FUSIONner, PHIer, ARMer, ATOMiser... Comment GPU faire cela ?

Emmanuel Quemener

FUSIONner, PHIer, ARMer, ATOMiser... Comment GPU faire cela ?

Emmanuel Quémener
Centre Blaise Pascal / ENS-Lyon
46, allée d'Italie
69364 Lyon cedex 07

Résumé

*L'hégémonie du x86 est-elle en train de vaciller ? Nous avions été trop habitués, ces dernières années, au règne sans partage du x86, dupliquant ses cœurs après ses bits, inondant des ultra-portables aux serveurs les plus imposants. Cependant, les détournements de technologies jadis de « niches » (l'accélération graphique et la gestion de l'énergie) ont changé la donne : le TOP 500 comporte ainsi une bonne dizaine de clusters de PC associés à des cartes graphiques équipées de **GPU** Nvidia ou AMD ; les processeurs embarqués dans nos ordiphones de technologie **ARM** commencent timidement à investir le monde des serveurs. Pour contrer cette menace, les acteurs majeurs du monde x86, Intel et AMD, recyclent de vieilles recettes : **Phi**, **Atom** et **Fusion** en sont la matérialisation. Ces détournements de technologies vont-ils bouleverser nos serveurs, nos postes de travail dans les années à venir ? Nous nous proposons d'y répondre sommairement, partant de « vraies » machines, en comparant leurs performances de la jungle du parallélisme massif à quelques applications plus classiques. Nous découvrirons que ces technologies offrent (certes plus sur le papier que dans le rack) débauche de puissance ou sobriété de chameau, mais que leurs utilisations s'avèrent délicates : les coûts d'entrée (et de sortie) vers de nouveaux langages ou une nouvelle architecture sont loin d'être négligeables, leur exploitation toujours gourmande en personnels « affûtés »... Mais la liberté face aux constructeurs n'est-elle pas à ce prix ?*

Mots-clefs

GPU, ARM, Phi, Atom, Fusion, BYTEmark, HPC, OpenCL, CUDA, TDP, SoC, BLAS, LAPACK

1 Introduction

Comment des architectures aussi spécialisées pour le graphique ou la basse consommation peuvent, aujourd'hui, offrir presque 3.5 Tflops sur une carte unique ou prétendre une puissance de plusieurs dizaines de GigaFlops dans quelques Watts ? Nous verrons ainsi que toutes partagent un point commun : une multiplication du nombre d'unités de traitement de quelques dizaines à plusieurs milliers de cœurs. Cependant, une « *green attitude* » s'associe maintenant à la performance pure avec un indicateur de marque : combien de GigaFlops par Watt pouvons-nous espérer ?

2 Architectures spécialisées

Un point commun rapproche les accélérateurs de calcul des processeurs basse consommation (que nous qualifierons de « légers » dans la suite) et tient dans l'acronyme anglais *KISS* (*Keep It Simple Stupid*). Ainsi, une architecture « simple », voire simpliste par opposition à « compliquée » s'avère, dans bien des cas, la plus efficace.

2.1 Les accélérateurs, graphiques et autres...

2.1.1 Du GPU au GPGPU : de puces hyper-spécialisées aux modèles de programmation généraux

Le GPU (pour *Graphics Processing Unit*) originel avait une unique fonction : il permettait un affichage sur un écran. Lorsque la 3D s'est développée, ce passage 3D vers 2D a évolué du « lancer de rayon » vers la « rastérisation »[1]: cette technique se compose presque exclusivement de multiplications matricielles, lesquelles ont un avantage majeur ; elles

se parallélisent très bien ! Les « processeurs de rendu » permettant de réaliser ces opérations sont passés de quelques unités (3 dans une 3Dfx Voodoo de 1996) à presque 3000 (dans une Nvidia GTX Titan de 2013), pour une puissance brute 100000 fois supérieure (en comparaison d'un facteur 50 à 500 pour les CPU sur la même période). C'est donc indubitablement dans le GPU que se trouve le concentré de puissance de traitement, à la condition expresse de pouvoir lui distribuer efficacement le travail à réaliser.

Pour exploiter ces GPU, chaque fabricant y est allé de son API, pour ses propres matériels (ou logiciels). L'exploitation de ces ressources pour des usages plus généralistes s'est développée dès 2002 avec Cg de Nvidia puis rationalisée avec CUDA[2] en 2007. Réponse initiée par Apple, OpenCL[3] est arrivée dès 2009 sur MacOSX. Quatre implémentations, pour GPU et CPU, venant de Nvidia, AMD, Intel et une Open Source coexistent.

2.1.2 Le Xeon Phi ou le recyclage d'un vieux Pentium

Lorsque fin 2008, Nvidia sort sa première carte Tesla GPGPU, Intel se trouve face à un concurrent sérieux dans le domaine du calcul haute performance. Après son échec du Larrebee fin 2009, Intel développe le MIC (pour Many Integrated Cores) : la récupération des vieux P54C subsiste mais plus massivement intégrés, les registres vectoriels sont étendus. L'actuel premier (d'octobre 2013) du TOP 500 Tianhe-2 en est équipé : chaque Xeon Phi offre 1 Tflops.

Selon Intel, l'avantage du MIC est d'exécuter directement les « vieux » codes : si cet argument marketing est vrai, sans une analyse et l'utilisation massive des outils d'investigation et de développement Intel, difficile d'exploiter la puissance brute de cet accélérateur.

