Optimisation d'un algorithme Galerkin Discontinu en OpenCL appliqué à la simulation en électromagnétisme

Bruno Weber, Philippe Helluy, Thomas Strub

Optimisation d'un algorithme Galerkin Discontinu en OpenCL appliqué à la simulation en électromagnétisme

Bruno Weber[1,2], Philippe Helluy[1], Thomas Strub[2]

[1]IRMA, Université de Strasbourg, Strasbourg, France
[2]Axessim, Illkirch-Graffenstaden, France
E-mail: bruno.weber@axessim.fr

***Résumé* — Dans cet article, nous présentons les résultats d'optimisation sur GPU et CPU d'un algorithme Galerkin Discontinu appliqué à l'électromagnétisme et codé en OpenCL et MPI. Cet algorithme a initialement été optimisé pour être exécuté en parallèle sur plusieurs GPUs et ensuite adapté pour CPUs. Les GPUs et CPUs nécessitent une implémentation propre à leur architecture matérielle. Nous commençons par préciser le contexte d'application. Dans un second temps, nous présentons les optimisations GPU ainsi que les performances obtenues sur GPU et CPU avec cette version du code. Enfin, nous décrivons les adaptations qui ont permis de décupler les performances sur CPU.**

I. INTRODUCTION

Le solveur CLAC (Conservation Laws Approximation on many Cores), développé par l'IRMA en partenariat avec la société AxesSim, est une implémentation OpenCL-MPI du schéma Galerkin Discontinu (GD) appliqué aux systèmes hyperboliques de lois de conservation [1,2]. L'algorithme GD est donné par l'expression analytique de la formulation faible du problème d'évolution lié au système hyperbolique étudié :

$$\int_L \partial_t W\psi \, dx - \int_L F(W, W, \nabla\psi) \, dx + \int_{\partial L} F(W_L, W_R, n_{LR})\psi_L \, ds = 0 \quad (1)$$

Avec W le vecteur des variables conservatives, L le volume courant, R un volume voisin, n_{LR} le vecteur normal à l'interface entre les volumes, orienté vers R (Fig. 1), F un flux d'interface consistant et conservatif et ψ une fonction test quelconque, continue sur l'union des ouverts L et R.

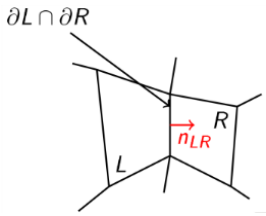

Figure 1 : Interface entre les volumes L et R

Chaque terme de la formule (1) (nommés dans l'ordre : masse, volume et flux) est implémenté par un kernel OpenCL distinct afin d'y optimiser les accès mémoire. Les transferts MPI sont effectués de manière asynchrone au niveau des interfaces, et de ce fait masqués par l'exécution des kernels OpenCL. L'intégration en temps est effectuée à l'aide du schéma de Runge-Kutta d'ordre 2 (RK2).

Dans le cas particulier de l'électromagnétisme, AxesSim développe la surcouche TETA qui spécialise le solveur CLAC pour les équations de Maxwell. Dans ce cas, le vecteur des variables conservatives contient les champs électrique et magnétique $W = (E_x, E_y, E_z, H_x, H_y, H_z)$.

II. OPTIMISATIONS OPENCL GPU

Les GPUs sont conçus pour exécuter en parallèle la même tâche sur de multiples données (architecture SIMD). Ils sont composés de plusieurs milliers de cœurs de calcul et de différents types de mémoire, dont notamment : la mémoire globale et la mémoire partagée (ou locale).

En OpenCL, chaque donnée est traitée par un work-item. Les work-items ont tous accès à la mémoire globale et peuvent être regroupés en work-groups, au sein duquel ils partagent de la mémoire locale, plus rapide d'accès que la mémoire globale.

Afin de tirer pleinement parti de leur architecture, les GPUs nécessitent une implémentation spécifique des kernels OpenCL. Quelques-unes de ces spécificités sont présentées ci-dessous.

