Université de Bordeaux

Programmation Multicœur et GPU

Simulation de particules sur architectures parallèles

Auteur :
Guillaume Thibault

Référents :
Raymond Namyst
Pierre-André Wacrenier

18 avril 2014

1 Contexte

Ce compte-rendu s'incrit dans le cadre du projet de l'Unité d'Enseignement Programmation Multicœur et GPU dont le but est de concevoir une application de simulation de particules dans un domaine en trois dimensions, en calculant des interactions à courte distance entre particules. Il nous a été demandé d'y présenter les expérimentations que l'on a réalisé sur la version séquentielle et sur celle programmée en OpenMP et ce sur différentes machines multicœurs.

Ce compte-rendu présente donc ces expérimentations dans une première partie, puis une analyse des observations dans un deuxième partie.

2 Présentation des expérimentations

Les résultats seront présentés sous forme de graphiques, avec pour l'axe des abscisses le nombre de threads utilisés lors du test et sur l'axe des ordonnées la valeur de l'accélération par rapport à la version séquentielle.

La première étape fut d'observer les résultats de l'éxecution de la version OpenMP sur une machine de la salle 203 du CREMI (*odo*, 24 cœurs) en faisant varier le nombre de processeurs utilisés. Nous avons reproduit cette étape sur un serveur AMD (*boursouf*, 48 cœurs). Puis, nous avons observé la différence avec ces deux tests en faisant varier le placement des threads et de la mémoire ($GOMP\_CPU\_AFFINITY$).

Le schéma ci dessous présente le résultat avec et sans la variation de la variable $GOMP\_CPU\_AFFINITY$ sur la machine *odo*.

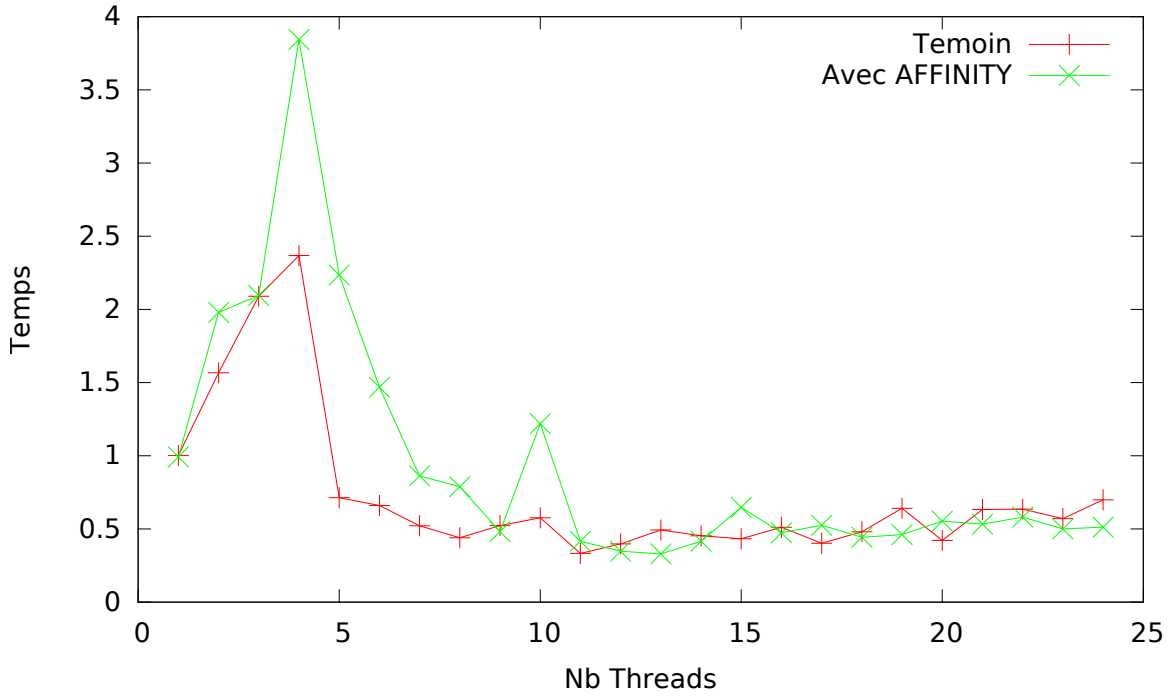

FIGURE 1 – Graphique présentant la différence entre le test sans (tracé rouge) et avec la variation de l'affinité (tracé vert) sur *odo*.

Le schéma ci dessous présente le résultat avec et sans la variation de la variable *GOMP_CPU_AFFINITY* sur *boursouf*.