2.1.3 De la pertinence de la règle de trois...

Avant d'aborder l'architecture de ces processeurs « légers », reprenons un petit cours d'électronique : la puissance électrique consommée par un composant électronique s'exprime par le produit entre sa capacitance, sa tension d'alimentation élevée au carré et sa fréquence. Cependant, les constructeurs livrent rarement la capacitance de leurs équipements mais se bornent bien souvent à la seule TDP[4], la *Thermal Design Power* ou l'enveloppe thermique.

2.1.4 De la course aux GHz à la gestion de la consommation

La course aux GHz sur les architectures x86 entamée par Intel au début du 21ème siècle s'est rapidement retrouvée dans une impasse, aux frontières des 4 GHz, par la consommation du processeur et la capacité du radiateur à en évacuer l'effet Joule. Les constructeurs toujours avides de plus de performances ont exploré d'autres voies, et le parallélisme en est une : les cœurs se sont multipliés dès fin 2005. La gestion de l'énergie s'est focalisée d'abord sur le contrôle de la fréquence (et son adaptation à la demande) essentiellement dans le monde de la mobilité. L'association des deux a invité les constructeurs à proposer des stratégies (ou plutôt des tactiques) complémentaires comme le contrôle de l'activité des unités de traitement : l'arrêt et le redémarrage de cœurs exigent simplement un appel système. Les technologies embarquées, dessinées pour leur sobriété, partagent-elles ces simples approches ? Nous allons le découvrir.

2.1.5 ARM : le « top » de l'architecture après le « flop » de la machine grand public

ARM[5] est l'acronyme de *Acorn Risc Machin*e. Malgré l'échec commercial de Acorn avec son Archimedes fin des années 80, le succès d'ARM repose toujours sur cette expertise technologique développée ces années-là : ARM fournit du design aux concepteurs pour fabriquer leurs propres puces. Les fondeurs recouvrent toute la planète, même en France avec ST Microelectronics ! Samsung , Texas Instruments voire Nvidia en sont certainement les plus connus. Ces derniers intègrent à proximité dans le SoC[6] (*System On Chip*) d'autres contrôleurs, ceux du *North Bridge*, allant même parfois jusqu'au *South Bridge* avec sa vingtaine de contrôleurs !

Premier processeur RISC « de masse », l'ARM est peut-être moins efficace en nombre d'instructions, mais l'association une instruction par cycle et *pipeline* est redoutable face aux processeurs CISC. Grâce à son nombre réduit de transistors issu de cette simplicité, l'empreinte thermique ne dépasserait pas 3W pour un SoC complet.

2.1.6 Atom : le recyclage de « ce qui marche » selon Intel

Jeter un coup d'œil sur la liste pléthorique des processeurs estampillés Atom[7] par Intel invite instantanément à la prudence : Atom est une famille, une très grande famille[8]. Cependant, sa phylogénie micro-électronique repose sur du recyclage : l'architecture Bonnel reprend des concepts du Pentium Pro (conversion CISC/RISC des instructions), des éléments du Pentium 4 (*l'Hyperthreading*), du Core2 (64 bits, VT-x) mais peu de cache (pas de L3) : en somme, moins de transistors, moins de surface de silicium. À noter que les Atoms partagent généralement leur SoC avec d'autres circuits, notamment graphiques (souvent mauvais d'ailleurs...).

Il est difficile de suivre Intel dans sa stratégie d'évolution de l'Atom. Côté consommation électrique, Intel propose des SoC d'une sobriété déclarée impressionnante : de quelques Watts à la dizaine de Watts pour la totalité de sa gamme.

2.1.7 Fusion : la vieille idée du coprocesseur dans le processeur

Avec Fusion, AMD[9] rapproche, depuis 2011, CPU et GPU sur une même puce, la transformant en une architecture finalement assez proche des SoC que nous retrouvons chez les intégrateurs d'ARM ou Intel : l'idée est de placer une puce graphique récente (Radeon HD 6250 à HD 8400) au plus proche du processeur pour s'affranchir du goulot d'étranglement que représente le bus PCI Express.

Avec Fusion, AMD a plutôt visé le marché des portables que le marché des ordiphones ou des serveurs : la TDP associée à ces puces frôle la vingtaine de Watts (soit 10 fois plus que les « chameaux » à base d'Atom ou d'ARM).

3 Éléments de migration : les approches développeur & intégrateur

3.1 Vers les processeurs « légers » : des x86 et une architecture bien différente...

Si nous nous penchons sur les processeurs x86 et x86-64, nous restons dans le « monde » x86 : la migration se fera instantanément. Les binaires sont compatibles pour autant qu'ils aient été compilés avec des extensions génériques. En effet, les processeurs légers ne disposent pas de la pléthore de drapeaux relevés par le noyau Linux : 77 sur un i7-2860, 79 sur un E5-2670, 61 pour un Fusion E2-1800, 50 pour le N550, 49 pour un Z530. De plus, ces processeurs légers n'ont pas les fonctions vectorielles les plus récentes. La différence majeure viendra de la performance à l'exécution au sujet de laquelle nous reviendrons par la suite.

Si nous abordons l'exploitation des SoC à base d'ARM, les différences sont nombreuses. Commençons par les OS grand public supportés : Android (normal !), Windows 8 (plus récent) et certaines distributions Linux, les plus abouties étant la Debian et sa dérivée la plus répandue, Ubuntu. Ainsi, le principal risque, dans le monde ARM, vient de la capacité des éditeurs à fournir des versions de leurs outils sur cette plate-forme. Par exemple, sur la distribution Debian Wheezy, les nombres de paquets disponibles pour i386, amd64 et armel sont respectivement de 37359, 37269 et 35634. Ainsi, 95 % des paquets Debian disposent de leur version ARM : largement de quoi pouvoir programmer dans tous les langages de la Terre ! Pour finir, gardons à l'esprit qu'un éditeur, s'il veut toucher une large clientèle, ne peut ignorer complètement la mobilité et donc le monde Android : cette effervescence est un catalyseur pour le portage sur ces plates-formes.

