Accélération sur GPU d'une simulation radar avec OpenACC

Maxime MARTELLI[1,2,3], Cyrille ENDERLI[3], Nicolas GAC[1], Antoine VERMESSE[3], Alain MÉRIGOT[2]

[1]Laboratoire des Signaux et Systèmes, [2]Laboratoire des Systèmes et Applications des Technologies de l'Information et de l'Énergie, [3]Thales DMS France

Contexte industriel

1. **Modéliser nécessite une puissance de calcul toujours plus importante**, et, à défaut d'avoir les ressources de calcul suffisantes, le modèle en est souvent simplifié, donc moins précis.
2. L'adoption d'une **démarche d'adéquation algorithme architecture**, avec un système comprenant plusieurs architectures hétérogènes adaptées aux algorithmes à accélérer, permet de trouver une nouvelle dynamique de croissance dans le domaine des semi-conducteurs, à l'heure ou la fin de la Loi de Moore a été annoncée pour 2021[1].
3. La modélisation fine et rapide de systèmes radars est cruciale pour **réduire les coûts et en améliorer la fiabilité et la robustesse**.

Objectifs

1. Proposer une **méthodologie pour accélérer** un environnement d'une simulation RADAR, en passant d'une implémentation sur CPU à une implémentation sur GPU (Bibliographie correspondante[2],[3]).
2. **Évaluer la programmation par directives** avec OpenACC en terme de performances, mais aussi de facilité et de rapidité de mise en œuvre.

Radar : concepts préliminaires

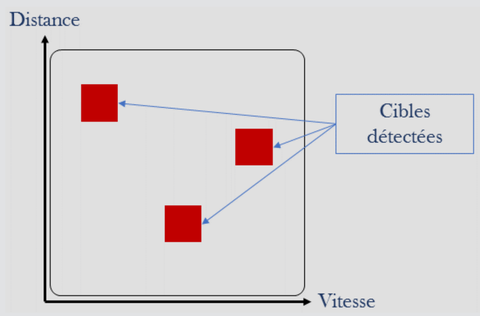

Figure 1: Carte d'ambiguïté en Distance et en Vitesse (DAVA)

La corrélation des fréquences des signaux émis et reçus par le radar permet de déduire la vitesse radiale de la cible (Effet Doppler-Fizeau). La carte DAVA permet d'**afficher de manière concise les cibles détectées** tout en les caractérisant en distance et en vitesse.

Cas d'étude : Simulateur d'Echos Numériques

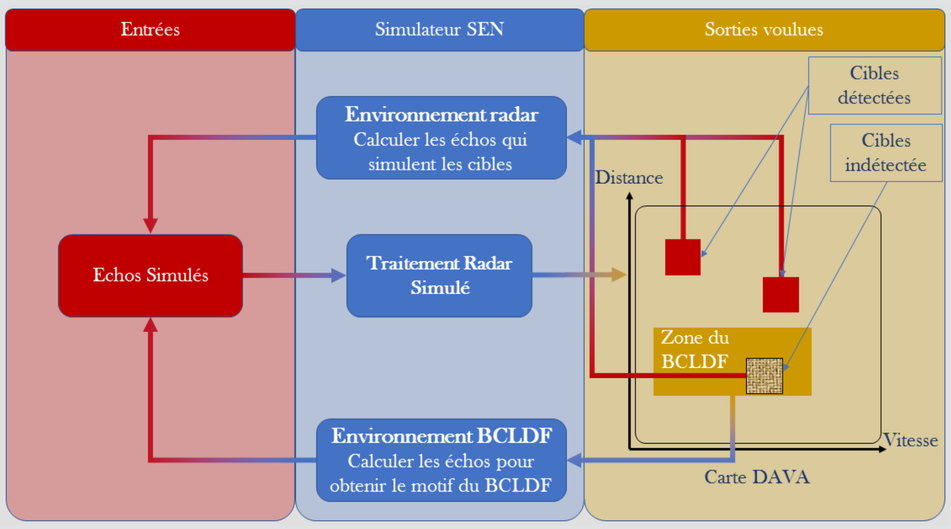

Figure 2: BCLDF et Simulateur SEN

En brouillant une zone rectangulaire sur la carte DAVA, le BCLDF empêche le radar de détecter une possible cible à l'intérieur du motif. Pour y parvenir, on doit **calculer les échos qu'un radar est supposé recevoir** pour que son traitement génère une telle carte.

