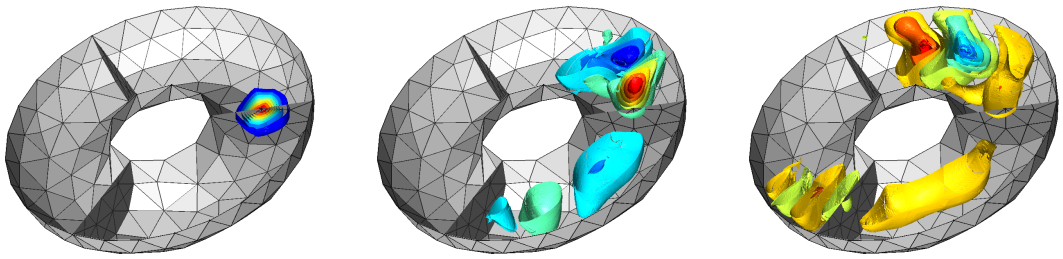
**Projet : Élément finis discontinus**

L'objectif de ce projet est d'améliorer les performances d'un outil de simulation numérique pour des calculs sur un processeur multi-cœur. L'outil 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 et 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.

### Étape 1 : Pré-requis et démarrage du projet

*Au niveau logiciel, vous aurez besoin du compilateur* `icc` *et d'une version récente du logiciel de maillage* `gmsh`*. 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.*

Pour démarrer ce projet, on fournit un code `C` de départ (non-optimisé) à télécharger sur le site du cours. L'archive contient un fichier `README` avec les commandes de base pour compiler le code, exécuter le programme, lancer un cas test et visualiser les résultats. On fournit également des maillages pour l'analyse de performance de la 3$^{e}$ partie.

Pour prendre en main le code, suivez les indications du fichier `README`.

### Étape 2 : Optimisation et parallélisation de deux implémentation

Concevez, optimisez, vectorisez et parallélisez deux implémentations, l'une utilisant les routines BLAS de la librairie `mkl` pour les opérations avec des matrices, et l'autre n'utilisant pas de librairie.

Seul le code `main.c` peut être modifié. Les codes finaux des deux variantes seront nommés `mainBLAS.c` et `mainNoBLAS.c`, respectivement. Ils seront compilés en utilisant les commandes `make blas` et `make noblas` (à inclure dans le `Makefile`).

*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`*).*

*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.*

*Pour les binômes, je recommande que, dans un premier temps, vous travailliez en parallèle sur ces deux implémentations.*

### Étape 3 : Analyse de performance

Évaluez, comparez et analysez les performances de la phase de *run (c'est-à-dire la partie avec la boucle temporelle)* des implémentations NoBLAS et BLAS *(pour les binômes)* ou bien du code de départ et de l'implémentation optimisée *(pour les monômes)*.

Les deux implémentations seront comparées pour des degrés polynomiaux $N$ allant de 1 à 7 avec (1) le temps de calcul et (2) le débit arithmétique *(le rapport doit contenir la formule du débit arithmétique que vous utilisez)*. Pour chaque degré polynomial, utilisez un maillage correspondant à environ 250.000 inconnues discrètes. Ignorez les phases d'initialisation et l'enregistrement des solutions (utilisez `OutputStep=0`) pour calculer les temps.

*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 parfaitement 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 que vous êtes contents des résultats, vous pouvez produire les résultats pour les autres valeurs de $N$ pour compléter votre analyse.*

**Consignes finales**

L'évaluation de ce projet se fera sur base d'un rapport écrit et des codes finaux *(nettoyé !)*, à envoyer par e-mail à `axel.modave@ensta-paris.fr` ***au plus tard le jeudi 5 mai 2022***. En outre, une soutenance est prévue le ***mardi 10 mai 2022***.

Dans votre rapport de ***maximum 6 pages***, expliquez les stratégies d'optimisation de votre *(ou de vos)* implémentation*(s)* et présentez les études de performance de cette *(ou ces)* stratégie*(s)*.

Vos codes finaux doivent donner la bonne solution. N'envoyez que les codes finaux (pas les versions intermédiaires). N'envoyez pas les maillages et vos fichiers de solutions, qui peuvent être très lourds.