Les supports d'exécution pour le calcul haute performance

Pierre-André Wacrenier, Raymond Namyst

Les supports d'exécution pour le calcul haute performance

Pierre-André Wacrenier et Raymond Namyst

Aux premières heures du calcul parallèle, dans les années 1970, les applications de calcul s'appuyaient sur des supercalculateurs spécialisés dont les processeurs accomplissaient des prouesses, mais qui exigeaient des efforts d'optimisation importants pour en tirer parti. Les chercheurs se concentraient sur l'élaboration de nouveaux langages de programmation — pour permettre une certaine indépendance des programmes vis-à-vis de la machine — et sur la conception de compilateurs capables d'accélérer les calculs au moyen d'instructions « vectorielles. » Le langage le plus utilisé était le Fortran et permettait la manipulation aisée de vecteurs et de matrices. Lorsque les supercalculateurs ont doucement basculé vers des architectures multiprocesseurs, il a fallu offrir des outils supplémentaires aux programmeurs pour qu'ils puissent contrôler explicitement la distribution des calculs sur les processeurs, ce qui dépassait ce qu'un compilateur était capable de faire automatiquement. Ces outils étaient principalement des bibliothèques de communication permettant des mouvements de données au sein de la machine. Les programmeurs, qui connaissaient bien l'architecture de la machine, déterminaient totalement l'enchaînement des calculs. Ce contrôle, qui constitue l'essence du mot « programmation », était fortement ancré dans la culture des informaticiens.

S'adapter à la machine… et aux imprévus

Toutefois, les programmeurs ont dû accepter de perdre une partie de leur contrôle, et ce dès la décennie suivante. L'augmentation de la puissance de calcul des supercalculateurs a en effet permis la naissance d'applications modélisant des phénomènes de plus en plus complexes, traitant des volumes de données plus grands et introduisant des modèles algorithmiques nouveaux. Les ensembles de données ne sont plus homogènes et certaines données, peu influentes sur le résultat de la simulation, peuvent parfois être négligées pour économiser des calculs. La quantité de calculs devient ainsi variable d'un point à l'autre du domaine. À l'inverse, certains algorithmes ne découvrent les calculs à effectuer qu'au fur et à mesure de leur progression… Ainsi, le calcul n'est plus distribuable équitablement sur les processeurs de manière statique. Cela passe par la mise en place de mécanismes qui corrigeront cette mauvaise répartition de la charge ou qui déplaceront les données d'une mémoire à une autre dynamiquement au cours de l'exécution. Ces mécanismes sont rassemblés au sein d'une couche logicielle appelée le « support d'exécution » (*runtime system* en anglais).

Si le périmètre de ces supports n'a cessé de croître depuis, il faut noter que leur évolution s'est effectuée dans un contexte à deux vitesses. D'un côté, les constructeurs se sont unis pour proposer des interfaces standards avec la volonté de proposer une approche générique laissant au programmeur un grand contrôle sur le déroulement de l'exécution, en se focalisant sur l'optimisation d'opérations

simples plutôt que sur l'ajout de mécanismes capables de prendre des décisions de manière autonome. D'un autre côté, les recherches du monde académique ont donné naissance à des supports d'exécution plus audacieux, introduisant de plus en plus d'intelligence en dehors de l'application. Par exemple, le placement des calculs sur les processeurs pouvait être automatiquement pris en charge par le support, l'application se contentant simplement de générer les calculs à faire. Toutefois, chacun de ces travaux proposait souvent une extension de langage différente, ce qui empêchait d'aboutir à une solution consensuelle.

Faire mieux que le système d'exploitation

Notons que les services fournis par un support d'exécution, en matière d'ordonnancement des processus ou de gestion mémoire, s'apparentent beaucoup à ceux fournis par un système d'exploitation. Quelques systèmes d'exploitation spécifiques au calcul haute performance ont d'ailleurs été développés par le passé, avec l'avantage d'offrir une interface de programmation standard et connue. Mais les systèmes d'exploitation doivent rester généralistes, ce qui les réduit à travailler « à l'aveugle », sans information sur les propriétés particulières d'une application de calcul (comme les liens privilégiés entre processus ou l'affinité des processus pour certaines zones mémoire). C'est pourquoi les supports d'exécution forment une couche logicielle à part entière, située juste au-dessus du système d'exploitation (Figure 1). Ils s'efforcent de neutraliser toutes les décisions impromptues que pourrait prendre le système d'exploitation afin de garder un contrôle total sur le placement des calculs et des données. Les systèmes d'exploitation en sont ainsi réduits à assurer un service minimal qui se cantonne à fixer exactement un processus par processeur.

