Analyse des textures en temps réel exploitant une architecture parallèle multi-coeurs et GPU

Moulay Akhloufi, *MScA, MBA* (moulay.akhloufi@crvi.ca)

Gilles Champagne (gilles.champagne@crvi.ca)

Mario Jr Laframboise (mario.laframboise@crvi.ca)

Plan

- Introduction
- Calcul générique sur processeur graphique (GPGPU)
- Techniques d’analyse de texture
- Résultats expérimentaux
- Conclusion

Introduction

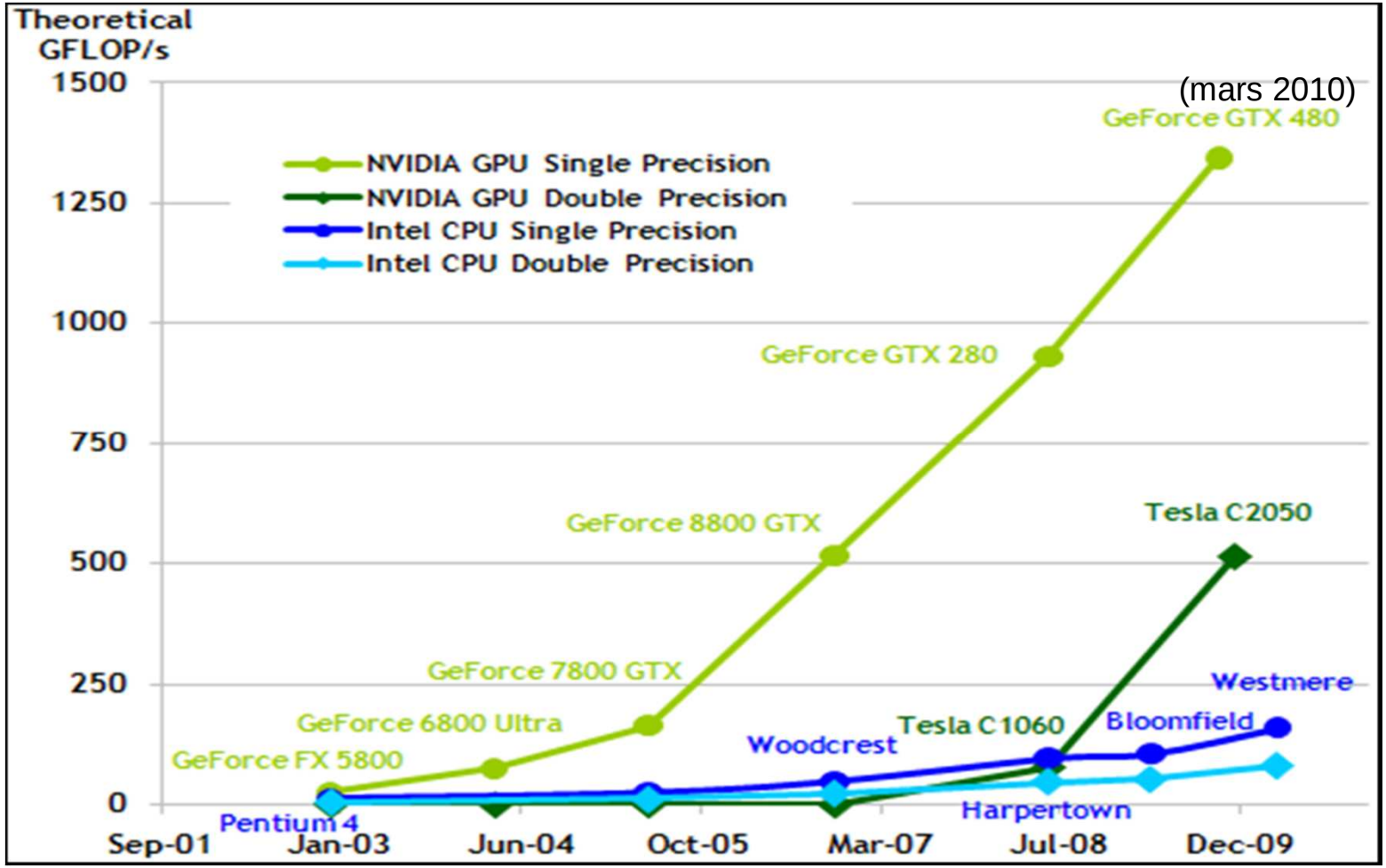
- Croissance de l'intérêt pour des techniques d'analyse de texture efficaces :
  - *Classification, segmentation, reconnaissance.*
- Alternative récente pour l'optimisation de la vitesse d'exécution :
  - *Systèmes parallèles, GPGPU, DSP, FPGA, clusters.*
- Technologie d'optimisation retenue :
  - GPGPU: CUDA
  - CPU: MMX-SSE, Multicoeur
- Techniques d'analyse de texture retenues :
  - LBP, LTP, LATP
  - Laws texture kernels
  - Gabor filters
  - Gray Level Co-Occurrence Matrix (GLCM)

GPGPU

- Très accessible, disponible dans la plupart des PC modernes.
- Moins de contraintes associées à cette technologie, ce qui permet une évolution plus intéressante.
- Architecture en mutation constante.
- Parmi les cinq ordinateurs les plus puissants de la planète, trois utilisent le GPGPU.

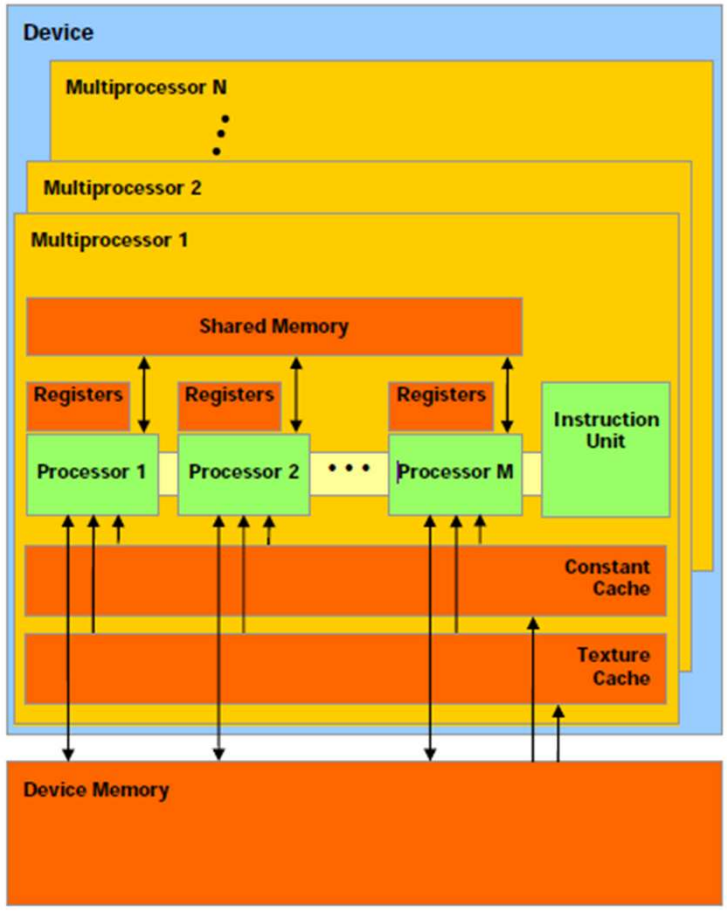
GPGPU

- Évolution du nombre d'opérations en point flottant par seconde.



