Université
de Technologie
Tarbes
Occitanie Pyrénées

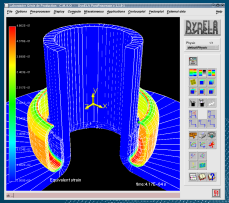
Reproduction des résultats de parallélisation du code DynELA

Une étude de cas sur la pérennité des résultats de recherche

Olivier Pantalé

olivier.pantale@uttop.fr

LGP, UTTOP, Université de Toulouse, 47 Avenue d'Azereix, F-65000, Tarbes, France



– 12ème Biennale Française des Mathématiques Appliquées et Industrielles – 02-06 juin 2025 –

Plan de la présentation

- Introduction et généralités
- Récupération du code source et compilation
- Comparaison des résultats des deux exécutions
- Conclusions

1. Introduction et généralités
2. Récupération du code source et compilation
3. Comparaison des résultats des deux exécutions 2005/2020
4. Conclusions

Le "Ten Years Reproducibility Challenge" en 2020

Challenge proposé par des chercheurs INRIA

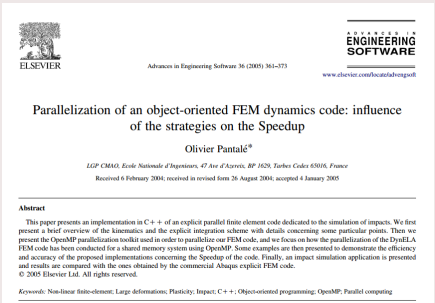
Extrait de l'Appel à contribution:
Avez-vous déjà essayé d'exécuter **un vieux code que vous avez écrit pour un article scientifique** que vous avez publié il y a des années? Avez-vous rencontré des problèmes? Avez-vous réussi? Nous sommes curieux de connaître votre histoire. C'est la raison pour laquelle nous publions un **numéro spécial de ReScience** pour recueillir ces histoires.

- Ré-exécuter un code utilisé pour une publication datant de plus de 10 ans.
- Tous types: analyse statistique, simulation numérique, traitement de données, etc.
- Le langage de programmation et le domaine scientifique sont libres.
- Condition principale: avoir publié un article avant 2010, dans une revue ou une conférence, contenant des résultats produits par code (même si le code n'a pas été publié à l'époque).
- Ne pas réécrire une nouvelle version du code.
- Faire fonctionner l'ancien code sur un système moderne, avec un minimum de modifications, pour vérifier si les mêmes résultats peuvent être reproduits.

Le "Ten Years Reproducibility Challenge" en 2020

Défi relevé avec le code de calcul Eléments Finis dynamique Explicite – DynELA du LGP/ENIT

Reproduction des résultats d'une publication de 2005[a].



ELSEVIER

Advances in Engineering Software 36 (2005) 361–373

ADVANCES IN ENGINEERING SOFTWARE

www.elsevier.com/locate/advengsoft

Parallelization of an object-oriented FEM dynamics code: influence of the strategies on the Speedup

Olivier Pantalé*

LGP CMAO, Ecole Nationale d'Ingenieurs, 47 Ave d'Azereix, BP 1629, Tarbes Cedex 65016, France

Received 6 February 2004; received in revised form 26 August 2004; accepted 4 January 2005

Abstract

This paper presents an implementation in C++ of an explicit parallel finite element code dedicated to the simulation of impacts. We first present a brief overview of the kinematics and the explicit integration scheme with details concerning some particular points. Then we present the OpenMP parallelization toolkit used in order to parallelize our FEM code, and we focus on how the parallelization of the DynELA FEM code has been conducted for a shared memory system using OpenMP. Some examples are then presented to demonstrate the efficiency and accuracy of the proposed implementations concerning the Speedup of the code. Finally, an impact simulation application is presented and results are compared with the ones obtained by the commercial Abaqus explicit FEM code.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Non-linear finite-element; Large deformations; Plasticity; Impact; C++; Object-oriented programming; OpenMP; Parallel computing

Code de calcul EF DynELA v.1.0:

- Formulation explicite en grandes déformations
- Code source en C++ (langage standard)
- Interface graphique sous QT (maintenu KDE)
- Langage de commandes structuré propriétaire (sources)
- Utilisation de bibliothèques standards (LAPACK, LIBM)
- Code source sauvegardé sur CD-ROM (ouf !)
- Documentation conservée (HdR, publications, ...)
- Savoir-faire préservé dans un cerveau opérationnel (plus ou moins !)

