Prototype de calcul parallèle sur carte graphique

Cinquièmes Journées Informatique de l'IN2P3 et du DAPNIA

Emmanuel Hornero, LPNHE

Prototype de calcul parallèle sur carte graphique

- Pourquoi les cartes graphiques ? Quelle idée bizarre… et pour quoi faire?

GPU : Graphics Processing Units

GPGPU : General-Purpose Computing on Graphics Processing Units

Prototype de calcul parallèle sur carte graphique

- Le GPGPU ou comment utiliser la carte graphique comme coprocesseur...
- Fonctionnement du GPU
- Programmation GPGPU
- Prototype de module de calcul parallèle par GPU
- Conclusions
- Evolutions et perspectives

Le GPGPU ou comment utiliser la carte graphique comme coprocesseur...

Un peu d'histoire sur le « Graphic Processor Unit » :

- 1991 :
  - « S3 graphics » introduit la 1ère puce accélératrice 2D le S3 86C911 et clonée à plusieurs reprises
- Fin des années 90 :
  - Arrivée de Direct X version 8 et d'OpenGL
  - Les GPU ajoutent un « Shading » programmable
- 2003 :
  - Introduction de la Nvidia GeForce Fx (NV30)
  - les pixels/vertex shaders deviennent aussi flexibles qu'un CPU
- 2005 :
  - Les GPU prennent une partie du calcul destiné au CPU : GPGPU

Le GPGPU ou comment utiliser la carte graphique comme coprocesseur...

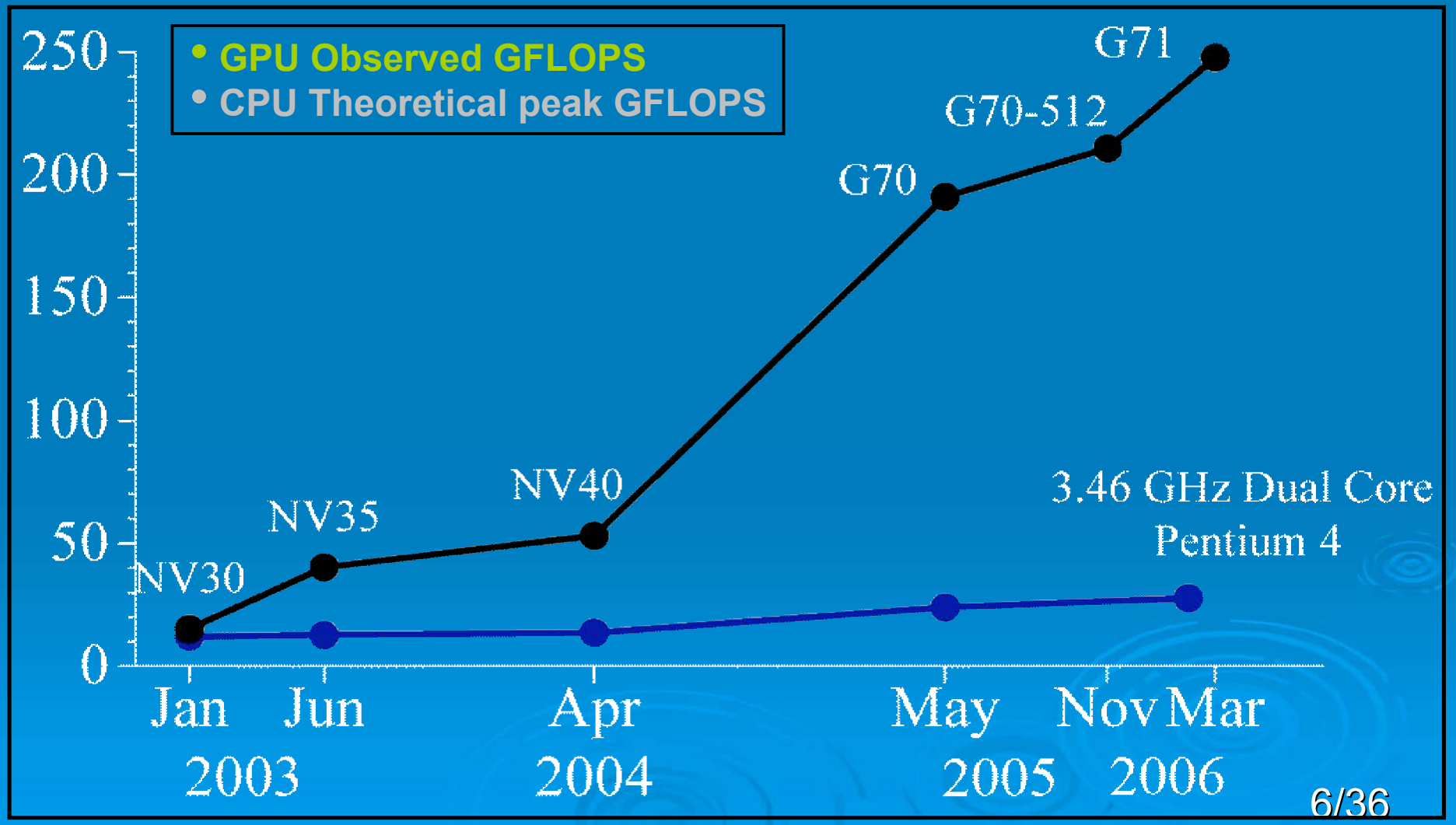
- Point de départ : l'industrie des jeux
  - Course à la puissance de calcul pour un coût « grand publique »
- Rapport Puissance / Coût élevé :

Nvidia GeForce 6800 Ultra
En juin 2005 :
120 GFLOPS
417$

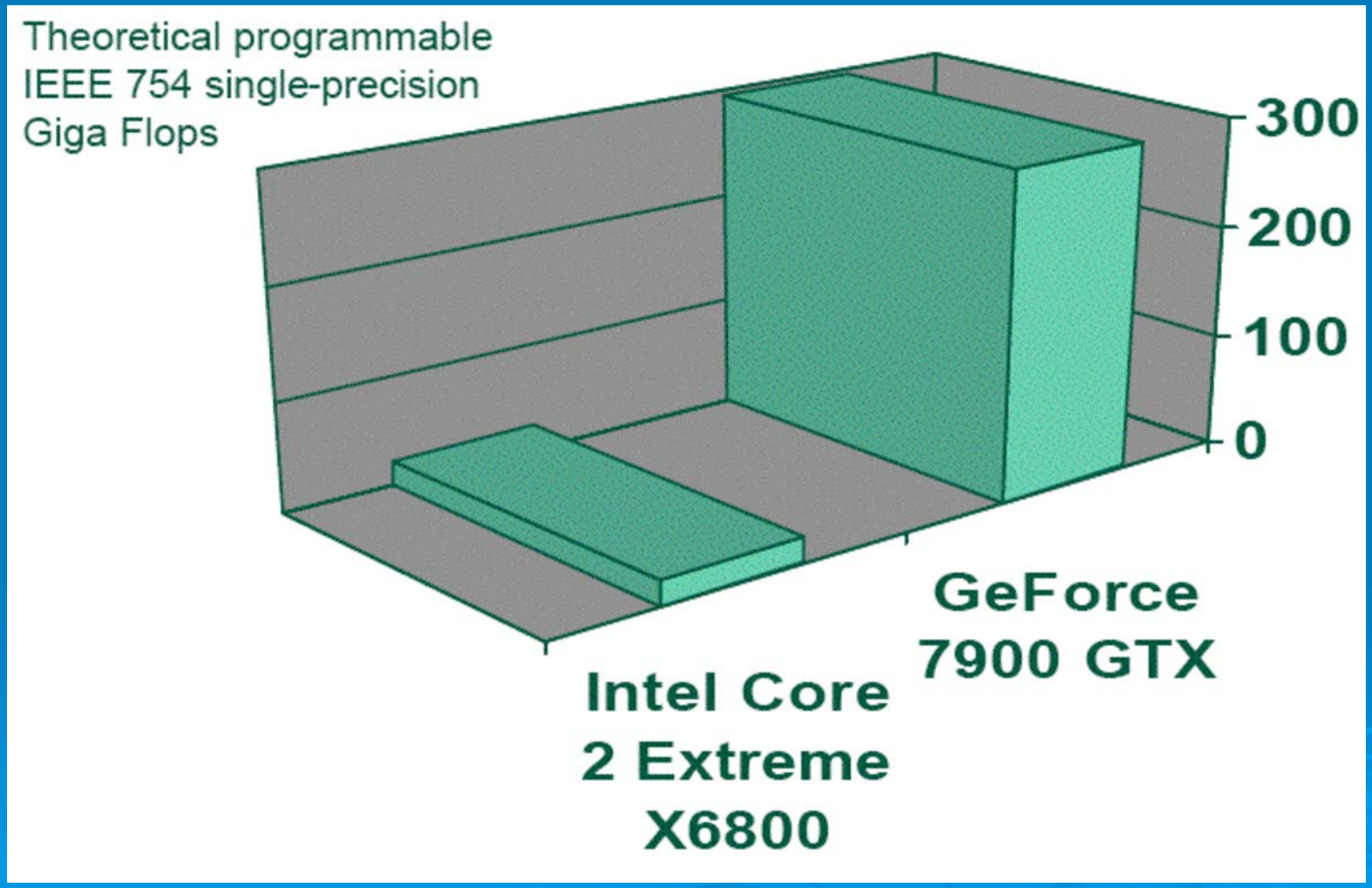
ATI X800 XT
En juin 2005 :
63 GFLOPS
447$

Pentium IV 3.7GHz
En juin 2005 :
14.8 GFLOPS
500$

Le GPGPU ou comment utiliser la carte graphique comme coprocesseur...



