OUTILS DE GÉNIE LOGICIEL

Les Outils de profilage

OBJECTIF

- Introduction à l'activité de profilage et l'analyse de performance des programmes ainsi que les outils de génie logiciels utilisés

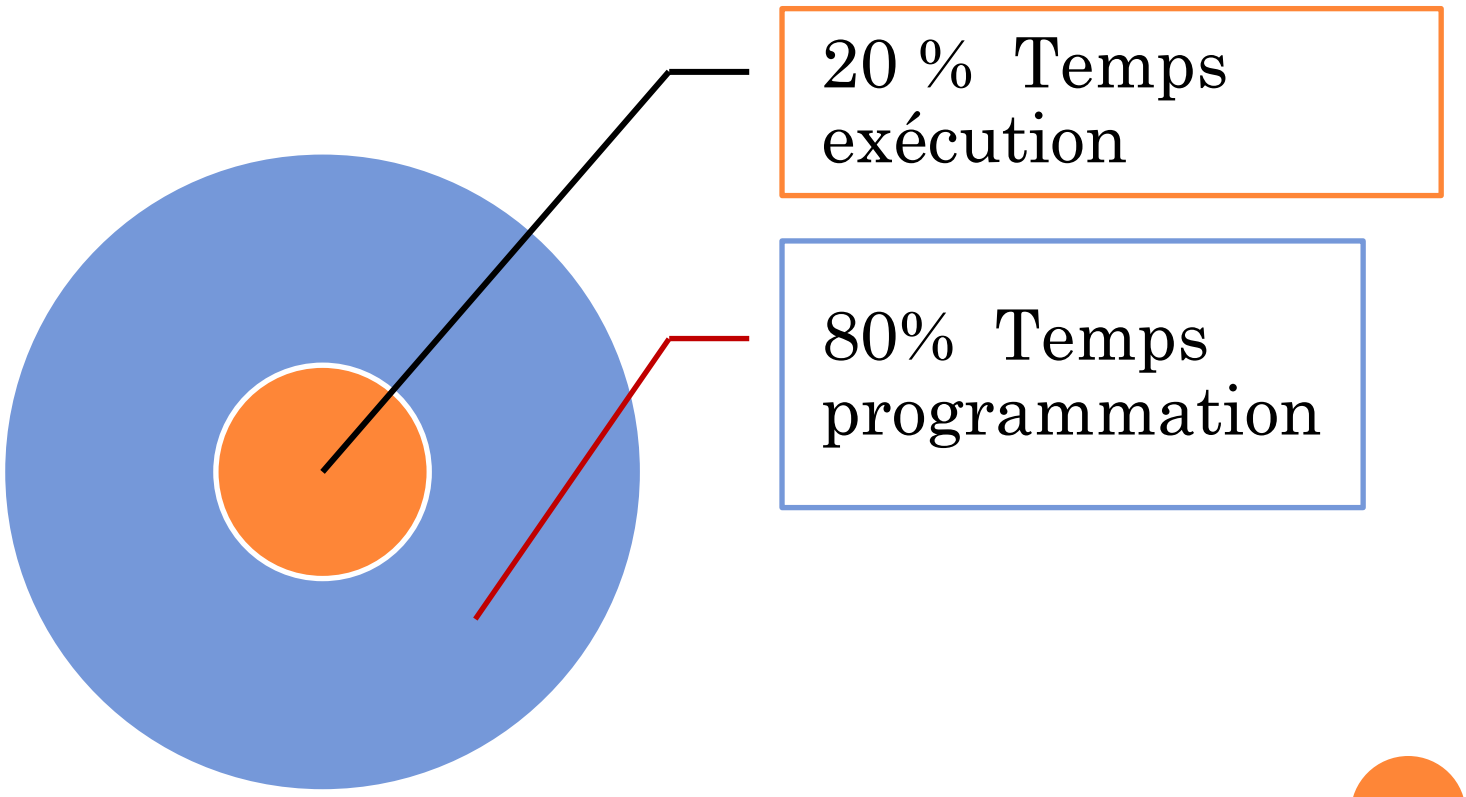
PLAN

1. Optimisation d'un programme
2. La performance d'un programme
3. Le profilage et type de profilage
4. Exemple
5. Outils de profilage

OPTIMISATION DES PROGRAMMES

- « Il y a plus de **péchés informatiques** commis au nom de l'efficacité (sans pour autant l'atteindre) qu'au nom de toute autre raison — y compris la stupidité aveugle. » (William A. Wulf)
- « Il faut oublier l'efficacité pour disons 97% du temps : **l'optimisation prématurée** est à la source de tous les maux. » (Donald E. Knuth)