Quelques informations complémentaires sur le code :

- Interpréteur : 10 000 lignes de C++, Flex et Bison
- Graphiques : 20 000 lignes de C++
- Solveur EF : 80 000 lignes de C++

[a] O. Pantalé, *Parallelization of an object-oriented FEM dynamics code: influence of the strategies on the Speedup*, Advances in Engineering Software, p. 361-373, vol 36, n°6, 2005

Environnement matériel et logiciel pour la reproduction des résultats de la publication originale

– 2005 –

- Compaq Proliant 8000 SMP.
- 8×32-bits Intel Xeon PIII 550/2Mb and 5GB of RAM.
- Linux Redhat 8.0 – 32 bits.
- Intel Cpp 7.1.

– 2020 –

- Dell R730.
- 2×64-bits Intel Xeon E5-2650 (12C) and 96GB of RAM.
- Ubuntu Bionic 18.04.4 LTS – 64 bits.
- g++ 7.4.0.

1. Introduction et généralités
2. Récupération du code source et compilation
3. Comparaison des résultats des deux exécutions 2005/2020
4. Conclusions

Perte partielle du code source de DynELA

En 2005, deux versions du code de calcul co-existaient sur deux machines différentes.

- Une version non parallèle déployée sur un ordinateur portable avec suivi des modifications avec Concurrent Version System (CVS).
- Une version parallèle déployée sur le Proliant (fork de la version non parallèle) spécifique pour les besoins de la publication.

Etat des lieux des **manquements à la reproductibilité des résultats**.

- Le code source de la version parallèle n'a pas été correctement archivé avec les sources de la publication (perte de lien, mémoire,...).
- Les codes sources étaient sauvegardés sur plusieurs supports (CDROM, Zip, HDD,...), sans réelle cohérence.
- Les codes sources étaient mal "étiquetés" (pas de date fiable), nuisant à la chronologie.
- Le serveur Proliant a été débranché et mis "**à la casse**" sans sauvegarder les tous derniers développements du code.
- Nécessité de **reconstruire** en 2020 une version aussi proche que possible de la version finale 2005 utilisée pour la publication.
- Pas de **garantie formelle** que la version "reconstruite" en 2020 soit la même que la version de 2005.

Le premier point qui ressort des difficultés liées à la **récupération d'anciens codes sources** concerne la nécessité d'améliorer les procédures d'archivage des codes sources des logiciels développés dans notre laboratoire.

Nécessité de modifier les pratiques d'archivage et de conservation des codes et des données (git, softwareheritage, zenodo, ...).

Perte partielle du code source de DynELA

En 2005, deux versions du code de calcul co-existaient sur deux machines différentes.

- Une version non parallèle déployée sur un ordinateur portable avec suivi des modifications avec Concurrent Version System (CVS).
- Une version parallèle déployée sur le Proliant (fork de la version non parallèle) spécifique pour les besoins de la publication.

Etat des lieux des **manquements à la reproductibilité des résultats**.

- Le code source de la version parallèle n'a pas été correctement archivé avec les sources de la publication (perte de lien, mémoire,...).
- Les codes sources étaient sauvegardés sur plusieurs supports (CDROM, Zip, HDD,...), sans réelle cohérence.
- Les codes sources étaient mal "étiquetés" (pas de date fiable), nuisant à la chronologie.
- Le serveur Proliant a été débranché et mis "**à la casse**" sans sauvegarder les tous derniers développements du code.
- Nécessité de **reconstruire** en 2020 une version aussi proche que possible de la version finale 2005 utilisée pour la publication.
- Pas de **garantie formelle** que la version "reconstruite" en 2020 soit la même que la version de 2005.