En conclusion, le travail de migration sur des processeurs « légers » x86 se limitera souvent à de l'optimisation sur les ressources. Sur les processeurs ARM, les outils sont assez largement répandus et la masse critique de développeurs Android suffisante pour espérer combler rapidement les manques en manière de portage.

3.2 Vers les accélérateurs, entre développement & intégration

3.2.1 L'approche « intégrateur » : la bonne ?

L'approche « intégrateur » se propose d'utiliser l'accélérateur comme un « composant sur étagère » (*Component On The Shelf*) et donc de modifier à minima son code.

Paradoxalement, les modifications seront moins nombreuses si des librairies externes sont largement exploitées. En algèbre linéaire, les librairies BLAS[10] et LAPACK[11] sont incontournables : il existe donc les versions optimisées pour tous les parallélismes possibles, y compris pour exploiter les GPU. De même pour les transformées de Fourier rapide. Une attention particulière doit cependant être portée aux limitations matérielles des composants !

Dans les faits, si l'usage de l'intégration permet d'obtenir des gains très substantiels en vitesse d'exécution, il apparaît rapidement que la principale difficulté réside dans la communication entre l'hôte et la carte : l'espace mémoire n'est pas complètement fusionné et la maîtrise de la localité mémoire est un gage de performances. Il convient donc, en amont, de définir ses objectifs : quel temps pouvons-nous passer à optimiser et pour quel gain en exécution ?

Dans notre expérimentation, nous allons présenter un exemple où cela se passe « bien » pour l'accélérateur, la multiplication matricielle avec les fameuses commandes SGEMM et DGEMM.

3.2.2 L'approche développeur : la brute ?

L'approche précédente consistait à appliquer le « principe de perturbation » à son programme : ajouter avec parcimonie quelques lignes par des directives pour détourner les appels de librairies classiques vers les accélérateurs. Ce n'est parfois pas suffisant et il faut savoir revenir à la base, et la base commence par l'apprentissage. En voici une approche en quatre étapes. Tout d'abord, la première étape est de « prototyper » sa simulation, son traitement ou sa visualisation dans un langage haut niveau. Dans le domaine scientifique, Python est largement répandu, essentiellement autour de l'usage de Numpy et de Scipy : en faire la promotion est hors sujet mais Python intègre tout ce qu'il faut en matière de parallélisme, y compris pour l'utilisation des GPU avec PyOpenCL[12] et PyCUDA[13] (merci à Andreas Klöckner) : supposons que notre programme est écrit en Python et fonctionne. Puis, la seconde étape est de « profiler » l'exécution pour mieux appréhender son déroulement : Python dispose là encore des outils idoines. Les parties coûteuses en temps sont alors identifiées. Ensuite, la troisième étape est une traduction des parties coûteuses en OpenCL ou CUDA : cela semble une opération assez simple : convertir son code de Python en C99 n'est pas difficile mais segmenter son code en petites tâches indépendantes est plus délicat. Enfin, la quatrième étape, la plus pénible à notre avis, est de bien initialiser les appels aux périphériques, de bien vérifier le *cast* de chacun des espaces de mémoire partagés entre le code Python et le langage OpenCL ou CUDA, de bien échanger les données, libérer les périphériques : trop de précipitation et le programme tournera, mais mal...

Voici quelques étapes que nous avons expérimentées et c'est sur cette base que seront présentées les évaluations de GPU.

3.2.3 L'approche minimaliste : la truande ?

Une troisième voie existe : elle propose de directement confier à des programmes la parallélisation du code. Des éditeurs comme CAPS (avec HMPP) ou PGI (avec Cuda Fortran Compiler) proposent des « facilitateurs » de portage basés sur le pointage de zones à paralléliser avec des directives comparables à OpenMP. Cela fonctionne, certes, mais les tarifs et l'addiction à ces outils ont clairement représenté un frein.

Une autre voie se propose de directement transformer du code C vers OpenCL, CUDA (ou OpenMP). Un outil Open Source est extraordinairement efficace sur des exemples simples : par4all[14]. Pour résumer, par4all va détricoter les boucles de votre programme pour trouver ce qui peut être parallélisé : il permet, sur des cas simplistes, d'obtenir la limite théorique de parallélisation, pourvu que les allocations de mémoire soient statiques ! Mais, dès que le programme perd en simplicité, son efficacité est limitée voire nulle (impossible d'obtenir le moindre exécutable). C'est cependant un outil pédagogique intéressant pour qui « veut » voir comment convertir un programme simple en code GPU.

4 Expérimentation

De manière à pouvoir comparer les matériels et uniquement les matériels, nous nous devions de proposer une plate-forme système et logicielle homogène. Notre choix s'est portée sur Linux Debian version 7.0 (la Wheezy), disponible aussi bien sous les architectures x86 (i386), que x86-64 (amd64) et ARM (armel). Toute la difficulté a donc été de choisir des équipements qui puissent héberger ce système (à défaut du noyau) complété de tout l'environnement de développement nécessaire à la compilation et à l'exécution de nos outils.