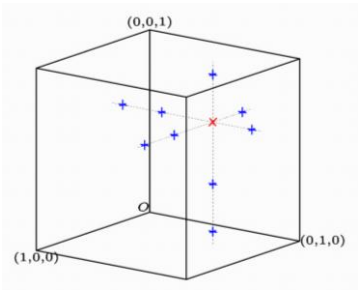

Figure 2 : Interactions entre les points d'interpolation à l'ordre 3 dans le cube de référence. Le point rouge n'interagit qu'avec les points bleus.

A. *Interpolation spatiale et flux de volume*

Le solveur CLAC est optimisé pour traiter des éléments hexaédriques, contrairement à d'autres solveurs qui traitent des maillages tétraédriques [6,7]. Pour chaque hexaèdre, le calcul des flux GD s'effectue dans le cube de référence $[0;1] \times [0;1] \times [0;1]$. A l'ordre d, l'élément de référence est discrétisé dans les 3 directions de l'espace par les $d+1$ points de Gauss-Legendre. Les polynômes d'interpolation de Lagrange qui en découlent sont choisis comme fonction de test dans (1). Ces polynômes ont la propriété d'annuler le gradient de ψ entre deux points non alignés dans la grille de discrétisation (Fig. 2), ce qui réduit le nombre d'interactions volumiques pour chaque point de $(d+1)^3$ à $3 \cdot (d+1)$.

B. *Utilisation de la mémoire locale*

Sur GPU, les points d'interpolation sont traités en parallèle par les kernels en les associant dans un work-group (ensemble de cœurs) par hexaèdre. A chaque point sont associées 6 variables conservatives, soit $6 \cdot (d+1)^3$ variables par élément. La mémoire locale étant plus rapide que la mémoire globale, et partagée dans un work-group, chacune de ces variables est chargée en mémoire locale

puis mise à jour en mémoire globale une seule fois par kernel et par élément en accès coalescent.

C. *Résultats de simulation*

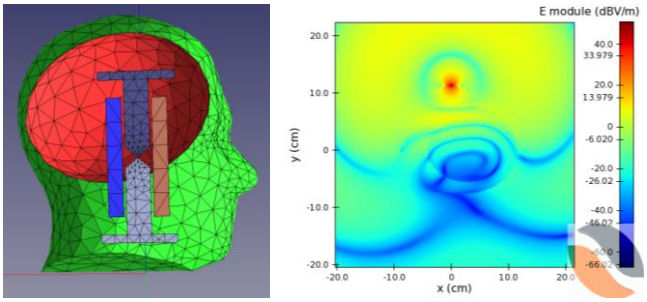

Figure 3 : Résultats de simulation du rayonnement d'une antenne ELPOSD [5] à 1 GHz à proximité d'une tête humaine. Gauche : surfaces du maillage tétraédrique découpé en hexaèdres pour la simulation. Droite : plan de coupe horizontal passant par la source de l'antenne représentant le module du champ E au temps t = 5ns.

Ces optimisations permettent au solveur TETA d'obtenir des temps de calcul comparables à ceux d'un solveur différences finies (FD) implémenté en OpenMP sur des maillages à géométries courbes. Les résultats obtenus dans le cas de la simulation d'une antenne à proximité d'une tête humaine (Fig. 3) sont identiques et les temps d'exécutions comparables (TABLEAU I).

TABLEAU I. COMPARAISON DES TEMPS D'EXECUTION FD-CPU/GD-GPU DANS LE CAS DE LA SIMULATION D'UNE ANTENNE A PROXIMITE D'UNE TETE HUMAINE

Solveur / Accélérateur	**Temps d'exécution**
TETA / GPU NVIDIA GeForce GTX 1070	5,6 heures
TEMSI-FD / CPU Intel Xeon E5645 (22 cœurs)	4,9 heures

III. OPTIMISATIONS OPENCL CPU

Les CPUs sont majoritairement conçus pour exécuter plusieurs tâches distinctes en parallèle (architecture MIMD). C'est pourquoi ils comportent moins de cœurs de calcul mais à plus haute fréquence qu'un GPU. De plus, ils bénéficient d'un accès direct à la mémoire vive (RAM) du nœud de calcul, généralement plus importante que celle d'un GPU. Ces différences d'architecture entre CPU et GPU imposent une approche algorithmique radicalement différente pour exploiter pleinement les ressources [3,4].