Le premier point qui ressort des difficultés liées à la **récupération d'anciens codes sources** concerne la nécessité d'améliorer les procédures d'archivage des codes sources des logiciels développés dans notre laboratoire.

Nécessité de modifier les pratiques d'archivage et de conservation des codes et des données (git, softwareheritage, zenodo, ...).

Perte partielle du code source de DynELA

En 2005, deux versions du code de calcul co-existaient sur deux machines différentes.

- Une version non parallèle déployée sur un ordinateur portable avec suivi des modifications avec Concurrent Version System (CVS).
- Une version parallèle déployée sur le Proliant (fork de la version non parallèle) spécifique pour les besoins de la publication.

Etat des lieux des **manquements à la reproductibilité des résultats**.

- Le code source de la version parallèle n'a pas été correctement archivé avec les sources de la publication (perte de lien, mémoire,...).
- Les codes sources étaient sauvegardés sur plusieurs supports (CDROM, Zip, HDD,...), sans réelle cohérence.
- Les codes sources étaient mal "étiquetés" (pas de date fiable), nuisant à la chronologie.
- Le serveur Proliant a été débranché et mis "**à la casse**" sans sauvegarder les tous derniers développements du code.
- Nécessité de **reconstruire** en 2020 une version aussi proche que possible de la version finale 2005 utilisée pour la publication.
- Pas de **garantie formelle** que la version "reconstruite" en 2020 soit la même que la version de 2005.

Le premier point qui ressort des difficultés liées à la **récupération d'anciens codes sources** concerne la nécessité d'améliorer les procédures d'archivage des codes sources des logiciels développés dans notre laboratoire.

Nécessité de modifier les pratiques d'archivage et de conservation des codes et des données (git, softwareheritage, zenodo, ...).

Reconstruction du code source de DynELA

Procédure de reconstruction du code source à partir des archives.

- Récupération de l'ensemble des fichiers sources de la version 1.0 à partir d'une sauvegarde datant de 2004.
- Fusion "une par une" des modifications de la version parallèle dans l'ensemble des fichiers sources de la version 1.0.
- Reconstruction d'un post-processeur issu de la version 2010 pour la visualisation des résultats et la production des fichiers graphiques.
- Remplacement la classe de mesure du temps du processeur par celle de la version la plus récente de DynELA.

Recompilation bibliothèque après bibliothèque de l'ensemble du code et adaptation aux changements de norme.

- **basicTools** : bibliothèque de base qui contient les classes de base de DynELA (gestion des objets, listes,...).
- **linearAlgebra** : bibliothèque des opérations algébriques. Elle définit en particulier les notions de vecteurs, matrices, tenseurs et fonctions mathématiques. Elle encapsule une partie des bibliothèques Lapack et Blas.
- **interpretor** : bibliothèque qui définit le langage de commande interprété de DynELA. Elle était l'un des points forts du logiciel original.
- **femLibrary** : bibliothèque de calcul par éléments finis (c'est le cœur du solveur d'éléments finis).

A partir de ces différentes bibliothèques, nous avons créé un ensemble de programmes exécutables qui correspondent aux différents modules du code FEM : le solveur d'éléments finis, le post-processeur graphique, le programme d'analyse de courbes, le générateur de langage, etc...

Reconstruction du code source de DynELA

Procédure de reconstruction du code source à partir des archives.

- Récupération de l'ensemble des fichiers sources de la version 1.0 à partir d'une sauvegarde datant de 2004.
- Fusion "une par une" des modifications de la version parallèle dans l'ensemble des fichiers sources de la version 1.0.
- Reconstruction d'un post-processeur issu de la version 2010 pour la visualisation des résultats et la production des fichiers graphiques.
- Remplacement la classe de mesure du temps du processeur par celle de la version la plus récente de DynELA.

Recompilation bibliothèque après bibliothèque de l'ensemble du code et adaptation aux changements de norme.