4.1 De l'objectif à la performance, de la performance au banc d'essai

Nous avons, dans le résumé, évoqué le « détournement » possible de ces architectures (accélérateurs et processeurs « légers ») vers d'autres utilisations dans notre écosystème : l'éducation/recherche. Pour les accélérateurs, clairement, il s'agit d'accélérer le traitement informatique de certaines tâches en les parallélisant à l'extrême. Pour les processeurs légers, l'objectif est de catalyser la recherche ou l'apprentissage en mettant à disposition des environnements de travail : des « paillasses numériques » complètes sur lesquelles les cœurs de métier vont pouvoir se « lâcher », expérimenter à l'envi sur des machines COMOD (*Compute On My Own Device*) proposées par exemple par SIDUS[15] (*Single Instance Distributing Universal System*) : un utilisateur exploite son propre poste de travail en y intégrant en quelques secondes un environnement numérique complet, comparable à un poste de formation ou un nœud de cluster de calcul.

Côté processeurs « légers », la performance sera donc évaluée suivant deux axes : une comparaison sur des traitements séquentiels avec Nbench[16] puis une comparaison sur des traitements parallèles avec PiMC. Côté « accélérateurs », ce seront des tests à base de multiplications matricielles et du même programme PiMC, porté en OpenCL afin de disposer exactement du même code exécuté sur les différentes plates-formes.

4.1.1 La forêt des accélérateurs graphiques face à quelques processeurs récents

Pour les accélérateurs, malheureusement, nous nous bornerons aux GPU, aux GPGPU des deux principaux fabricants de cartes vidéo, Nvidia et AMD : cinq cartes AMD/ATI : des cartes d'entrée de gamme HD 6450 et HD 6670, des cartes de *gamer,* une assez ancienne HD 5850, une plus récente HD 7970, et une « professionnelle », la FirePro v5900, neuf cartes GPU Nvidia : de petites cartes GT 220, GT430, et GT 620, des cartes de *gamers* GTX 260, GTX 680, GTX 690, GTX Titan, et des cartes « professionnelles » Quadro 2000M et 4000), deux cartes GPGPU Tesla M2090 et Tesla K20m.

Ces cartes seront opposées à quatre processeurs dont trois Intel Sandy Bridge (portable i7 à 4 cœurs i7-2860, station E5-2620 à 6 cœurs, serveur E5-2670 à 2x8 cœurs) et un Intel Westmere (serveur X5650 à 2x6 cœurs).

4.1.2 ARM, Atom et Fusion ou de la difficulté d'un boot Debian natif

Pour les processeurs ARM, la quête a été difficile : fin 2012, peu de boîtiers à base d'ARM de développement : seul de Rasperry Pi, « trop » léger, était largement diffusé. Il a fallu se tourner vers les hôtes naturels de ces puces : les ordiphones et les tablettes tactiles. Comme processeur, notre choix s'est porté sur un Cortex A9 quadri-cœur, équipant un Galaxy S3 de Samsung (SoC Exynos 4412) et un TF700 de Asus (SoC Tegra 3) ! L'installation d'une Debian native sur ces plates-formes est, avouons-le, quasi impossible. Ces deux plates-formes ont été rejointes en mi septembre 2013 par des équipements entre le kit de développement et le nano-serveur : un Odroid-U2 (un Cortex A9 dans un SoC Exynos 4412) et un Odroid-XU (un Cortex A15 dans un SoC Exynos 5410). Notons que ce dernier est une architecture Big.LITTLE. L'installation de l'OS des ARM est bien différente. Pour les Odroid, aucun souci : une communauté très réactive fournit des images complètes démarrant sur carte SD. Pour l'ordiphone et la tablette, nous avons à la suite : un break du système originel, les installations d'un gestionnaire de boot et d'un Android CyanogenMod supérieur à 10, la création d'une racine Debian ARM et un binding serré entre le CyanogenMod et le chroot Debian : c'est du hack !

Pour les processeurs Atom, nous avions a disposition deux NetBook : un Dell Mini 1010 équipé d'un Atom Z530 déjà ancien (fin 2008) et un Clevo M1111 équipé d'un N550 : les deux Netbook, équipés d'un boot en réseau, ont démarré instantanément SIDUS. La recherche d'une processeur plus récent (un Z2760) a levé un problème pour tourner au fiasco : un choix s'est porté par défaut sur un Acer W510 équipé nativement d'un Windows 8, et d'un Secure Boot impossible à désactiver... A notre grand regret, les Atoms testés se limiteront donc à des processeurs anciens.

Pour les processeurs Fusion, la quête a été plus fructueuse, même si elle a frisé la berezina du Z2760. Notre choix s'est porté par défaut sur l'Asus X55U, un portable *cheap*. Équipé d'un processeur E2-1800, composé d'un CPU bi-cœur à 1.7 GHz et un GPU Radeon HD 7340, cette machine est livrée pré-installée avec Windows 8, donc Secure Boot... Côté installation, nous n'évoquerons pas les heures passées dans les modifications de BIOS et l'installation standard...

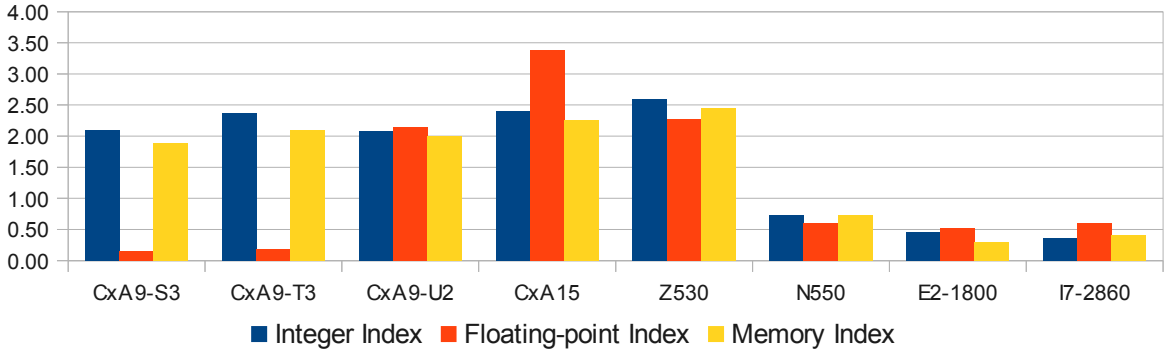
4.2 Processeurs légers : fonctionnement en mode séquentiel

