Exploration de la librairie OpenCL

Objectifs

Ce laboratoire vise à explorer les fonctionnalités de la librairie OpenCL, à déterminer l'accélération et l'efficacité d'un programme utilisant la programmation sur GPGPU (General-Purpose processing on Graphics Processing Units) et à comparer le fonctionnement avec les librairies OpenMP et MPI utilisées durant les laboratoires précédents.

Le problème traité est exactement le même que celui du laboratoire #3. Il s'agit de simuler le transfert de chaleur sur une plaque en deux dimensions. La simulation doit se faire à l'aide de la librairie OpenCL afin d'exploiter le traitement sur la carte graphique.

Énoncé du problème

On partitionne une plaque en subdivisions (m par n). La température de la plaque évolue dans le temps pour un certain nombre de pas de temps (np). La taille du pas de temps (td) et la taille d'un côté d'une subdivision (h) influence le transfert de chaleur entre les subdivisions, mais n'affecte pas votre implémentation. Contrairement au laboratoire précédent, il n'y a pas de paramètre pour le nombre de processus puisque OpenCL utilise plutôt un nombre d'unités de calcul concurrentes selon le matériel.

Votre programme recevra donc les paramètres suivants dans cet ordre :

- m : Le nombre de colonnes de la matrice de travail (peut varier de 3 à 10000)
- n : Le nombre de lignes de la matrice de travail (peut varier de 3 à 10000)
- np : Le nombre d'itérations à effectuer (peut varier de 1 à 10000)
- td : Taille d'un pas de temps discrétisé
- h : Taille d'une subdivision

Vous pouvez débugger votre programme avec des paramètres. Pour ce faire, ouvrez les propriétés du projet, allez sous « Configuration properties -> Debugging » et écrivez votre paramètre dans l'entrée « Command Arguments ».

Sur une matrice de dimensions U[m, n, np], la formule discrétisée d'Euler à appliquer sur chaque cellule de la matrice à chaque itération est la suivante :

$$U(i,j,k+1) = \left(1 - \frac{4\mathrm{td}}{h^2}\right) * \mathrm{U(i,j,k)} + \left(\frac{\mathrm{td}}{h^2}\right) * [\mathrm{U(i-1,j,k)} + \mathrm{U(i+1,j,k)} + \mathrm{U(i,j-1,k)} + \mathrm{U(i,j+1,k)}$$

De plus, la matrice au temps k = 0 doit être initialisée grâce à l'équation suivante :

$$U(i,j,0) = \mathrm{i} * (\mathrm{m} - \mathrm{i} - 1) * \mathrm{j} * (\mathrm{n} - \mathrm{j} - 1)$$

Laboratoire

Vous devez concevoir un programme qui pourra simuler le transfert de chaleur d'une plaque chauffante de façon séquentielle, puis à l'aide d'OpenCL. Votre programme affichera ensuite le temps d'exécution parallèle, le temps d'exécution séquentiel et l'accélération correspondante.

Environnement de travail

Pour ce laboratoire, vous utiliserez Visual Studio 2010 sur Windows. La procédure pour configurer OpenCL dans Visual Studio est en annexe. Cette procédure ne s'applique qu'aux cartes NVidia et elle requiert l'installation du SDK disponible ici : https://developer.nvidia.com/cuda-downloads .

Vous trouverez aussi en annexe, une fonction pour lire un fichier .cl, un squelette de kernel (le contenu du fichier .cl) et la liste des fonctions OpenCL à utiliser pour rouler ledit kernel.

Contraintes supplémentaires :

- Dans le cas des deux problèmes, le logiciel doit effectuer le traitement séquentiellement, puis à l'aide de OpenCL. Le logiciel doit aussi afficher le temps d'exécution et la matrice finale dans les deux cas ainsi que l'accélération de la version parallèle vis-à-vis de la version séquentielle.
- Contrairement au laboratoire précédent, il n'y a pas de usleep. Comme l'application roule sur Windows, il n'y a pas non plus de script de lancement, ni de makefile. Tout doit simplement rouler dans Visual Studio 2010.
- Votre programme ne doit provoquer aucun avertissement de compilation.

Questions :

- Quelle est l'accélération de votre version parallèle relativement à votre version séquentielle?
- Comment est-ce que les unités d'exécution communiquent-elles entre elles?
- Comment la communication se compare-t-elle avec MPI?
- Comment la communication se compare-t-elle avec OpenMP?
- Pourquoi est-il nécessaire d'utiliser la fonction clCreateBuffer plutôt que de simplement passer un pointeur vers les données à la carte vidéo?

Remise du laboratoire

Date de remise

- Jeudi 13 avril 2017 avant 23h59 (3 périodes)

Livrables

- La solution Visual Studio 2010 (.sln) et ses dépendances dans la structure de dossier originale (je dois pouvoir décompresser le dossier, ouvrir la solution et compiler/rouler votre programme sans étape supplémentaire)
    - Le fichier de projet (.vcxproj)
    - Le ou les fichiers C++ contenant votre code (.cpp)
    - Le fichier OpenCL contenant votre kernel (.cl)
    - Ne m'envoyez PAS l'exécutable, les .obj, .pdb, .log, .user, .filters, .sdf, .ipch ou autres fichiers qui n'iraient pas dans un *source control*. Soit ce sont des fichiers résultants (donc redondants avec le code source), soit ils contiennent des informations sur votre machine (et sur l'utilisateur) que vous ne voudriez probablement pas partager.
- Un rapport au format **PDF**
- Compressez tous les fichiers pour n'avoir qu'un seul fichier d'archive