- **basicTools** : bibliothèque de base qui contient les classes de base de DynELA (gestion des objets, listes,...).
- **linearAlgebra** : bibliothèque des opérations algébriques. Elle définit en particulier les notions de vecteurs, matrices, tenseurs et fonctions mathématiques. Elle encapsule une partie des bibliothèques Lapack et Blas.
- **interpretor** : bibliothèque qui définit le langage de commande interprété de DynELA. Elle était l'un des points forts du logiciel original.
- **femLibrary** : bibliothèque de calcul par éléments finis (c'est le cœur du solveur d'éléments finis).

A partir de ces différentes bibliothèques, nous avons créé un ensemble de programmes exécutables qui correspondent aux différents modules du code FEM : le solveur d'éléments finis, le post-processeur graphique, le programme d'analyse de courbes, le générateur de langage, etc...

Reconstruction du code source de DynELA

Procédure de reconstruction du code source à partir des archives.

- Récupération de l'ensemble des fichiers sources de la version 1.0 à partir d'une sauvegarde datant de 2004.
- Fusion "une par une" des modifications de la version parallèle dans l'ensemble des fichiers sources de la version 1.0.
- Reconstruction d'un post-processeur issu de la version 2010 pour la visualisation des résultats et la production des fichiers graphiques.
- Remplacement la classe de mesure du temps du processeur par celle de la version la plus récente de DynELA.

Recompilation bibliothèque après bibliothèque de l'ensemble du code et adaptation aux changements de norme.

- **basicTools** : bibliothèque de base qui contient les classes de base de DynELA (gestion des objets, listes,...).
- **linearAlgebra** : bibliothèque des opérations algébriques. Elle définit en particulier les notions de vecteurs, matrices, tenseurs et fonctions mathématiques. Elle encapsule une partie des bibliothèques Lapack et Blas.
- **interpretor** : bibliothèque qui définit le langage de commande interprété de DynELA. Elle était l'un des points forts du logiciel original.
- **femLibrary** : bibliothèque de calcul par éléments finis (c'est le cœur du solveur d'éléments finis).

A partir de ces différentes bibliothèques, nous avons créé un ensemble de programmes exécutables qui correspondent aux différents modules du code FEM : le solveur d'éléments finis, le post-processeur graphique, le programme d'analyse de courbes, le générateur de langage, etc...

Récupération du code source et compilation

Compilation du code source de DynELA

Le code DynELA comprend plusieurs milliers de lignes de C++ organisées dans une structure arborescente complexe.

Problématique de compilation initiale :

- Compilation chaînée répertoire par répertoire (séparation des fonctions)
- Makefiles écrits manuellement (simples mais efficaces)
- Compilation directement dans les répertoires sources (mauvaise habitude)
- Mélange de fichiers sources, en-têtes et objets compilés dans les mêmes répertoires (je sais, c'est mal !).

Solution mise en œuvre :

- Adoption de l'utilitaire CMake pour générer les différents Makefiles
- Réorganisation avec séparation des sources et d'un arbre de compilation (Build)
- Création de directives de compilation spécifiques

Gestion des dépendances :

- Prise en compte des bibliothèques externes (principalement Flex et Bison)
- Installation facilitée via les packages Ubuntu standard
- Récupération des informations nécessaires depuis les anciens Makefiles

Récupération du code source et compilation

Compilation du code source de DynELA

Le code DynELA comprend plusieurs milliers de lignes de C++ organisées dans une structure arborescente complexe.

Problématique de compilation initiale :

- Compilation chaînée répertoire par répertoire (séparation des fonctions)
- Makefiles écrits manuellement (simples mais efficaces)
- Compilation directement dans les répertoires sources (mauvaise habitude)
- Mélange de fichiers sources, en-têtes et objets compilés dans les mêmes répertoires (je sais, c'est mal !).

