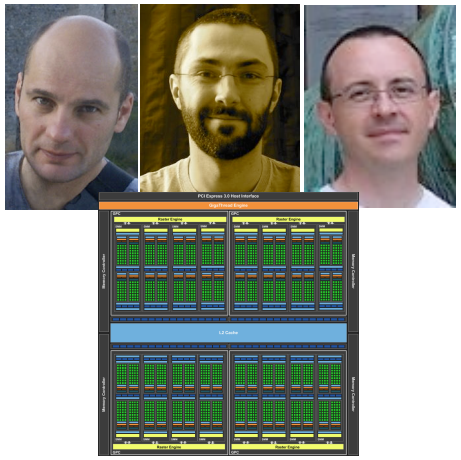
INSA-5IF

Programmation massivement parallèle sur GPU

Séance 4 : débogage et profiling

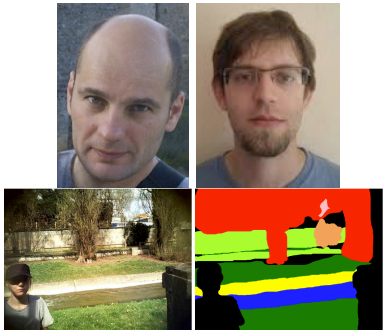
Christian Wolf, Eric Lombardi
INSA-Lyon, Dép. IF, LIRIS

Sem GPU

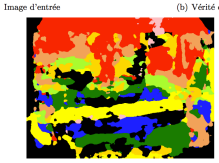


Sem Calcul P.

Projet dC Image



(a) Image d'entrée (b) Vérité de terrain



(c) Solution estimée

Projet dC Eval

Programmation massivement parallèle sur GPU

Christian Wolf	Cours	Introduction : architecture d'un GPU
Christian Wolf	Cours + TP	Programmation en CUDA : bases
Christian Wolf, Lionel Morel	Cours + TP	Programmation en CUDA : bases + Programmation en CUDA : mémoire partagé
Christian Wolf, Lionel Morel	Cours + TP	Programmation en CUDA : mémoire partagé
Christian Wolf, Eric Lombardi	Cours + TP	Débogage et optimisation sur GPU
Eric Lombardi	Cours + TP	Pattern : Multiplication matrice creuse - vecteur
Eric Lombardi	Cours + TP	Programmation Open-CL