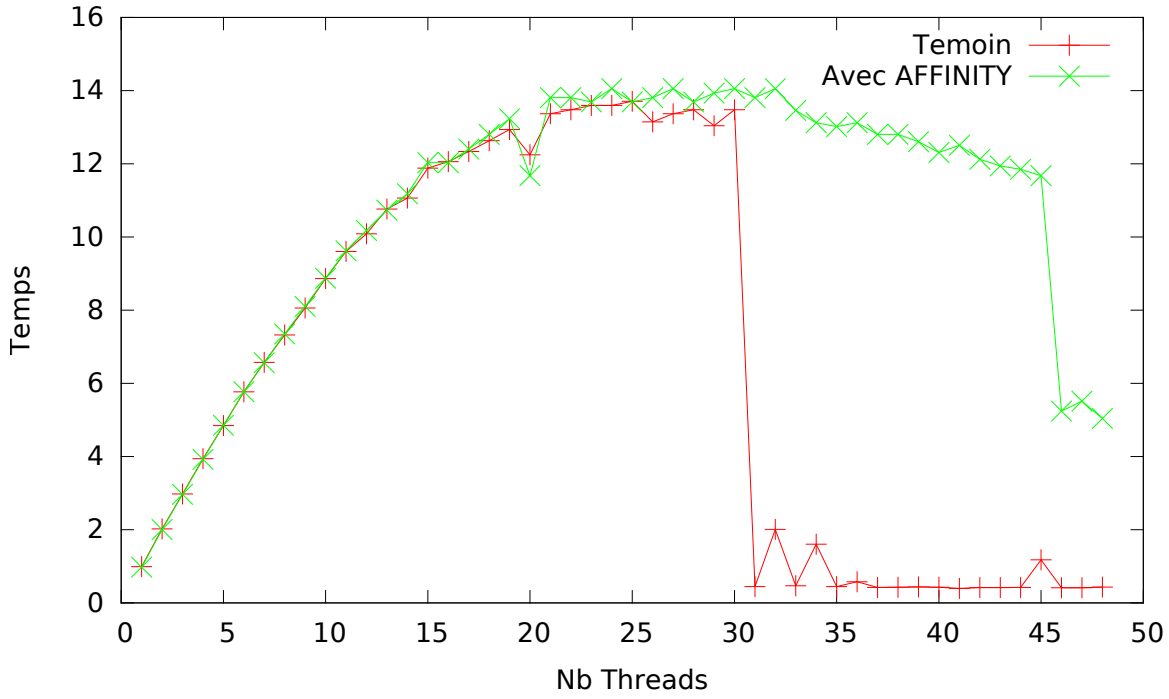

FIGURE 2 – Graphique présentant la différence entre le test sans (tracé rouge) et avec la variation de l'affinité (tracé vert) sur *boursouf*.

La deuxième étape consistait à comparer les exécutions en fonction des politiques de distributions d'indice. Le témoin restant le même que précédemment, les autres version sont respectivement les résultats d'une exécution avec une politque de distribution *STATIC*, *DYNAMIC* puis *GUIDED*.

Le schéma ci-dessous présente le résultat en changeant la politique de distribution sur *odo*.