De manière à comparer les processeurs « légers » entre eux, nous disposons donc des équipements suivants : un Samsung Galaxy S3 (CxA9-S3), un Asus TF700 (CxA9-T3), un Odroid U2 (CxA9-U2), un Odroid XU (CxA15), un Asus x55u (Fusion E2-1800), un Dell Mini 1010 (Atom Z530), un Clevo M1110 (Atom E550). Comme processeur de comparaison, nous avons choisi celui équipant un portable récent, le Sandy Bridge Intel équipant un Dell Precision M4600(i7-2860) : Intel fournissant ses processeurs d'abord pour les gammes grand public, la puissance « atomique » du i7-2760 sera comparable à celle d'un Sandy Bridge de serveur, avec certes moins de cache L3.

Le test que nous allons exploiter pour évaluer ces processeurs en mode séquentiel est Nbench[16], issu de BYTEmark, un test que le magazine BYTE utilisait il y a une bonne quinzaine d'années pour évaluer les processeurs. Son avantage est qu'il est assez polyvalent : il permet de comparer les modes de fonctionnement, notamment entre les domaines purement entier ou virgule flottante, mais uniquement sur un seul cœur. Les résultats sont présentés avec pour référence l'AMD K6 cadencé à 233 MHz. Chaque plate-forme a compilé sa propre version de Nbench, dans son environnement Debian Wheezy, avec un compilateur GCC 4.7 et un niveau d'optimisation en O3.

Les résultats préliminaires montrent sans la moindre ambiguïté la performance d'un processeur traditionnel i7-2860 par rapport aux processeurs légers au point que toute la métrique en est écrasée. L'essentiel présente une accélération entre 40 et 70 fois, ce qui ne paraît pas extraordinaire compte-tenu que nous avons, au moins, une fréquence 10 fois supérieure entre le K6-233 et le i7-2860. Côté processeurs légers, l'examen est plus nuancé : les Atom Z530 et N550 flirtent avec la barre des 10, le Fusion E2-1800 la dépasse. C'est sur les résultats des différents ARM que va se porter plus attentivement notre analyse : les Cortex A9, intégrés aux Galaxy S3 et Tegra 3, sont moins puissants en flottant

qu'un processeur de 17 ans leur aîné. Pourtant, l'Odroid-U2, équipé a priori exactement du même SoC que le Galaxy S3 (un Exynos 4412 avec un Cortex A9 4 cœurs) est 20 fois plus rapide... Pourtant, même compilateur, même noyau 3.0, mêmes sets d'instructions ! Ainsi, le Cortex A9 de l'Odroid-U2 voire le Cortex A15 de l'Odroid-XU présente des performances très honorables en virgule flottante (second du test).



Ainsi, il convient de normaliser ces valeurs par rapport à deux autres paramètres, la fréquence de processeurs d'une part et la TDP d'autre part. La renormalisation des résultats a pour effet immédiat de réduire les prétentions du i7, 15 fois plus gourmand que les ARM. L'Atom Z530 redevient compétitif, mais pas le N550 trop gourmand, tout comme le Fusion, devenant bon dernier. Ainsi, les performances des ARM sont particulièrement homogènes sur les index mémoire et entier. Finalement, le Cortex A15 est une très bonne surprise qu'il nous faudra confirmer avec le Cortex A57, sa version 64 bits.

4.3 Processeurs légers : fonctionnement en mode parallèle

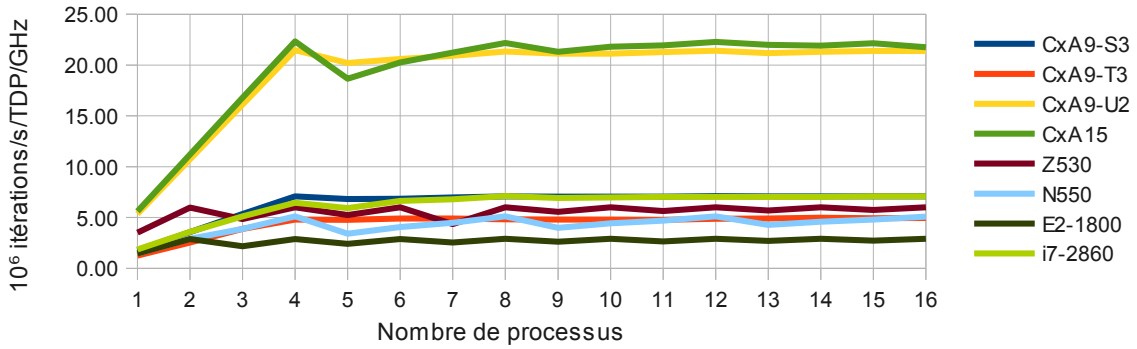
Tous les processeurs de notre banc d'essai ont plusieurs cœurs de CPU, ne serait-ce que virtuels par *Hyperthreading* : de 4 à 8 pour les ARM (enfin 4 permanents), de 2 à 4 (logiques) pour les Atom, 2 pour le Fusion et 8 logiques pour le i7. Maintenant, évaluons leur capacité à distribuer des calculs sur certains d'entre eux.

