Traitement hétérogène de grandes images avec StarPU

QUÔC-DINH NGUYEN, PATRICK HORAIN

Institut Mines-Télécom / Télécom SudParis,
9 rue Charles Fourier, 91011 Évry Cedex, France

Nguyen.QuocDinh@gmail.com, Patrick.Horain@Telecom-Sudparis.eu

Résumé - Le traitement d'images de très grande taille peut être accéléré en utilisant le système StarPU pour ordonnancer les tâches sur une architecture parallèle et hétérogène (CPU et GPU). Nous analysons d'abord expérimentalement les performances pour déterminer un partitionnement optimal de l'image. Nous proposons ensuite une extension du système StarPU consistant à permettre de prendre en compte la valeur de certains arguments scalaires dans le modèle de performance et nous montrons expérimentalement que cela peut améliorer l'ordonnancement des tâches.

Abstract - Processing very large images can be accelerated using the StarPU task scheduler on a parallel and heterogeneous platform (CPU and GPU). We first experimentally analyze performances for the sake of an optimal image partitioning. Then, we propose to consider the value of some scalar variables in the StarPU performance model, and we experimentally show that it can result in a more efficient task scheduling.

1 Introduction

Des applications, telles que l'imagerie satellite ou la microscopie de grands champs, exploitent des images atteignant ou dépassant plusieurs dizaines de Go [**1**]. Leur traitement peut-être accéléré au moyen de CPU multi-cœurs ou de GPU massivement parallèles [**2**][**3**]. La répartition des tâches sur les ressources de calcul est souvent délicate car elle doit être adaptée à la plateforme matérielle et logicielle, et parfois anticiper des évolutions futures de leurs caractéristiques et de leurs performances. Des systèmes de gestion de ressources parallèles et hétérogènes ont été développés pour le calcul intensif [**4**] [**5**] [**6**], qui ordonnancent les tâches sur l'ensemble des ressources de calcul, CPU multi-cœurs, accélérateurs de type GPU ou CELL.

Dans ce papier, nous étudions et adaptons le système StarPU pour traiter une image de grande taille. La section 2 est une présentation rapide de StarPU. Dans la section 3, nous étudions expérimentalement la configuration de StarPU sur un exemple de traitement d'image. Enfin, en section 4, nous proposons d'enrichir le modèle de performance de StarPU pour améliorer l'ordonnancement des tâches en prenant en compte les arguments scalaires et ainsi réduire le temps d'exécution.

2 Présentation de StarPU

StarPU [**4**] est un environnement d'exécution parallèle et unifié. Il permet de distribuer des calculs sur un ensemble de ressources hétérogènes, CPU multi-cœurs (pour lesquels OpenMP peut être utilisé) ou accélérateurs de type GPU (avec CUDA, OpenCL), ou CELL (OpenCL). Un traitement est développé pour différentes architectures sous forme de noyaux de code écrits en CUDA, OpenCL ou en langage C sur CPU et regroupés en un *codelet* qui est une structure décrivant ces noyaux et le type (l'architecture) d'unité de traitement sur lequel ils peuvent être exécutés. StarPU exécute la version du code correspondant à l'architecture du processeur auquel chaque tâche est affectée.

Les images sont partitionnées en blocs de données traités en parallèle. Une tâche est l'activation d'un codelet pour un bloc de données sur une unité de traitement. Le module de gestion de mémoire de StarPU s'assure de la disponibilité des données en mémoire de cette unité et prend en charge les transferts éventuels avant l'activation de la tâche.

La durée d'exécution des tâches est variable et doit être déterminée statistiquement. Le modèle de performance enregistre les temps d'exécution des tâches en fonction de la taille des données dans une table avec hachage. En fonction de ces mesures de performance et de la présence des données en mémoire, l'ordonnanceur distribue les tâches aux unités de traitement disponibles. Il optimise l'exploitation de la puissance de calcul suivant des stratégies d'ordonnancement des tâches qui sont sélectionnées au niveau applicatif.

Ces modèles de performance permettent à l'ordonnanceur de StarPU de prévoir la durée d'exécution des tâches en fonction de la taille des données afin de distribuer les tâches sur l'ensemble des ressources de calcul.

3 Mise en œuvre de StarPU pour le traitement d'image

StarPU a été conçu en particulier pour le calcul matriciel. Nous nous intéressons ici à son utilisation pour le traitement parallèle et hétérogène de très grandes images.

