



RETOUR D'EXPERIENCE GPU CARTES GRAPHIQUES PROGRAMMABLES EN CALCUL HAUTE PERFORMANCE

Hervé Jourdren, Marc Pérache

CEA/DIF/DSSI

Clément KOYESSE

Stage Master 2 Informatique Professionnelle

Université de Versailles S^t Quentin

CEA/DIF

**Utilisation des technologies alternatives de type GPU ou Cell dans le domaine du
calcul scientifique**

11 avril 2008



1. **Description et intérêt**
2. **Mesures de performance**
3. **Limitations**
4. **Conclusions**



- **Approche GPGPU**

- Pipeline graphique parallélisé
 - Opérations indépendantes sur les pixels et/ou vertex
- Prédominance des calculs sur les opérations mémoires
- Extension récente du domaine d'application
 - Calculs scientifiques
 - Calculs financiers

- **Solutions disponibles**

- ATI Close To Metal
 - R600 475 GFLOP/s (crête)
 - ...
- NVIDIA Compute Unified Device Architecture
 - G80 518 GFLOP/s (crête)
 - ...

Évolutions récentes GPU

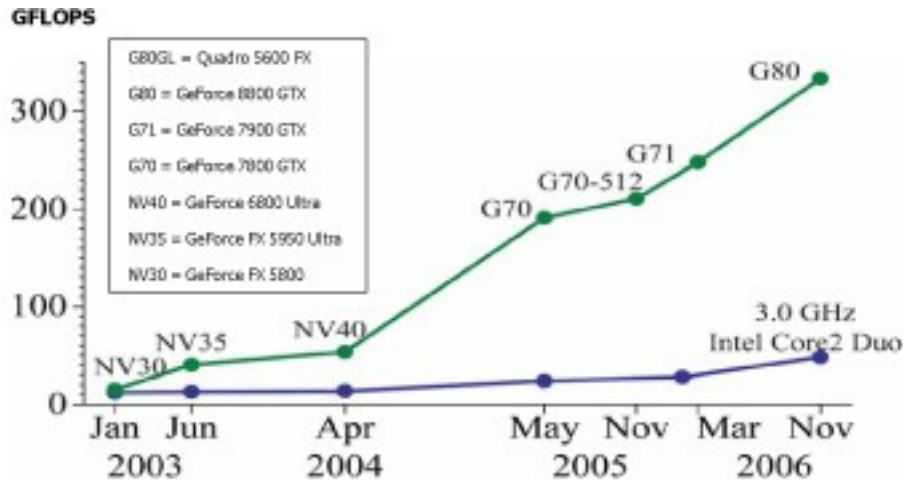
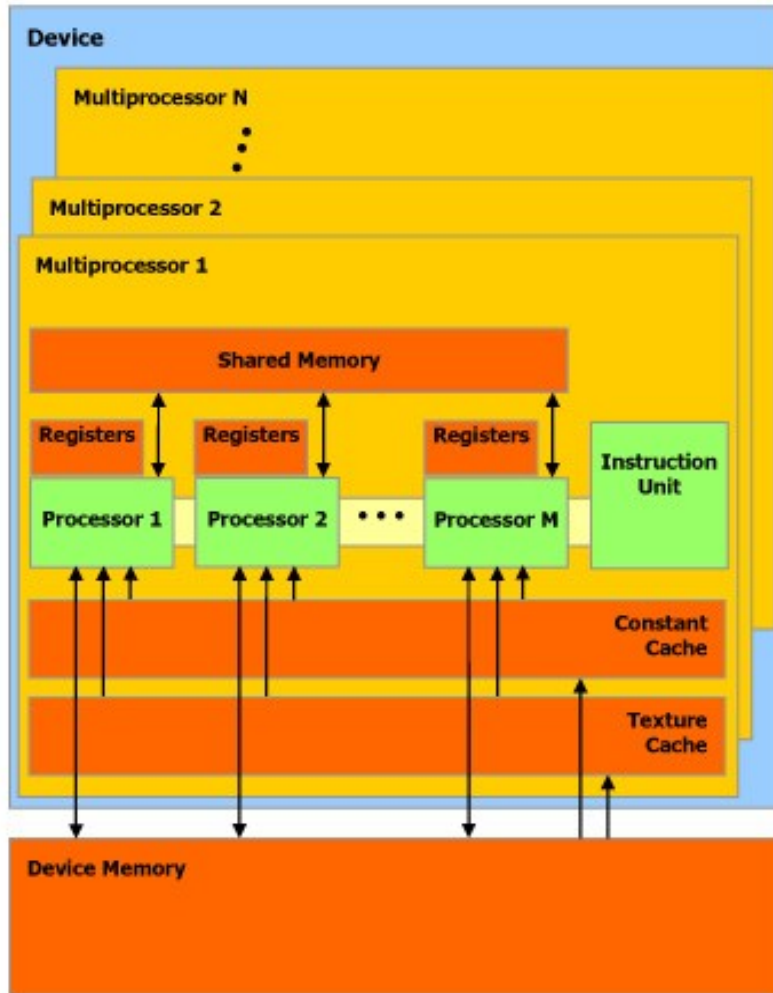


