-oriented view for "high-level" algorithms. This abstraction endows the code with properties such as re-usability, genericity, and ease-of-extension. But for "low-level" processes, such as loops on elements, nodes or material points, the mechanisms of object-oriented programming such are virtual calls can be slow. Consequently, in Akantu the choice was made to code these critical loops in a vectorial way, i.e. similar to a C or Fortran manner. This hybrid implementation style uses the advantages of both the object-oriented and the procedural programming paradigms. To understand better this hybrid

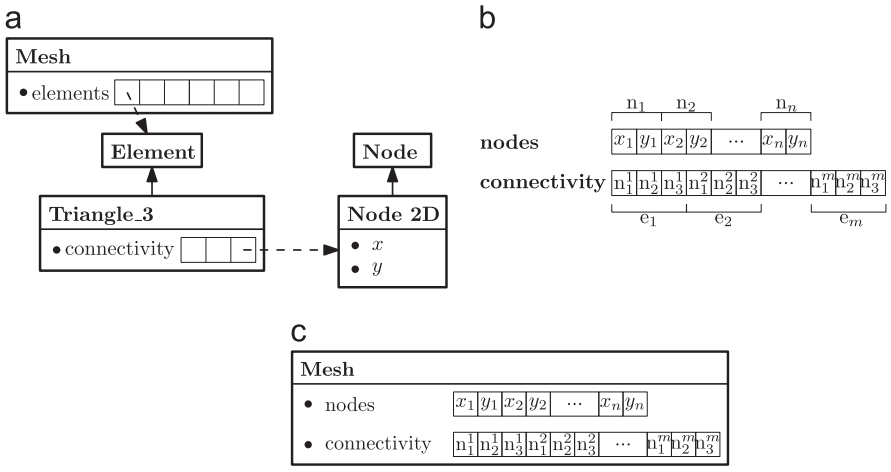

Fig. 1. Mesh data structures for different programming paradigms. (a) Object-oriented, (b) vectorial and (c) object/vector.