Pour exploiter le parallélisme, l'image doit être découpée en blocs. Or, beaucoup d'opérations, par exemple des convolutions ou des filtres morphologiques, nécessitent pour calculer chaque pixel

résultat d'accéder aux pixels voisins dans l'image. Pour gérer les voisinages, nous générons des blocs de pixels qui se recouvrent sur leurs bords. Ainsi, chaque bloc peut être traité indépendamment des autres, dans une tâche de StarPU.

Les expérimentations ci-après concernent une opération intrinsèquement parallèle, une érosion morphologique (minimum local), que nous appliquons à une très grande image (13 723 × 6 812 pixels couleur, soient 280 Mo). Les temps d'exécution sont mesurés sur un PC sous Windows 7 intégrant un CPU Intel Xeon E5540 (4 cœurs à 2,53 GHz) avec 4 Go de RAM et une carte graphique composée d'un GPU Nvidia GTX 480 avec 1,5 Go de mémoire.

3.1 Intérêt du traitement hétérogène d'image

Selon le traitement effectué, la taille de l'image traitée et l'emplacement des données, l'exécution peut être plus rapide sur l'une ou l'autre des unités de traitement [**3**]. L'ordonnanceur de StarPU répartit les calculs sur les ressources de calcul disponibles, éventuellement hétérogènes.

Nous vérifions l'intérêt du calcul hétérogène sur notre exemple de traitement d'une grande image. Le tableau Tab 1 présente les temps d'exécution mesurés pour l'érosion par un élément structurant (voisinage) de 11 × 11 pixels (soit un recouvrement entre blocs de 5 pixels) de l'image découpée en 26 × 13 = 338 blocs d'environ 512 × 512 pixels. StarPU est configuré pour n'utiliser que le CPU, que le GPU, puis les deux processeurs.

Tab 1 : Exemples de durées d'exécution d'une érosion sur une grande image en fonction de l'unité de traitement utilisée

Unités de traitement	Durée d'exécution
CPU seul	760 ± 0,5 s
GPU seul	14,5 ± 0,2 s
CPU + GPU	10,4 ± 0,4 s

Bien que l'image soit initialement stockée en mémoire centrale, ce filtre très régulier est beaucoup plus rapide sur GPU que sur CPU. Le surcoût du transfert des données est largement compensé par la puissance de calcul du GPU.

Surtout, le traitement hétérogène, qui cumule les puissances de calcul des deux unités de traitement et évite de transférer une partie des blocs de données vers le GPU, est bien le plus rapide sur cet exemple.

3.2 Partitionnement de l'image

Le nombre de blocs de pixels, et donc le nombre de tâches générées dans StarPU, est inversement proportionnel à la taille des blocs. De petits blocs induisent des transferts de données moins efficaces, un surcoût de gestion des nombreuses tâches, voire une occupation insuffisante de l'architecture massivement parallèle du GPU (480 cœurs pour notre GPU NVIDIA GTX 480). De grands blocs ne permettent pas d'équilibrer finement la charge entre CPU et GPU.

La figure 1 détaille les temps d'exécution pour notre exemple de traitement hétérogène sur StarPU en fonction de la taille et du nombre de blocs. L'élément est encore de dimensions 11 × 11. Sur notre machine de test, la meilleure performance est obtenue pour des blocs d'environ 512 × 512 pixels.

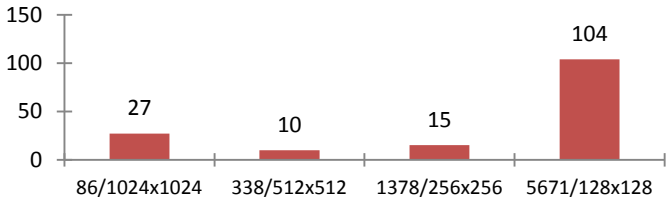


Figure 2 : Exemples de durées de traitement en fonction du nombre et de la taille des blocs de pixels

4 Prise en compte des paramètres scalaires

Pour distribuer les tâches sur les unités de traitement, l'ordonnanceur de StarPU exploite des prévisions de durées d'exécution issues des modèles de performance. Ceux-ci prennent en compte la taille des données, déterminante en calcul matriciel.

En traitement d'image, le temps d'exécution dépend bien sûr de la taille des images, mais dépend aussi de paramètres scalaires tels que la taille d'une fenêtre de convolution ou d'un élément structurant de morphologie, ou encore d'un nombre d'itérations. Pour cette raison, la bibliothèque de traitement d'image GpuCV [**3**] prend en compte les paramètres scalaires pour indexer les performances des fonctions. Bien sûr, toutes les variables scalaires n'ont pas un impact sur les performances et il incombe au développeur de l'application d'indiquer les variables pertinentes pour le modèle de performance.