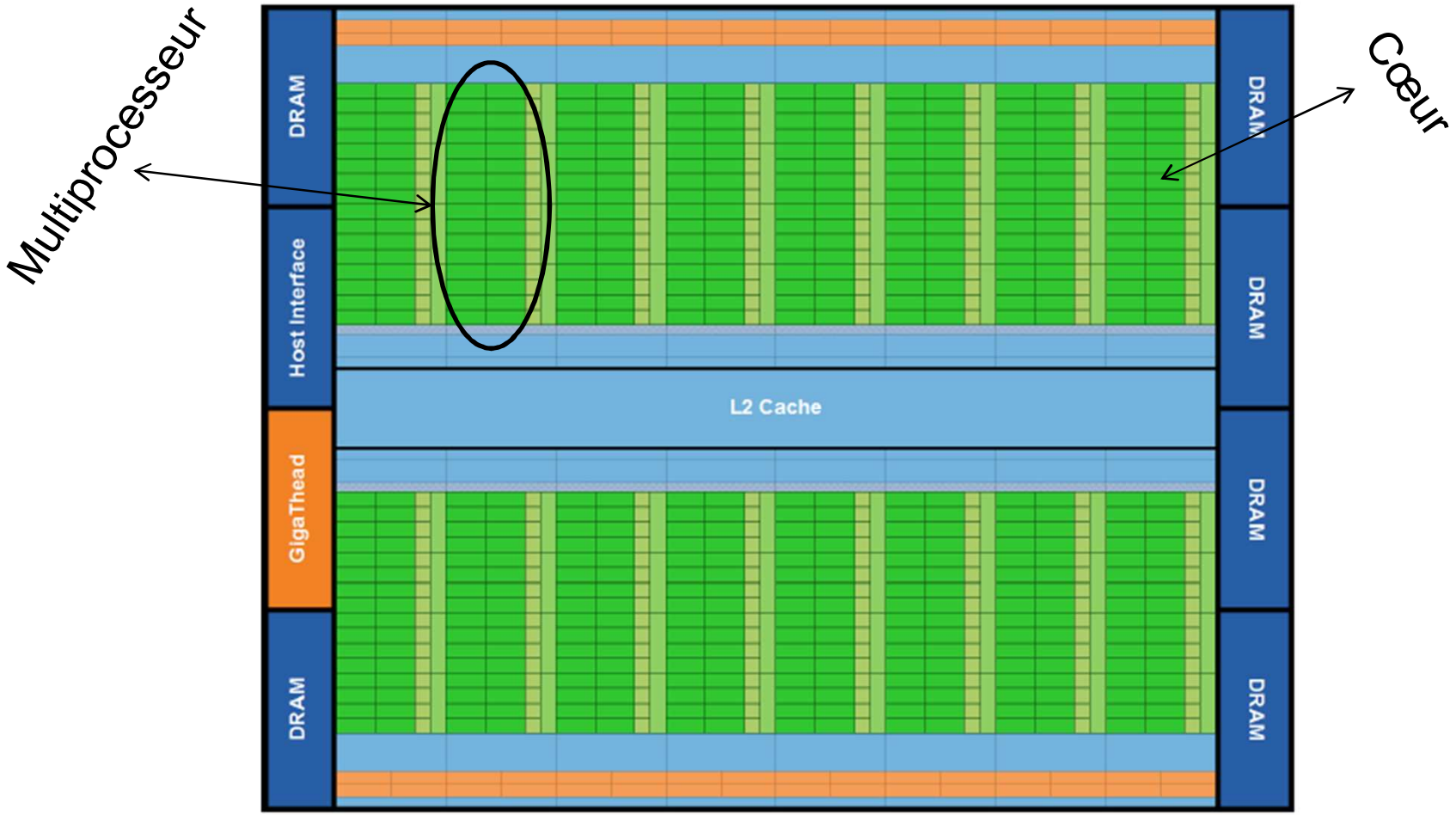
GPGPU (NVIDIA)

- Cœur : unité d'exécution matérielle de base
- Multiprocesseur : regroupement de 8,32 cœurs qui partagent différentes ressources.
- Conçu pour les opérations en point flottant.



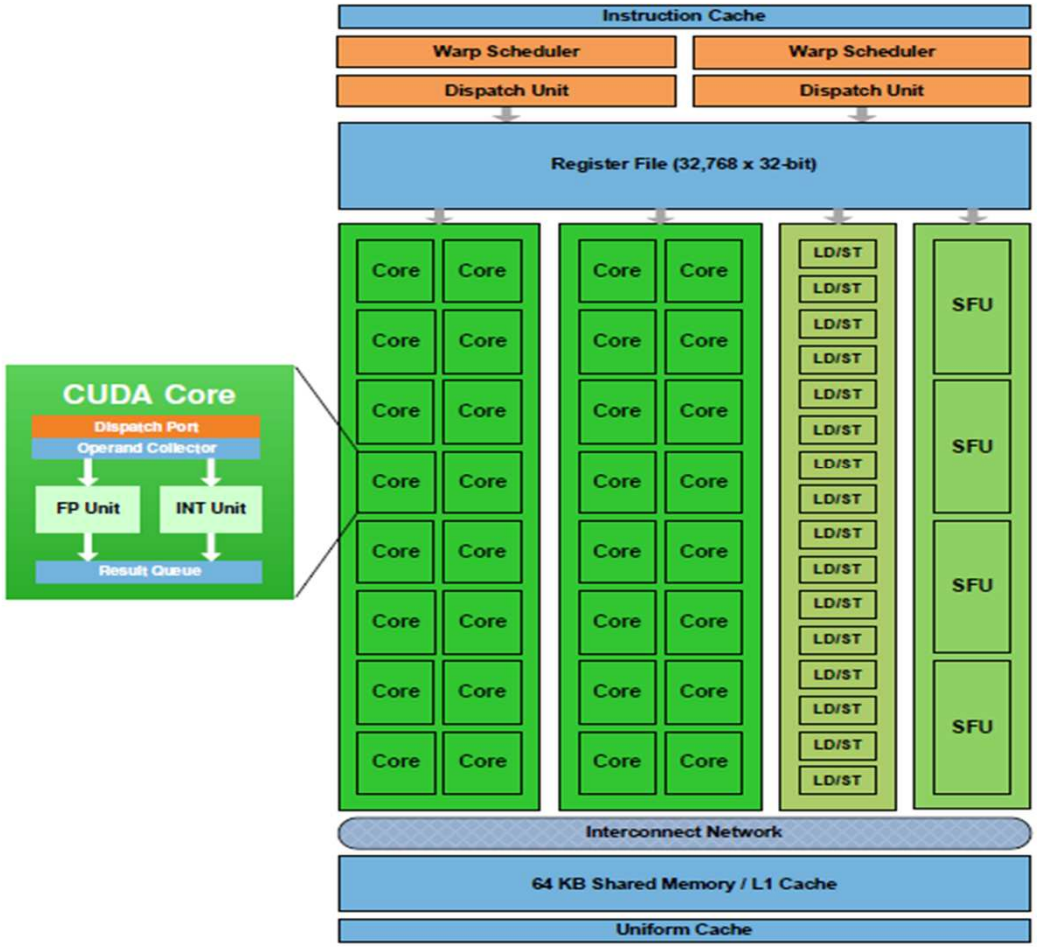
Architecture GT200

GPGPU (NVIDIA)