Le GPGPU ou comment utiliser la carte graphique comme coprocesseur...



Le GPGPU ou comment utiliser la carte graphique comme coprocesseur...

- Les GPU modernes possèdent :
  - une Unité de rendu composée d'unités de calcul pour accélérer les calculs géométriques
    - ces unités de calculs opèrent en parallélisme SIMD (Single Instruction on Multiple Data) ou en MIMD (Multiple Instruction on Multiple Data)
  - des Shader programmables pour la manipulation de vertex (sommets), textures, pixels

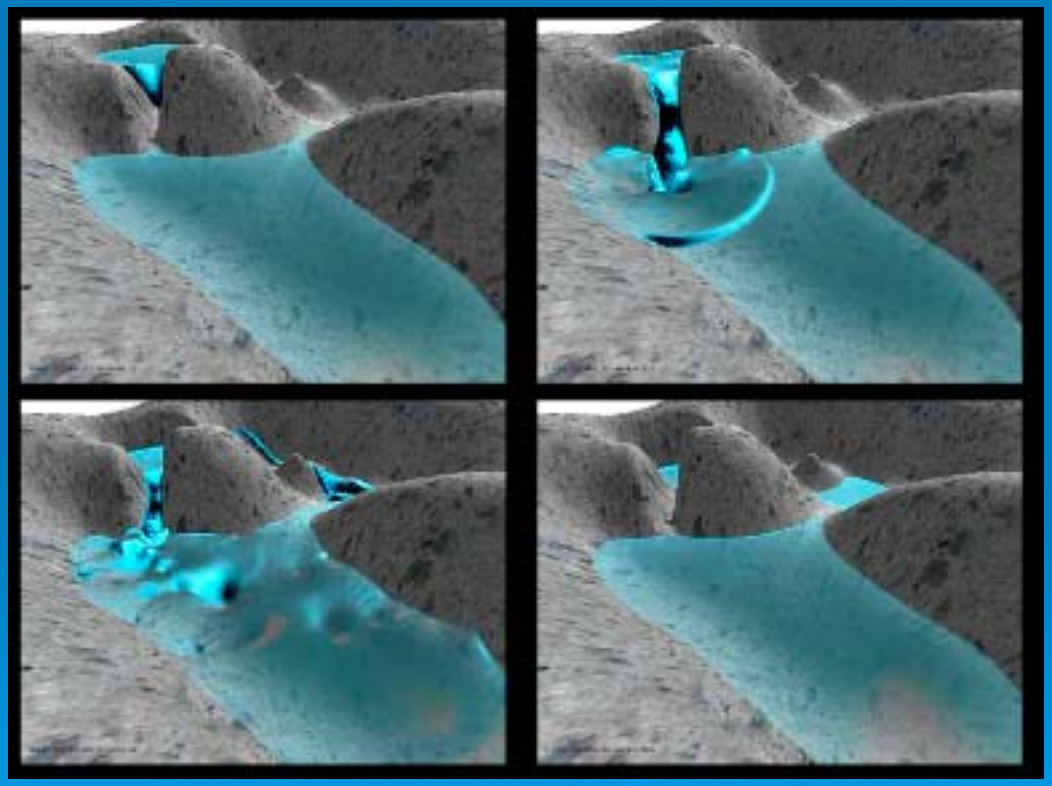
Le GPGPU ou comment utiliser la carte graphique comme coprocesseur...

- Le GPGPU (General-Purpose Computing on Graphics Processing Units) exploite ces unités de calcul programmables pour effectuer des tâches génériques :
  - Un GPU = un processeur SIMD
  - Une texture = un tableau d'entrée
  - Une image = un tableau de sortie
- Les applications types du GPGPU :
  - Algèbre linéaire
  - Résolution de systèmes d'équations à N inconnues
  - Calcul par éléments finis
  - Transformées de Fourier (FFT)
  - Tri

Le GPGPU ou comment utiliser la carte graphique comme coprocesseur...

- Visual Simulation of Shallow-Water Waves

(Septembre 2005)



✓NVIDIA Geforce 7800 GTX

✓2.8 GHz Intel Xeon (EM64T)

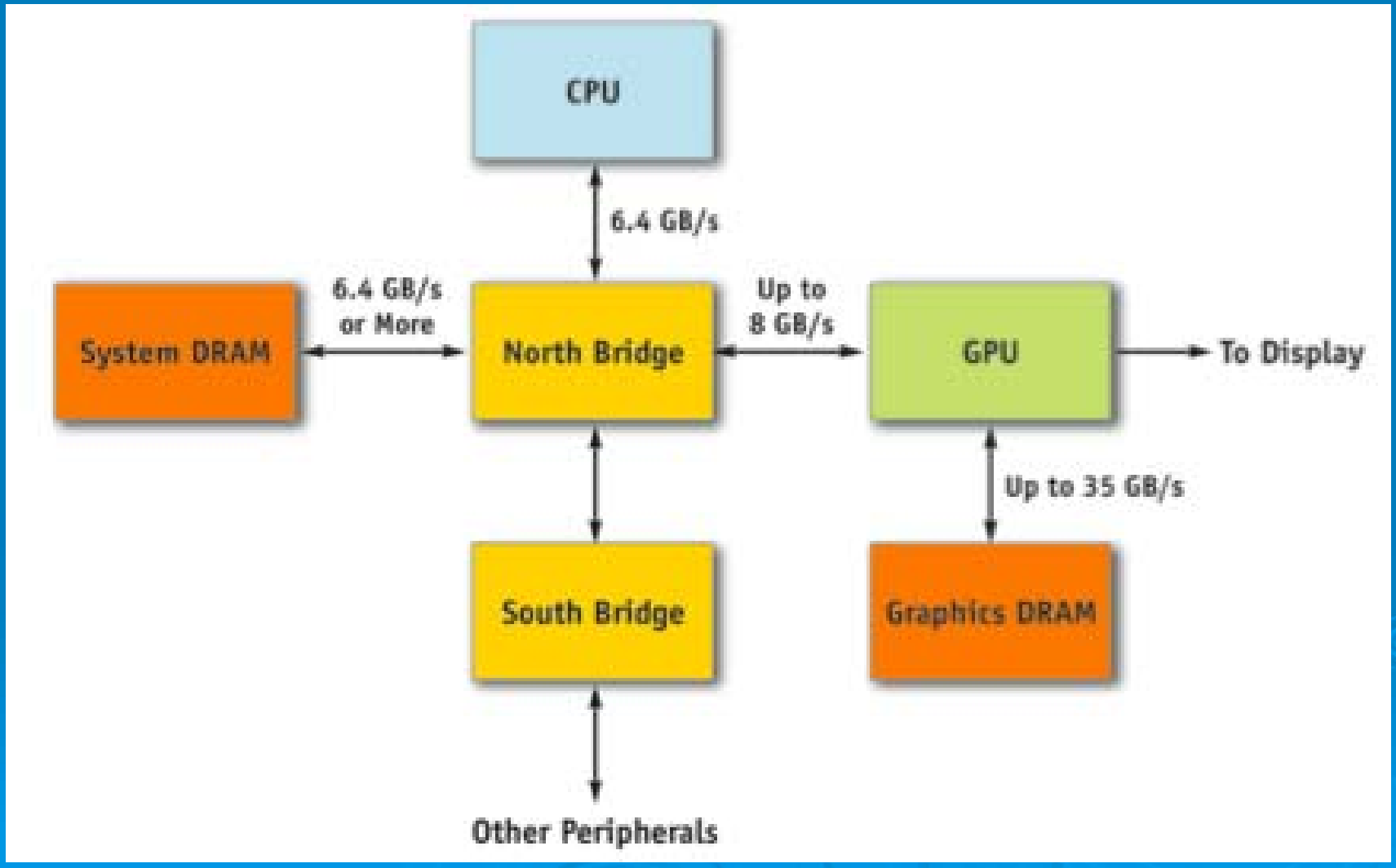
	Lax–Friedrichs		
N	CPU [ms]	GPU [ms]	ν
128	2.22	0.23	9.53
256	9.09	0.46	19.8
512	37.1	1.47	25.2
1024	148	5.54	26.7

Algorithme basé sur les équations différentielles partielles (EDP)

T.R. Hagen, J.M. Hjelmervik, K.–A. Lie, J.R. Natvig, M. Ofstad Henriksen
(SINTEF ICT, Applied Math., Oslo, Norway)