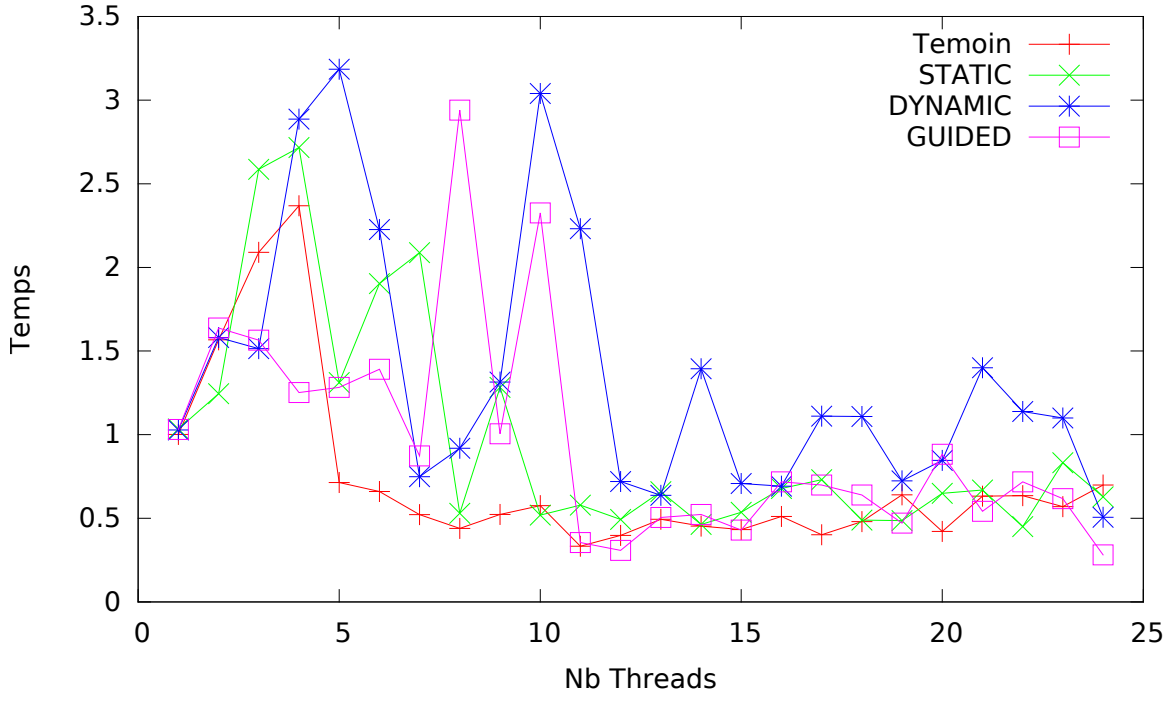

FIGURE 3 – Graphique présentant la différence entre le test sans (tracé rouge) et avec la variation de la politique de distribution sur *odo*. *STATIC* (tracé vert), *DYNAMIC* (tracé bleu), *GUIDED* (tracé violet).

Le schéma ci-dessous présente le résultat en changeant la politique de distribution sur *boursouf*.

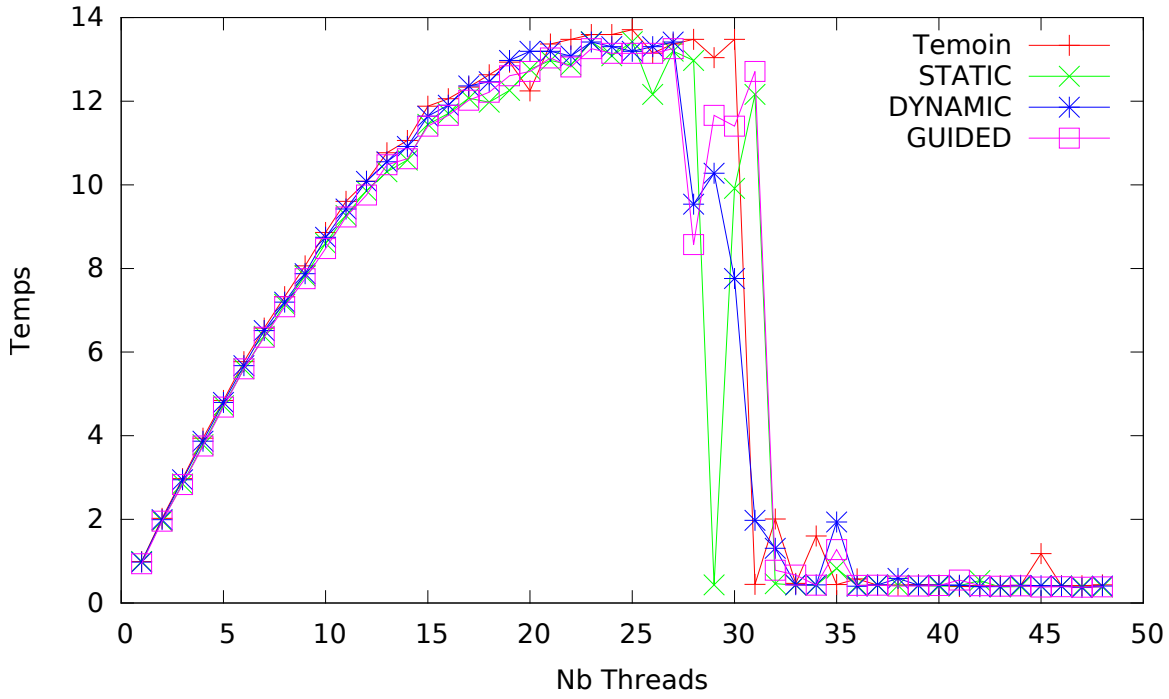

FIGURE 4 – Graphique présentant la différence entre le test sans (tracé rouge) et avec la variation de la politique de distribution sur *boursouf*. *STATIC* (tracé vert), *DYNAMIC* (tracé bleu), *GUIDED* (tracé violet).