Nous restons sur un parallélisme « gros grain » laissant la distribution exploitant les registres vectoriels à l'efficacité du compilateur. Ces parallélismes se basent sur : la gestion des processus par les Pthreads, la distribution des boucles par OpenMP, et la distribution par passage de messages avec MPI. Pour évaluer cette capacité, nous utilisons un programme très simple de Monte Carlo basé sur l'exploration d'un domaine par des nombres aléatoires pour le calcul de Pi : le programme comprend une trentaine d'opérations par itération. Le parallélisme gros grain sur ce programme consiste alors à distribuer équitablement le nombre d'itérations entre les différents processus : ces processus prennent la forme de processus MPI, OpenMP ou de Pthreads.

L'examen des résultats pour les 8 architectures (dont 3 Cortex A9, 1 Cortex A15, 2 Atom, 1 Fusion et 1 i7) nous permet de tirer quelques conclusions. Tout d'abord, les résultats sont comparables en MPI, OpenMP et Pthreads, mais MPI présente une moins bonne efficacité de parallélisation que les deux autres approches, notamment sur les processeurs Tegra. Puis, le i7 reste au moins 8 fois plus rapide que tous ses concurrents. Ensuite, les Cortex A9 et Cortex A15 d'Odroid sont seconds dès que la sollicitation des cœurs est importante. De plus, l'*Hyperthreading* des Atoms est particulièrement efficace : presque 85 % alors que le i7 ne l'a présente même pas sur ses 4 cœurs réels. Enfin, la scalabilité est parfaite pour les ARM en OpenMP et Pthreads, mais seulement de 85 % en MPI.

De manière à disposer de valeurs comparables, nous allons, une fois encore, renormaliser par rapport à la TDP et la fréquence. Tout d'abord, les Cortex A9 et Cortex A15 de Odroid écrasent la concurrence d'un facteur 3 mais restent équivalents, le Xeon i7 est même surclassé par le Cortex A9 du Galaxy S3 (pourtant calamiteux en virgule flottante). Puis, l'Atom Z530 (le 32 bits) résiste bien mais moins que son successeur 64 bits, le N550. Enfin, un décrochage entre 5 et 8 processus est bien perceptible ici, surtout sur le Cortex A15.

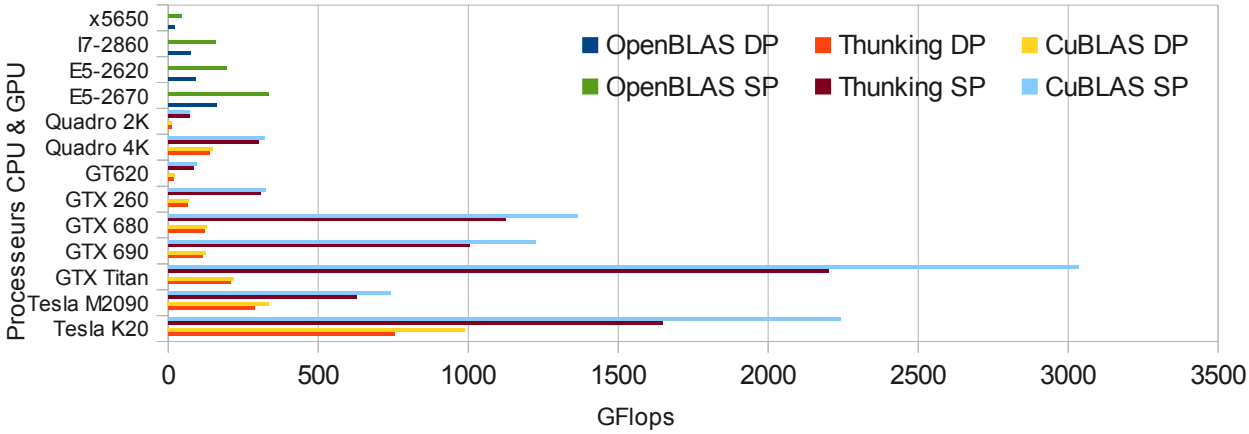
Quelles conclusions pouvons-nous tirer de ces analyses de processeurs « légers » ? Les processeurs ARM présentent, relativement à leur consommation, une capacité de traitement très supérieure aux autres. Les processeurs Atom, efficaces en *Hyperthreading*, résistent avec leur TDP faible. Le Fusion demeure pour l'heure complètement distancé. Pour finir, ces processeurs peuvent supporter de belles charges sans s'effondrer. Que le marché du *Cloud* s'y intéresse n'est donc pas une surprise !



4.4 Accélérateurs : parallélisme grain fin ou gros grain ?

4.4.1 Grain fin sur des opérations matricielles « qui marchent »

Comme banc de test, nous avons 9 cartes Nvidia : des cartes GPGPU Tesla (K20 récente et M2090 plus ancienne), des cartes de station de travail (GTX Titan récente, GTX 690 et GTX680 passées, GTX 260 ancienne), des cartes professionnelles Quadro. La GT620 est une carte à moins de 30€. Ces cartes seront exploitées avec CuBLAS, en utilisant soit les fonctions Thunking (gestion des E/S mémoire par la librairie elle-même), soit standard (gestion de E/S par le le programmeur). Elles sont comparées à différentes machines ayant des processeurs essentiellement Intel Sandy Bridge (E5-2670, E5-2620, I7-2860) et Westmere (X5650) ayant respectivement 16, 6, 4 et 12 cœurs par station, avec l'implémentation très efficace OpenBLAS. Les tests sont menés avec l'appel des fonctions SGEMM (en simple précision et notée SP) et DGEMM (double précision et notée DP). Les éléments de comparaison sont les valeurs maximales en GFlops obtenues à partir de multiplication de matrices allant de 1000² à 16000².