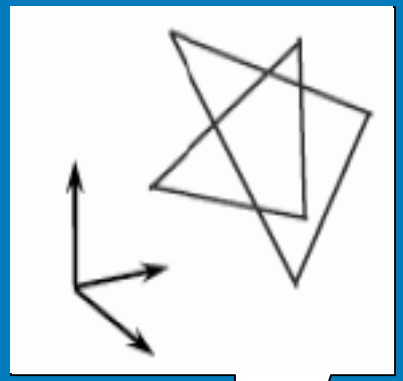
Fonctionnement du GPU

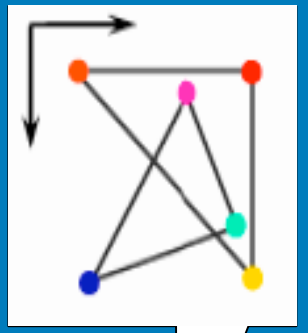
Place du GPU au sein de la machine

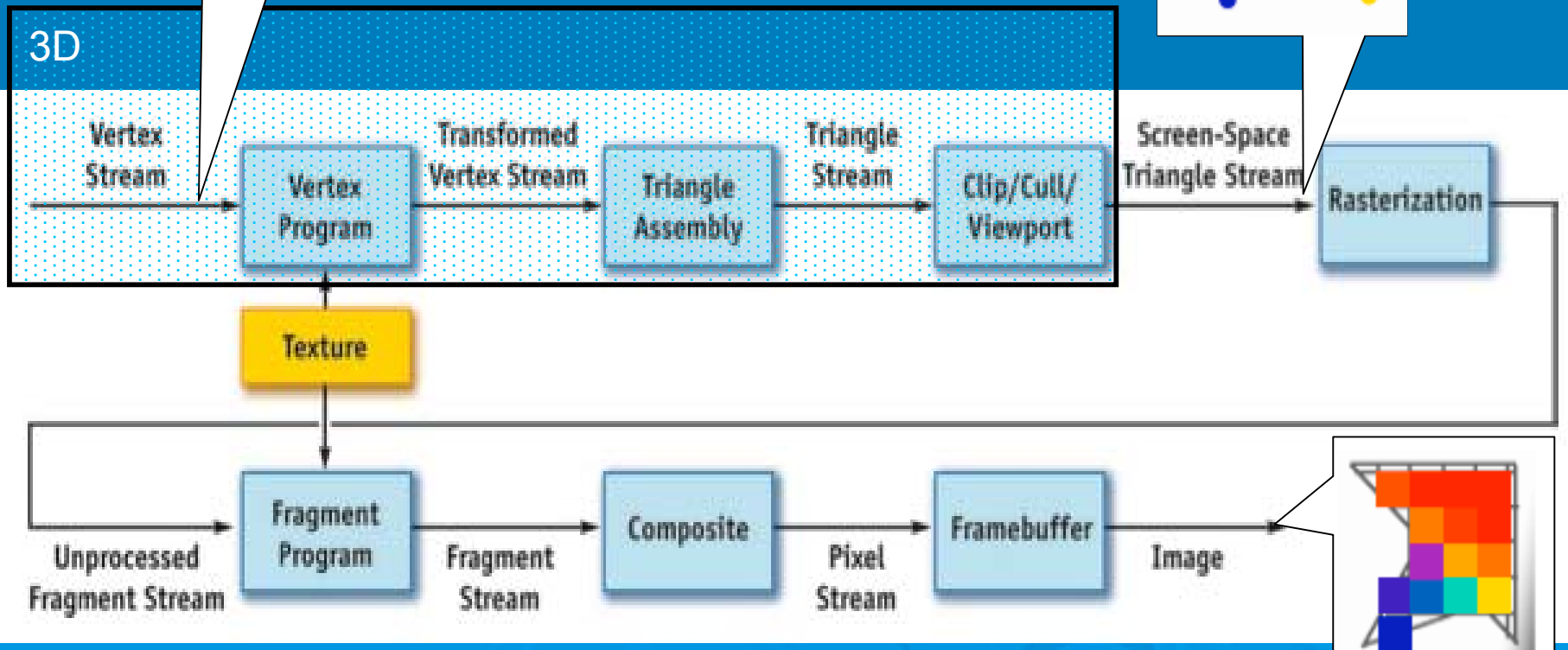


Fonctionnement du GPU

Le pipeline graphique

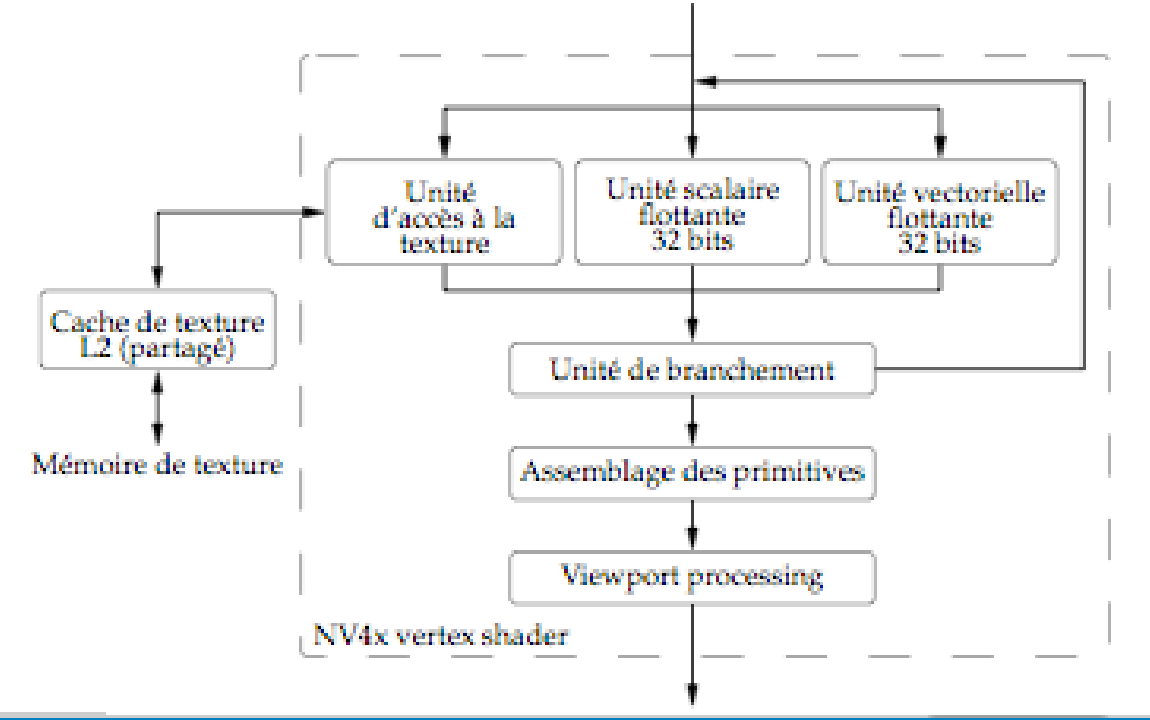






Fonctionnement du GPU

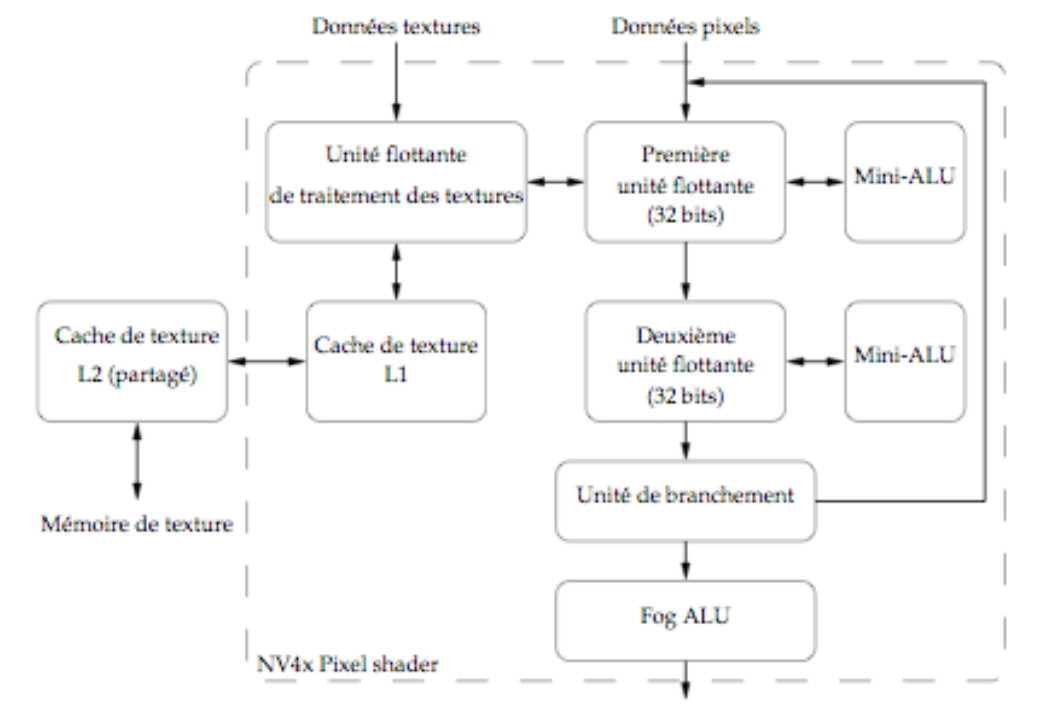
Le vertex shader



- Fonctionne en MIMD (Multiple Instruction on Multiple Data)
- Actuellement 8 vertex shader en parallèle
- A chaque cycle d'horloge :
  - Une opération Multiply and Accumulate (MAD) sur 4 triplés dans l'unité vectorielle.
  - Une opération special dans l'unité scalaire (exp, log, cos, sin, 1/x, $1/\sqrt{x}$)

Fonctionnement du GPU

Le fragment shader



- Fonctionne en SIMD (Single Instruction on Multiple Data)
- Actuellement 24 fragment shaders en parallèle
- A chaque cycle d'horloge :
  - 4 opérations MAD dans chaque unité flottante ⇒ 8 ops / cycle.
- Calcule sur des quadruplés RGBA de flottants 32 bits.