Figure 1-1. Floating-Point Operations per Second for the CPU and GPU



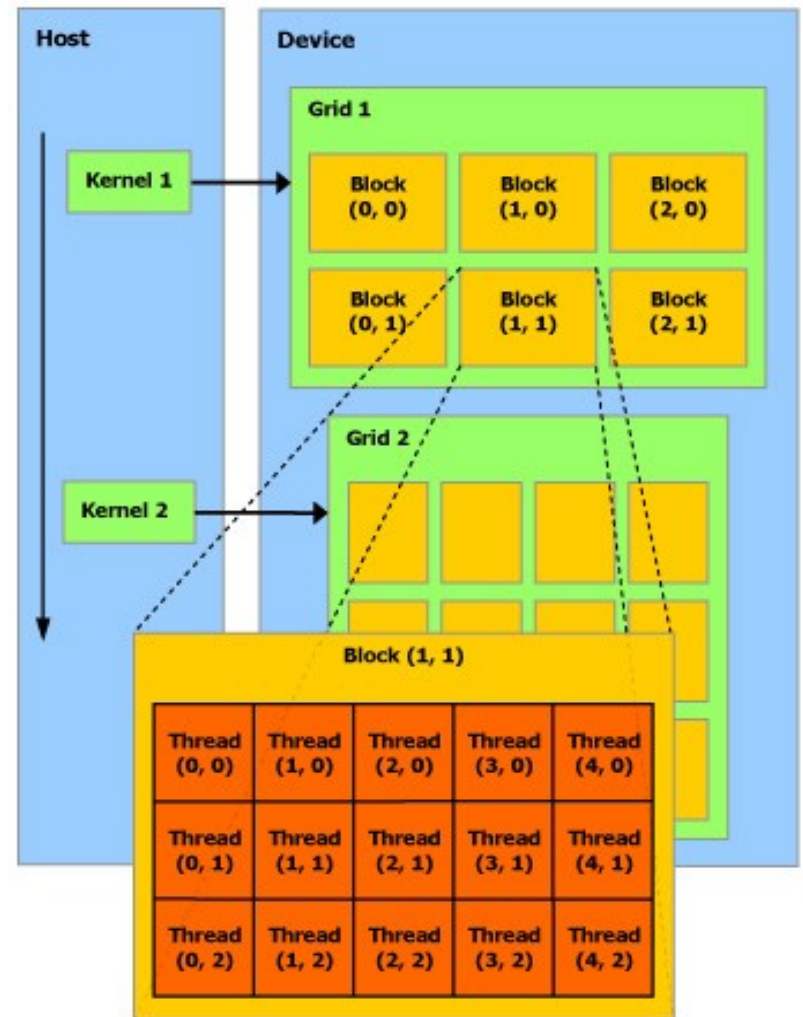
Figure 1-2. The GPU Devotes More Transistors to Data Processing

Architecture G80 Nvidia



A set of SIMD multiprocessors with on-chip shared memory.