La dernière étape consistait à faire varier le domaine initial. Ainsi nous avons réalisé les tests avec 1000 atomes, la configuration *choc2.conf* puis *choc4.conf*, consécutivement sur *odo* et sur *boursouf*.

Le schéma ci-dessous présente le résultat en changeant le domaine initial sur *odo*.

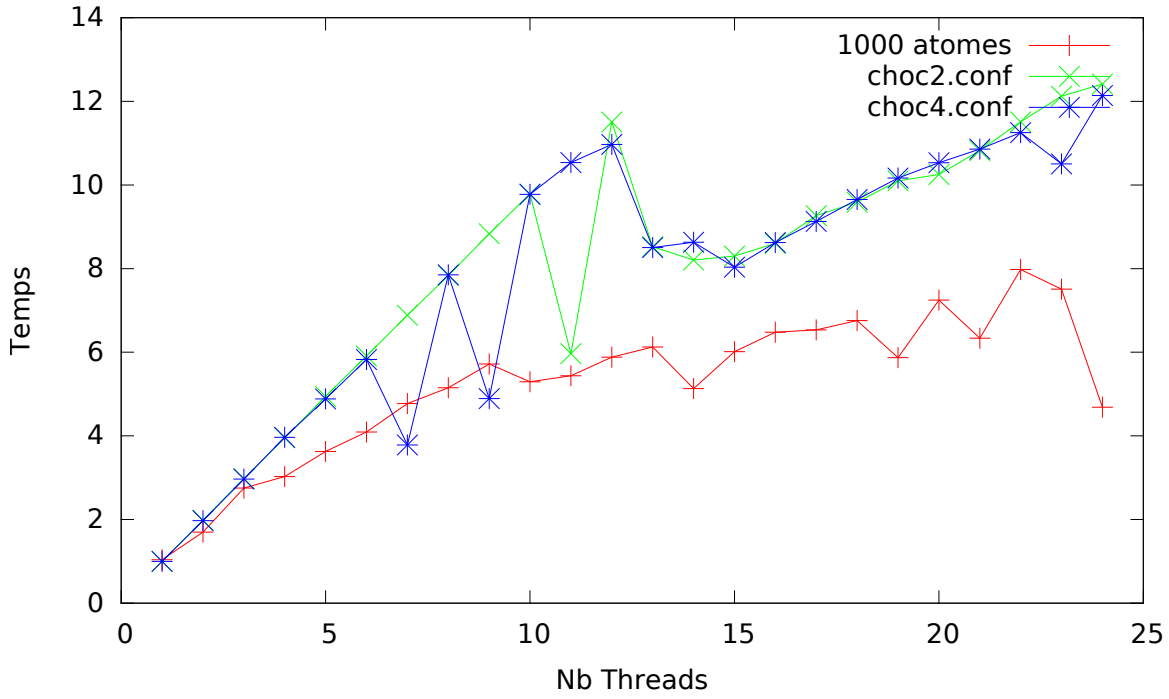

FIGURE 5 – Graphique présentant la différence entre le test avec 1000 atomes (tracé rouge), avec la configuration de départ *choc2* (tracé vert) et avec la configuration de départ *choc4* (tracé bleu) sur *odo*.

Le schéma ci-dessous présente le résultat en changeant le domaine initial sur *odo*.