Fonctionnement du GPU

Précision des calculs sur GPU

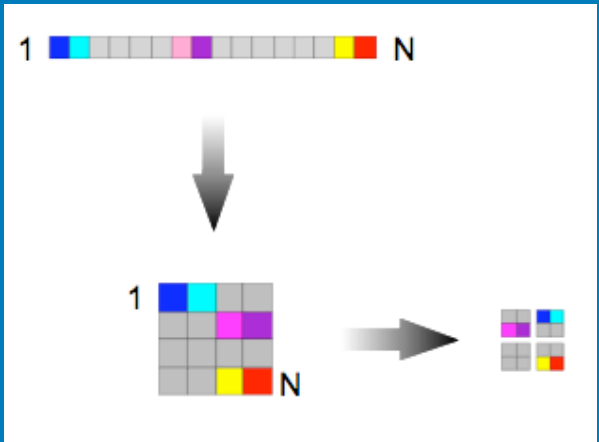
Name	Sign	Exponent	Mantissa	Largest Values	Smallest Values	Whole Number Range[1]	Supports Specials (NaN, Inf, etc.)
NVIDIA 16-bit	15 (1)	14:10 (5)	9:0 (10)	±65,504	$\pm 2^{-14} \cong 10^{-5}$ ($\pm 2^{-24}$ with denorms)	±2048	Yes
ATI 16-bit	15 (1)	14:10 (5)	9:0 (10)	±131,008	$\pm 2^{-15} \cong 10^{-5}$	±2048	No
ATI 24-bit	23 (1)	22:16 (7)	15:0 (16)	$\pm \sim 2^{64} \cong 10^{19}$	$\pm \sim 2^{-62} \cong 10^{-19}$	±131,072	No
NVIDIA 32-bit (IEEE 754)	31 (1)	30:23 (8)	22:0 (23)	$\pm \sim 2^{128} \cong 10^{38}$	$\pm \sim 2^{-126} \cong 10^{-38}$	±16,777,216	Yes

1. This is the contiguous zero-centered range of exactly representable whole numbers.

- Capable de calculer sur l'ensemble du pipeline graphique en flottants 32 bits identiques au standard IEEE-754
- Mais incapable à ce jour de travailler sur des entiers 32 bits (reste néanmoins les 23 bits de mantisse des flottants 32 bits)