Composer avec la complexité croissante des architectures

Le début des années 2000 a marqué un tournant important pour l'informatique et une accélératation du développement des supports d'exécution. Les constructeurs, parvenus aux limites de l'architecture traditionnelle des processeurs, ont été contraints de basculer vers la technologie multicœur, renommant au passage les processeurs en « cœurs » et appelant « processeur » la puce qui en contient plusieurs. Leur diffusion est devenue massive, des supercalculateurs aux téléphones, et désormais la programmation parallèle est incontournable. L'éventail des applications parallèles s'en trouve naturellement élargi et il devient nécessaire de générer du parallélisme à tous les étages : boucles, fonctions récursives ou encore pipelines de traitement.

Un paradigme de programmation, pourtant exploré dès les années 1990, effectue alors un retour en grâce au sein des supports d'exécution, y compris ceux proposés par les constructeurs : la programmation à base de tâches. Dans ce modèle, l'application génère des tâches qui sont des appels de fonctions différés et dont elle attendra, à certains points, la fin de l'exécution. Toutes les décisions

relatives à leur exécution sont déléguées au support d'exécution, qui a la délicate charge de maintenir dynamiquement un équilibre entre le stock de tâches prêtes et la disponibilité des cœurs de calcul (Figure 2). Il faut éviter qu'une pénurie de tâches provoque l'oisiveté des cœurs de calcul, tout en prévenant l'engorgement du système par un trop grand nombre de tâches.

Le paradigme de tâche permet au programmeur de structurer le flux des calculs en tâches tout en laissant au support d'exécution le soin de l'exploiter au mieux. Pour des informaticiens habitués à contrôler finement l'exécution, cela peut paraître un renoncement. Mais force est de reconnaître que contrôler l'exécution d'un programme sur des processeurs multicœurs où les caches sont partagés et la fréquence de chaque cœur varie en fonction, par exemple, de la chaleur dégagée relève désormais de la gageure.

Assurer la portabilité des performances

Dès 2007, un nouvel élément majeur vient malmener à nouveau la communauté du calcul parallèle : le recours massif aux accélérateurs de calculs matériels, et en particulier aux cartes graphiques (GPU pour *Graphical Processing Unit*). Mais cette fois, la décision n'émane pas des architectes. C'est en découvrant les travaux d'utilisateurs pionniers, qui révèlent comment ils ont réussi à accélérer l'exécution de leur application en détournant des cartes graphiques programmables de leur usage premier (l'accélération des graphismes en 3D), que la communauté se tourne vers ces accélérateurs pleins de promesses. À l'origine, ces cartes graphiques sont destinées à accélérer l'affichage d'objets en trois dimensions dans les jeux vidéo. Pour ce faire, elles possèdent de très nombreux processeurs élémentaires qui sont capables d'appliquer un même opérateur sur des millions (voire des milliards) de points en une fraction de seconde. Mais, depuis 2001, l'architecture des cartes graphiques est programmable : il est possible de demander à la carte graphique d'appliquer n'importe quel traitement sur un ensemble de données qui ne représentent pas forcément des points, à condition de renoncer à l'affichage. Ainsi, ces accélérateurs peuvent notamment être utilisés pour effectuer des calculs matriciels de manière très efficace. La difficulté de programmation est toutefois à la hauteur des gains en performance escomptés (au moins un ordre de grandeur).

Bien que les constructeurs de cartes graphiques — Nvidia en tête — comprennent rapidement les enjeux et proposent des interfaces de programmation destinées à faciliter la programmation des GPU, la technicité exigée devient une nouvelle frontière à dépasser tant du point de vue algorithmique que du point de vue exécutif. C'est le début d'une période foisonnante où de nombreux travaux donnent naissance à des bibliothèques spécifiques à chaque grande classe de calcul, des langages et des compilateurs permettant de s'affranchir des caractéristiques techniques des GPU. Mais, si puissants soient-il, ces accélérateurs ne remplacent pas les processeurs traditionnels qui restent nécessaires — et parfois même compétitifs — pour exécuter certaines parties de l'application.