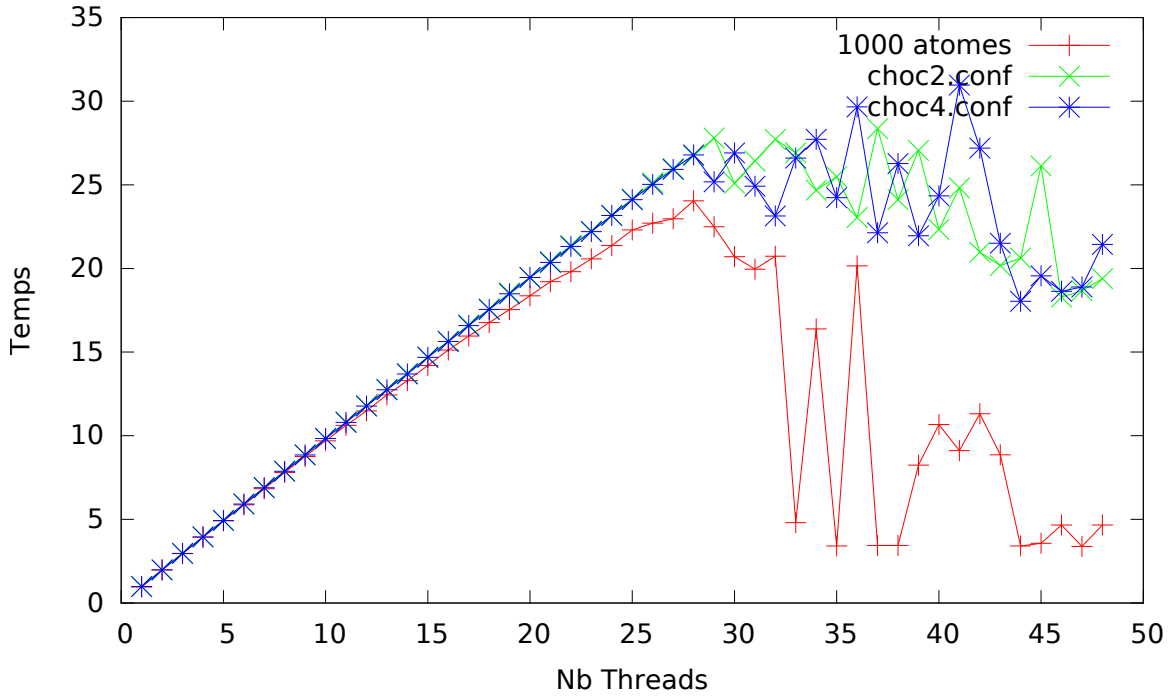

FIGURE 6 – Graphique présentant la différence entre le test avec 1000 atomes (tracé rouge), avec la configuration de départ *choc2* (tracé vert) et avec la configuration de départ *choc4* (tracé bleu) sur *boursouf*.

3 Analyse des observations

D'après le premier graphique, on observe de meilleurs résultats en utilisant la variable *GOMP_CPU_AFFINITY* et en faisant varier le nombre de threads utilisé tout au long de la simulation.

Cela est plus remarquable sur le serveur AMD *boursouf*, où l'accélération au lieu de diminuer drastiquement au 31e thread, reste plus ou moins linéaire jusqu'au 46e thread. Après cela elle commence à chuter.

Les résultats obtenus en faisant varier la politique de distribution d'indice sur *odo* sont peu cohérents mais il semblerait que *DYNAMIC* donne les meilleurs résultats, devant *GUIDED* et *STATIC*.

Sur *boursouf* les 4 courbes sont assez similaires. jusqu'à 26 threads elles croissent toutes de la même manière jusqu'a atteindre une accéleration 13 à 14 fois supérieure à la version séquentielle. Du 32e thread jusqu'au dernier les courbes suivent de nouveau le même comportement et affichent de moins bonnes performances que la version séquentielle. C'est au passage entre une éxecution avec 26 threads et avec 32 threads que l'on observe des différences. La version sans politique de distribution d'indice chute de performance au 30e thread et passe d'une accélération de près de 14 à des performances moins bonnes qu'en séquentiel. La version en *STATIC* chute dès le 27e thread, mais refait un pic a une accélération de 12 avant de rechute au 31ethread. Les perfommance de la version en *DYNAMIC* baissent régulièrement entre le 26e et le 32e thread. Il semblerait donc que les meilleurs performances soient réalisé par la version utilisant la politique de distribution *GUIDED*, puisque les perfomances de celle-ci diminuent en formant une marchant entre le 26e et le 32e thread, où elles stagnent avec une accélération de 8 à 12, avant de rejoindre le comportement des autres simulations.

Pour la dernière étape, nous avons aussi séparé les résultats obtenus sur *odo* et *boursouf*. Sur *odo* les trois cas de variation du domaine initial, 1000 atomes, configuration *choc2* et *choc4*, donnent des résultats assez similaire. Les performances augmentent de manière plus ou moins régulière jusqu'a l'utilisation de 13 threads, même si cette augmentation est en dents de scie avec les 2 configurations à partir de l'utilisation du 6^{e} thread. Puis on observe une faible de chute de performance avec que celle-ci ne recommence à croître. Alors que les performances se stabilisent voir commencent à diminuer sur la fin avec un domain initial de 1000 atomes, les autres simulation semble continuer de gagner en performance.

Sur *boursouf*, le resultat est plus ou moins proche. Les trois simulations observent la même accélération jusqu'au 27^{e} thread. C'est à cette étape là que la première simulation perd en efficacité et chute de manière assez irrégulière vers une accélération proche de 5 avant d'y stagner à partir de l'utilisation de 44 threads. Les deux autres simulation atteignent un palier à partir de 27 threads proche d'une accélération de 25. Elles oscillent ensuite entre une accélération de 20 et de 25 jusqu'à l'utilisation du 48^{e}thread.