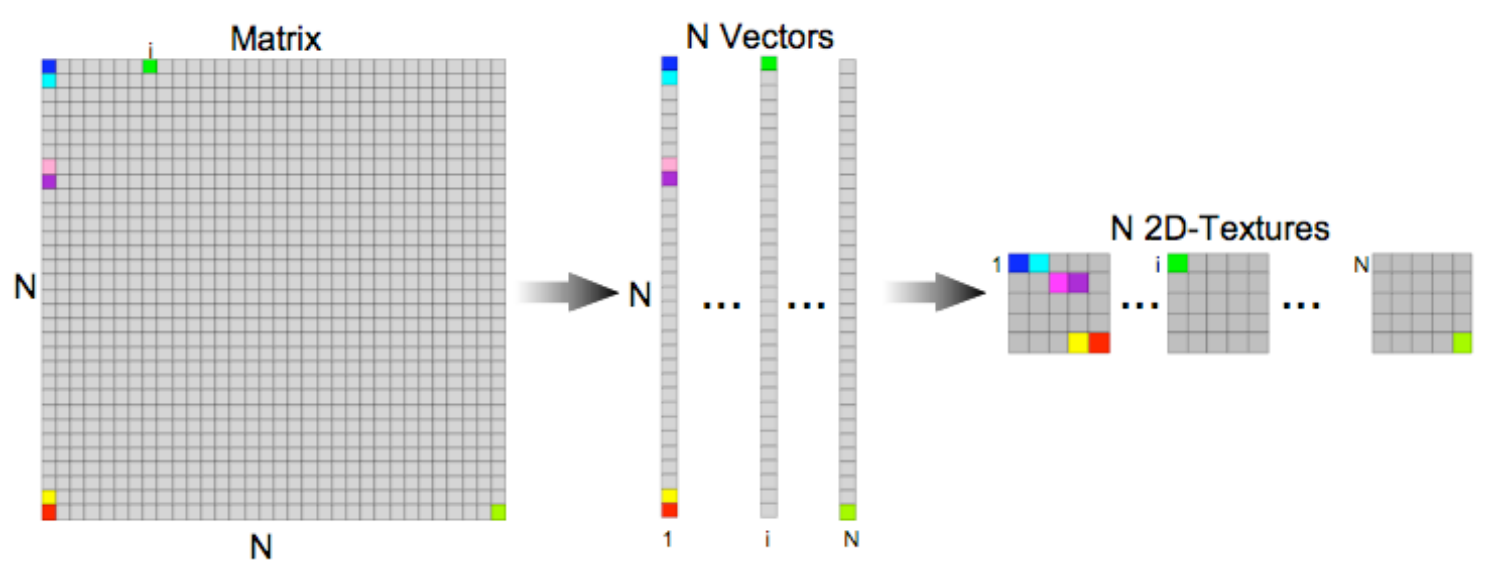
Programmation GPGPU

Une texture = un tableau d'entrée



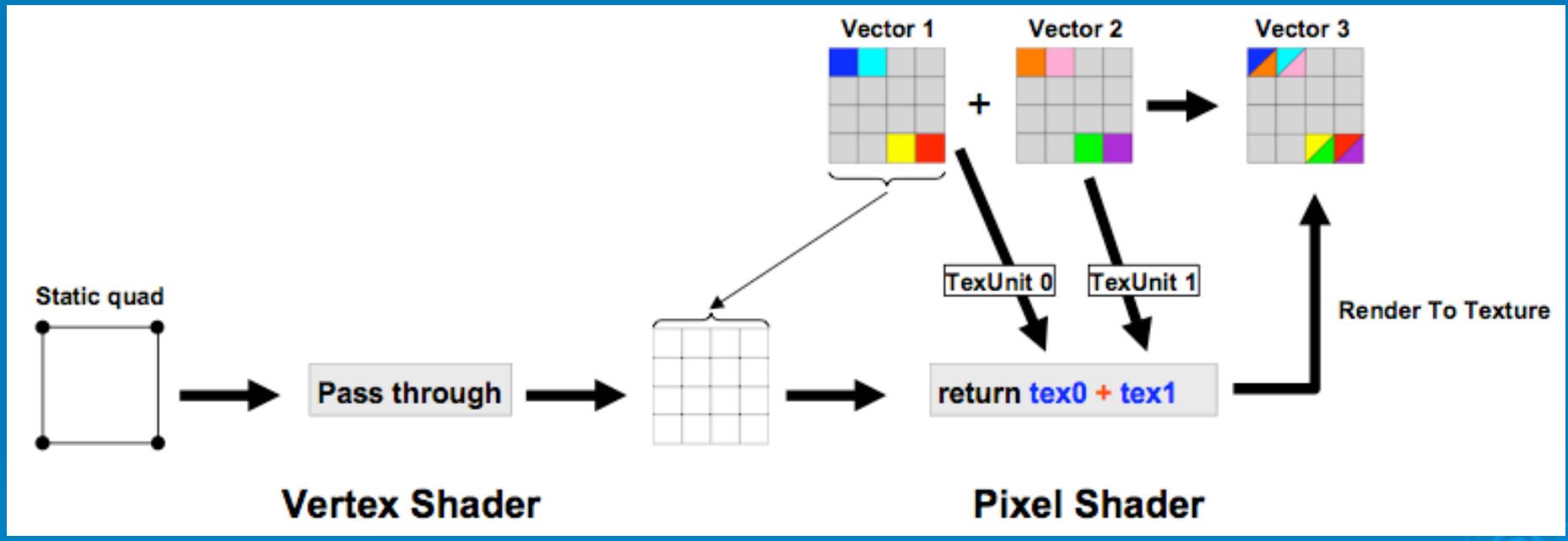
Représentation d'un vecteur

Représentation d'une matrice dense



Programmation GPGPU

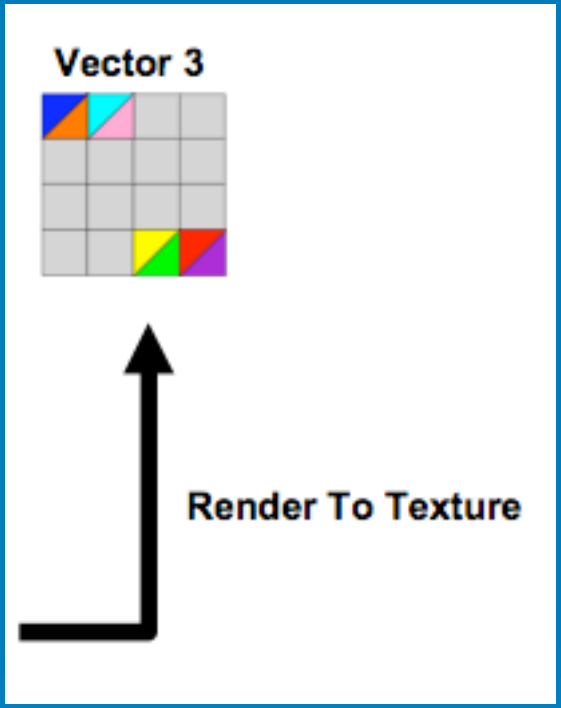
Un GPU = un processeur SIMD



Programmation GPGPU

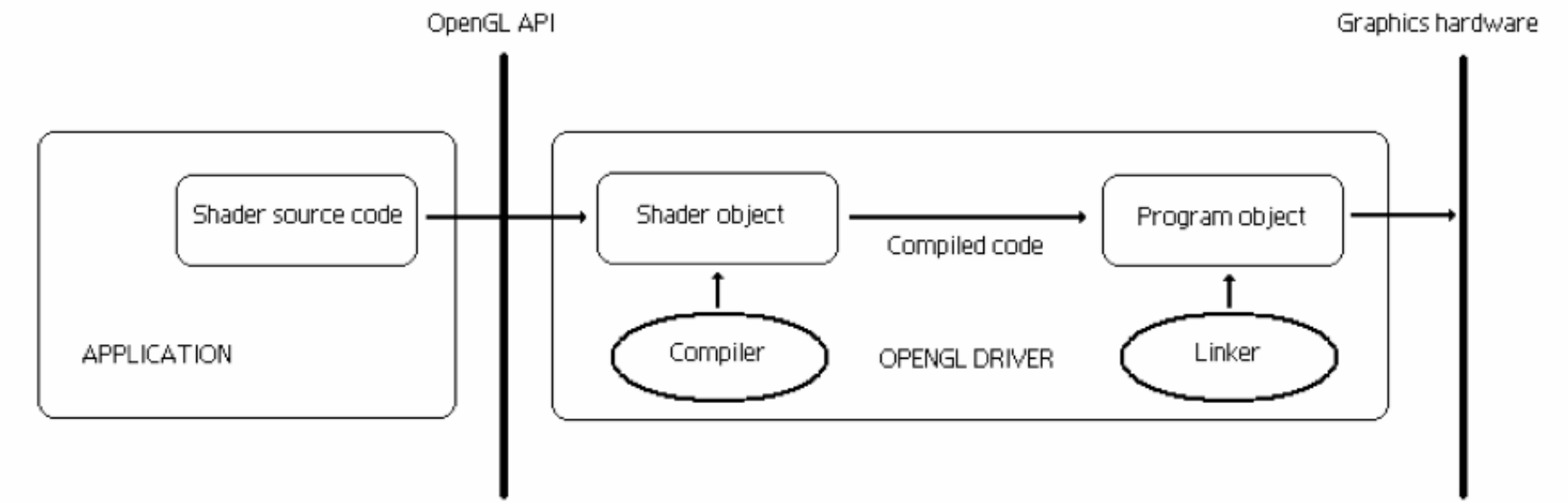
Une image = un tableau de sortie

- Le résultat est le rendu effectué dans le frame buffer
- FBO ou Frame Buffer Object : permet d'effectuer le rendu dans une texture
- La texture est transférée et le résultat est stocké dans la RAM de la machine
  - Ordres d'idées (ATI : X1800XT)
    - Download (GPU -> CPU): 900 MB/s
    - Upload (CPU -> GPU): 1.4 GB/s



Programmation GPGPU

Langages de programmation évolués pour GPU



- High Level Shading Language (HLSL, Microsoft)
  - Fait partie de l'API DirectX, Windows uniquement, hardware indépendant
- "C for graphics"(Cg, NVIDIA)
  - Compile sous OpenGL et DirectX, platforme et hardware indépendant
- OpenGL Shading Language (GLSL, ARB)
  - Fait partie de l'API OpenGL, platforme et hardware indépendant

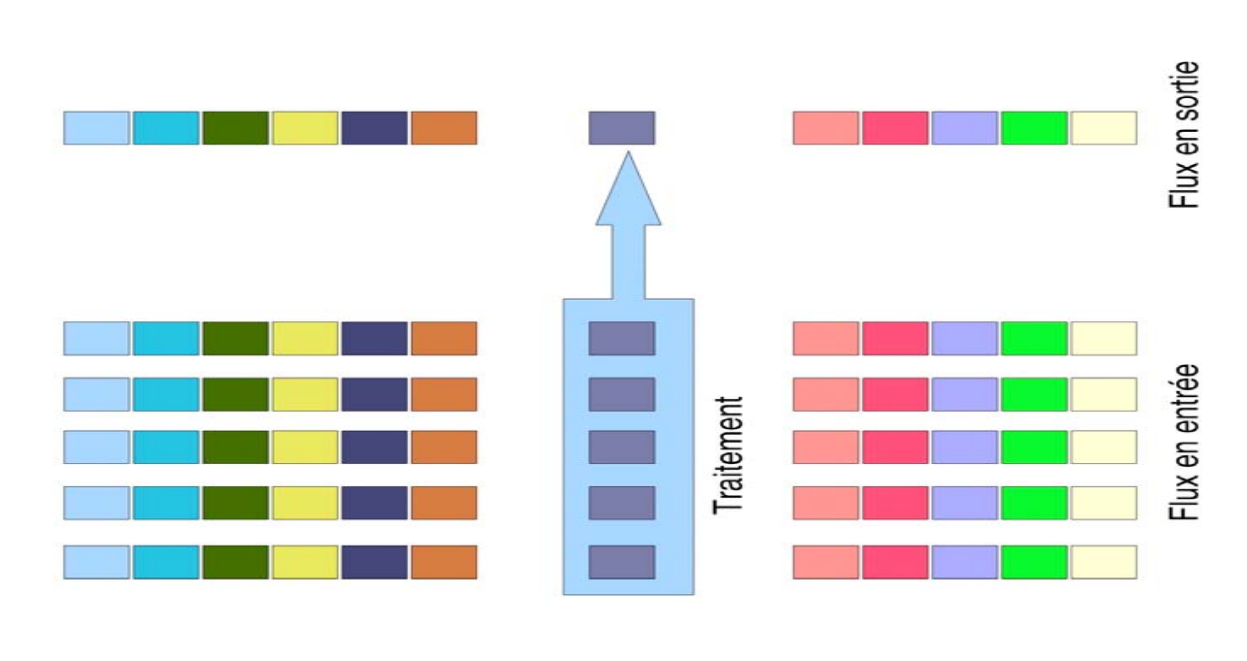
Programmation GPGPU

« C for graphics » (Cg, NVIDIA)

- Langage très proche du C
- Définitions de fonctions, types similaires au C (vecteurs, tableaux, etc…), boucles
- Fonctions spécialisées performantes : mathématiques, géométrie, …
- A venir, des fonctionnalités C++:
  - Classes
  - Templates
  - Surcharge d'opérateurs
  - Mais pas de foncionnalités telles que : Run-Time Type Information (RTTI), new/delete, ou les exceptions

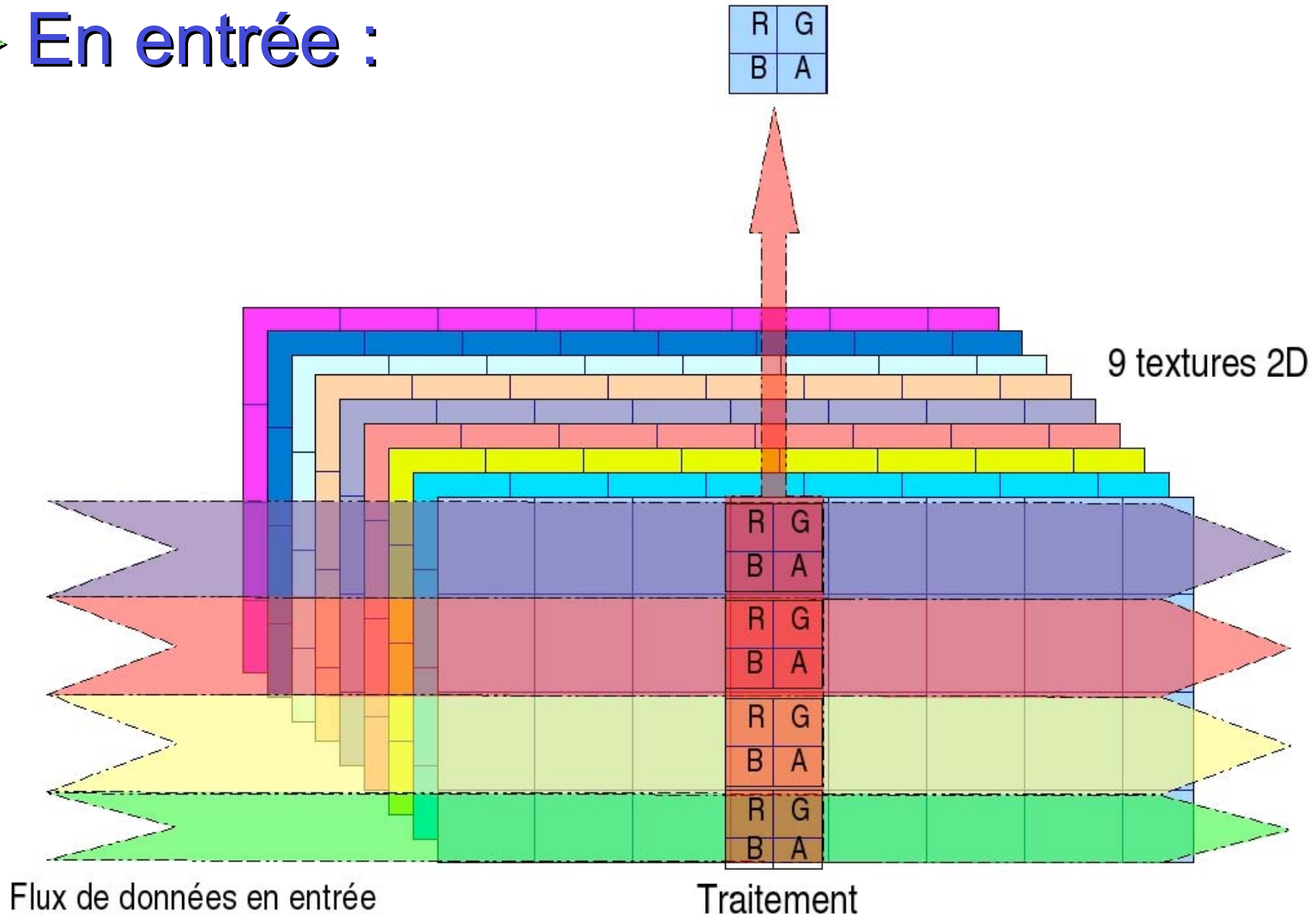
Prototype de module de calcul parallèle par GPU