Solution mise en œuvre :

- Adoption de l'utilitaire CMake pour générer les différents Makefiles
- Réorganisation avec séparation des sources et d'un arbre de compilation (Build)
- Création de directives de compilation spécifiques

Gestion des dépendances :

- Prise en compte des bibliothèques externes (principalement Flex et Bison)
- Installation facilitée via les packages Ubuntu standard
- Récupération des informations nécessaires depuis les anciens Makefiles

Récupération du code source et compilation

Compilation du code source de DynELA

Le code DynELA comprend plusieurs milliers de lignes de C++ organisées dans une structure arborescente complexe.

Problématique de compilation initiale :

- Compilation chaînée répertoire par répertoire (séparation des fonctions)
- Makefiles écrits manuellement (simples mais efficaces)
- Compilation directement dans les répertoires sources (mauvaise habitude)
- Mélange de fichiers sources, en-têtes et objets compilés dans les mêmes répertoires (je sais, c'est mal !).

Solution mise en œuvre :

- Adoption de l'utilitaire CMake pour générer les différents Makefiles
- Réorganisation avec séparation des sources et d'un arbre de compilation (Build)
- Création de directives de compilation spécifiques

Gestion des dépendances :

- Prise en compte des bibliothèques externes (principalement Flex et Bison)
- Installation facilitée via les packages Ubuntu standard
- Récupération des informations nécessaires depuis les anciens Makefiles

Récupération du code source et compilation

Compilation du code source de DynELA

La compilation du code DynELA s'effectue avec le compilateur standard sur Ubuntu 18.04.4 LTS : le C++ 7.4.0.

Parallélisation :

- Utilisation de l'option de compilateur –fopenmp
- Bibliothèques de parallélisation OpenMP fournies par gcc-7

Problèmes de compilation :

- Plusieurs messages d'avertissement générés lors de la compilation
- Erreurs principalement liées à des fonctions non explicitement définies à la compilation
- Evolution du standard C++ au cours des 15 dernières années
- Warnings ignorés sans conséquence apparente sur le fonctionnement du code

Bibliothèques mathématiques :

- Compilation des bibliothèques mathématiques Lapack et Blas sans problème
  Note : ces bibliothèques ont été traduites par f2c et intégrées au code source.

Introduction et généralités

Récupération du code source et compilation

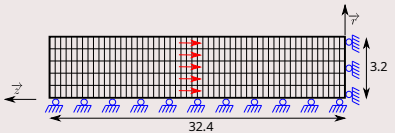
Comparaison des résultats des deux exécutions 2005/2020

Conclusions

Comparaison des résultats des deux exécutions 2005/2020

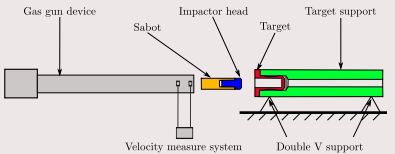
Essai d'impact de Taylor

- Matériau : cuivre OHFC
- Loi d'écoulement : Elasto-plastique
- Vitesse impact : 227 m/s



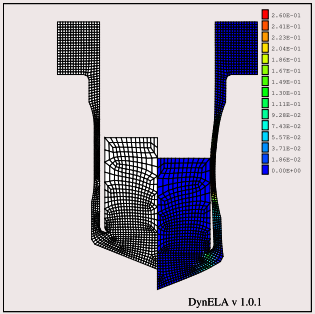
Essai d'impact de Taylor inversé

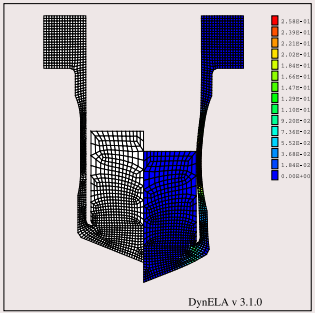
- Matériau : aluminium 2017T3
- Tête impact : 42CrMo4
- Loi d'écoulement : Elasto-plastique
- Vitesse impact : 80 m/s



Comparaison des résultats des deux exécutions 2005/2020

Comparaison des résultats obtenus / publication originale





- Simulation sur 4 cœurs et résultats concernant la déformation plastique équivalente.
- Les valeurs numériques **ne sont pas identiques**, mais la **différence est négligeable**.