Spécifications matérielles

Spécifications	CPU	GPU
Type	Intel Xeon E5-2667	GTX 1080Ti
Fréquence	2.9 GHz	1.48 GHz
Cœurs physiques	6	3584
Mémoire Globale	96 GB	11 GB
Bande passante	51.2 GB/s	484.4 GB/s
Performance FP32	500 GFLOPS	11.34 TFLOPS

Table 1: Spécifications matérielles des architectures utilisées

Précision : dans la section "Résultats et discussions", **le facteur d'accélération inclut les transferts mémoires CPU <-> GPU.**

Optimisations GPU implémentées

- **CPU Référence :** Version optimisée de référence sur CPU.
- **OpenACC Opt1 :** Conversion fonctionnelle du code CPU en code OpenACC GPU, transformation du code.
- **OpenACC Opt2 :** Segmentation des objets mémoires, répartition CPU/GPU et transformation des structures de données.
- **OpenACC Opt3 :** Optimisation de la localité des données sur CPU et GPU.
- **OpenACC Opt4 :** Paramétrage fin des structures mémoires et de leurs accès (taille des bus, ...).
- **OpenACC Opt5 :** Parallélisme de boucle, de fonction, et vectorisation.
- **CUDA Optimisé :** Version optimisée CUDA qui reprend les concepts d'optimisation de la version OpenACC Opt5.

Résultats et discussions

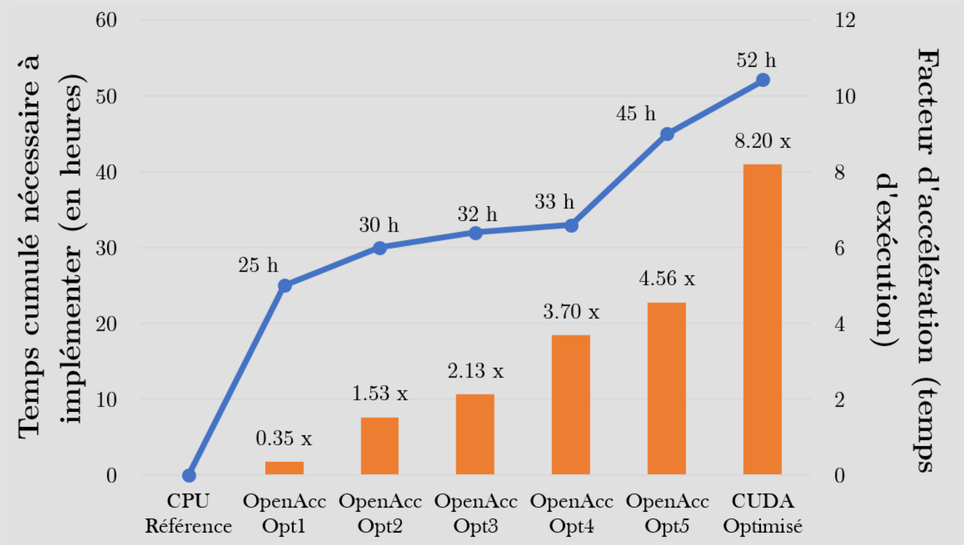

Figure 3: Facteur d'accélération du temps d'exécution & Temps d'implémentation : CPU - GPU OpenACC &CUDA

- L'optimisation OpenACC Opt4 montre **l'importance de la gestion mémoire** en co-processing.
- La meilleure optimisation OpenACC (Opt5) reste derrière la version CUDA optimisée, à cause de **lacunes au niveau de la gestion mémoire de l'outil**.
- Une fois la conversion effectuée (Opt1), **le temps d'implémentation des optimisations est rapide** (Opt2 à Opt4) pour une accélération significative.
- L'avantage d'utiliser OpenACC est de pouvoir **obtenir rapidement un facteur d'accélération correct**, et qu'il est ensuite facilement possible de continuer les optimisations en CUDA puisque la vue conceptuelle de l'architecture GPU est partagée par les deux outils.

Conclusion

1. Avec un **facteur d'accélération** maximal de **4.56** avec OpenACC et de **8.2** avec CUDA (par rapport au code de référence CPU), nous avons pu mettre en valeur les optimisations qui impactent les performances d'une implémentation GPU.
2. **OpenACC est rapide à prendre en main**, et ne modifie que peu le code CPU, qui reste donc lisible.
3. Si les algorithmes utilisés ne sont pas adaptés au calcul parallèle, il faut par contre les réécrire et transformer ainsi significativement le code de référence.
4. **La meilleure optimisation est obtenue en utilisant le langage CUDA**. Cela s'explique par son optimisation forte par rapport à OpenACC qui se veut de plus haut niveau.

En résumé, **OpenACC est une bonne solution de prototypage rapide d'un algorithme sur GPU**, qui permet, le cas échéant, de passer sur une programmation en CUDA pour tirer au mieux parti des GPUs NVIDIA.

Références

[1] The International Technology Roadmap For Semiconductors 2.0. *Semiconductor Industry Association*, 2015.

[2] Qian Zhang and Yangdong Deng. Toward of a GPU accelerated software navigation radar. *IEEE 4th International Conference on Software Engineering and Service Science*, 2013.

[3] Jean-François Degurse et al. Architecture GPU pour radar de surveillance spatial et pour radar aéroporté. *GRETSI*, 2013.