- Objectifs de ce module de calcul :
  - Accepter en entrée plusieurs centaines de flux de données de longueur indéterminée
  - Pour chaque position dans le flux de données, effectuer le même traitement sur tous les flux de données
  - Rendre un flux de données unique en sortie



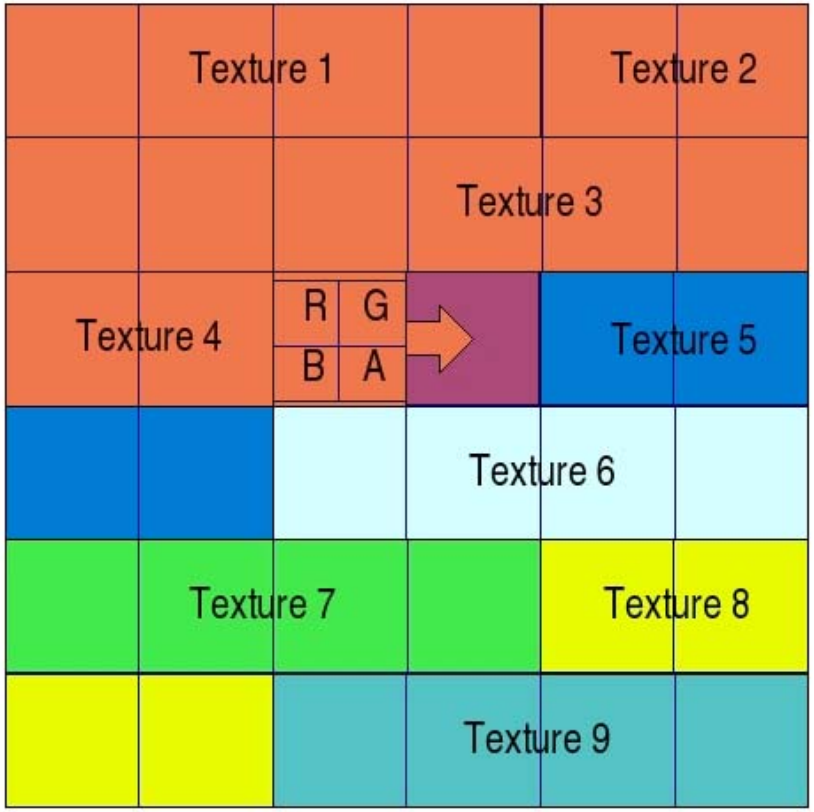
Prototype de module de calcul parallèle par GPU

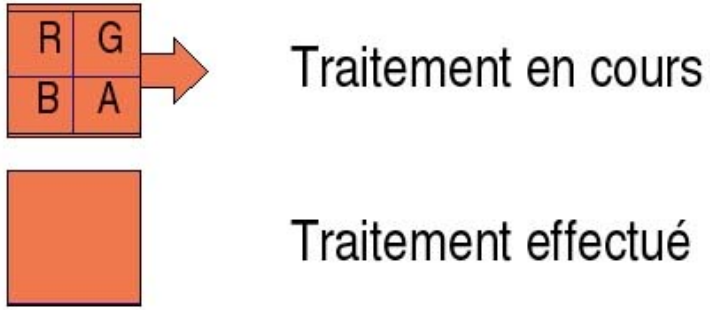
- En entrée :



Prototype de module de calcul parallèle par GPU

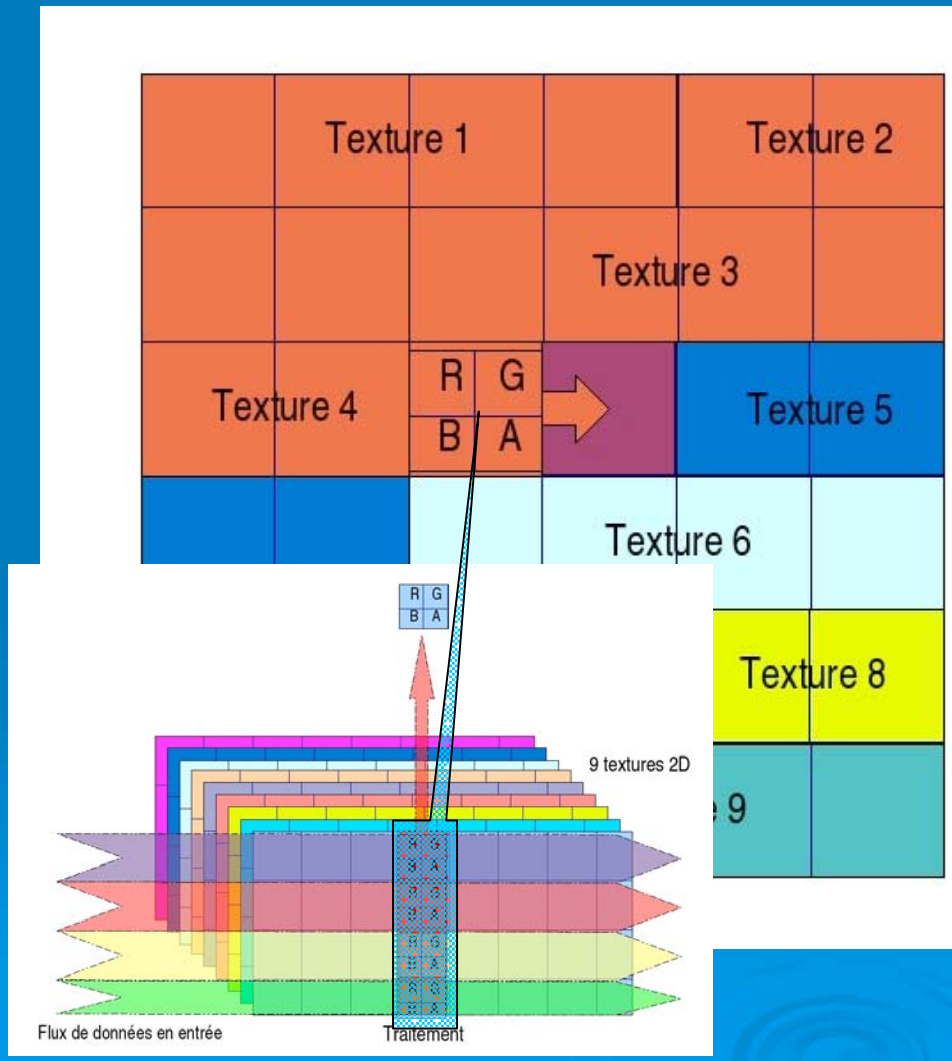
- En sortie :





Prototype de module de calcul parallèle par GPU

- Traitement effectué lors du « rendu »



Algorithme 1 Fragment shader

Entrées: float2 *coords*, samplerRECT $texture0 - 8X$, float *alpha*, float *sqrtN*.

Sorties: Traite toutes les données d'une colonne d'une texture donnée.

Calculer le numéro de pixel pointé par le fragment shader à partir de *coords*.

A partir de ce numéro, calculer le numéro de texture de stockage concernée.

Calculer la coordonnée en x de la colonne de valeurs en flux d'entrée sur la texture de stockage : *xcoord*.

pour tout Numéros de textures de 0 à 8 **faire**

  si C'est le bon numéro de texture de stockage **alors**

    pour tout Chaque flux d'entrée de 0 à *alpha* **faire**

      Récupérer la valeur stockée dans la texture de stockage d'après *xcoord* et *alpha*.

      Effectuer un traitement.

    fin pour

  finsi

fin pour

Retourner *resultat*.