Architecture Fermi

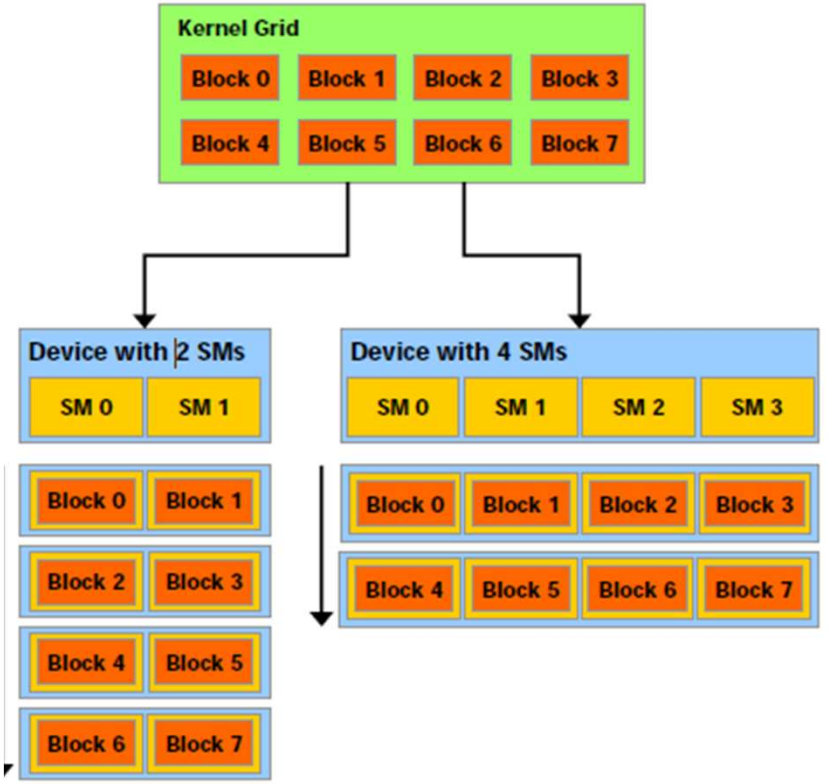
GPGPU (NVIDIA)



Multiprocesseur Fermi

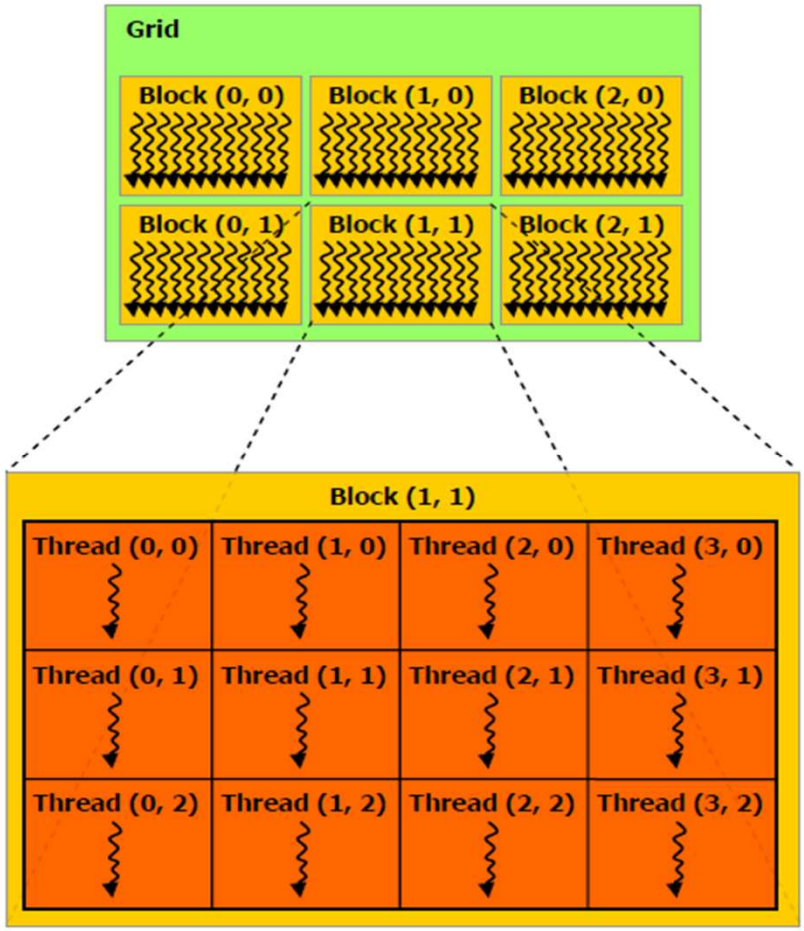
GPGPU (NVIDIA)

- Kernel : fonction C exécutée N fois par N threads.
- Thread: unité d'exécution de base.
- Bloc: regroupement de X thread lancé sur un même multiprocesseur.
- Grille: regroupement de Y bloc formant une unité d'exécution lancée sur le GPU par le CPU.



Nombre de thread indépendant du nombre de cœur.

GPGPU (NVIDIA)



Grille de bloc

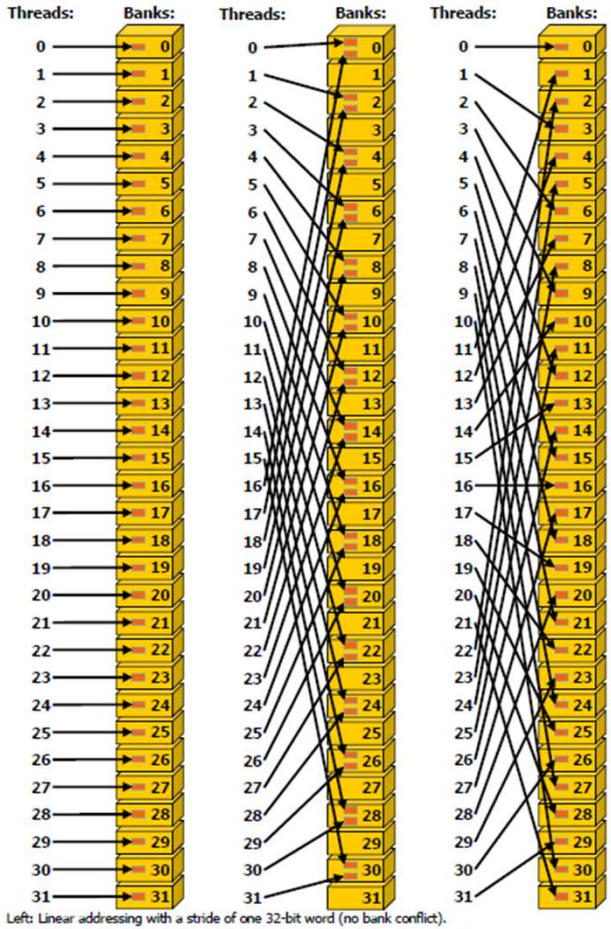
GPGPU (NVIDIA)

- Caractéristique de Fermi
  - Architecture SIMD.
  - 32 cœurs/ multiprocesseur
  - Espace d'adressage unifié.
  - Kernel concurrent (CUDA 4.0)
  - Cache L1/ Mémoire partagé configurable
  - Opération double précision
  - Opération point flottant IEEE
  - Opération atomique.