Barème de correction

- Introduction
- Temps d'exécution
    - Résultats (5%)
    - Discussion (5%)
- Questions (20%)
- Conception de l'algorithme
    - Code (50%)
    - Conception (10%)
    - Discussion (10%)
- Conclusion
- Des points peuvent être retirés pour la présentation :
    - Rapport paginé (-1%)
    - Rapport de 15 pages maximum (-1%)
    - Sujet du courriel (-1%)
    - Structure de l'archive (-2%)
- Retard : (-5% par heure)
- Français : (Jusqu'à -10%)

Procédure de remise

- Envoyer votre fichier compressé à stonkie@gmail.com.
    - S.V.P., mettez comme sujet du courriel « **LOG645-Lab4-EquipeX** »
    - Le fichier compressé « **LOG645-Lab4-EquipeX.zip** » doit avoir la structure suivante :
        - LOG645-Lab4-EquipeX/
            - Rapport.pdf
            - Lab4.sln
            - Lab4/
                - Lab4.vcxproj
                - Lab4.cpp
                - Lab4.cl

Annexe A : Configuration de OpenCL dans Visual Studio 2010

Cette procédure permet de configurer OpenCL dans Visual Studio 2010 avec « CUDA Toolkit 6.5 ». Cette procédure est aussi applicable à Visual Studio 2012 et 2013.

- Ouvrir le menu « Tools -> Options ». Sous la section « Projects and Solutions -> VC++ Project Settings » et ajouter à l'entrée « Extensions To Include » l'extension « .cl »
- Créer un projet de type « Win32 Console Application ». Faire « Next », cocher « Empty projet » puis cliquer sur « Finish ».
- Dans les propriétés du projet sous « Configuration Properties -> VC++ Directories », ajouter à l'entrée « Include Directories » une nouvelle valeur qui sera « $(CUDA_PATH)/include ».
- Dans les propriétés du projet sous « Configuration Properties -> Linker -> General », ajouter à l'entrée « Additional Library Directories » une nouvelle valeur qui sera « $(CUDA_PATH)/lib/Win32 ».
- Dans les propriétés du projet sous « Configuration Properties -> Linker -> Input », ajouter à l'entrée « Additional Dependencies » une nouvelle valeur qui sera « OpenCL.lib ».
- Ajouter un nouvel item au projet (CTRL-SHIFT-A) et choisir le template « C++ File (.cpp) ». Nommer le fichier « Lab4.cpp » (il devrait automatiquement être déplacé dans le dossier virtuel « Sources »). Créer un nouveau fichier et nommer le fichier « Lab4.cl ».

Annexe B : Liste de fonctions nécessaires

Pour compléter le laboratoire, vous aurez besoin d'utiliser les fonctions suivantes dans cet ordre. Elles vous sont fournies afin d'accélérer la mise en place de votre environnement de départ. Vous pouvez aussi trouver des exemples sur le site Web suivant : https://developer.nvidia.com/opencl

- clGetPlatformIDs
- clGetDeviceIDs
- clCreateContext
- clCreateCommandQueue
- clCreateBuffer
- clCreateProgramWithSource
- clBuildProgram
- clCreateKernel
- clSetKernelArg
- clEnqueueWriteBuffer
- clEnqueueNDRangeKernel
- clEnqueueReadBuffer

Annexe C : Squelette de kernel

Voici un squelette de kernel auquel vous pouvez ajouter des paramètres et votre traitement.

```
__kernel void HeatTransfer()
{
	int id = get_global_id(0);
}
```

Annexe D : Fonction de lecture du fichier .cl

Source : https://developer.nvidia.com/opencl

```
////////////////////////////////////////////////////////////////////////////
//! Loads a Program file and prepends the cPreamble to the code.
//!
//! @return the source string if succeeded, 0 otherwise
//! @param cFilename        program filename
//! @param cPreamble        code that is prepended to the loaded file, typically a set of
#defines or a header
//! @param szFinalLength    returned length of the code string
////////////////////////////////////////////////////////////////////////////
char* oclLoadProgSource(const char* cFilename, const char* cPreamble, size_t*
szFinalLength)
{
  // locals
  FILE* pFileStream = NULL;
  size_t szSourceLength;

  // open the OpenCL source code file
  if (fopen_s(&pFileStream, cFilename, "rb") != 0)
  {
    return NULL;
  }

  size_t szPreambleLength = strlen(cPreamble);

  // get the length of the source code
  fseek(pFileStream, 0, SEEK_END);
  szSourceLength = ftell(pFileStream);
  fseek(pFileStream, 0, SEEK_SET);

  // allocate a buffer for the source code string and read it in
  char* cSourceString = (char *)malloc(szSourceLength + szPreambleLength + 1);
  memcpy(cSourceString, cPreamble, szPreambleLength);
  if (fread((cSourceString)+szPreambleLength, szSourceLength, 1, pFileStream) != 1)
  {
    fclose(pFileStream);
    free(cSourceString);
    return 0;
  }

  // close the file and return the total length of the combined (preamble + source)
string
  fclose(pFileStream);
  if (szFinalLength != 0)
  {
    *szFinalLength = szSourceLength + szPreambleLength;
  }
  cSourceString[szSourceLength + szPreambleLength] = '\0';

  return cSourceString;
}
```