Comparaison des résultats des deux exécutions 2005/2020

Parallélisation du calcul des forces internes

- Le principal sujet de la publication originale de 2005 concernait la parallélisation du code de calcul DynELA et **l'influence des stratégies de parallélisation**.
- Nous avons donc ré-exécuté les simulations numériques afin de comparer les stratégies de parallélisation et de **répliquer les résultats obtenus en 2005**.
- On se focalise ici sur la **parallélisation du calcul des forces internes** pour un essai d'impact de Taylor avec un maillage axi-symétrique de 6500 éléments finis (25×250).
- Ce calcul est la partie la plus **consommatrice en ressources de calcul**.
- Dans l'article original de 2005, **4 stratégies de parallélisation ont été comparées**, parmi les 8 méthodes disponibles à l'époque dans le code DynELA.

- **Méthode 1**: Utilise une directive '#pragma omp parallel for' avec un vecteur Fint partagé et une section critique pour l'assemblage, provoquant des goulots d'étranglement.
- **Méthode 2**: Emploie une région parallèle où les threads accèdent à une liste partagée d'éléments à traiter avec des vecteurs Fint privés locaux, évitant les sections critiques pour l'assemblage.
- **Méthode 3**: Distribue un nombre prédéterminé et égal d'éléments à chaque thread via la classe Jobs, éliminant l'accès concurrent à une liste partagée.
- **Méthode 4**: Combine la méthode 3 avec un équilibrage de charge dynamique qui redistribue les éléments en fonction des temps d'attente mesurés pour optimiser l'utilisation des processeurs.

Comparaison des résultats des deux exécutions 2005/2020

Parallélisation du calcul des forces internes

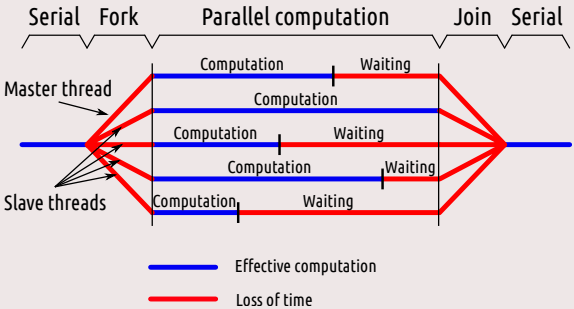
- Le principal sujet de la publication originale de 2005 concernait la parallélisation du code de calcul DynELA et **l'influence des stratégies de parallélisation**.
- Nous avons donc ré-exécuté les simulations numériques afin de comparer les stratégies de parallélisation et de **répliquer les résultats obtenus en 2005**.
- On se focalise ici sur la **parallélisation du calcul des forces internes** pour un essai d'impact de Taylor avec un maillage axi-symétrique de 6500 éléments finis (25×250).
- Ce calcul est la partie la plus **consommatrice en ressources de calcul**.
- Dans l'article original de 2005, **4 stratégies de parallélisation ont été comparées**, parmi les 8 méthodes disponibles à l'époque dans le code DynELA.

- **Méthode 1**: Utilise une directive '#pragma omp parallel for' avec un vecteur Fint partagé et une section critique pour l'assemblage, provoquant des goulots d'étranglement.
- **Méthode 2**: Emploie une région parallèle où les threads accèdent à une liste partagée d'éléments à traiter avec des vecteurs Fint privés locaux, évitant les sections critiques pour l'assemblage.
- **Méthode 3**: Distribue un nombre prédéterminé et égal d'éléments à chaque thread via la classe Jobs, éliminant l'accès concurrent à une liste partagée.
- **Méthode 4**: Combine la méthode 3 avec un équilibrage de charge dynamique qui redistribue les éléments en fonction des temps d'attente mesurés pour optimiser l'utilisation des processeurs.