GPGPU (NVIDIA)

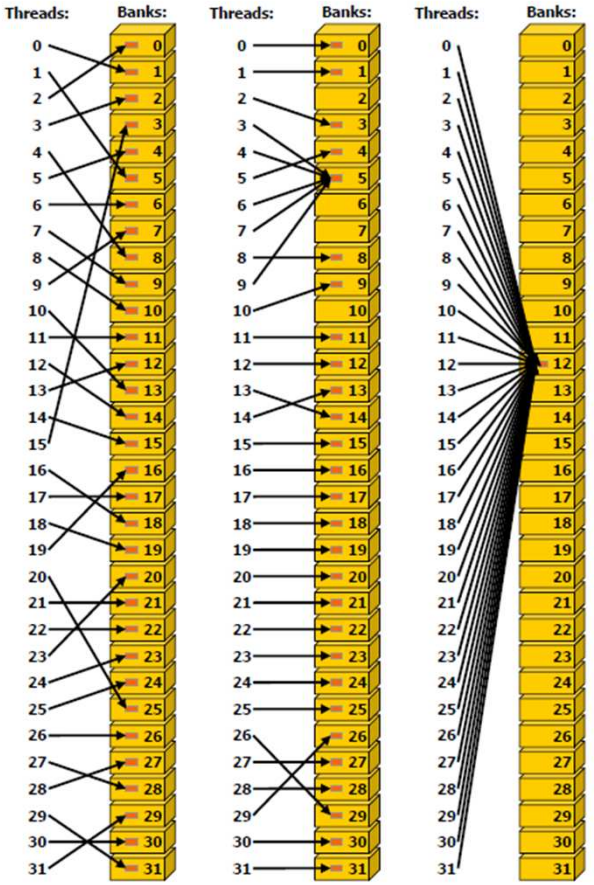
- Limitation:
  - Communication inter bloc non prévue
  - Connaissance du matériel nécessaire
  - Indépendances des résultats
  - Courbe d’apprentissage
  - Accès mémoire coordonnée

GPGPU (NVIDIA)



Left: Linear addressing with a stride of one 32-bit word (no bank conflict).
Middle: Linear addressing with a stride of two 32-bit words (2-way bank conflicts).
Right: Linear addressing with a stride of three 32-bit words (no bank conflict).

Figure G-2 Examples of Strided Shared Memory Accesses for Devices of Compute Capability 2.x



Left: Conflict-free access via random permutation.
Middle: Conflict-free access since threads 3, 4, 6, 7, and 9 access the same word within bank 5.
Right: Conflict-free broadcast access (all threads access the same word).

Figure G-3 Examples of Irregular and Colliding Shared Memory Accesses for Devices of Compute Capability 2.x

GPGPU (NVIDIA)

- Environnement de développement CUDA:
  - Windows, Linux, MacOS
  - Visual Studio 2008,2010
    - Code mixte HOST-DEVICE
  - Parallel Nsight pour le debugging.
    - 2 cartes GPU
    - Remote debugging
- OpenCL

GPGPU (NVIDIA)

- Exemple de code CPU VS GPU

```
main()
{
        ImageSize =1024*1024;
        for (int i = 0; i < ImageSize; i++)
        {
            ImageLBP[i] = LBP(Image, i);
        }
}
```

```
main()
{
    ImageSize =1024*1024;
    DimBloc = 32;
    DimGrid = ImageSize/DimBloc;
    cudaMalloc (GPUImage, ImageSize,…);
    cudaMemcpy (GPUImage, Image,..);
    cudaMalloc (GPUImageLBP, ImageSize,…);
    MonKernel<<DimGrid,DimBloc >>(GPUImage,GPUImageLBP)
    cudaMemcpy (ImageLBP,GPUImageLBP,..);
    cudaFree(GPUImage);
    cudaFree(GPUImageLBP);
}
MonKernel(Image,ImageLBP)
{
    index = ThreadIdx * BlockIdx;
    ImageLBP[index] = LBP(Image, index);
}
```

Techniques d'analyse de texture

- Local Binary Pattern (LBP)
  - *L'opérateur de texture LBP est une mesure invariante d'une texture en niveau de gris, calculée à partir de l'analyse du voisinage.*

$$P_i' = \begin{cases} 0 & if\ P_i < P_0 \\ 1 & otherwise \end{cases}$$

$$I_{LBP} = \sum_{i=1}^{8} P_i' 2^{i-1}$$

100	90	40
212	100	12
200	75	120

(a)

1	0	0
1		0
1	0	1

(b)

1	2	4
8		16
32	64	128

(c)

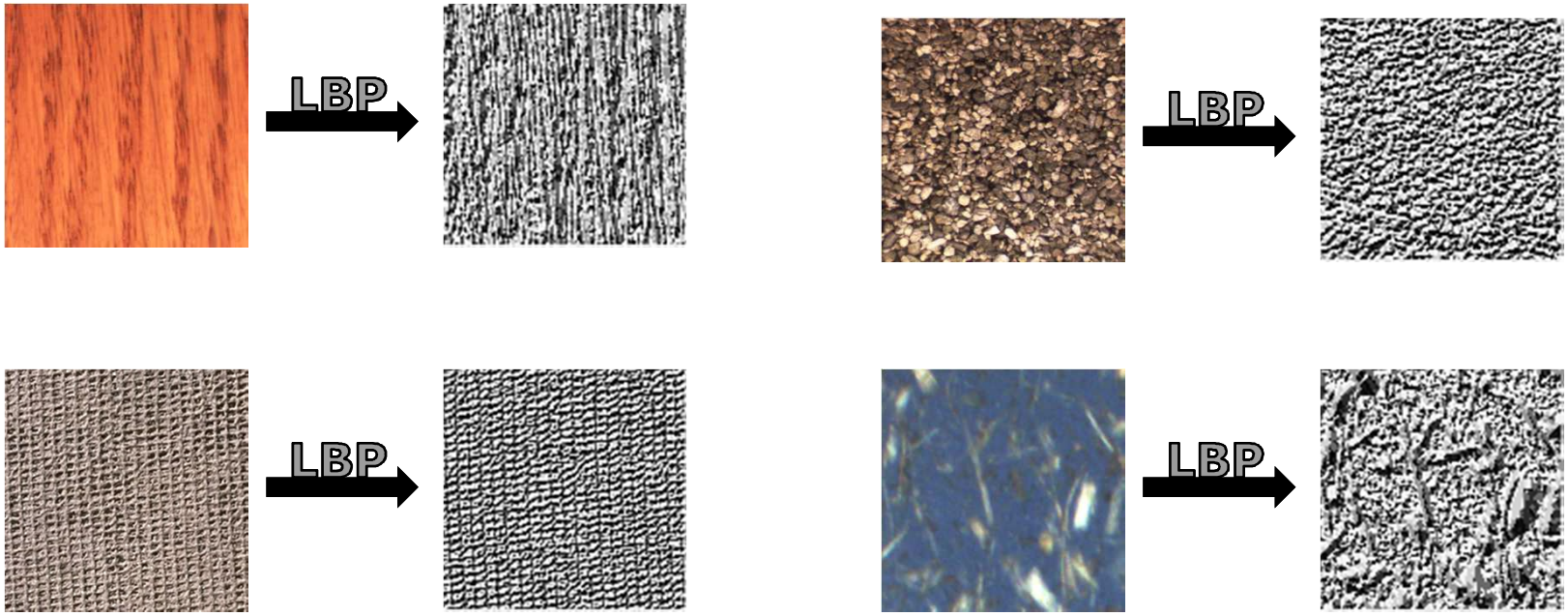
1	0	0
8		0
32	0	128

(d)

LBP = 1+8+32+128 = 169

Techniques d’analyse de texture

- Local Binary Pattern (LBP)