Ce qui frappe le plus, c'est la différence d'écart entre simple et double précision. Pour les cartes Tesla, cet écart va du simple au double, ce qui est comparable à ce qui est mesuré sur un processeur standard. Cependant, sur les GTX et les Quadro de portable, le calcul en simple est entre 3 et 15 fois plus lent (que pour la Titan). D'autre part, le GPU n'est supérieur au CPU récent que pour les cartes Tesla ou GTX récentes développant plus de 100 GFlops en double précision. Quelques conclusions s'imposent : d'un côté, la puissance existe, mais demeure essentiellement concentrée sur de la simple précision. De l'autre, une carte GPGPU récente est équivalente à 6 nœuds récents en DP, une GTX récente à 10 nœuds en SP.

Nous n'avons pas détaillé les autres contraintes liées au GPU, essentiellement dues à sa mémoire limitée (moins d'un dixième de la RAM typique d'un nœud), ni sur les petites matrices sur lesquelles les GPU sont peu efficaces. Ainsi, le GPU développe une puissance colossale, mais pas dans toutes les situations.

4.4.2 Gros grain sur Pi par Monte Carlo

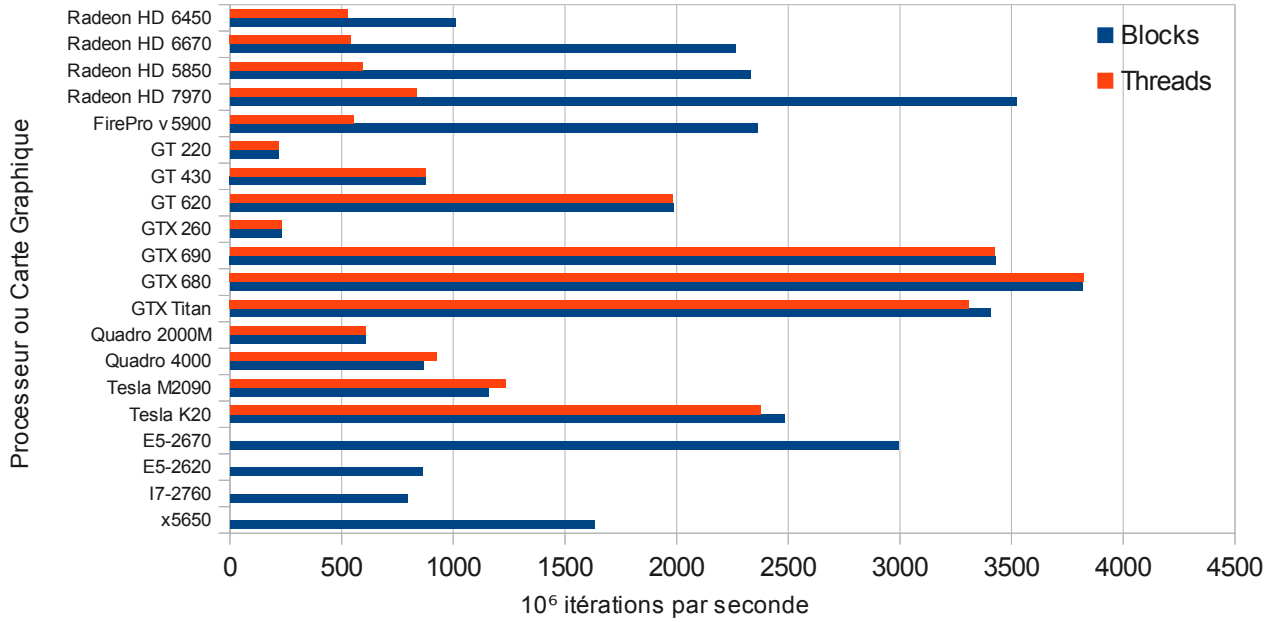
Le calcul précédent montrait la capacité d'un GPU à multiplier rapidement les matrices, essentiellement en simple précision. Est-il aussi efficace si nous l'utilisons pour des programmes fortement parallélisés, mais à gros grain (de grosses tâches séquentielles bien séparées) ?

Nous avons donc investigué sur 16 GPGPU/GPU, avec 11 Nvidia (2 GPGPU Tesla, 4 cartes GTX, 4 « petits » gabarits,) et 5 AMD/ATI, le tout en OpenCL. Ces résultats ont été comparés avec 4 processeurs différents (les mêmes que au

4.4.1). Nous avons exploré en OpenCL de 1 à 1024 processus parallèles suivant les méthodes par *blocks* ou par *threads*.Quels résultats sur ces 20 tests ? Là encore, son lot de surprises. Tout d'abord, les cartes GTX récentes s'en sortent haut la main, mais ce n'est pas la plus récente la meilleure, ni même la plus performante sur le papier : la GTX 680 est première ! Puis, seuls les 16 cœurs de 2 E5-2670 parviennent à rivaliser avec les GPU récents, mais restent inférieurs aux GTX et à la Radeon HD récentes. Ensuite, les modèles chers de GPGPU Nvidia Tesla M2090 et K20m sont en retrait significatif face aux GTX récentes. Enfin, les modèles AMD/ATI n'exécutent pas plus de 256 *Threads* :c'est par *Blocks* qu'ils s'en sortent honorablement.

Cependant, derrière ces résultats manifestement très encourageants se dissimule une information capitale : si nous prenons le GPU le plus puissant de notre test (GTX 680) et la configuration CPU la plus rapide (2 E5-2670 totalisant 16 cœurs), ce n'est qu'en parallélisant sur plus de 700 processus que le GPU supplante le CPU. Si nous regardons pour un seul et unique processus, le GPU est presque 30 fois plus lent !