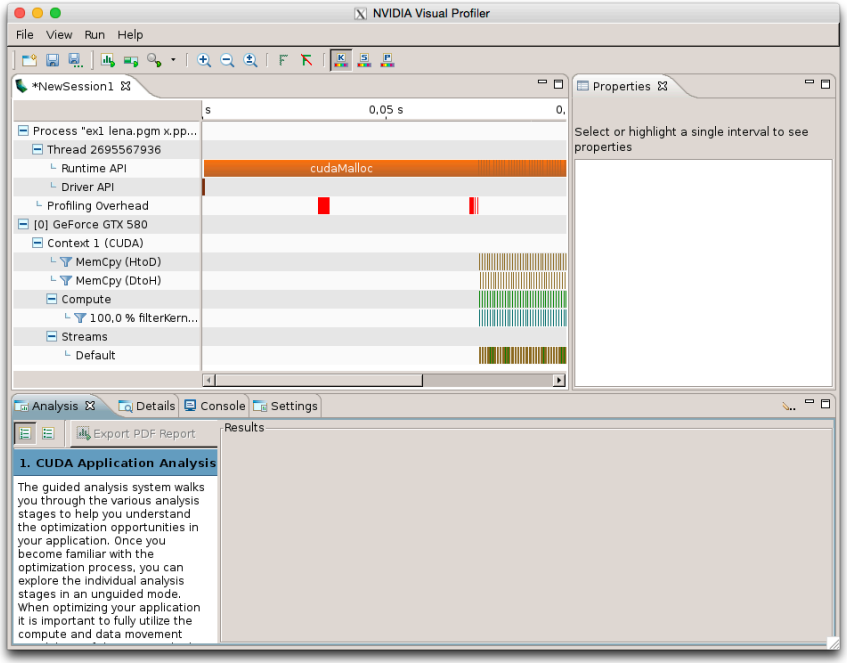
Sommaire

Débogage (cudo-gdb)

```
chris — ssh — 80×42
6.5 release
Portions Copyright (C) 2007-2014 NVIDIA Corporation
GNU gdb (GDB) 7.6.2
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/chris/chris/sem-gpu/cuda-tutorial-exos/ex1...done.
(cuda-gdb) break filterKernel
Breakpoint 1 at 0x403bf9: file ex1.cu, line 116.
(cuda-gdb) run
Starting program: /home/chris/chris/sem-gpu/cuda-tutorial-exos/./ex1 lena.pgm x.
ppm
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7fffe314b700 (LWP 16122)]
==========================================
Loaded image of size 512x512.
Time: CPU version: 0.948388
[New Thread 0x7fffe19e3700 (LWP 16128)]
[Switching focus to CUDA kernel 0, grid 1, block (0,0,0), thread (0,0,0), device
 0, sm 0, warp 0, lane 0]

Breakpoint 1, filterKernel<<<(1024,1,1),(256,1,1)>>> (
    _imarr=0xb03520000 "\243\242\242\242\244\243\241\237\233\233\234\235\235\236
\235\235\236\236\235\233\233\235\235\233\233\231\231\235\237\235\235\236\233\235
\236\237\237\236\241\246\246\243\240\237\237\242\244\243\243\244\243\241\242\241
\237\235\241\235\236\243\243\237\240\245\246\246\245\254\247\253\253\260\254\256
\255\254\254\250\245\247\242\236\241\243\235\231\230\226\222\216\211\205\202}xuj
e`\\YYZXTVVY[[[]XaZ[YWc^WU\\Z[_]a\\]___acahcgb_fefcedcdhjh`b``c`_feb^_b`^b^`^`he
`edbacdda^dddfjljglmmopmkk", _resarr=0xb03560000 "", rows=512, cols=512)
    at ex1.cu:117
117             int i=threadIdx.x + blockDim.x*blockIdx.x;
(cuda-gdb) step
118             if (i<rows*cols)
(cuda-gdb) p i
$1 = 0
(cuda-gdb)
```

Profiling (nvprof + nvvp)



Compilation en mode DEBUG

Ajouter les options –g –G

Par exemple :

```
nvcc `pkg-config --cflags opencv`
-g -G -o ex1 ex1.cu  `pkg-config
opencv --libs` -lstdc+
```

Cela va désactiver tout tentative d'optimisation du code, donc forcer –O0

Lancement de cuda-gdb

```
cuda-gdb -args <arguments-appel>
```

Par exemple :

```
cuda-gdb -args ./ex1 lena.pgm x.ppm
```

Gestion des points d'arrêt

Certaines commandes ne sont pas spécifiques à la version cuda

```
break <nomDeFonction>
break <nomFichier>:<numeroLigne>
delete <numeroBreakPoint>
```

Configurer ou supprimer un point d'arrêt. Le nom de fonction peut être un *kernel*. Dans ce cas, le point d'arrêt est actif pour tous *les threads.*

```
set cuda break_on_launch application
```

Demande un break point pour chaque lancement de *kernel*

```
run
```

Démarre le programme configuré

```
continue
```

Continuer après un arrêt

Navigation + inspection

```
step
next
```

Exécuter la prochaine ligne du code source. S'il s'agit d'une fonction, rentrer (« step ») ou pas « next ».

```
where
```

Affichage du lieu de l'exécution (fonction, ligne code source etc.)

```
print <variable>
p <variable>
```

Afficher la valeur d'une variable

```
p *<pointeur>@<nombreElements>
```

Affichage de plusieurs élements d'un tableau

Débogage de code parallèle

Pour déboguer un code parallèle on faire un « focus » sur un *thread* donné dans un *block* donné.