Techniques d'analyse de texture

- Local Ternary Pattern (LTP) - *With t = 5*

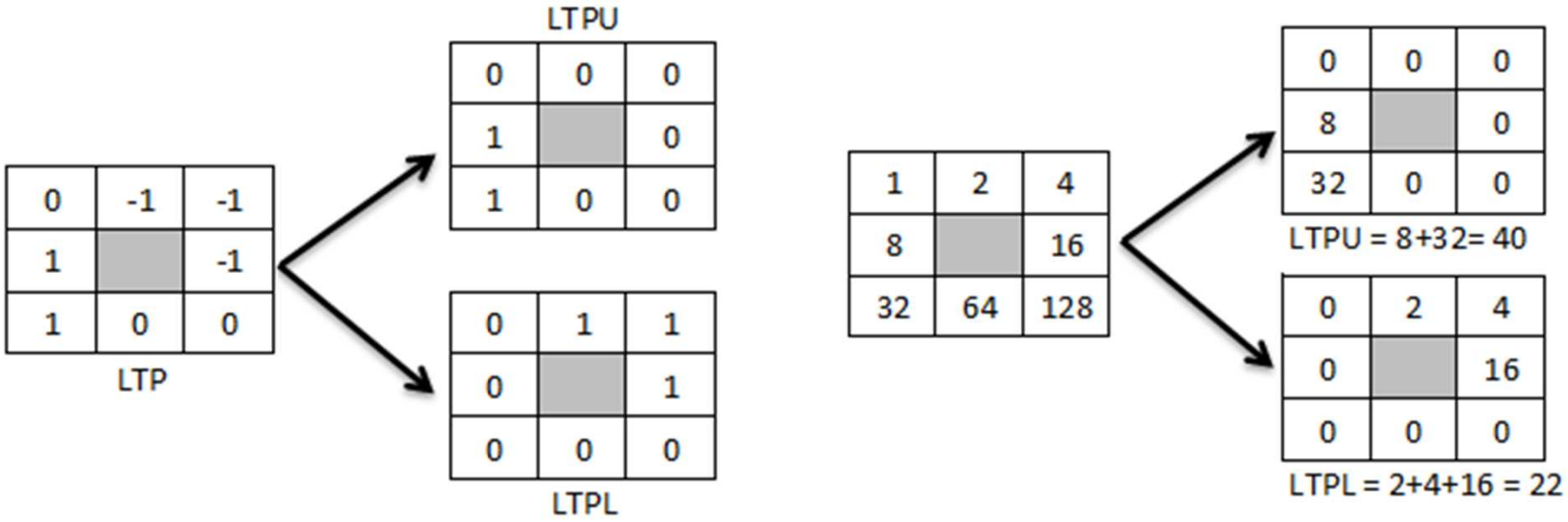
$$P_i' = \begin{cases} 1 & if\ P_i \geq P_0 + t \\ 0 & if\ |P_i - P_0| < t \\ -1 & if\ P_i \leq P_0 - t \end{cases}$$

100	90	40
212	100	12
200	98	103

(a)

0	-1	-1
1		-1
1	0	0

(b)



Techniques d'analyse de texture

- Local Adaptive Ternary Pattern (LATP)

$$P_i^{'} = \begin{cases} 1 & if\ P_i \geq (\mu + k\sigma) \\ 0 & if\ |P_i| < (\mu + k\sigma) \\ -1 & if\ P_i \leq (\mu + k\sigma) \end{cases}$$

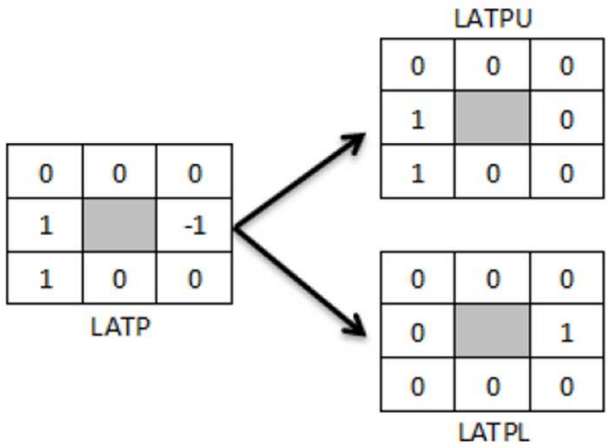
100	90	40
212	100	12
200	75	120

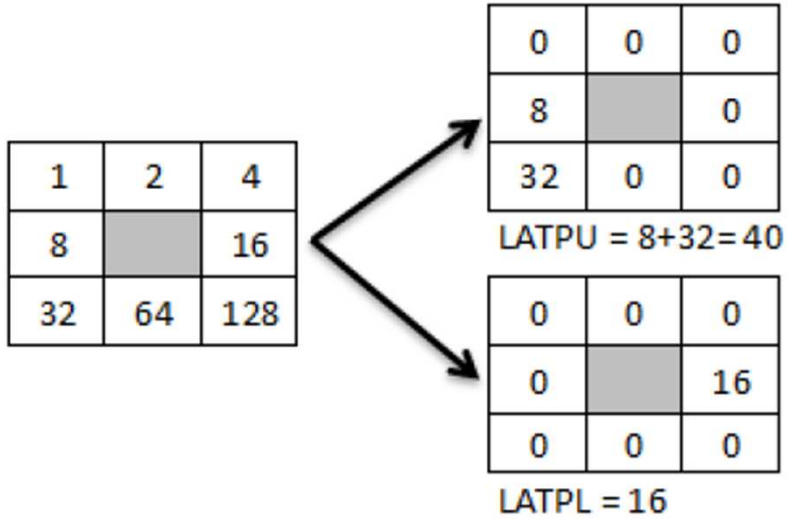
(a)

μ = 105,44
σ = 65,90
t_u = 171,35
t_l = 39,54
(b)

0	0	0
1		-1
1	0	0

(c)





Techniques d’analyse de texture

- Law texture kernels
  - Masque 2D dérivé à partir de 5 noyaux 1D:
    - Center-weighted local averaging (L5 = [1 4 6 4 1])
    - Edge detection (E5 = [-1 -2 0 2 1])
    - Spot detection (S5 = [-1 0 2 0 -1])
    - Wave detection (W5 = [-1 2 0 -2 1])
    - Ripple detection (R5 = [1 -4 6 -4 1])
  - Pour obtenir les masques de convolution 2D:
    - Convolution d’un noyau 1D vertical avec un noyau 1D horizontal.

Techniques d’analyse de texture

- Law texture kernel

*	L5	E5	S5	W5	R5
L5	L5L5	E5L5	S5L5	W5L5	R5L5
E5	L5E5	E5E5	S5E5	W5E5	R5E5
S5	L5S5	E5S5	S5S5	W5S5	R5S5
W5	L5W5	E5W5	S5W5	W5W5	R5W5
R5	L5R5	E5R5	S5R5	W5R5	R5R5

$$L5E5 = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}$$

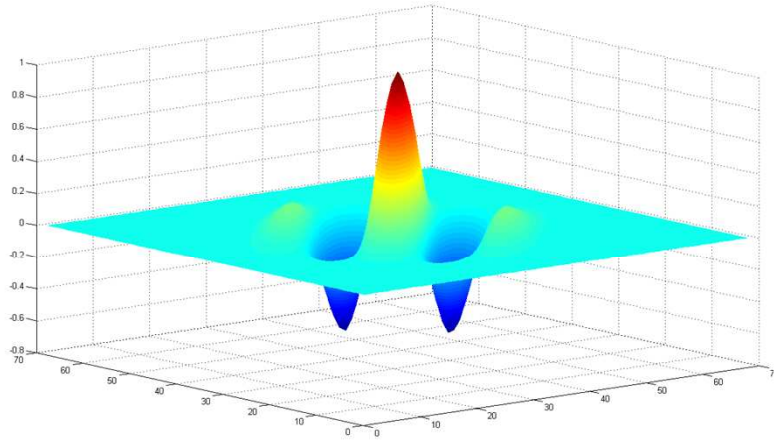
$$L5S5 = \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \\ -4 & 0 & 8 & 0 & -4 \\ -6 & 0 & 12 & 0 & -6 \\ -4 & 0 & 8 & 0 & -4 \\ -1 & 0 & 2 & 0 & -1 \end{bmatrix}$$

