Génie Logiciel pour le Calcul Scientifique

Les objectifs

- Présentation des concepts de génie logiciel dans un cadre HPC
  - Méthodes de conception d'API, UML, design patterns, …
- Comprendre l'interaction génie logiciel / performances
  - 2 objectifs importants, parfois contradictoires
  - Savoir quoi favoriser, dans quelle situation (2 niveaux dans les codes)
- Identifier les outils pour combiner ces objectifs
  - Les « briques logicielles » existantes, bibliothèques, algos, ...
  - Les approches émergentes

GLCS : en pratique

- 1ère phase
  - Cours
  - TDs (mini-projets) => notés
- 2ème phase
  - Le projet principal
    - Modularité pour le post-traitement « post-hoc » et « in situ » de données de simulation
  - Quelques encadrés de cours et rappels utiles pour le projet
- Évaluation finale
  - Rendu du projet
  - Soutenance de présentation du projet

GLCS : Introduction

Génie logiciel pour le calcul scientifique
Introduction des objectifs de base

Génie logiciel : Définition

Le génie logiciel (en anglais *software engineering*) est une science de génie industriel qui étudie les méthodes de travail et les bonnes pratiques des ingénieurs qui développent des logiciels.

Le génie logiciel s'intéresse en particulier aux procédures systématiques qui permettent d'arriver à ce que des logiciels de grande taille correspondent aux attentes du client, soient fiables, aient un coût d'entretien réduit et de bonnes performances tout en respectant les délais et les coûts de construction.

https://fr.wikipedia.org/wiki/Génie_logiciel

Code scientifique, objectifs

- Correction (“correctness”)
- Temps d’exécution mono-cœur
- Passage à l’échelle du temps d’exécution
  - Intra-nœud / Inter-nœud
- Consommation mémoire
- Passage à l’échelle mémoire
  - Intra-nœud / Inter-nœud

Performance

- Lisibilité
- Évolutivité
- Maintenabilité
- Ré-utilisabilité

Génie logiciel

- …

Performance & simulation

- Temps de calcul = argent
  - P.ex. machine *Genci/TGCC Curie* (2012-2018)
    - ~ 80k.cœurs x ~ 6 ans => ~ 4,4 Gh.cœur
    - Achat (~ 150M€) + électricité + refroidissement (~300M€)

      ⇒ ~ 0,1€ / h.cœur
- Des simulations longues
  - P.ex. 1 simulation Gysela5D
    - 1 mois sur Curie : 60 Mh.cœur (>68 siècles seq) : ~ 6 M€
- Optimisation = gain d'argent
  - ex. Gysela : Gain 10 % => 600k€ / simulation

Génie Logiciel & simulation

- Temps de développement = argent
  - Ingénieur débutant ⇒ ~ 100k€ / an (+env)
  - Efficacité non linéaire (passage à l'échelle)
    - Cf. modèle COCOMO
- Des codes complexes
  - P.ex. Gysela5D
    - ~ 450 kloc (MPI + OpenMP + ...)
    - 20 ans: 3 Chercheurs + Thèses + Stages ⇒ >20 M€

      ⇒ 0,025€ / loc
- Maintenance ~ 75% coûts
  - Durée de vie d'un code ~30 ans
  - Durée de vie d'une architecture ~10 ans
    - Vectoriel, multi-cœur, distribué, hiérarchique, hybride, ...
- Simplification = gain d'argent

Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

Génie Logiciel pour le HPC

- Un aspect important
  - Évolutivité / Réutilisabilité
  - Ré-écriture d'un code de 0 très cher

> *Premature optimization is the root of all evil (or at least most of it) in programming.*
>
> Donald Knuth

> *The first rule of program optimization: don't do it. The second rule of program optimization (for experts only!): don't do it yet.*
>
> Michael A. Jackson

- HPC ⇒ pas au prix des performances
  - Temps de calcul très cher aussi

- 2 Questions :
  - Quand favoriser le génie logiciel ?
  - Comment favoriser le génie logiciel ?

Plusieurs sortes de codes

- Des codes « bac à sable »
  - Des codes en mouvement
  - Test de nouveaux modèles, nouvelle physique
  - Peu d'utilisateurs, souvent aussi développeurs

  ⇒ Maximiser **l'évolutivité** et la **maintenabilité**

- Des codes « industriels »
  - Physique fixe
  - Durée de vie plus limitée
  - Beaucoup d'utilisateurs distinct des développeurs

  ⇒ Maximiser les **performances**

Plusieurs niveaux dans le code

- <10% du code, >90% du temps de calcul
  - Nids de boucle ($O(x^n)$)
  - Code peu complexe
    - Peu de lignes, appels de fonctions, condition, …
  - « ***noyaux de calcul*** »

```
Foreach x
  Foreach y
    Foreach z
      t[x,y,z] = a*t[x,y,z] - b*t[x-1,y,z]
```

⇒ Maximiser les **performances**

- >90% du code, <10% du temps de calcul
  - Structure du code, appel des noyaux
  - Souvent variable d'une exécution à l'autre

⇒ Maximiser la **maintenabilité**, la **ré-utilisabilité**

Optimisation : perf. ou GL ?

- Code lisible > efficace pour commencer
- On optimise seulement si nécessaire
  - « Code industriel » ?
  - Temps d’exécution trop long
- On optimise seulement là où nécessaire
  - « Noyau de calcul » ?
  - Évaluation de performances ! (*Profiling*)
- Si possible on fait les deux
  - **Réutilisation** !