OPTIMISATION DES PROGRAMMES - LA RÈGLE DU 80:20 -



ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

OPTIMISATION DES PROGRAMMES
LA RÈGLE DU 80:20

- Optimisation du temps / efficacité de la programmation:
  - Plus de temps à consacrer à la logique qu'aux subtilités de l'architecture de la machine sous-jacente
  - Avènement de langages comme Java et C#, qui réduisent le temps nécessaire à la programmation
  - Le refactoring (ré-usinage) du code, les Frameworks…
- **Économise le temps de programmation mais augmente le temps d'exécution des programmes**

OPTIMISATION DES PROGRAMMES
LA RÈGLE DU 80:20

- Optimiser le temps d'exécution pour accélérer l'exécution d'un programme:
  - Lors de la génération du code = une étape d'optimisation
  - Utilisant des algorithmes implémentant des techniques d'optimisation :
    - Optimiser la distribution de registre
    - Optimiser les accès en mémoire
    - L'option *-O du compilateur GCC dans le* niveau d'optimisation
- La problématique = Efficacité de l'optimisation
- Exemple :
  - Optimiser une fonction qui est appelée des milliers de fois lors de l'exécution qu'une fonction appelée une dizaine de fois dans un programme

LA PERFORMANCE DE PROGRAMME

- La performance mesure le temps que le programme passe dans le processeur (les boucles)
- Le temps passer à appeler des fonctions
- Le temps passer en lecture écriture en mémoire (cache ou externe)
- Savoir quelles sont les parties du code fréquemment utilisées et quelles fonctions prennent le plus de temps processeur

PROFILAGE DE PROGRAMME

- Le profilage est la collecte des informations sur l'exécution d'un programme pour trouver les sections du code/des fonctions à optimiser pour améliorer les performances d'un programme :
  - Parties du code fréquemment utilisées et quelles fonctions prennent le plus de temps processeur.
  - Liste des fonctions appelées et le temps passé dans chacune d'elles
  - Utilisation processeur;
  - Utilisation mémoire.
- Principe de KAPLAN

  « Si vous ne pouvez pas le mesurer,

  vous ne pouvez pas le gérer »

PROFILAGE DE PROGRAMME

- Le profilage est implémenté en rajoutant des instructions au code source pour mesurer le comportement du logiciel lors de l'exécution.
- Ensuite, un scénario d'utilisation est défini et exécuté sur le logiciel instrumenté.
- Les données récoltées et analysées forment un "Profil d'utilisation"
- Une recompilation à base de ces données pour que le compilateur optimise ce profil d'utilisation.
- Optimisation dirigée par les profils

« *profile-guided optimization* »

TYPE DE PROFILAGE

- Le profil plat (*Flat Profile*)
  - Détaille combien de temps processeur dure chaque fonction et combien de fois elle a été appelée.
  - Bref résumé des informations de profilage rassemblées.
  - Donne une idée des fonctions qui peuvent être réécrites ou affinées pour gagner en performance.
- EXEMPLE avec gprof

```
Each sample counts as 0.01 seconds.
% cumulative self self total
time seconds seconds calls ms/call ms/call name
33.33 0.01    0.01     207 0.05    0.05    file_hash_2
33.33 0.02    0.01      38 0.26    0.26    new_pattern_rule
33.33 0.03    0.01       6 1.67    2.81    pattern_search
 0.00 0.03    0.00    2881 0.00    0.00    hash_find_slot
 0.00 0.03    0.00    2529 0.00    0.00    xmalloc
```

TYPE DE PROFILAGE

- Le graphe d'appels (*Call Graph*) :
  - Montre le nombre de fois où chaque fonction a été appelée
  - Montre les relations existant entre les différentes fonctions
  - Permet de découvrir des bogues dans le code :
  - Les appels de fonctions à éliminer
  - A remplacer par d'autres fonctions plus efficaces
  - Chemins du code à optimiser
- Exemple avec gprof

index	% time	self	children	called	name
				6	eval_makefile [49]
[25]	3.7	0.00	0.00	6	eval [25]
		0.00	0.00	219/219	try_variable_definition [28]
		0.00	0.00	48/48	record_files [40]
		0.00	0.00	122/314	variable_expand_string [59]
		0.00	0.00	5/314	allocated_variable_expand_for_file[85]

OUTILS DE PROFILAGE

- Intégrés aux IDE ou en ligne de commande:
  - Microsoft Studio intégré un outil "Assistant de profilage"
  - Profileur Java (open sources) intégré à Éclipse comme plugin
- Libre non intégré à l'IDE
  - Valgrind (débugger et profileur)
  - **Gprof sous linux avec GCC**

EXEMPLE D'OPTION D'OPTIMISATION

- Optimization Options GCC:
  - `-falign-functions[=n]`
  - `-falign-jumps[=n]`
  - `-falign-labels[=n]`
  - `-falign-loops[=n]`
  - `-fassociative-math`
  - `-fauto-inc-dec`
  - `-fbranch-probabilities`
  - `-fbranch-target-load-optimize`
  - `-fbranch-target-load-optimize2`
  - `-fbtr-bb-exclusive`
  - `-fcaller-saves`
  - `-fcheck-data-deps`