Comparaison des résultats des deux exécutions 2005/2020

Algorithme d'équilibrage de charge

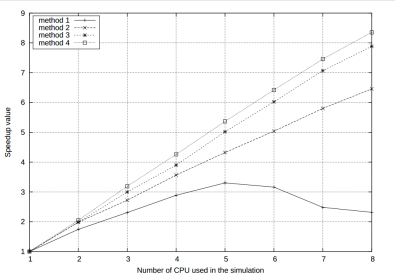
- On compare les résultats obtenus par l'algorithme d'équilibrage de charges pour le calcul du vecteur des efforts internes.
- Cet algorithme minimise le temps d'attente des threads parallèles.

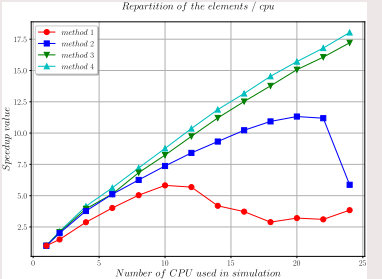


Comparaison des résultats des deux exécutions 2005/2020

Parallélisation du calcul des forces internes

- Le Dell R730 dispose de 24 cœurs, on a donc pu **étendre** l'analyse au delà des 8 processeurs de l'article original.





- Malgré certaines différences, on note que la version 2020 **reproduit** les mêmes tendances que l'original en 2005.
- Les 4 méthodes se comportent de manière analogue.

Analyse des différences de performance

Comparaison des performances de parallélisation entre l'architecture de 2005 (Compaq Proliant) et celle de 2020 (Dell R730)

Les speedups obtenus en 2020 sont globalement inférieurs au-delà de 8 processeurs.

Système d'exploitation : Meilleure gestion du multitâche en 2020 qu'en 2005

- Impact sur la charge machine et la répartition des ressources

Architecture matérielle :

- 2005: 8 processeurs 32-bits indépendants
- 2020: 2 processeurs 64-bits multi-cœurs (24 cœurs au total)

Infrastructure réseau :

- 2020: Disques sur un autre serveur avec protocole NFS
- Impact potentiel sur les performances de parallélisation

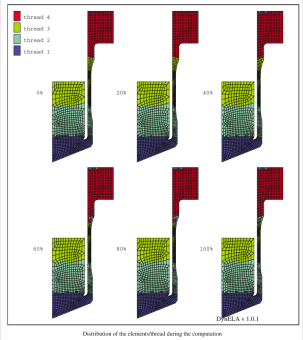
Taille du modèle :

- Modèle non optimisé pour grand nombre de processeurs (>8)
- Temps fork/join non négligeables quand charge/processeur diminue

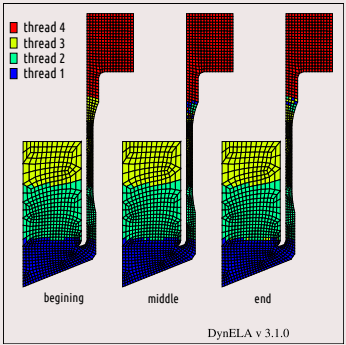
Malgré ces différences, la version 2020 reproduit les mêmes tendances qu'en 2005.

Comparaison des résultats des deux exécutions 2005/2020

Algorithme d'équilibrage de charge



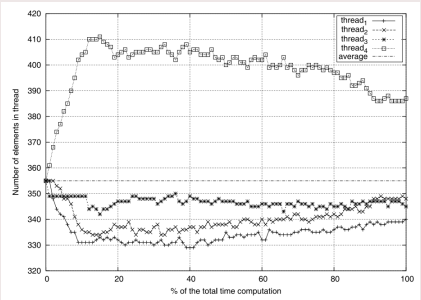
Distribution of the elements/thread during the computation

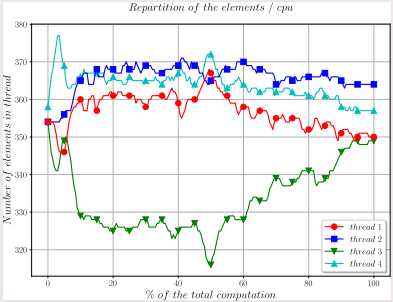


- Distribution spatiale des éléments sur 4 cœurs, au début de la simulation (à gauche), à 50% du calcul (milieu) et à la fin du calcul (à droite).

