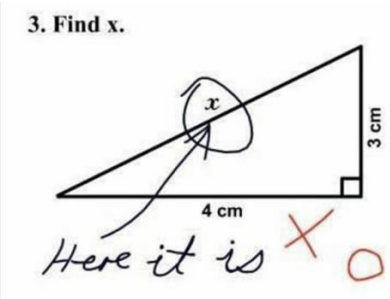
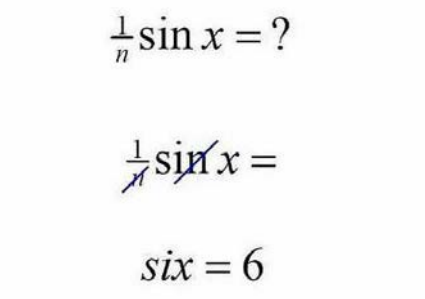
Evaluation TP GPU





By Professeur
Cédric Bilat
Cedric.Bilat@he-arc.ch

Formule

Pour un TP

Score OpenMP

Chaque TP sera noté de 0 à 1

$$score_{omp}(tp) = \begin{cases} 1 & \text{code fonctionel, résultat juste} \\ 0 & \text{sinon} \end{cases}$$

Score Cuda

Chaque TP sera noté de 0 à 3

$$score_{omp}(tp) = \begin{cases} 1 & \text{code orienté projet, presque fonctionel} \\ 2 & \text{code fonctionel, résultat juste} \\ 3 & \text{code parfait et soigné! Plusieurs implémentations avec speed up} \\ 0 & \text{sinon} \end{cases}$$

Score finale TP

$$score(tp) = \max\left(score_{omp}(tp), score_{cuda}(tp)\right) \in [0,3]$$

Pour tous les TP

Le score final de tous les TP est une moyenne pondérée des TP, où le poids dépend de la complexité du TP

$$score(tps) = \sum_{tp} score(tp) \frac{1}{poids(tp)} \in [1,3]$$

La note est obtenue par linéarisation

$$note(tps) = \frac{5}{3} score(tps) + 1 \in [1,6]$$

La note finale tiendra compte de la gaussienne des notes obtenus par le groupe. Un ajustement selon ce critère pourrait être appliqué !

Délivrable

Ou

Dans le folder approprié de votre « home », sur le serveur *Cuda1*

/home/NNN/CUDA/toProf/code

déposer la version de votre workspace qui sera utilisé pour l'évaluation.

Quoi

Tout votre workspace, ie le folder complet W….

Finalisation

Launcher

Dans le folder

./RELEASE/bin

du workspace, il doit y avoir, un sous folder XXX par TP. XXX sera le nom du tp. Dans ce folder on doit trouver :

- un ***.run*** pour chaque implémentation différente d'un même TP
- un ***.sh*** permettant de lancer ce *.run*

Pour les noms de ces fichiers vous choisirez

- ***XXX.run***
- ***XXX.sh***

Attention

(A1) Le .sh doit initialiser des variables d'environnement avec ***cbirt*** (cbiRuntime), et lancer le wraper ***gl*** en cas de bureau à distance.

Exemple1 : *Sans OpenGL*

```
# !/bin/bash
cbirt ./Student_Cuda_64.run
```

Exemple2 : *Avec OpenGL*

```
# !/bin/bash
cbirt gl ./Student_Cuda_Image_64.run
```

Par défaut, c'est le runtime *gcc* qui est utilisé. Pour activer le runtime *intel* il suffit d'ajouter *intel* apres *cbirt :*

cbirt intel …

C'est utile avec les projets OMP si vous avez compilé avec intel !

(A2) N'oubliez pas de rendre les *.sh* executable!

Multi-Implémentation

Dans le cas d'implémentations différentes pour un même TP, post fixé le XXX par _YYY ou YYY représente la description de votre implémentation. Il y aura ainsi une paire de

XXX_YYY.run
XXX_YYY.sh

par implémentation différente d'un même TP.

Procédure

(P1) Compiler votre projet
Le *XXX.run* se trouve dans *./RELEASE/bin*

(P2) Renommer ce *XXX.run*

(P3) Déplacer ce .run dans le sous folder approprié.

Cas OpenMP

Pour les TP non graphiques implémentés en OpenMP, on se contentera d'un seul *.run* et *.sh* par TP et par compilateur. Les différentes implémentations s'exécuteront alors séquentiellement, avec des affichages consoles soignées permettant de comprendre le fil des événements. Autrement-dit :

XXX_gcc.run
XXX_intel.run

Compilation globale

Assurez-vous que votre workspace globale est compilable, et ceci avec tous les compilateurs (GCC et Intel). Utilisez à cet effet depuis *Eclipse*

linux_workspace_gcc
linux_workspace_intel
linux_workspace_doc

Readme

Réalisé un fichier ***readme.txt*** indiquant les TP réalisés.

Auto-Evaluation

Remplissez le classeur excel d'auto-evaluation ! Celui-ci se trouve sur le serveur ftp du cours et sera remis dans

/home/NNN/CUDA/toProf/evaluation/auto

Backup

Les données sur les serveurs du cours (Cuda1 et cuda2) ne sont pas backupées.
A vous d'assurer un backup jusqu'au jour de l'examen.

Acompte

Dès la fin du cours, vos acomptes sont susceptibles d'être effacer à tout instant sans préavis.
Penser donc à effectuer une copie des données que vous souhaitez conserver.

Contraintes

(C1) Les implémentations et *compilation/linkage* doivent tourner avec succès

- Sur les server ***linux*** Cuda1 et Cuda2 mis à disposition pour le cours.

(C2) Tous les projets doivent être compilables :

- Avec les **makefiles** fournis.
- Avec les compilateurs fournis
  - gcc
  - intelLinux
  - nvcc

(C3) L'environnement de développement est **Eclipse**.

(C4) L'objectif est de fournir le code le plus performant possible en utilisant les techniques vu au cours, et uniquement celle-ci !

(C5) Une API permettant de dessiner des images sera mise à disposition.
Il est obligatoire d'utiliser cette API !!
Deux versions sont mises à disposition :

- ***GLImages***
- ***GLImagesCuda***

La deuxième sera utilisée lors des implémentations en *Cuda*. Elle offre de l'interopérabilité *OpenGL* et des performances accrues ! Une documentation séparée est aussi fournie pour l'API.

(C6) Tous les TP images seront rendus sous la forme d'une animation, avec overlays textuels indiquant des informations pertinente, sur l'animation, sur la méthode utilisée lors de l'implémentation,…

(C7) Tous les TP numériques, dont le résultat est prédictible, seront testé avec l'api *cppTest* fournie dans le *workspace* donnée au début du cours.

(C8) Tous les TP images « zoomables » seront implémentés de telle sorte que le zoom soit fonctionnel !

End