Figure 3-1. Hardware Model



The host issues a succession of kernel invocations to the device. Each kernel is executed as a batch of threads organized as a grid of thread blocks

Figure 2-1. Thread Batching



● API CUDA

- Disponible à partir du GPU G80
 - Gamme Cartes Graphiques DirectX-10
 - Quadro FX 5600 1.5 Go de DRAM
 - GeForce 8800 GTX et GTS
 - GeForce 8600, 8400, etc
 - Gamme GPGPU Tesla
- Langage de haut niveau
 - Plus de programmation de shaders OpenGL
 - Extension SIMD du langage C
- Environnement d'exécution
- Mode émulation sur CPU (*)

(*) comportement légèrement différent par rapport à la carte, chaque thread étant exécuté séquentiellement



Exemple de multiplication d'une matrice par une constante version CPU:

```
void CMMul(float a, int len, float A[]){  
    for(int i = 0; i < len; i++){  
        for(int j = 0; i < len; i++){  
            A[i][j] = a * A[i][j];  
        }  
    }  
}
```



Exemple de multiplication d'une matrice par une constante version GPU:

```
void __global__ Cmmul_KER(float a, float A[]){
    //kernel executé en mode SIMD sur le device

    int tx = threadIdx.x + threadIdx.y * blockDim.x; //calcul du rang
    A[tx] = a * A[tx];
}

float A = (float*) malloc(N*sizeof(float));

//...initialisation du tableau

//allocation sur la carte
float *A_d;
cudaMalloc( (void**) &A_d, N*sizeof(float));

//copie vers la carte des données
cudaMemcpy(A_d, A, N*sizeof(float));

//appel du kernel
Cmmul_KER <<< N/256, 256>>> (a,A);

//rapatriement des résultats
cudaMemcpy(A, A_d, N*sizeof(float));
```




- **Equation d'état JWL (Jones–Wilkins–Lee)**
 - Produits de détonation
 - Calcul de la pression et de la vitesse du son en fonction de la densité, de l'énergie interne et de la fraction brûlée
 - Équation d'état relativement coûteuse avec plusieurs exponentielles (en version CPU, 10% du coût d'un solveur de dynamique des gaz)
- **Condition des mesures**
 - Version CPU
 - Version GPU
 - Mesure avec et sans transferts
 - Vecteur d'indirection
- **Écrit en « CUDA naïf »**