Comparaison des résultats des deux exécutions 2005/2020

Algorithme d'équilibrage de charge





- Evolution du nombre d'éléments pris en charge par chaque cœur de calcul au cours de la simulation.

Distinction entre réplication et reproduction

Définitions conceptuelles

- **Reproduction :** Capacité à obtenir exactement les mêmes résultats numériques en utilisant les méthodes, matériaux et données originaux
- **Réplication :** Capacité à confirmer les tendances et comportements généraux, sans nécessairement obtenir des résultats identiques

Application au cas DynELA (2005 $\longrightarrow$ 2020)

Reproduction

- Résultats numériques des simulations
- Valeurs de déformation plastique
- Comportement mécanique des structures

Réplication

- Stratégies de multithreading
- Comportement relatif des 4 méthodes
- Tendances des courbes de speedup
- Algorithmes d'équilibrage de charge

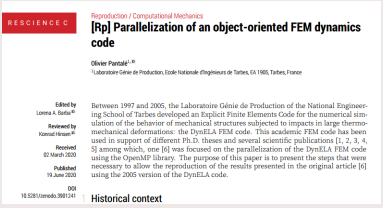
Conclusion

Malgré les différences d'architecture (32-bits single core $\longrightarrow$ 64-bits multi-core), les résultats sont à la fois :

- **Reproduisibles** concernant les résultats numériques de simulation
- **Réplicables** concernant les stratégies de parallélisation où la méthode 1 sature et la méthode 4 offre le meilleur scaling

1. Introduction et généralités
2. Récupération du code source et compilation
3. Comparaison des résultats des deux exécutions 2005/2020
4. Conclusions

Récapitulatif général de la procédure



RESCIENCE C

Reproduction / Computational Mechanics

[Rp] Parallelization of an object-oriented FEM dynamics code

Olivier Pantalé[1,*]

[1] Laboratoire Génie de Production, Ecole Nationale d'Ingénieurs de Tarbes, EA 1905, Tarbes, France

Edited by
Lorena A. Barba[ed]

Reviewed by
Konrad Hinsen[rev]

Received
02 March 2020

Published
19 June 2020

DOI
10.5281/zenodo.3901241

Historical context

Between 1997 and 2005, the Laboratoire Génie de Production of the National Engineering School of Tarbes developed an Explicit Finite Elements Code for the numerical simulation of the behavior of mechanical structures subjected to impacts in large thermo-mechanical deformations: the DynELA FEM code. This academic FEM code has been used in support of different Ph.D. theses and several scientific publications [1, 2, 3, 4, 5] among which, one [6] was focused on the parallelization of the DynELA FEM code using the OpenMP library. The purpose of this paper is to present the steps that were necessary to allow the reproduction of the results presented in the original article [6] using the 2005 version of the DynELA code.

- Il est encore possible, avec quelques ajustements, de recompiler en 2020 le code tel qu'il a été proposé en 2005.
- L'exécution du code sur les architectures informatiques modernes n'a pas posé de problème, aucun plantage n'ayant été constaté lors des simulations numériques et de l'utilisation du code.
- Les résultats en termes de performance sont cohérents avec les résultats obtenus il y a 15 ans, compte tenu de l'évolution radicale du matériel.

- Cet article[a] traite à la fois de replication (s'applique à l'analyse des stratégies de multithreading), et de reproduction (s'applique au benchmark sur la simulation de la traction dynamique) de l'article original.
- Nous avons constaté que, même si l'architecture des processeurs diffère (32-bits single core d'une part et 64-bits multi-core d'autre part), les résultats sont reproducibles concernant les résultats numériques et replicables concernant les stratégies de multithreading avec OpenMP où la méthode 1 sature et n'évolue pas, et la méthode 4 évolue le mieux avec l'augmentation du nombre de processeurs.

[a] O. Pantalé, *[Rp] Parallelization of an object-oriented FEM dynamics code*, ReScience C, vol 6, n°1, 2020