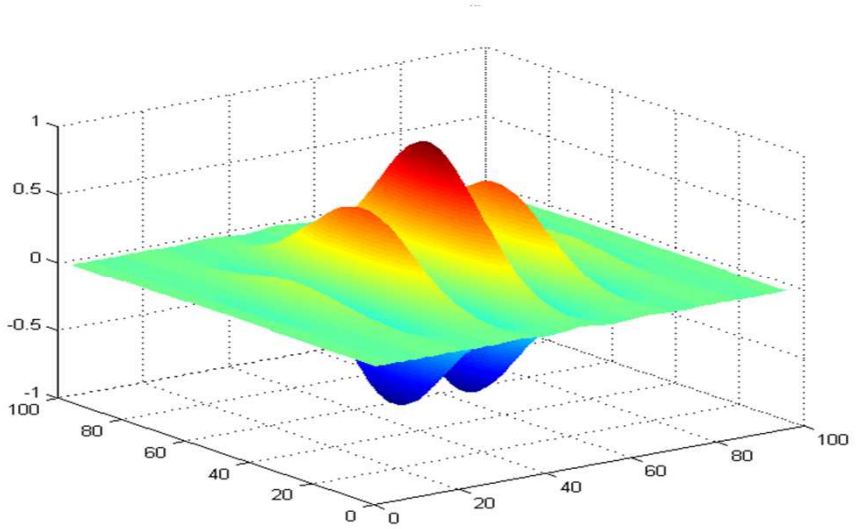
$$E5S5 = \begin{bmatrix} 1 & 0 & -2 & 0 & 1 \\ 2 & 0 & -4 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 4 & 0 & -2 \\ -1 & 0 & 2 & 0 & -1 \end{bmatrix}$$

Techniques d'analyse de texture

• Filtre de Gabor: Produit d'une sinusoïde complexe et d'une fonction gaussienne.

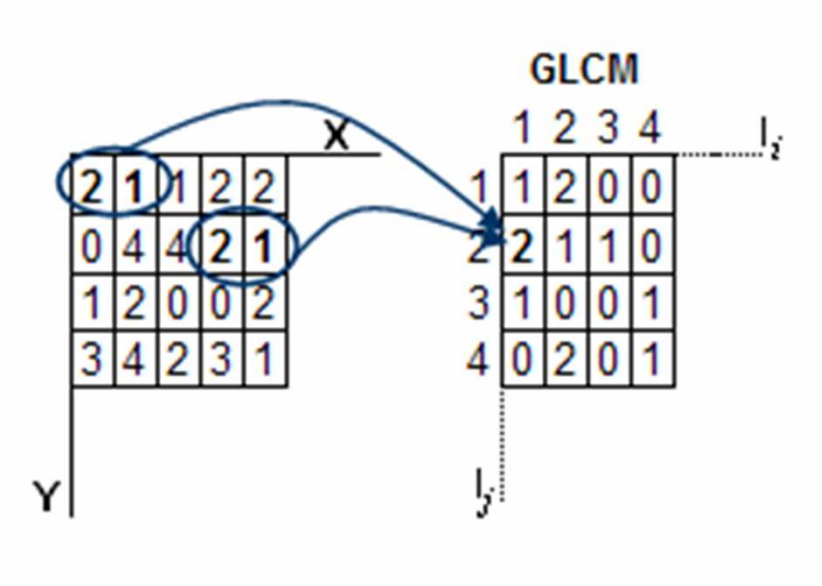
$$f_m(x,y) = \frac{1}{2\pi\sigma_m^2} \exp^{-\frac{x^2+y^2}{2\sigma_m^2}} \times \cos 2\pi\left(u_m x \cos\theta + u_m y \sin\theta\right)$$





Techniques d’analyse de texture

- Gray Level Co-occurrence Matrix (GLCM)
  - Statistique de second niveau reliée aux propriétés de l’image.
  - Case plutôt difficile pour le GPU.
  - Règles d’accès coordonnées à la mémoire du GPU non respecté.



Résultats expérimentaux

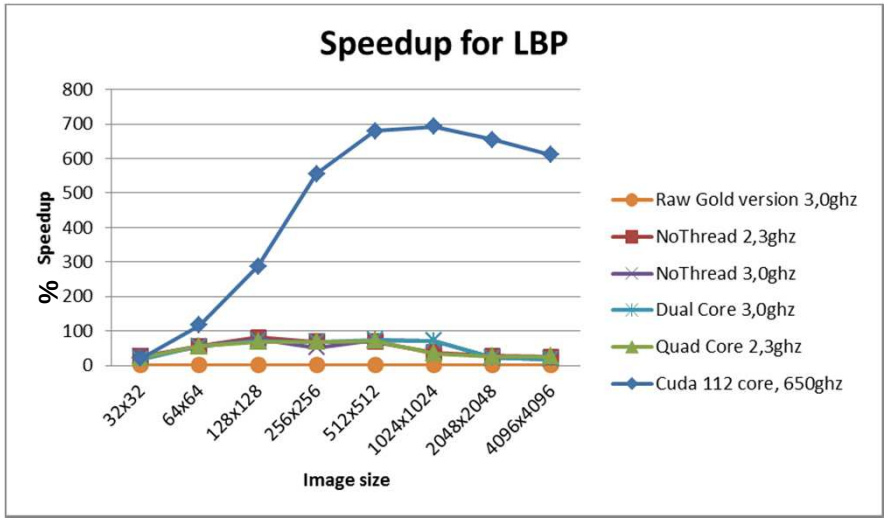
- Systèmes utilisés:
  - Intel E8400
    - *Dual Core (3.0 Ghz) – 2 GB RAM (1.98 Ghz)*
  - Intel Q8200
    - *Quad Core (2.33 Ghz) – 3 GB RAM (1.97 Ghz)*
  - Geforce 8800 GT
    - *112 CUDA Cores (650 Mhz) – 512 MB RAM (900Mhz)*
  - Geforce GTX 260
    - *192 CUDA Cores (1 Ghz) – 1.7 GB RAM (1 Ghz)*
  - Tesla C2050
    - *448 CUDA Cores (1.1 Ghz) – 2.6 GB RAM (1.5 Ghz)*

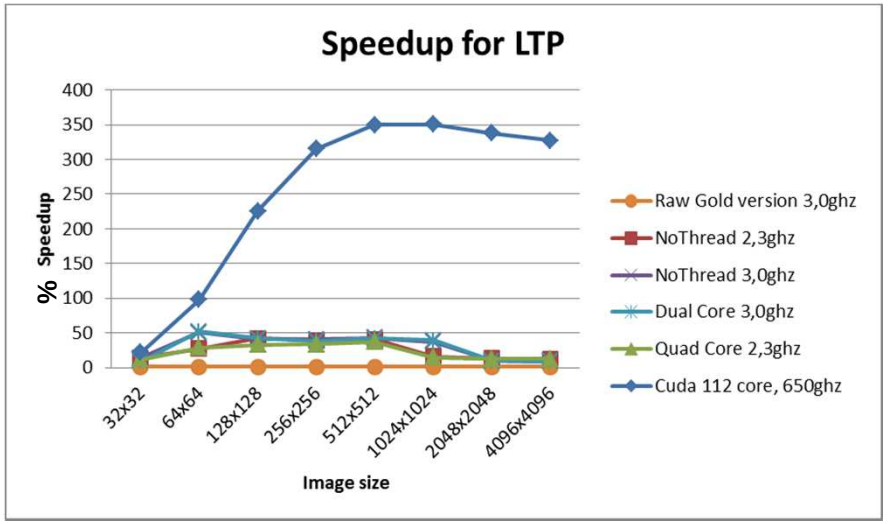
Résultats expérimentaux

- Implantations utilisées:
  - Raw Gold version 3Ghz: Basic C++ used as a reference
  - Intel NoThread 2.3 Ghz: MMX-SSE
  - Intel NoThread 3 Ghz: MMX-SSE
  - Intel Dual Core 3 Ghz: MMX-SSE + Multithreading
  - Intel Quad Core 2.3 Ghz: MMX-SSE + Multithreading
  - Nvidia CUDA 112 core, 650 Mhz: GPGPU
  - Nvidia CUDA 192 core, 1 Ghz: GPGPU
  - Nvidia CUDA 448 core, 1.1 Ghz: GPGPU