Mesures de performance : EOS JWL (2/4)



```
Fichier  Édition  Affichage  Terminal  Onglets  Aide
[ckoyesse@localhost JWL_100707]$ ./JWL -a -c 1000000 -e
Number of cell(s): 1000000
Block size: 256
Direct adressng
Memory transfert not included in time mesures
GPU mesure
block size: 256x1, grid size: 3906x1
JWL_EOS_GPU time: 0.591000 msec
JWL_EOS_GPU grind time: 5.910000e-04 µs/cell
JWL_EOS_GPU: 111.675125 Gflop/s

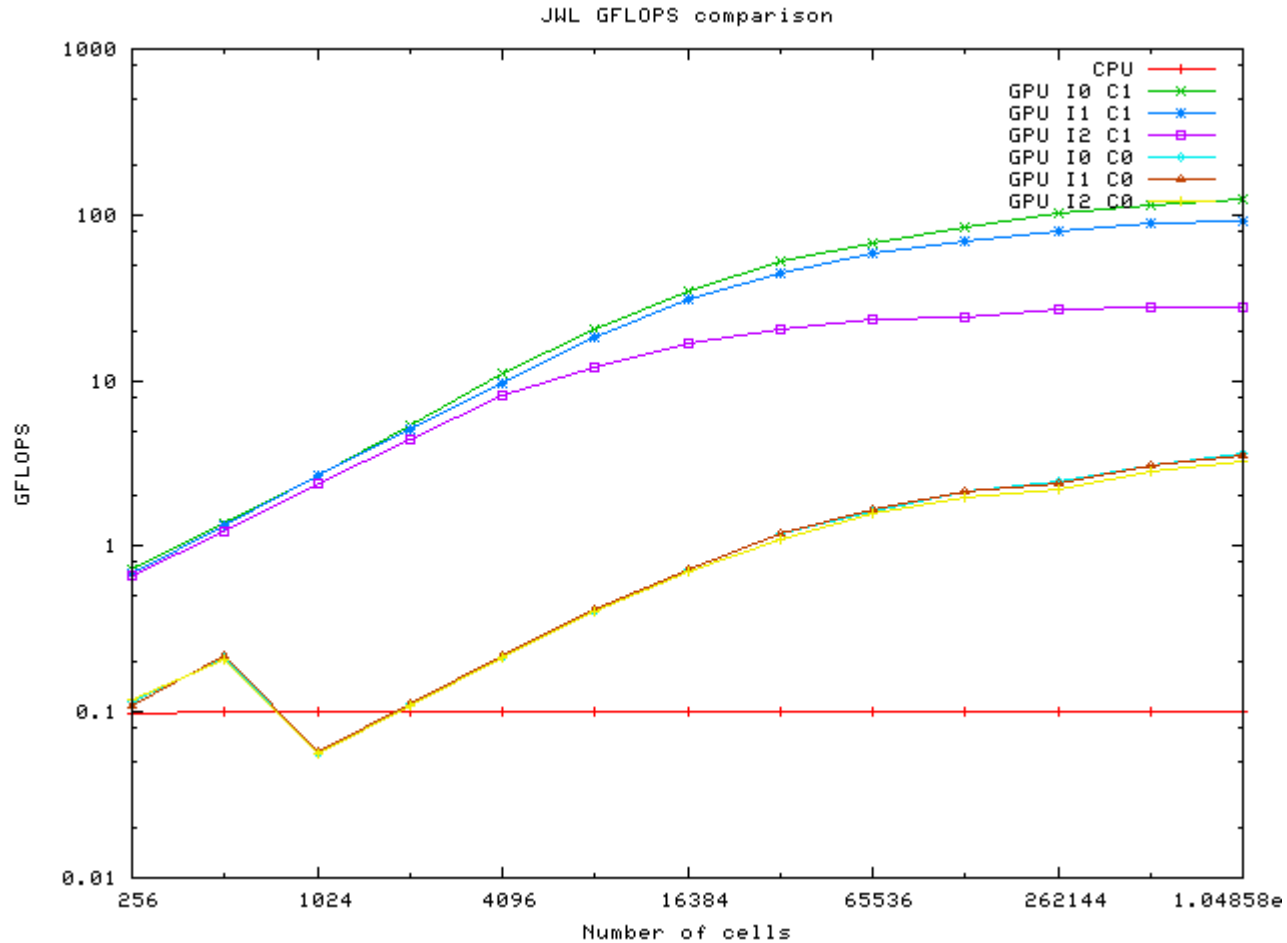
JWL_EOS_CPU time: 655.562012 msec
JWL_EOS_CPU grind time: 6.555620e-01 µs/cell
JWL_EOS_CPU: 100.676971 Mflop/s

GPU Speed up : 1109.241943

pmat max error : 2.822562e-07
cmat max error : 2.125488e-07

[ckoyesse@localhost JWL_100707]$
```

Mesures de performance : EOS JWL (3/4)





- **100 GFLOP/s simple précision en adressage direct et tableaux résidants sur la carte**
- **Très léger abattement (99 GFLOP/s) avec « fausse » indirection $\text{index}[i] = i$**
- **Facteur 3-4 d'abattement (27 GFLOP/s) avec « vraie » indirection, mais encore 2 ordres de grandeur de gain par rapport à la version CPU (0.1 GFLOP/s)**
- **En mode déporté avec transferts CPU-GPU/GPU-CPU, l'écart n'est plus que d'un ordre de grandeur (3.5 GFLOP/s), conséquence de la bande passante limitée du port PCIE**



- **Solveur Lagrange 1D**

- Équations de la dynamique des gaz
- Solveur GAD de type Godunov ordre 2 avec limiteur

- **Condition des mesures**

- Gestion optimale des différentes mémoires
 - Shared memory
 - Texture memory
- Block bi-dimensionnels
 - 16x16
- Implémentations avec et sans mailles fantômes
- Taille de block optimale
 - CUDA GPU occupancy calculator