Reprenant notre exemple d'érosion d'une grande image, le tableau Tab 2 montre l'influence du paramètre de taille de l'élément structurant sur la durée d'exécution, qui varie dans un rapport de plusieurs dizaines. La version actuelle du modèle de performance de StarPU ne permet pas de prendre en compte la valeur de ce paramètre numérique et moyenne toutes les durées d'exécution. Les prédictions dont dispose l'ordonnanceur sont alors très approximatives.

Tab 2 : Variation des durées moyennes d'exécution des tâches, enregistrées par le modèle de performance, en fonction d'un paramètre scalaire (taille de l'élément structurant de l'érosion)

Taille de l'élément structurant	Sur CPU	Sur GPU
3 × 3	19 ms	0,08 ms
7 × 7	70 ms	0,20 ms
11 × 11	145 ms	0,41 ms
21 × 21	504 ms	1,60 ms

Nous avons ajouté une interface aux modèles de performance de StarPU pour prendre en compte la valeur d'arguments scalaires pertinents. Les enregistrements de performances sont indexés par une clef calculée par hachage sur les paramètres déclarés du traitement. Nous avons donc ajouté une interface pour déclarer la valeur de paramètres numériques. Ainsi, la clef de hachage de ces enregistrements intègre, outre les dimensions des vecteurs et matrices, les valeurs des arguments scalaires déclarés. Les modèles de performance, qui accèdent aux mesures de temps d'exécution à travers la même fonction de hachage prennent ainsi en compte ces paramètres scalaires. Le modèle de performance indexe alors les performances distinctement en fonction de tous ces paramètres, et l'ordonnanceur dispose de prédictions plus précises de durées d'exécution.

L'intérêt de cette modification peut-être constaté lorsque deux exécutions parallèles de l'érosion précédente sont lancées avec des tailles différentes d'élément structurant. La Figure 2 montre la durée totale du traitement, avec et sans prise en compte du paramètre scalaire dans le modèle de performance, pour diverses stratégies d'ordonnancement : `dm` (*deque model*), `dmda` (*deque model data aware* [**5**]), `heft` (*heterogeneous earliest finish time* [**6**]). Pour les stratégies `dm`, `dmda` et `heft`, qui exploitent les prédictions de performance, la prise en compte du paramètre scalaire améliore la performance, au contraire de la stratégie `eager` qui n'exploite pas les prédictions de performances. L'enrichissement des modèles de performance de StarPU par la prise en compte des variables scalaires permet donc, pour les stratégies d'ordonnancement qui les utilisent, de réduire le temps d'exécution.

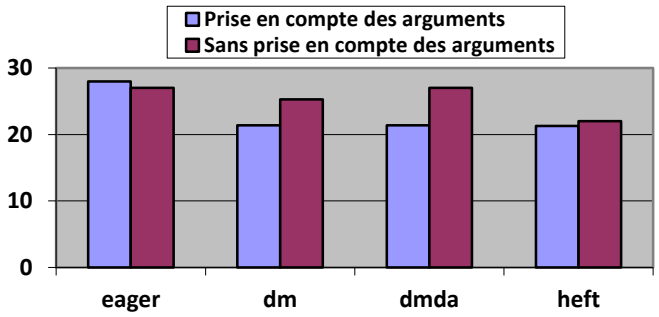
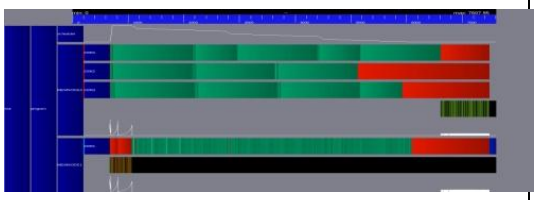
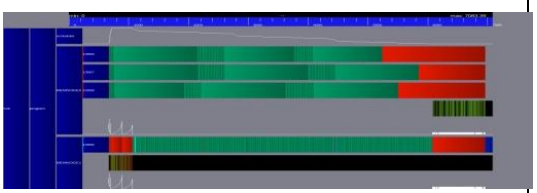
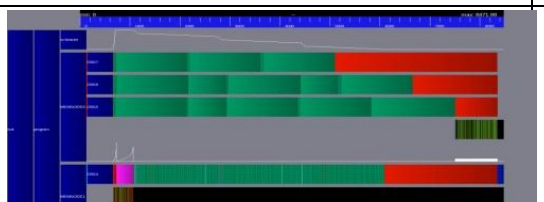
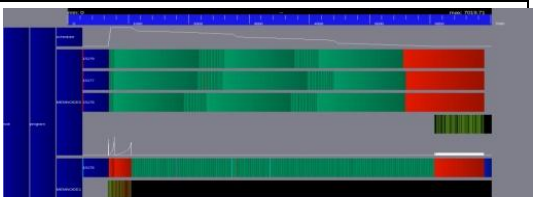
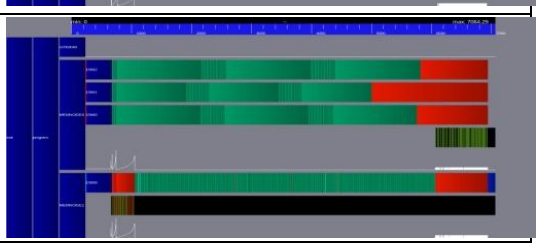


