

Initiation à la GPGPU

Introduction

David Odin

Forma3Dev pour CPE-Lyon

2010

QU'EST-CE QUE LA GPGPU ?

- **General Programing on Graphic Processing Units.**
- Utilisation de la puissance des cartes graphiques pour autre choses qu'afficher des triangles.

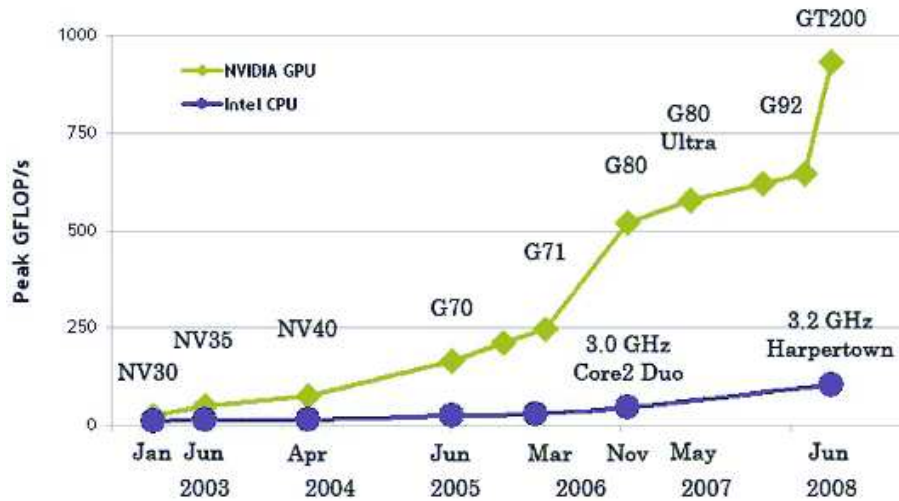
PROGRAMMATION PARALLÈLE

- Idée assez vieille, au moins 30 ans (transputers)
- Implémenté via MMX, SSE sur les processeurs x86 depuis des années.
- Activec sur PowerPC, Paired Single sur Wii, etc.
- **Single Instruction on Multiple Data.**

POURQUOI SUR GPU ?

- Les GPU sont SIMD par nature (*fragment shader*).
- Shaders unifiés.
- GPGPU commence à devenir mature (standardisation).
- CPU = 8 cœurs d'exécutions max.
- GPU = 1600 cœurs pour la AMD Radheon 5870 (comparable pour NVidia).

COMPARAISON CPU / GPU



LIMITATIONS

CPU \neq GPU.

- Beaucoup de données en même temps, oui, mais un seul programme à la fois.
- Impossibilité d'écrire et de lire les mêmes données en même temps.
- Pas applicable à tous les algorithmes.

TRAITEMENTS D'IMAGE SIMPLES

- Traitements les plus simple possible.
- Noir & blanc.
- Sépia.
- Ajustement de contraste.
- Ajustement de luminosité.
- Seuillage.

CONVOLUTIONS

- Flou.
- Érosion / Dilatation.
- Post-effects :
 - DoF (profondeur de champ),
 - HDR (simulation de conditions de lumière extrêmes)

PLUS AMBITIEUX

- Calculs de pliage de protéines.
- Équation aux dérivées partielles.
- Calculs météo.
- Mécanique des fluides.
- Cryptage / décryptage.
- Détection de virus.

MISE EN ŒUVRE

- OpenGL 2.1 : fragment shader et FBO.
- OpenGL 3.0 : transform feedback
- OpenCL : Computing library

CONSTAT

- Donnée de base (sortie) : pixel/fragment.
- Programme identique pour beaucoup de données : *Fragment shader*.
- Données d'entrées :
 - Variables *uniform*,
 - Variables *varying*,
 - Textures 1D, 2D, 3D, etc.

EXEMPLE : IMPLÉMENTATION DU FLOU

À partir d'une image stockée dans une texture, on affiche un *quad* de la même taille à l'écran avec le *fragment shader* suivant :

```
uniform sampler2D unite_texture;
uniform vec2      taille_texture;

void main(void)
{
    vec2 offset = vec2 (1.0) / taille_texture;
    vec2 tex_coord = gl_TexCoord[0].xy;

    vec4 color = texture2D (unite_texture, tex_coord + offset * (-1.0, -1.0));
    color += texture2D (unite_texture, tex_coord + offset * (-1.0, 0.0));
    color += texture2D (unite_texture, tex_coord + offset * (-1.0, 1.0));
    color += texture2D (unite_texture, tex_coord + offset * (0.0, -1.0));
    color += texture2D (unite_texture, tex_coord + offset * (0.0, 0.0));
    color += texture2D (unite_texture, tex_coord + offset * (0.0, 1.0));
    color += texture2D (unite_texture, tex_coord + offset * (1.0, -1.0));
    color += texture2D (unite_texture, tex_coord + offset * (1.0, 0.0));
    color += texture2D (unite_texture, tex_coord + offset * (1.0, 1.0));

    gl_FragColor = color / 9.0;
}
```

FBO

- Possibilité de dessiner dans une texture.
- Peut contenir 4 textures : ajoute des données de sorties !
- Peut contenir une "texture de profondeur".
- Peut contenir des textures de flottant.
- Se comporte comme un *framebuffer* "normal".
- Utilisation en "Ping-Pong".

IMPLÉMENTATION DES FBO

```
GLuint fbo, texture;

glGenFramebuffers (1, &fbo);
glBindFramebuffer (GL_FRAMEBUFFER, fbo); // On dessine maintenant dans fbo.

glGenTextures (1, &texture);
glBindTexture (GL_TEXTURE_RECTANGLE, texture);
glTexImage2D (GL_TEXTURE_RECTANGLE, 0, GL_RGBA, 800,600, 0,
              GL_RGBA, GL_UNSIGNED_BYTE, NULL);
glFramebufferTexture2D (GL_FRAMEBUFFER,
                       GL_COLOR_ATTACHMENT0,
                       GL_TEXTURE_RECTANGLE, texture, 0);
check_framebuffer_status ();

glBindFramebuffer (GL_FRAMEBUFFER, 0); // On dessine de nouveau normalement.
```

PRÉSENTATION

- Standard ouvert pour le calcul sur les GPU.
- Conçu pour s'exécuter sur GPU, CPU, SPU, DSP, etc.
- Utilisable sur GPU "compatible OpenGL 3.x".
- Langage "parallèle" semblable à GLSL ou au C++.

ABSTRACTION

OpenCL prend en charge :

- La gestion de la mémoire.
- La gestion des textures.
- L'écriture et la lecture des données.
- La répartition de la charge entre les cœurs du GPU.
- L'utilisation de MMX, SSE, AltiVec sur CPU.

Exemple d'addition de vecteur en OpenCL :

```
__kernel void square (__global float *input1,
                    __global float *input2,
                    __global float *output,
                    unsigned int count)
{
    int i = get_global_id (0); // Le rang de notre processeur
    if (i < count)
        output[i] = input1[i] + input2[i];
}
```

- <http://gpgpu.org/>
- <http://www.khronos.org/opencv/>