Outil d'optimisation



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	CUDA GPU Occupancy Calculator				Click Here for detailed instructions on how to use this occupancy calculator For more information on NVIDIA CUDA, visit http://developer.nvidia.com/cuda															
2																				
3																				
4	Just follow steps 1, 2, and 3 below! (or click here for help)																			
5																				
6	1.) Select a GPU from the list (click):		G80	(Help)																
7																				
8	2.) Enter your resource usage:																			
9	Threads Per Block		256	(Help)																
10	Registers Per Thread		10																	
11	Shared Memory Per Block (bytes)		4096																	
12																				
13	(Don't edit anything below this line)																			
14																				
15	3.) GPU Occupancy Data is displayed here and in the graphs:																			
16	Active Threads per Multiprocessor		768	(Help)																
17	Active Warps per Multiprocessor		24																	
18	Active Thread Blocks per Multiprocessor		3																	
19	Occupancy of each Multiprocessor		100%																	
20	Maximum Simultaneous Blocks per GPU		48																	
21	(Note: This assumes there are at least this many blocks)																			
22																				
23	Physical Limits for GPU:		G80																	
24	Multiprocessors per GPU		16																	
25	Threads / Warp		32																	
26	Warps / Multiprocessor		24																	
27	Threads / Multiprocessor		768																	
28	Thread Blocks / Multiprocessor		8																	
29	Total # of 32-bit registers / Multiprocessor		8192																	
30	Shared Memory / Multiprocessor (bytes)		16384																	
31																				
32	Allocation Per Thread Block																			
33	Warps		8																	
34	Registers		2560																	
35	Shared Memory		4096																	
36	These data are used in computing the occupancy data in blue																			
37																				
38	Maximum Thread Blocks Per Multiprocessor		Blocks																	
39	Limited by Max Warps / Multiprocessor		3																	
40	Limited by Registers / Multiprocessor		3																	
41	Limited by Shared Memory / Multiprocessor		4																	
42	Thread Block Limit Per Multiprocessor is the minimum of these 3																			
43																				
44	CUDA Occupancy Calculator																			
45	Version:		1.2																	
46	Copyright and License																			
47																				
48																				
49																				
50																				
51																				
52																				
53																				
54																				
55																				

Varying Block Size

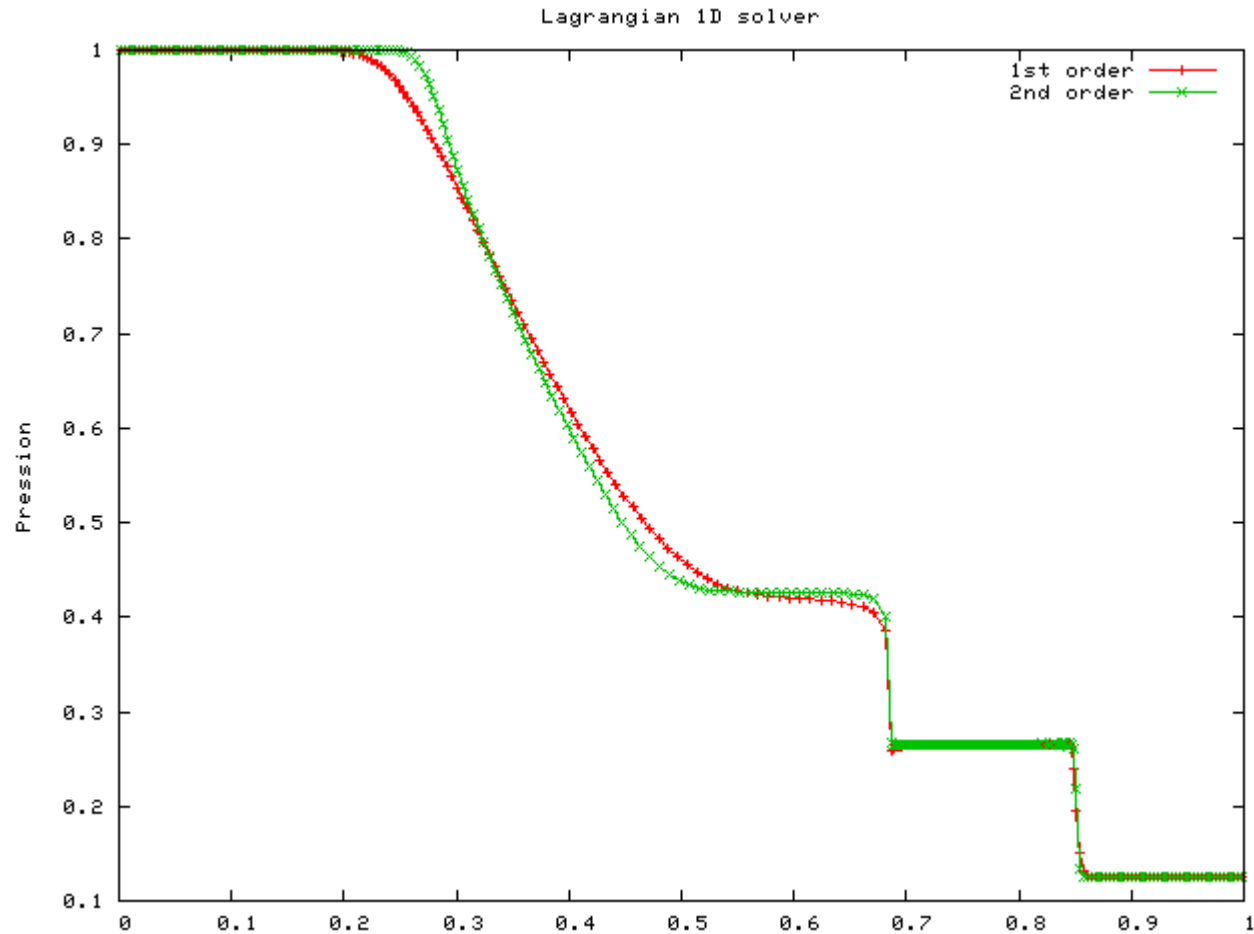
This graph plots Multiprocessor Warp Occupancy (Y-axis, 0 to 24) against Threads Per Block (X-axis, 16 to 464). The occupancy starts at approximately 4 for 16 threads and increases to a peak of 24 between 128 and 256 threads, before dropping to about 16 for 464 threads.