De nombreux supercalculateurs sont désormais formés de nœuds hétérogènes associant des processeurs multicœurs et des cartes graphiques (Figure 3) et tout l'enjeu, dans les domaines du calcul haute performance, du *big data* ou de l'intelligence artificielle (cf. ANTONIU), est désormais d'être capable d'exploiter toutes les unités de calcul (des millions dans les plus gros supercalculateurs au monde) simultanément ! Mais planifier de façon optimale l'enchaînement des calculs et des transferts de données entre les GPU et les processeurs n'est plus faisable *a priori* : il faut constamment inspecter l'état de la machine, utiliser des modèles prédictifs calibrés sur le tas, et compenser les aléas par un ajustement dynamique du découpage de l'application.

C'est précisément sur ce plan que les supports d'exécution à base de tâches ont tiré leur épingle du jeu et sont devenus indispensables. Moyennant quelques informations supplémentaires demandées au programmeur (pour chaque tâche, il faut spécifier les données consultées et celles qui seront modifiées), un support d'exécution peut ainsi inférer des dépendances entre certaines tâches (une tâche B ne peut pas s'exécuter tant qu'une tâche A n'est pas terminée, car B a besoin de consulter le résultat produit par A) et, à tout moment, ordonnancer les tâches prêtes (dont toutes les dépendances sont satisfaites) sur les unités de calcul disponibles. Les tâches et leurs dépendances forment un graphe de tâches (Figure 2) qui, associé aux données manipulées, donne de précieuses informations au support d'exécution pour prendre de bonnes décisions de placement, pour anticiper les transferts de données de manière à ce qu'elles soient disponibles au plus tôt à l'endroit où elles seront accédées, ou encore pour prévoir l'état de la machine dans un futur proche (figure 3).

Cette séparation des rôles, avec d'un côté le programmeur qui exprime le parallélisme sous forme logique en découpant les calculs en tâches, et de l'autre côté le support d'exécution qui orchestre l'exécution de ces tâches du mieux possible en fonction de la machine sous-jacente, permet d'assurer la portabilité des performances d'une machine à une autre. Il devient possible d'obtenir des performances nominales sur un ordinateur de bureau et sur un supercalculateur composé de milliers de nœuds avec le même programme !

Et demain ?

Il reste encore beaucoup à faire pour que l'utilisation des supports d'exécution devienne transparente pour le programmeur. L'existence d'une multitude de supports d'exécution différents montre que la nécessaire convergence vers un seul standard sera encore longue. Par ailleurs, l'utilisation de supports prenant de nombreuses décisions de manière autonome complique la compréhension des performances obtenues pour le programmeur.

En dépit de ces difficultés, les supports d'exécutions sont amenés à jouer un rôle très important à l'avenir, au vu de la tendance confirmée vers l'utilisation de machines de plus en plus hétérogènes, notamment dans le domaine de l'intelligence artificielle (accélérateurs spécialisés), des stations de

travail (processeurs formés de plusieurs types de cœurs) ou des serveurs de calcul (fusion des processeurs et accélérateurs autour d'une mémoire commune).

Figures

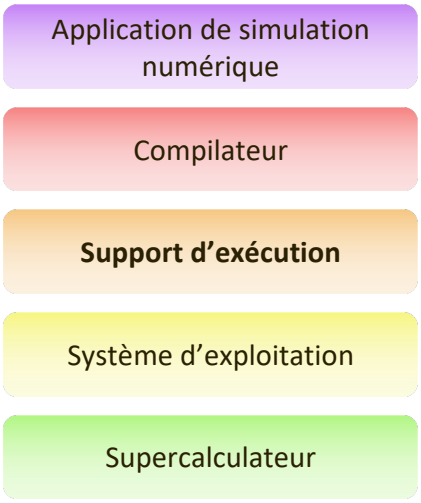

Figure 1. La place des supports d'exécution dans la pile logicielle d'un supercalculateur.

