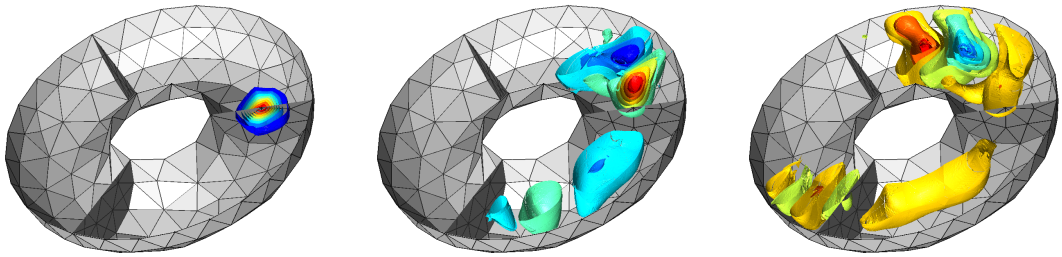
Projet : Élément finis discontinus

L'objectif de ce projet est d'améliorer les performances d'un code de simulation numérique pour des calculs sur un processeur multi-cœur. Ce code permet de simuler la propagation d'ondes scalaires en 3D. Il est basé sur une méthode d'éléments finis discontinus de type Galerkin pour des maillages composés de tétrahèdres avec des fonctions de base polynomiales de degré $N \in [1, 7]$.



$t = 0$ $t = 1.96$ $t = 3.92$

Figure 1: Exemple de simulation. Propagation d'une onde scalaire dans un tore composé de 3 milieux physiques différents. Les parties inférieures du maillage et de la solution sont montrées.

Consignes

Ce projet peut se faire seul ou à deux. L'évaluation se fera en deux temps.

Un rapport intermédiaire correspondant aux étapes 1 et 2 du projet, accompagné de votre fichier `main.cpp`, devront être envoyés par e-mail ***au plus tard le mercredi 24 avril 2024***.

- Dans un rapport intermédiaire ***(maximum 4 pages)***, expliquez les stratégies d'optimisation utilisées/envisagées (étape 1) et présentez les études de performance (étapes 1 et 2).
- Votre code `main.cpp` ***(nettoyés!)*** accompagné des codes de départ doit compiler et donner la bonne solution, mais plus rapidement que le code initial.

Un rapport final correspondant à l'ensemble du projet (dont les étapes 1 et 2), accompagné des codes finaux, devront être envoyés par e-mail ***au plus tard le vendredi 10 mai 2024***.

- Dans le rapport final, expliquez les stratégies d'optimisation de vos implémentations et présentez les études de performance de ces stratégies.
- Les codes finaux ***(nettoyés!)*** doivent compiler et donner la bonne solution, mais plus rapidement que le code initial.

Dans tous les cas, n'envoyez pas les maillages et vos fichiers de solutions, qui peuvent être très lourds.

Étape 0 : Pré-requis et démarrage du projet

Pour ce projet, on fournit un code `C++` de départ non-optimisé. Pour la prise en main, suivez les indications du fichier `README`. Il vous indique comment compiler le code, comment lancer une simulation, et comment visualiser les résultats.

Au niveau logiciel, vous aurez besoin du compilateur `icpc` *et d'une version récente du logiciel* `gmsh`*. La visualisation des solutions pourra se faire sur votre ordinateur personnel. Je vous recommande d'utiliser la dernière version stable de* `gmsh`*, à télécharger sur le site* `http://gmsh.info/`*. Sur ce site, utilisez les liens de la ligne en dessous de "Current stable release". N'utilisez pas les versions des dépôts Linux.*

Étape 1 : Analyse préliminaire de la performance sur le code non-optimisé

Analysez les performances de la phase de *run (c'est-à-dire la partie avec la boucle temporelle)* du code de départ. Ignorez les phases d'initialisation et l'enregistrement des solutions en utilisant `OutputStep=0` lors du calcul des temps.

Plus spécifiquement, étudiez l'évolution (1) du temps de calcul et (2) du débit arithmétique *(le rapport doit contenir la formule du débit arithmétique que vous utilisez)* pour des degrés polynomiaux N allant de 1 à 7. Pour chaque degré polynomial, utilisez un maillage correspondant à environ 250.000 inconnues discrètes. Des maillages sont fournis dans le dossier `benchmark/cube/` du code.

Le calcul du pas de temps est inclus le code de calcul. Pour une durée de simulation donnée, il n'y aura pas le même nombre de pas de temps suivant le maillage et le degré polynomial utilisés. Pour les analyses de performance, vous pouvez comparer les implémentations en utilisant "le runtime pour une durée de simulation donnée" ou bien "le runtime moyen par pas de temps". Les deux choix sont valables.

Cette étape n'est pas compliquée, mais elle peut prendre du temps sans une bonne organisation. Dans un premier temps, vous pouvez ne considérer que les cas $N = 1$, 3 et 5 par exemple. Une fois satisfait des résultats, vous pouvez complétez votre analyse en produisant les résultats pour les autres valeurs de N.

Étape 2 : Optimisation et analyse de performance

Optimisez la phase de *run* du code de calcul *(i.e. parallélisation et vectorisation)*. Seul le code `main.cpp` peut être modifié.

Pour le rapport intermédiaire, répéter l'analyse de performance sur le code optimisé, et comparer avec le code de départ.

Les deux difficultés principales de cette étape (et du projet) sont de rentrer dans le code de départ qui vous a été fourni, et ensuite de garder un code qui donne la bonne solution.

1. *Une fois familiarisé avec le code, identifiez les boucles qui pourraient être vectorisées et parallélisées. Certaines boucles imbriquées pourraient être inversées. Certaines opérations répétées plusieurs fois pourraient être pré-calculées. Énormément d'optimisations sont possibles, mais toutes ne sont pas efficaces. Je vous recommande de tester l'efficacité de chaque optimisation que vous imaginez "une à la fois".*
2. *Le code final optimisé doit donner la bonne solution. Au cours de votre travail, des bugs pourraient être introduits. Je vous recommande donc de faire des sauvegardes régulières, et de vérifier régulièrement si votre code donne bien la bonne solution. En général, j'ai une simulation de référence avec une solution écrite dans un fichier "de référence". Pendant l'optimisation d'un code, je reproduit régulièrement la simulation, et je vérifie sur le nouveau fichier de résultat est bien identique au fichier "de référence" (avec un `diff`).*

Étape 3 : Implémentation alternative avec BLAS

Concevez, optimisez, vectorisez et parallélisez une implémentation alternative utilisant les routines BLAS de la librairie `mkl` pour les opérations avec des matrices. De nouveau, seul le code `main.cpp` peut être modifié. Le code final de la variante sera nommé `mainBLAS.cpp`. Il sera compilé en utilisant la commande `make blas` (à inclure dans le `Makefile`).

Pour le rapport final, répétez l'analyse de performance sur le code optimisé utilisant BLAS, et comparez avec le code de départ et sa version optimisée.

Dans tous les cas, n'utilisez les routines BLAS que pour les opérations d'algèbre linéaire qui impliquent des matrices. N'utilisez pas ces routines pour des produits scalaires par exemple, c'est inefficace.