Varying Register Count

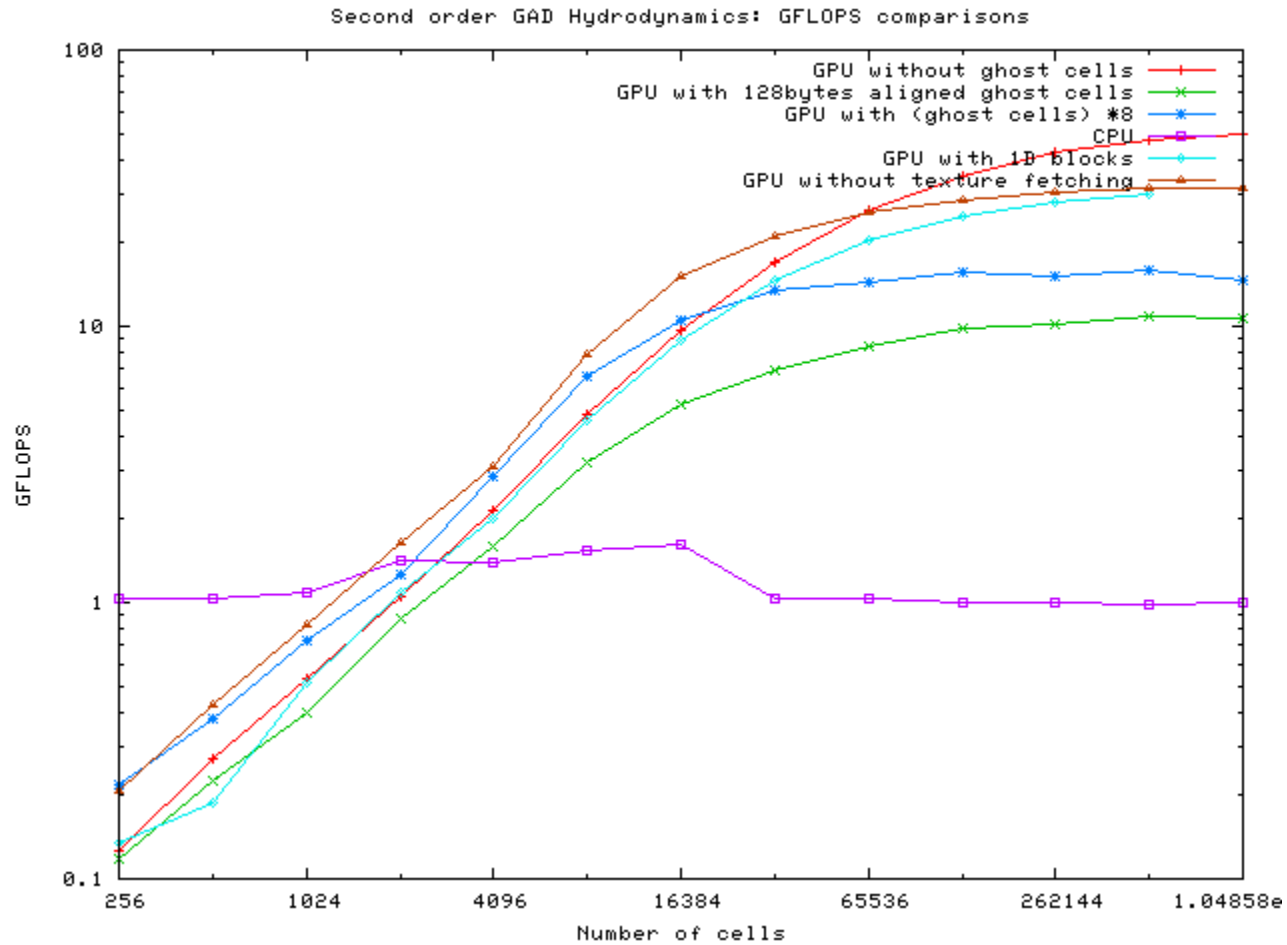
This graph plots Multiprocessor Warp Occupancy (Y-axis, 0 to 24) against Registers Per Thread (X-axis, 0 to 32). The occupancy is constant at 24 for 0-10 registers, then drops to 16 for 12-16 registers, and finally to 8 for 18-32 registers.