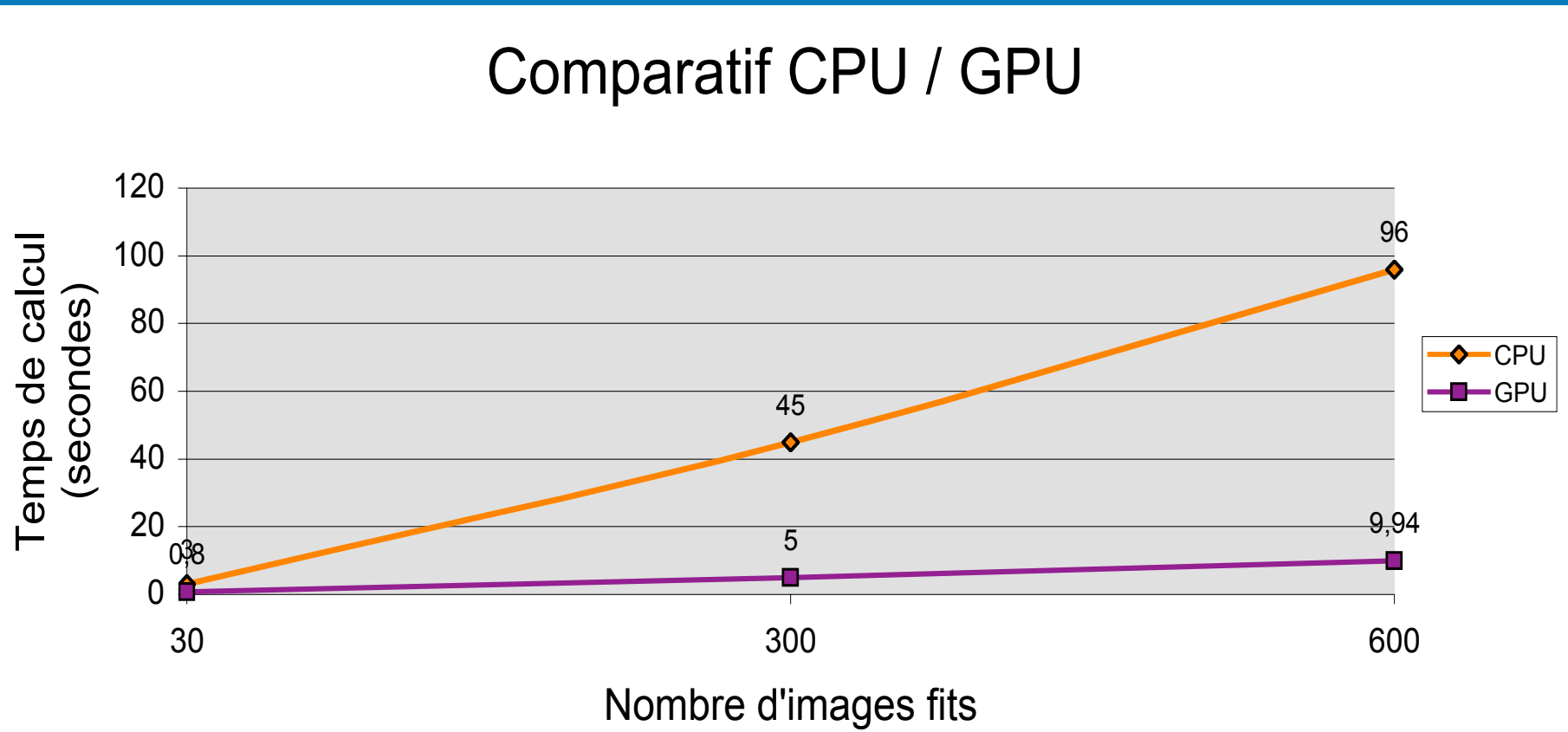
Prototype de module de calcul parallèle par GPU

Mise en pratique du module

- Développement du module en C avec l'API OpenGL 2.0.2 (glx version 1.3, glu version 1.3) et la librairie Cg 1.4 sous Ubuntu 5.10
- Traitement effectué et programmé dans le fragment shader : moyenne
- Intégration du module dans la librairie Poloka de l'expérience Supernovae permettant de manipuler des fichiers d'acquisition « fits » (permet de lire les fichiers fits sans les charger entièrement et ainsi travailler par bandes horizontales (slides) si l'on veut).
- Utilisation du module afin d'effectuer les calculs de moyenne d'images « fits » sur 30, 300 et 600 images fits (les fichiers fits sont chargés et traités par bandes de 100 pixels de hauteur).

Prototype de module de calcul parallèle par GPU

Benchmarks GPU / CPU



- Matériel utilisé :
  - Un Athlon 64 de 3,2GHz avec 1Go DDR
  - Une carte graphique Nvidia GeForce 6600GT sur bus Pci Express

Conclusions

Sur le module

- Ce module n'est pas programmé de manière optimale pour le GPGPU à cause :
  - De son profil « générique » : n'est pas développé pour un problème particulier
  - D'une algorithmie lourde et non optimisée : trop de conditions et de boucles coûteuses en nombres de cycles

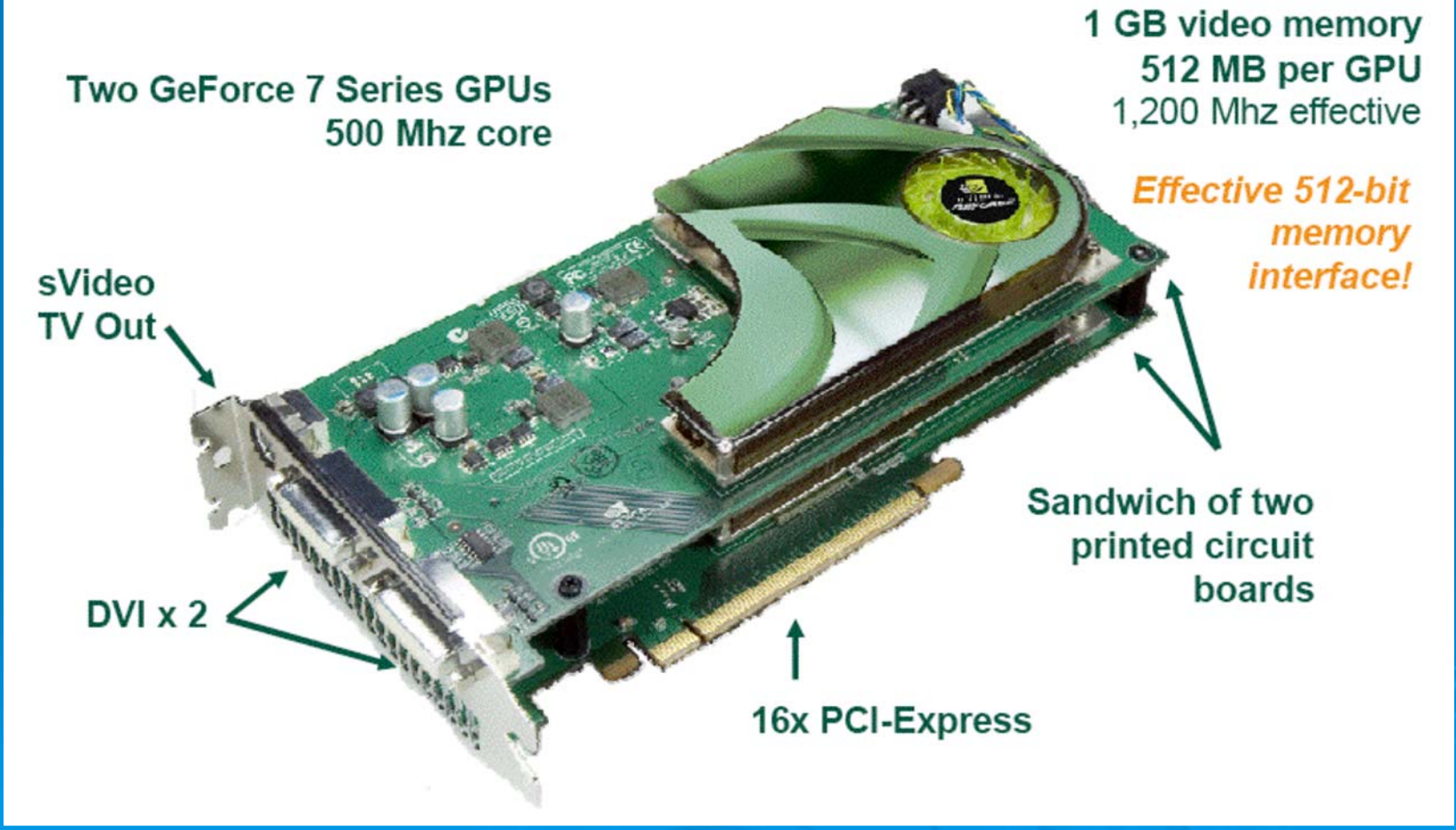
Instruction	Cost (Cycles)
If/endif	4
If/else/endif	6
Call	2
Ret	2
Loop/endloop	4

  - De la non utilisation du vertex shader pour « pointer » le fragment shader sur les données d'entrée (permettrait de supprimer de nombreux « if » dans le fragment shader)
- Mais cela ne reste qu'un prototype dont les performances restent importantes (un facteur 10 par rapport à un CPU) malgré ses défauts de conception et son manque d'optimisation.