Ainsi, à partir de la loi d'Amdahl, dans le cas théorique précédent (1 cœur de GTX680 30 fois plus lent que 1 cœur E5-2670), avec 16 cœurs de CPU, nous devons disposer d'un facteur d'accélération de 480 (30 fois 16). Avec 1000 *Threads*, le taux de parallélisation doit être d'au moins 0.999 : tous les codes ne peuvent être parallélisés de la sorte !



5 Retour aux applications

5.1 L'accélérateur : incontournable en HPC, timide sur le poste, et ailleurs ?

C'est dans le HPC que les GPU sont devenus des GPGPU : ils ont rapidement offert des premières places au TOP 500. Derrière la profusion d'articles vantant des facteurs d'accélération de l'ordre de la centaine, la réalité opérationnelle est nettement plus nuancée. Cependant, la mise à disposition de librairies « compatibles » avec les canons du calcul scientifique (BLAS, LAPACK, FFTw, …) permet, avec une approche d'intégrateur, d'obtenir des gains substantiels : nous l'avons expérimenté avec succès, au Centre Blaise Pascal, sur les programmes BigDFT, Lammps et Abinit, exploitant Cuda, CuBLAS, CuFFT, MAGMA. Les accélérations atteignent rarement celles présentées dans les articles (une centaine de fois) mais frôlent la dizaine, ce qui, économiquement, est viable sur des systèmes correctement dimensionnés.

Dans les domaines liés à la sécurité, les GPU sont très présents sur les crackages de mot de passe via CryptoHaze ou JohnTheRipper. Malheureusement la prometteuse librairie libgpucrypto[17] ne s'est pas développée, pas plus que le projet sslshader[18] de pare-feu.

Dans d'autres domaines, nous assistons à des annonces liées à l'exploration de base de données[19] (sous PostgreSQL), à des applications Java. Les journaux regorgent d'articles sensationnalistes mais gardons à l'esprit ce nombre que nous avons évalué : si le traitement n'est pas parallélisable à mieux que 999 pour 1000, la prudence est de mise.

5.2 Les processeurs basse consommation : un marché universel ?

Avec le « *Green IT* » , la presse se fait écho chaque semaine de l'intérêt des Atoms. Ces derniers offrent une intégration maximale pour les usages « nuageux ». Tous les grands constructeurs sont en mesure de proposer « leur solution » : les solutions à base d'ARM et de Fusion sont déjà dans les cartons.

Cependant, diviser par 20 sa consommation pour diviser par 10 sa puissance a-t-il un sens ? La gestion de la consommation, pendant nos investigations, a montré qu'il fallait faire très attention au comportement par défaut que prennent les processeurs pour limiter leur consommation : le *On Demand* s'avère particulièrement inadapté pour les services réseaux à faible latence notamment. Il convient donc d'appréhender ces nouvelles architectures avec des protocoles de tests complets pour ne pas sacrifier le service sur l'autel de cette nouveauté.

6 Conclusion

Du côté des architectures, il ne serait pas étonnant que nous assistions à la fusion de ces deux approches entre accélérateur et basse consommation : AMD ne tente-t-elle pas de supprimer son unité de calcul flottante vectorielle pour confier ses calculs au GPU à proximité ? Pourquoi Nvidia s'intéresse-il au marché de l'embarqué alors que son leadership s'est établi largement sur les cartes graphiques ?

Si nous revenons à notre analyse, ces architectures « légères » ou « accélératrices » sont-elles matures pour des usages autres que le HPC ou les terminaux ultra-mobiles ? Nous avons tenté, en quelques mots et quelques tests sommaires, de lever un voile sur ces technologies : la puissance brute d'un côté, la puissance par Watt et la contrainte de la consommation de l'autre.

Ces deux technologies exigent l'une comme l'autre de la maîtrise pour les exploiter : programmer un accélérateur efficacement demande de solides connaissances en « profilage » et en parallélisme. De plus, le traitement doit particulièrement bien s'y prêter. Utiliser un processeur basse consommation dans un environnement serveur réclame une connaissance du fonctionnement interne de la gestion de l'énergie. Nous en revenons inéluctablement à la même conclusion : seules l'exploration des nouvelles technologies par une expérimentation largement partagée et l'anticipation des besoins par une connaissance approfondie de nos cœurs de métiers nous permettront de rester dans la course de l'informatique scientifique.

Bibliographie

[1] http://fr.wikipedia.org/wiki/Rast%C3%A9risation
[2] https://developer.nvidia.com/category/zone/cuda-zone
[3] http://www.khronos.org/opencl/
[4] http://en.wikipedia.org/wiki/Thermal_design_power
[5] http://en.wikipedia.org/wiki/ARM_architecture
[6] http://en.wikipedia.org/wiki/Exynos_%28system_on_chip%29
[7] http://en.wikipedia.org/wiki/Atom_%28Intel%29
[8] http://en.wikipedia.org/wiki/List_of_Intel_Atom_microprocessors
[9] http://en.wikipedia.org/wiki/AMD_Accelerated_Processing_Unit
[10] http://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms
[11] http://en.wikipedia.org/wiki/Lapack
[12] http://mathema.tician.de/software/pyopencl/
[13] http://documen.tician.de/pycuda/
[14] http://www.par4all.org/
[15] http://www.cbp.ens-lyon.fr/sidus/
[16] http://www.tux.org/~mayer/linux/bmark.html
[17] http://shader.kaist.edu/sslshader/libgpucrypto/
[18] http://shader.kaist.edu/sslshader/
[19] http://wiki.postgresql.org/wiki/PGStrom