Résultats expérimentaux

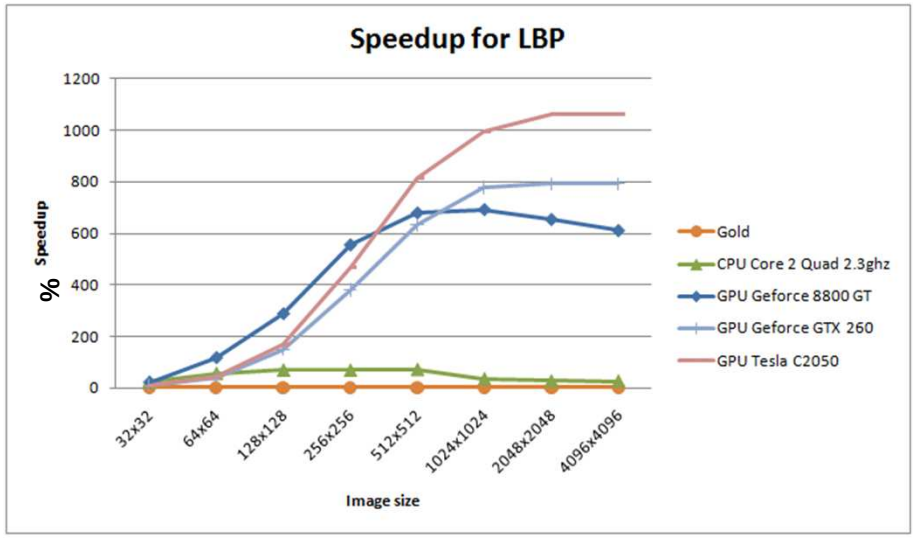
- Le GPGPU permet un gain plus important lorsque la grosseur de l'image augmente.
- Petite diminution pour LBP et LTP lorsque l'image est supérieure à 1024x1024.

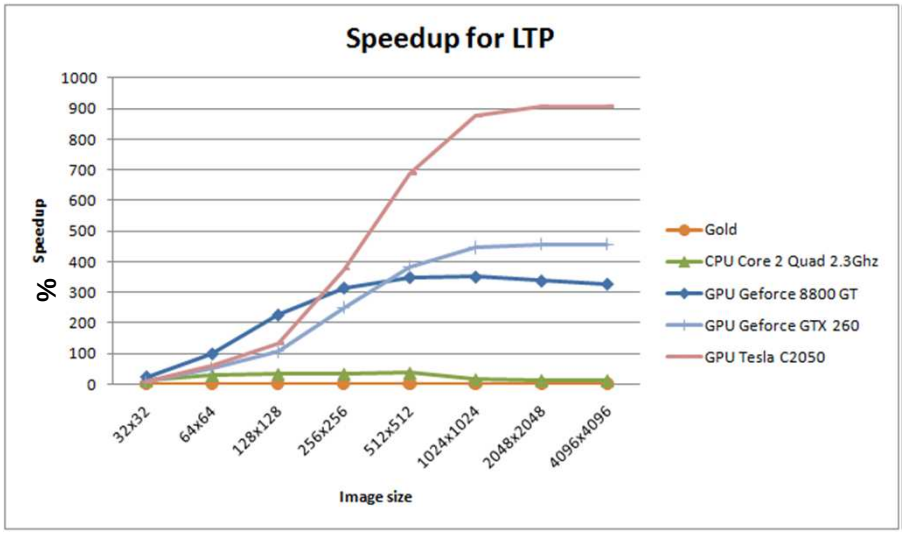




Résultats expérimentaux

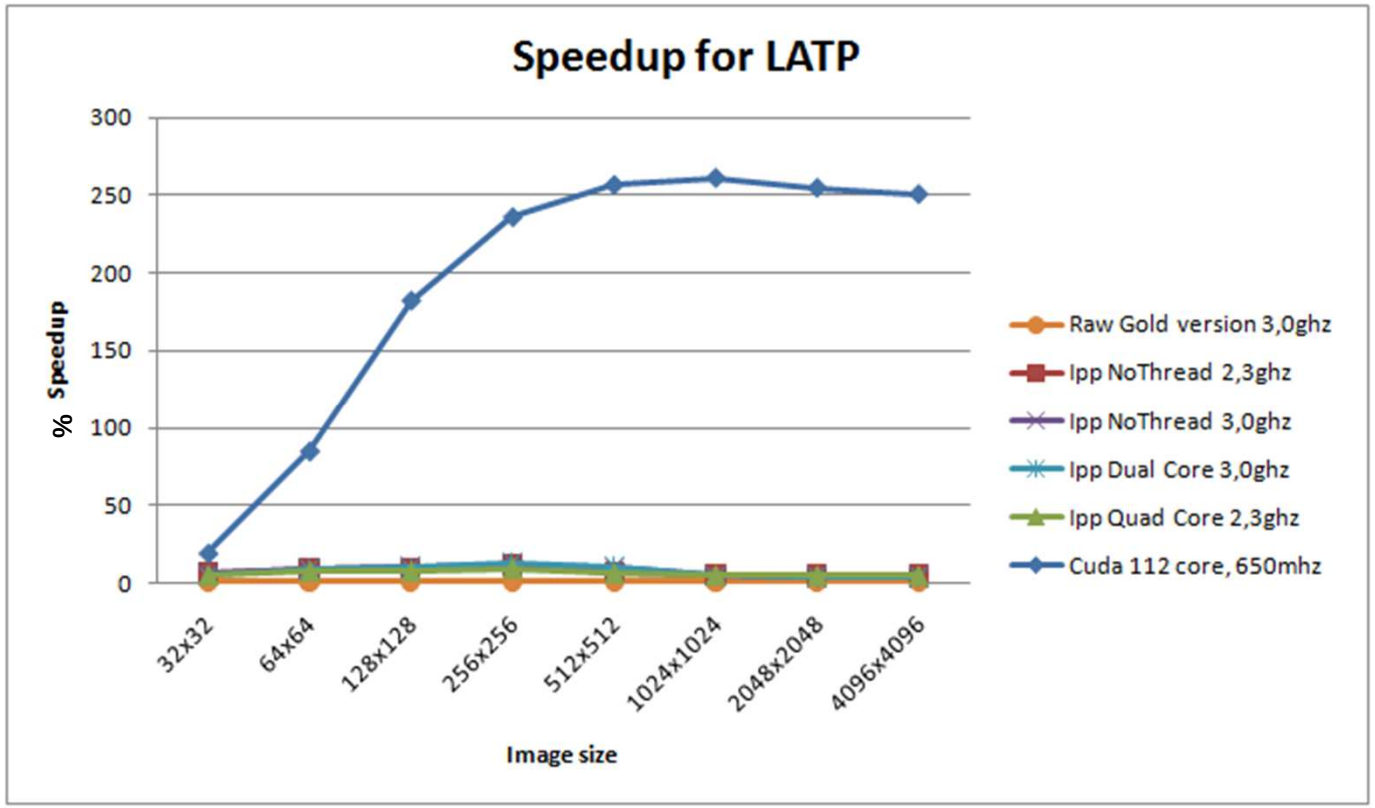
- Image de 256*256, les 448 cœurs de la C2050 font une différence.
- Meilleure optimisation pour l'architecture Fermi (C2050).