Evolutions et perspectives

Au niveau hardware

- Cartes graphiques double GPU : NVIDIA GeForce 7950 GX2



Evolutions et perspectives

Au niveau hardware

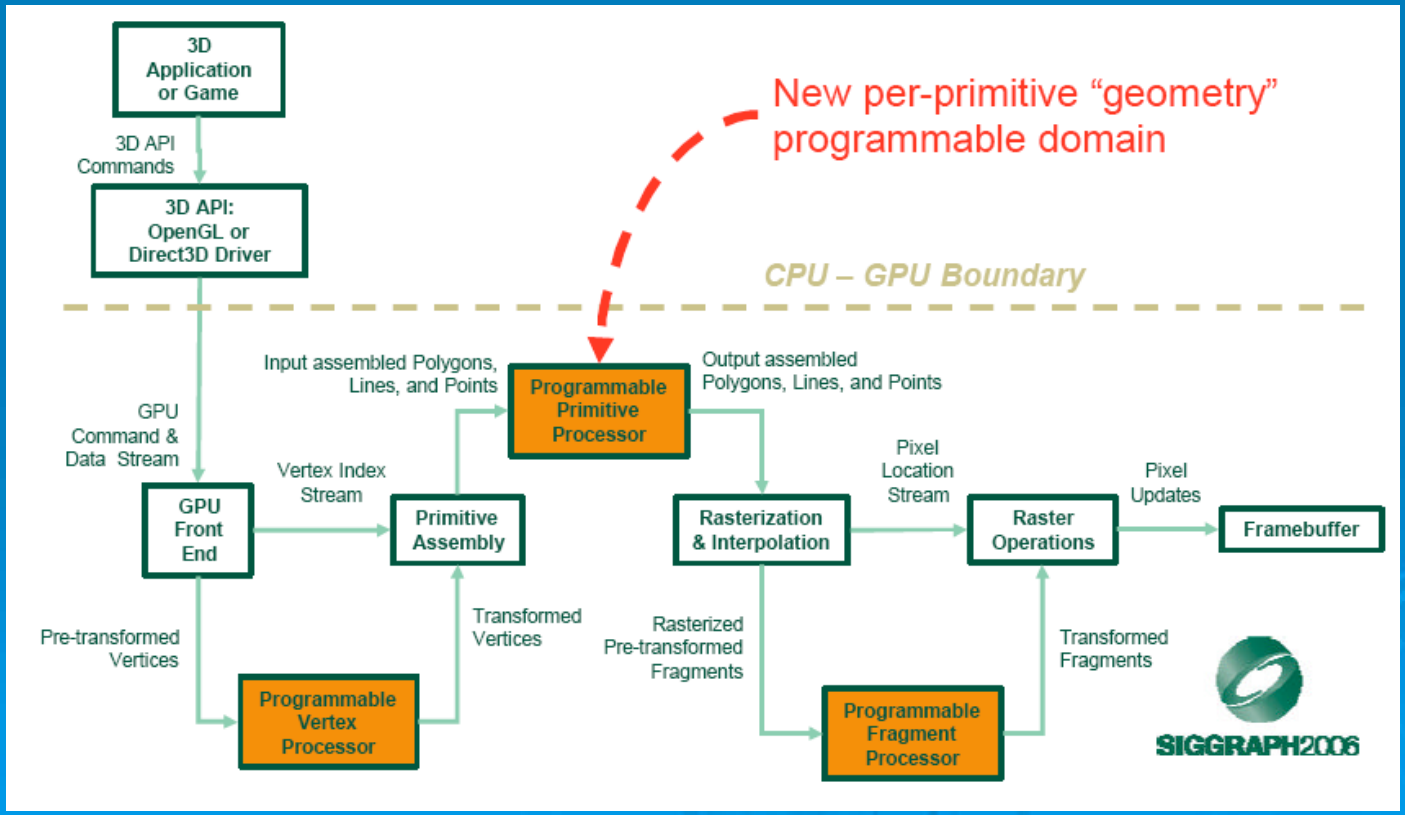
- Plusieurs cartes graphiques dans la même machine : SLI

(ici deux Geforce 7950 GX2)

Evolutions et perspectives

Au niveau hardware

- Un nouveau processeur : le « Primitive Processor »
- Un registre d'instructions GPU plus étendu
- Une programmation de plus en plus souple et de moins en moins limitée par le modèle du pipeline graphique



Evolutions et perspectives

Au niveau des bibiliothèques GPGPU

- La bibliothèque Sh (University of Waterloo) :
  - Permet de programmer en GPGPU sans connaître OpenGL ou DirectX
  - Implémentée en classes C++
  - Notions de « Channel » : séquence d'éléments d'un type donné :
    - ShChannel<element_type>
  - Notions de « Stream » : séquence de « Channels » et combinaison de « Channels » avec & :
    - ShStream = a & b & c;
  - Notions de « Kernel » : programme appliqué aux « Streams » avec <<:
    - ShStream t = (x & y & z);
    - s = p << t;
    - (a & b & c) = p << (x & y & z);

Evolutions et perspectives

Au niveau des bibiliothèques GPGPU

- La bibliothèque Brook (Stanford University) :
  - Egalement des kernels et des streams
  - Langage C avec des extensions

```
kernel void foo (float a<>, float b<>,
                 out float result<>) {
   result = a + b;
}

float a<100>;
float b<100>;
float c<100>;

foo(a,b,c);
```

```
for (i=0; i<100; i++)
      c[i] = a[i]+b[i];
```

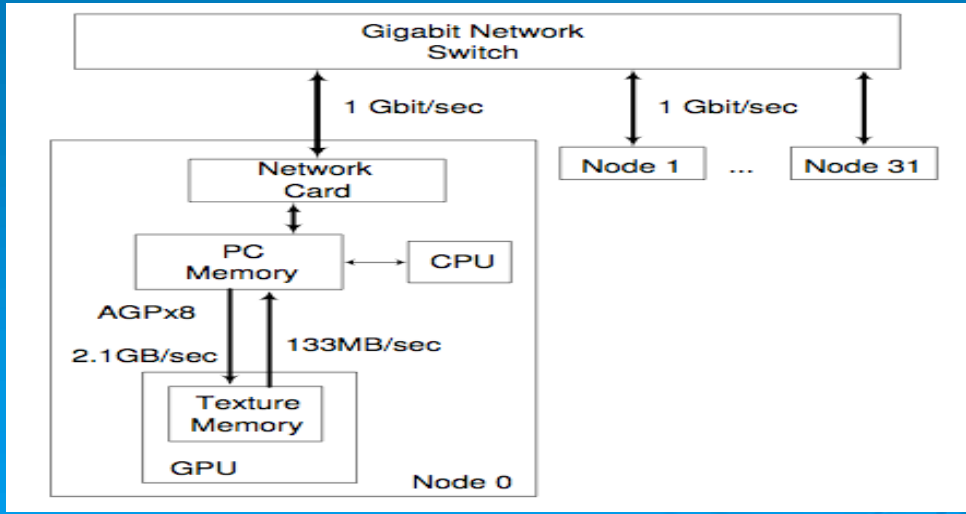
Evolutions et perspectives

Des clusters de calcul CPU/GPU apparaissent

Center For Visual Computing and Department of Computer Science

Stony Brook University

- 32 noeuds (PC HP) composés chacun de :
  - 2 processeurs Pentium Xeon 2.4GHz et 2.5 GB de RAM
  - Carte GeForce 5800 Ultra 128 MB
- Interconnectés sur un switch Gigabit

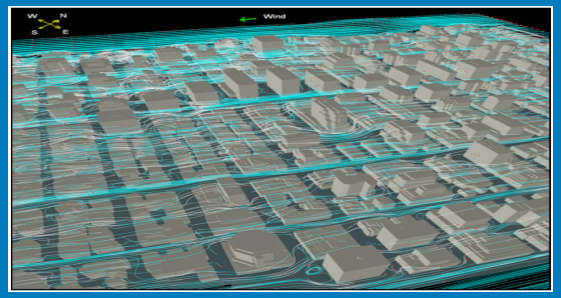


Novembre 2004

Evolutions et perspectives

Des clusters de calcul CPU/GPU apparaissent

- Simulation de la progression d'un flux d'air dans Times Square (New York).



- Temps (en ms) nécessaire pour calculer une étape de la simulation.
  - L'espace de la simulation est discrétisé en 15,4 millions de cellules
  - Chaque noeud calcule 512 000 cellules

Number of nodes	CPU cluster	GPU cluster				Speedup
	Total	Computation	GPU and CPU Communication	Network Communication: Non-overlapping Cost (Total)	Total	
1	1420	214	-	-	214	6.64
2	1424	216	13	0 (38)	229	6.22
4	1430	224	42	0 (47)	266	5.38
8	1429	222	50	0 (68)	272	5.25
12	1431	230	50	0 (80)	280	5.11
16	1433	235	50	0 (85)	285	5.03
20	1436	237	50	0 (87)	287	5.00
24	1437	238	50	0 (90)	288	4.99
28	1439	237	50	11 (131)	298	4.83
30	1440	237	50	25 (145)	312	4.62
32	1440	237	49	31 (151)	317	4.54

Pour conclure (brièvement) …

- Un concepte novateur qui a de l'avenir (évolution du hardware graphique) …
- Multiplication des publications de chercheurs dans ce domaine (**Siggraph** : *Special Interest Group in GRAPHics*, *ACM / IEEE Supercomputing Conference,* www.gpgpu.org, Nvidia, ATI, …)
- Des applications proches de nos problématiques déjà expérimentées
- Pourquoi pas?

Un point de départ pour se documenter …

- www.gpgpu.org