Varying Shared Memory Usage

This graph plots Multiprocessor Warp Occupancy (Y-axis, 0 to 24) against Shared Memory Per Thread (X-axis, 0 to 16384). The occupancy is constant at 24 for memory usage up to 4096 bytes, then drops to 16 for 4096-8192 bytes, and finally to 8 for 8192-16384 bytes.



Mesures de performance : solveur hydrodynamique (3/4)





- **50 GFLOP/s sans mailles fantômes**
- **Facteur 3 à 4 d'abattement (13 GFLOP/s) avec les mailles fantômes en dépit d'un alignement sur 128 octets**
- **A partir de 2^{16} mailles**
 - Utiliser des Blocks 2D plutôt que 1D
 - Utiliser le cache de texture
- **Meilleurs résultats de l'alignement 'simple'(*) par rapport aux recommandations de Nvidia**

(*) le nombre de mailles fantômes est égale à 8 fois le numéro d'ordre

Principales limitations



- **Taille block**
 - 512 threads / block
- **Taille grille**
 - 2^{40} threads / grille
- **Quantité de registres/shared memory utilisé**
- **Alignement en mémoire des données en cas d'indirection**
- **Bande passante**
 - PCIE 1.1 (4 Go/s)
 - PCIE 2.0 (8 Go/s)

- **Calculs simple précision non-IEEE**
- **Pas de code de correction d'erreur pour la mémoire GPU**

Conclusions de l'étude



- **Très bonnes performances en simple précision**
 - JWL : 125 GFLOP/s
 - GAD : 50 GFLOP/s
 - Schéma GAIA ordres très élevés : 200-300 GFLOP/s
- **Mise en oeuvre aisée *via* API CUDA**
 - JWL 480 lignes
 - GAD 1300 lignes
- **Mise au point optimale néanmoins assez pointue**
 - Alignement en mémoire globale
 - Conflits de bancs shared memory
 - Nombre limité de registres
 - Très longs vecteurs *indispensables*
 - Débit Port PCI-E limité (4 ou 8 Go/s)
 - Privilégier si possible les solutions tout GPU
 - Nouvelles difficultés pour de très gros maillages (programmation parallèle type Out-of-Core)