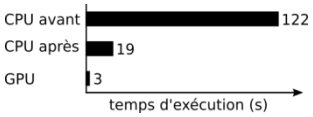

Figure 4 : Temps d'exécution sur un cas simple pour un GPU NVIDIA GeForce GTX 1070 et un CPU Intel Core i7-920 avant et après optimisations CPU

A. *Utilisation de la mémoire globale*

Sur CPU, la mémoire locale au sens d'OpenCL est physiquement de la mémoire globale (RAM). Ainsi, l'utilisation de la mémoire locale implique des transferts mémoire inutiles et coûteux. La première optimisation CPU a donc été de supprimer le chargement des variables conservatives en mémoire locale pour directement les accéder en mémoire globale. Dans le cas du kernel calculant le terme de volume, un speedup de 2 a été constaté.

B. *Augmentation de la taille des work-items*

Contrairement aux GPUs, pour lesquels le nombre de cœurs permet de paralléliser les traitements au niveau des points d'interpolation, ce choix multiplie les basculements mémoire sur CPU. De manière générale, il est préférable de sérialiser les traitements au sein d'un kernel CPU. Paramétrer les kernels afin que chaque work-item traite un élément a permis de constater une amélioration des performances de 25 %.

C. *Stockage des constantes*

Pour les données géométriques invariantes au fil des itérations – comme le gradient de ψ – leurs valeurs sont précalculées et stockées en mémoire, contrairement aux kernels GPU qui les recalculent à chaque itération. Les accès en mémoire vive sont plus rapides que la grande quantité de calculs que représentent ces données. Le speedup global constaté suite à cette modification est là aussi d'approximativement 2 (Fig. 4).

TABLEAU II. REPARTITION DES TEMPS DE CALCUL SUR CPU AVANT ET APRES OPTIMISATION DES KERNELS DE VOLUME ET DE SURFACE

Kernel OpenCL	**Avant (%)**	**Après (%)**
Volume	48	13
Surface	49	71
Masse	2	10
RK2	1	6

IV. CONCLUSIONS

Nous avons vu dans cet article les principales optimisations OpenCL implémentées dans le solveur GD TETA-CLAC. Avec ces optimisations, les performances du solveur GD sur GPU rivalisent avec celles des solveurs FD sur CPU. De plus, pour un cas de test simple, le speedup global du solveur GD sur CPU est d'environ 6 par rapport à l'implémentation GPU initiale exécutée sur CPU (TABLEAU II).

Un schéma en temps utilisant un pas de temps local par élément permettrait d'améliorer encore les performances du solveur, quelle que soit l'architecture. D'autre part, sur CPU, l'utilisation des unités de calcul vectorielles (SSE, AVX) serait un axe d'amélioration supplémentaire.

REFERENCES

[1] T. Strub. Résolution des équations de Maxwell tridimensionnelles instationnaires sur architecture massivement multicœur. Mémoire de thèse. Université de Strasbourg. 2015.

[2] P. Helluy, T. Strub, M. Massaro, M. Roberts. Asynchronous OpenCL/MPI numerical simulations of conservation laws. H.-J. Bungartz, P. Neumann, W. E. Nagel. Software for Exascale Computing - SPPEXA 2013-2015, Springer International Publishing, pp.547–565. 2016. <hal-01134222v2>

[3] Intel® SDK for OpenCL Applications. OpenCL Optimization Guide. 326542-001US. 2012.

[4] B. Catanzaro. OpenCL Optimization Case Study: Simple Reductions. 2010.

[5] T. G. Spence, D. H. Werner. A Novel Miniature Broadband/Multiband Antenna Based on an End-Loaded Planar Open-Sleeve Dipole. IEEE Transactions on Antennas and Propagation 54(12):3614-3620. 2006.

[6] T. Cabel, J. Charles, S. Lanteri. Multi-GPU acceleration of a DGTD method for modeling human exposure to electromagnetic waves. RR-7592, INRIA. 2011, pp.27.

[7] J. S. Hesthaven. High-order accurate methods in time-domain computational electromagnetics. Adv. Imaging Electron Phys. 127(2003), 59–123.