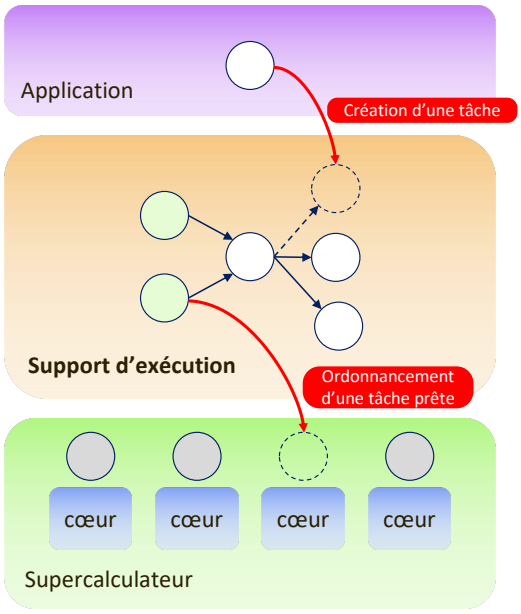

Figure 2. Principe de fonctionnement d'une exécution à base de tâches. Le support d'exécution (orange) maintient en permanence un stock de tâches qui augmente lorsque l'application en ajoute de nouvelles (flèche rouge du haut) et qui diminue lorsqu'une tâche prête (rond vert) est affectée à un cœur libre (flèche rouge du bas).

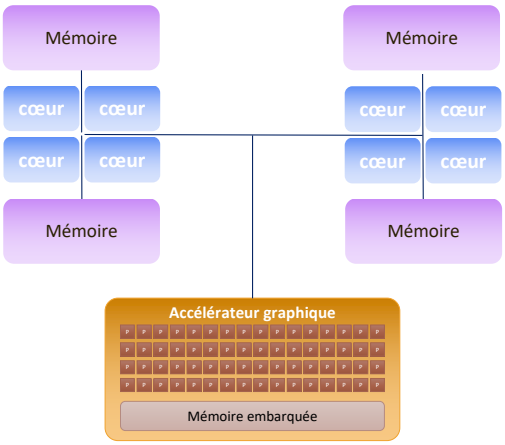

Figure 3. Schéma d'un ordinateur équipé d'un accélérateur de calcul (GPU).

Références

Brice Goglin, « Et plus vite si affinités… », https://interstices.info/et-plus-vite-si-affinites/

Claude Kaiser, « À quoi sert un système d'exploitation ? », https://interstices.info/a-quoi-sert-un-systeme-dexploitation/

Victor Eijkhout, "Parallel Programming in OpenMP and MPI", https://web.corral.tacc.utexas.edu/CompEdu/pdf/pcse/EijkhoutParallelProgramming.pdf

Affiliations

Pierre-André Wacrenier. Informatique du calcul haute performance, maître de conférences, Université de Bordeaux, LaBRI/Inria, pierre-andre.wacrenier@u-bordeaux.fr

Raymond Namyst. Informatique du calcul haute performance, Professeur, Université de Bordeaux, LabrI/Inria, raymond.namyst@u-bordeaux.fr

Glossaire

Instruction vectorielle. Une instruction vectorielle indique au processeur d'effectuer une même opération (par exemple une addition) en parallèle sur des données différentes. En traitant plusieurs données à la fois, ces instructions permettent donc d'accélérer les calculs s'appliquant sur des tableaux (vecteurs et matrices).

Bibliothèque de communication. Ensemble de fonctions permettant de programmer simplement le transfert de données d'une machine à une autre sans avoir à connaître les technologies réseaux sous-jacentes.

Support d'exécution. Couche logicielle intercalée entre le code applicatif et le système d'exploitation facilitant l'utilisation efficace des différents composants d'un ordinateur (mémoire, unités de calcul, cartes réseaux...).

Fonction récursive. Fonction dont le code effectue à son tour un ou plusieurs appel(s) à cette même fonction pour traiter des sous-cas.

Pipeline de traitements. Enchaînement de procédures de calcul où le résultat d'une procédure est utilisé comme entrée de la procédure suivante.

GPU (Graphical Processing Unit, processeur graphique en français). Circuit dédié au traitement et à l'affichage des images. Il possède une architecture hautement parallèle lui permettant d'effectuer des opérations de rendu géométrique ou d'anticrénelage des images de manière bien plus efficace que les processeurs classiques. Exclusivement dédiés aux fonctions graphiques à l'origine, ces coprocesseurs se sont également imposés dans le calcul intensif et l'intelligence artificielle.