

Parallel Task Force : le parallélisme, pourquoi, comment ?

Vincent C. LAFAGE

¹D2I, Institut de Physique Nucléaire
Université d'Orsay



mardi 14 octobre 2014

Pourquoi paralléliser ?

... doublement de la fréquence tous les 2 ans, 2 ans et demi ?

Vrai depuis 1980

Plus depuis ≈ 2004 !!! Pentium 4 : 3 GHz...

on attend entre 16 et 32 fois plus, > 50 à 100 GHz !!!

- $\mathcal{P} = C.V^2.f$
- 70–85 W \Rightarrow 2,5 kW !!!

\Rightarrow Trop d'énergie à dissiper !

\Rightarrow plusieurs processeurs sur un même circuit : **plusieurs cœurs**

Pourquoi paralléliser ?

- ⇒ pour continuer à bénéficier de la puissance croissante de la technologie, pas d'autre solutions que la programmation parallèle : **coordonner le travail** des différents cœurs
- ⇒ alors qu'on nous a toujours enseigné l'aspect essentiellement séquentiel de la programmation !

Pourquoi paralléliser ?

nouvelles technologies : part croissante d'une énergie rare & chère.
Consommation CC-IN2P3 ; 1,6 MW → 1,2 MW
budget électrique total 0,864 M€

- évolution vers un **meilleur ratio W/MIPS** ou W/MFLOPS :
Intel XScale, 600 MHz, 0,5 W
5 fois plus lent, 80 fois plus économe !
- **diminuer la fréquence**, vu qu'il y aura plus de coeurs

Comment paralléliser ?

- c'est un **paradigme de programmation (concurrence)** complémentaire des approches classiques : procédural, modulaire, générique, objet.
- ... et qui s'appuie sur différentes **technologies** matérielles également à différentes échelles
 - pipeline (DSP)
 - vectorisation (SIMD)
 - parallélisation par processus (lourds) assez indépendants
 - parallélisation par processus légers (*threads* = fils) partageant l'espace d'un même processus : **économie de mémoire**
 - distribution (parallélisation par processus tournant sur des machines différentes)

S2I ::groupe Parallèle



Christophe DIARRA
Grille, MPI, OpenMP



Ivana HRIVNACOVA
C++, ROOT / Geant4 multithread



Luz GUEVARA
OpenMP, GPU



Vincent LAFAGE
Fortran, optimisation calcul numérique

Success story

- Code de capture d'électrons
9 kSLOC de Fortran, 5 kSLOC de C(++)
- Accélération avant $//^n$
 - ... facteur 12 (stockage plutôt que recalcul)
 - ... facteur **>50** au total
 - utiliser les bibliothèques standard (`nearbyint`)
 - Spherical Bessel Benchmark (15 codes, 9 algos)
 - vectorisation des boucles. . .
- amélioration de la précision

Ces résultats sont le fruit d'une méthode :

- ⇒ on ne rentre pas dans 15 000 lignes de code comme dans un moulin : mise du code sous contrôle de version `svn`
- analyse statique `ftncheck`, `cppchecker`
 - * branches mortes (procédures, variables)
 - * métriques `sloccount`
 - analyse dynamique "profiling" `gprof`
 - * identification des goulets d'étranglements
 - * optimisation de fond de boucles
 - chasse aux problèmes de mémoire `valgrind`
 - typographie, indentation, documentation `doxygen`
 - pêche aux mauvaises pratiques numériques
- D. GOLDBERG, *What every computer scientist should know about floating point arithmetic*
- * constantification des constantes : extraction des constantes en dur, uniformisation : combien de valeurs de π distinctes ?
 - * accélération (Horner, Richardson, stockage intermédiaire...)

Conclusion

- + beaucoup de perspectives
 - pour les anciens codes. . .
 - . . .et pour les nouveaux
- + beaucoup de solutions
- pas si évident
- ⇒ offre de conseil
- ! d'abord, optimiser

Sans programmation parallèle, aucun homme n'est jamais assez fort pour ce calcul

lafage@ipno.in2p3.fr

☎ : 53138