Si plusieurs kernels sont actif, on peut choisir un *kernel* donné.

```
cuda kernel <numero-kernel>
cuda block <numero-block>
cuda thread <numero thread>
```

Ces choix vont automatique sélectionner un numéro de warp (dépendant du thread) et de streaming-multiprocesseur (dépendant du block et du GPU). On peut forcer par :

```
cuda sm <numero-sm>
cuda warp <numero-warp>
```

Un changement de *warp* sélectionnera bien sur un nouveau *thread*.

Un changement de *SM* sélectionnera bien sur un nouveau *block*.

Renseignements

Sans arguments, ces instruction renseignent sur le focus actuel :

```
cuda kernel
cuda thread
…
cuda kernel block thread warp
```

Information sur tous les threads :

```
info cuda threads
```

Rappel : warp drive

- Chaque SM organise les *thread* en groups de 32 appelés « *warp* » (*warp* <> *block* !!)
- Les *threads* d'un *warp* sont exécutés en parallèle
- La même instruction est exécutée par tous les threads, généralement sur des données différentes
- SIMT = *Single Instruction Multiple Data*
- Exemple de la multiplication de matrices :

```
1 void mult_kernel_simple(int c, int r)
2 {
3     output[r*mxWidth + c] = 0.0f;
4     for( int k = 0; k < mxWidth; k++)
5         output[r*mxWidth + c] += mx1[r*mxWidth + k] * mx2[k*mxWidth + c];
6 }
```

- Chaque thread du warp exécute une multiplication
- Les opérandes sont différents

Navigation en parallèle

```
step
next
```

Sur CPU, un pas est fait par thread.

Sur GPU, un pas est fait pour tous les threads du warp actuel, mais pas pour les autres threads.

Si on veut avancer tous les threads : mettre un breakpoint.

Exception: un pas sur une instruction *__syncthreads()*.

Dans ce cas, un point d'arrêt implicite est mis immédiatement après, et tous les thread continuent jusqu'à ce point.