Figure 2 : Durée du traitement exécuté concurremment avec différents paramètres scalaires, pris en compte ou non dans le modèle de performance, pour quatre stratégies d'ordonnancement

Le tableau Tab 3 détaille l'occupation des ressources de calcul sans puis avec utilisation de notre interface de variable scalaire. La fin du traitement correspond au moment où toutes les unités de traitement deviennent inactives (elles sont alors représentées en rouge). Avec les stratégies d'ordonnancement `dm` et `dmda`, les tâches sont mieux réparties sur les ressources de calcul lorsque la variable scalaire est prise en compte et le traitement se termine un peu plus tôt. La différence est moins marquée pour la stratégie `heft`, qui est plus robuste.

Tab 3 : Occupation des ressources de calcul sans puis avec prise en compte des arguments scalaires dans les modèles de performance, pour trois ordonnanceurs qui exploitent ces modèles. Le temps figure en abscisse. Les 3 barres en haut montrent l'occupation des 3 cœurs CPU utilisés, celle en dessous l'occupation du GPU. Les unités occupées sont en vert avec les tâches successives représentées en gris. Les unités inactives sont en rouge.

Ordonnanceur	Sans prise en compte des arguments scalaires	Avec prise en compte des arguments scalaires
`dm`		
`dmda`		
`heft`		

5 Conclusion

Nous avons montré expérimentalement que StarPU constituait une plateforme adaptée pour traiter efficacement de très grandes images sur une architecture hétérogène. Il est toutefois utile de l'enrichir par des fonctionnalités dédiées. En particulier, nous montrons l'importance de modéliser les performances des traitements en tenant compte, outre de la taille des données, de la valeur de certains arguments scalaires.

Remerciements. Ce travail a reçu le soutien du projet ANR 2009-CORD-025 MediaGPU. Nous remercions Samuel Thibault de l'INRIA/LABRI pour la mise à disposition de StarPU et pour ses conseils.

Références

[1] Vilppu Tuominen and Jorma Isola, "Linking Whole-Slide Microscope Images with DICOM by Using JPEG2000 Interactive Protocol," *Journal of Digital Imaging*, vol. 23, no. 4, pp. 454-462, August 2010.

[2] Wen-mei Hwu, Ed., *GPU Computing Gems, Emerald Edition*.: Morgan Kaufmann / Elsevier Science, 2011.

[3] Y. Allusse, P. Horain, A. Agarwal, and C. Saipriyadarshan, "GpuCV: An OpenSource GPU Accelerated Framework for Image Processing and Computer Vision," in *16th ACM international conference on Multimedia*, Vancouver, BC, Canada, 2008, pp. 1089-1092, Open Source Software Competition.

[4] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, February 2011, Special Issue: Euro–Par 2009.

[5] J. M. Perez, R. M. Badia, and J. Labarta, "A dependency-aware task-based programming environment for multi-core architectures," in *2008 IEEE International Conference on Cluster Computing*, 2008, pp. 142-151.

[6] H. Topcuoglu, S. Hariri, and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, March 2002.

[7] C. Augonnet, S. Thibault, and R. Namyst, "Automatic Calibration of Performance Models on Heterogeneous Multicore Architectures," in *Euro-Par 2009 – Parallel Processing Workshops*, vol. LNCS 6043 , 2009, pp. 56-65.