CUDAisation du code impératif

Université d'Angers

Vendredi 03 juin 2016

Tuteur pédagogique :Jean Michel RICHER

Alexis BRIARD
Jason JAMET
Guillaume GRANDJEAN

Plan

Objectif

- Transformer un code impératif en CUDA

```
# pragma cuda thread_loop ( i ) params (x ,y ,z , size )
void sum ( float *x , float *y , float *z , int size ) {
        for ( int i =0; i < size ; ++ i ) {
                z [ i ] = x [ i ] + y [ i ];
        }
}
```

```
__global__ void kernel ( float *x , float *y , float *z , int size ) {
        int gtid = ...
        if ( gtid < size ) {
                z [ i ] = x [ i ] + y [ i ];
        }
}
```

Analyse générale

Analyse de fonctions transformées en kernel

Les transformations à réaliser :

- Initialisation de la variable récupérée dans le thread_loop
- Remplacement de la boucle for en condition if
- Modification de l'entête de la fonction

Analyse générale

Exemple simple :

```
#pragma cuda thread_loop(i)
void sum(float *x, float *y, float *z, int size) {
        int i;
        for (i=0; i<size; ++i) {
                z[i] = x[i] + y[i];
        }
}
```

```
__global__ void kernel(float *x, float *y, float *z, int size) {
        int i = ((((blockIdx.x * gridDim.y + blockIdx.y) * gridDim.z + blockIdx.z) * blockDim.x + thread
        if (i < size) {
                z[i] = x[i] + y[i];
        }
}
```

Analyse générale

Exemple complexe :

```
#pragma cuda thread_loop(i) bloc_size(16,16)
void sum(float *x, float *y, float *z) {
        int i;
        for (i=4; i<size-2; i=i+3) {
                int j;
                for(j=0, j<size, j++)
                        z[i] = x[i] + y[j];
        }
}
```

Analyse générale

Eléments qui peuvent poser problème :

- Variable globale
- Parcours d'une boucle atypique
- Recherche de la boucle à paralléliser
- Fonction non void

Informations à extraire du code :

- Pragma
- Boucle for
- Déclaration de variables

Solutions envisagées

Les solutions :

- Analyseur syntaxique / lexical
- Compilateur gcc
- Regex

Solution retenue

Analyseur syntaxique / lexical

Simple d'utilisation, maitrisé par les membres de l'équipe.

Base de travail et correction

- 1ère version from scratch
- 2nd version basée sur l'implémentation yacc/lex de Jeff Lee
- 3ème version basée sur le projet "tc-parser" (github)

Analyse du pragma

- Définition du format
- Modification de l'arbre

#pragma cuda thread_loop(j) block_size(2,2) nbr_threads(16)

Recherche de la boucle à paralléliser

- Parcours de la fonction à CUDAiser
- Recherche du thread_loop dans les paramètre des boucles for

```
for(i=0; i<size; ++i) {
    ...
}
for(j=0; size>j; j++) {
    ...
}
```

Vérification de la portée des variables

- Accessibilité des variables

[WARNING] Variable(s) ”x” may not be reachable

Transformation de la fonction

- Modification de l'entête de la fonction
- Modification de l'initialisation de la variable sur la quelle itérer
- Modification de la boucle for vers une condition if

```
#pragma cuda thread_loop(j) block_size(2,2) nbr_threads(16)

int j = ((blockIdx.x * blockDim.x + threadIdx.x) * blockDim.y + threadIdx.y);
```

Wrapper de la fonction transformée

- Fonction qui appelle le kernel généré
- Récupération des paramètres contenus dans le pragma
- Affectation selon les paramètres s'ils existent

```
float *x_gpu;
float *y_gpu;
float *z_gpu;
int size_gpu;

cudaMalloc( (void**) &x_gpu, /*replace with size*/ );
cudaMalloc( (void**) &y_gpu, /*replace with size*/ );
cudaMalloc( (void**) &z_gpu, /*replace with size*/ );

/*insérer ici les cuda_memcopy*/

int nbr_thread_x = 2;
int nbr_thread_y = 2;
int nbr_block_x = (16 + 2*2 - 1 ) / 2*2;

kernel_sum <<< dim3(nbr_block_x) , dim3(nbr_thread_x, nbr_thread_y) >>> (x_gpu, y_gpu, z_gpu, size_gpu);

cudaFree( x_gpu );
cudaFree( y_gpu );
cudaFree( z_gpu );
```

Améliorations

- Parse complet du langage c
- Gestion plus complète de la boucle for
- Automatisation des allocations