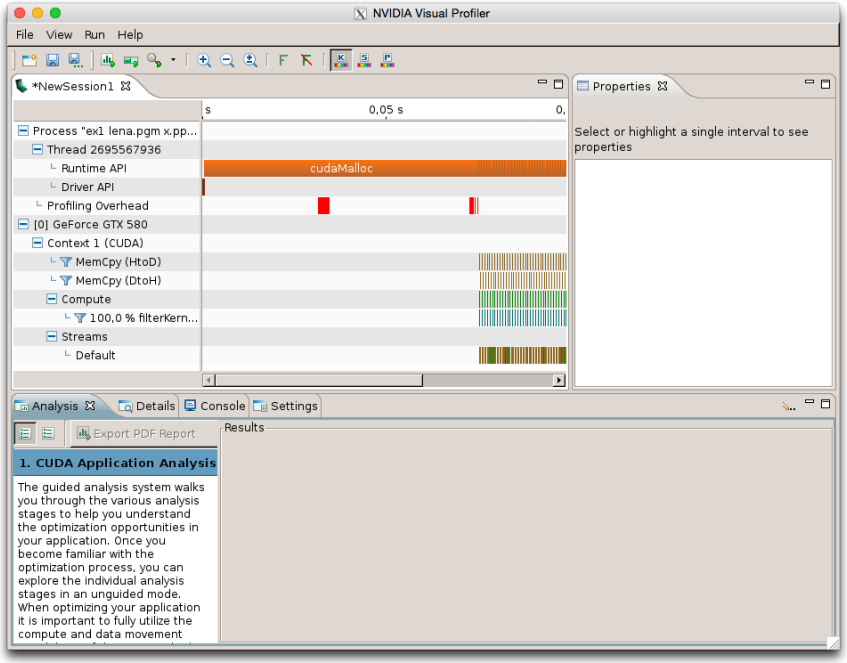
Sommaire

Débogage (cuda-gdb)

```
chris — ssh — 80×42
6.5 release
Portions Copyright (C) 2007-2014 NVIDIA Corporation
GNU gdb (GDB) 7.6.2
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/chris/chris/sem-gpu/cuda-tutorial-exos/ex1...done.
(cuda-gdb) break filterKernel
Breakpoint 1 at 0x403bf9: file ex1.cu, line 116.
(cuda-gdb) run
Starting program: /home/chris/chris/sem-gpu/cuda-tutorial-exos/./ex1 lena.pgm x.
ppm
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7fffe314b700 (LWP 16122)]
==========================================
Loaded image of size 512x512.
Time: CPU version: 0.948388
[New Thread 0x7fffe19e3700 (LWP 16128)]
[Switching focus to CUDA kernel 0, grid 1, block (0,0,0), thread (0,0,0), device
 0, sm 0, warp 0, lane 0]

Breakpoint 1, filterKernel<<<(1024,1,1),(256,1,1)>>> (
    _imarr=0xb03520000 "\243\242\242\242\244\243\241\237\233\233\234\235\235\236
\235\235\236\236\235\233\233\235\235\233\233\231\231\235\237\235\235\236\233\235
\236\237\237\236\241\246\246\243\240\237\237\242\244\243\243\244\243\241\242\241
\237\235\241\235\236\243\243\237\240\245\246\246\245\254\247\253\253\260\254\256
\255\254\254\250\245\247\242\236\241\243\235\231\230\226\222\216\211\205\202}xuj
e`\\YYZXTVVY[[[]XaZ[YWc^WU\\Z[_]a\\]___acahcgb_fefcedcdhjh`b``c`_feb^_b`^b^`^`he
`edbacdda^dddfjljglmmopmkk", _resarr=0xb03560000 "", rows=512, cols=512)
    at ex1.cu:117
117             int i=threadIdx.x + blockDim.x*blockIdx.x;
(cuda-gdb) step
118             if (i<rows*cols)
(cuda-gdb) p i
$1 = 0
(cuda-gdb)
```

Profiling (nvvp)



Nvidia profiler

Deux outils de profiling en mode console (nvprof) et avec GUI (nvvp)

```
chris — ssh — 80×29
chris exo1 $ nvprof ./ex1 lena.pgm x.ppm
==20159== NVPROF is profiling process 20159, command: ./ex1 lena.pgm x.ppm
Loaded image of size 512x512.
Time: CPU version: 0.944895
Time: GPU version: 0.334481
Program terminated correctly.
==20159== Profiling application: ./ex1 lena.pgm x.ppm
==20159== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 63.76%  23.150ms       100  231.50us  231.14us  232.07us  filterKernel(unsigned
 char*, unsigned char*, int, int)
 19.63%  7.1289ms       100  71.289us  65.119us  77.279us  [CUDA memcpy DtoH]
 16.61%  6.0307ms       100  60.307us  59.040us  61.984us  [CUDA memcpy HtoD]

==20159== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 58.54%  91.491ms       200  457.45us  6.5450us  78.769ms  cudaMalloc
 34.33%  53.665ms       200  268.32us  68.984us  483.71us  cudaMemcpy
  5.84%  9.1248ms       200  45.623us  9.6270us  121.50us  cudaFree
  0.98%  1.5365ms       100  15.364us  14.854us  29.848us  cudaLaunch
  0.15%  227.41us        83  2.7390us     239ns  93.323us  cuDeviceGetAttribute
  0.08%  129.45us       400     323ns     244ns  4.5750us  cudaSetupArgument
  0.04%  65.070us       100     650ns     620ns  2.1320us  cudaConfigureCall
  0.02%  31.415us         1  31.415us  31.415us  31.415us  cuDeviceTotalMem
  0.02%  26.540us         1  26.540us  26.540us  26.540us  cuDeviceGetName
  0.00%  1.9600us         2     980ns     493ns  1.4670us  cuDeviceGetCount
  0.00%     829ns         2     414ns     329ns     500ns  cuDeviceGet
chris exo1 $
```