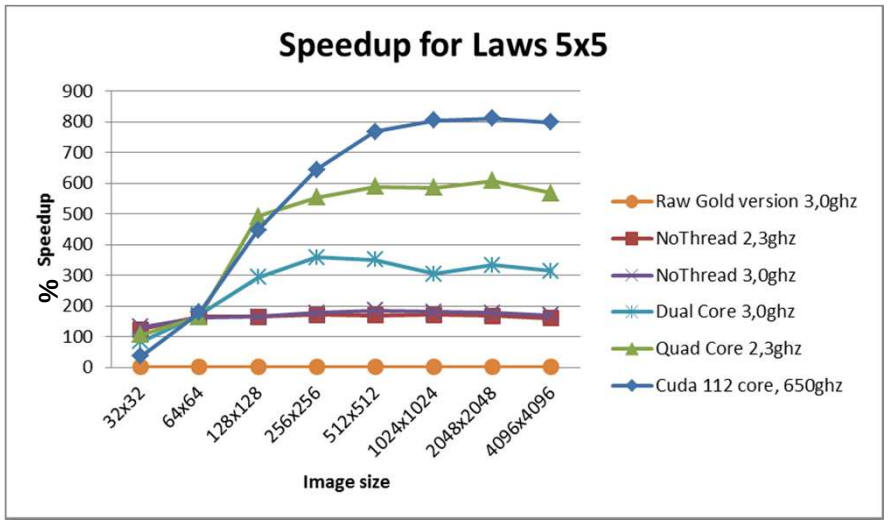
Résultats expérimentaux

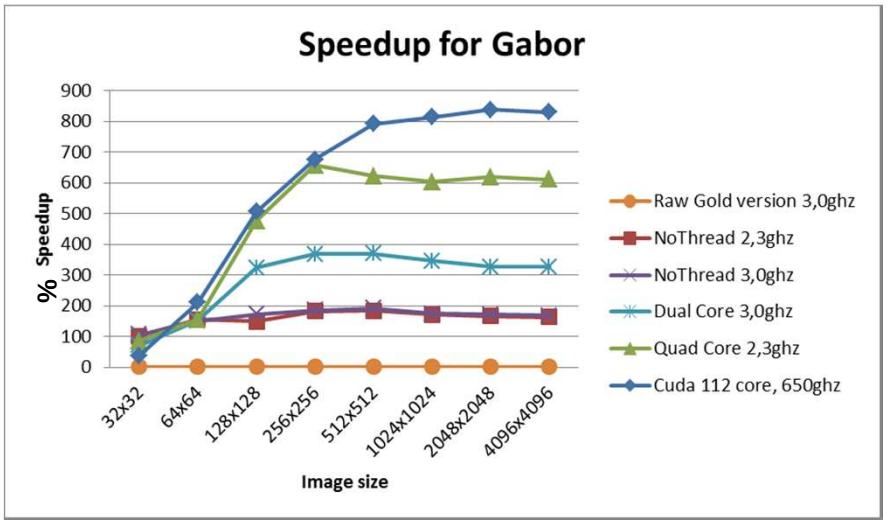
- Calcul complexe de l'operateur LATP marque la différence.



Résultats expérimentaux

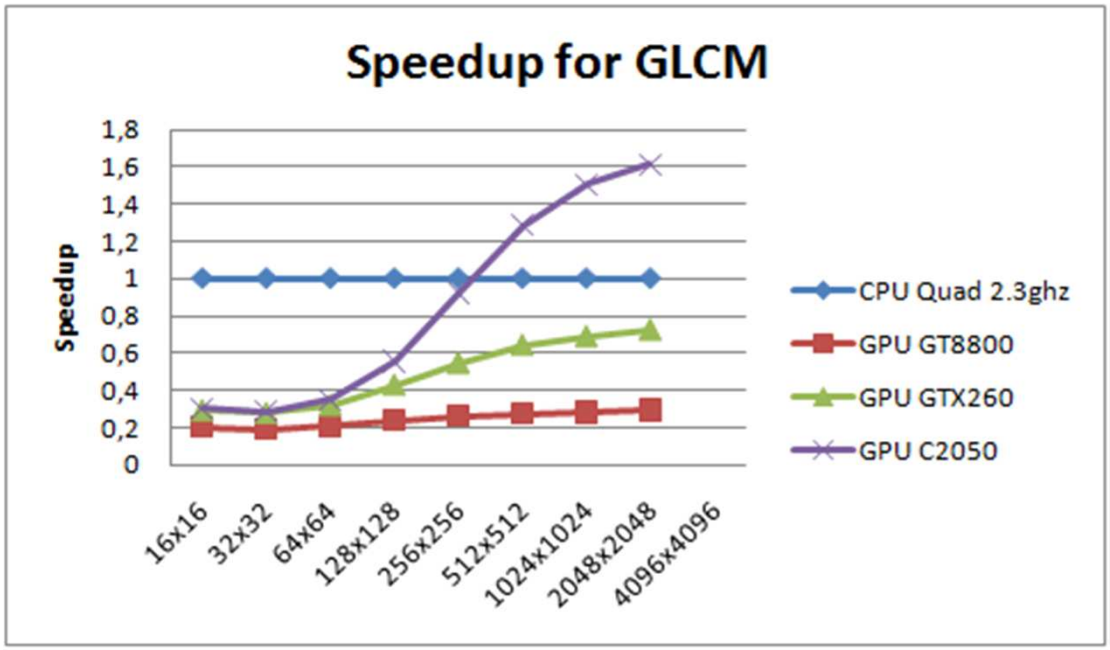
- Le MMX-SSE combiné avec Multicore donne des résultats intéressants pour de petites images.





Résultats expérimentaux

- L'accès mémoire non coordonné provoque des chutes de performance sur GPU.
- Optimisation plus intéressante sur C2050 (Fermi).



Résultats expérimentaux

- Avec optimisation sur GPU, mémoire partagé et autre.
- GPU GT8800 (112 coeurs)
- Gain LATP très intéressant.

	Réel 32 bits			Entier 32 bits			Entier 8 bits		
Dimension	**GAIN LBP**	**GAIN LTP**	**GAIN LATP**	**GAIN LBP**	**GAIN LTP**	**GAIN LATP**	**GAIN LBP**	**GAIN LTP**	**GAIN LATP**
16	0,2675	0,88	3,96	0,255	0,305	3,6925	0,255	0,285	3,795
32	0,165	0,61	3,595	0,17	0,1875	3,53	0,27	0,2475	4,5975
64	0,6275	2,3825	13,7625	0,64	0,7175	12,72	0,6475	0,725	12,7775
128	2,02	7,39	34,235	2,025	2,2475	37,025	2,1425	2,3	35,3975
256	4,9175	17,0325	74,26	4,875	5,1725	69,5925	4,895	5,0425	66,41
512	6,82	23,75	93,815	6,8425	7,445	88,235	6,955	7,265	78,11
1024	8,3625	28,8725	103,99	7,9925	8,575	96,8575	7,6125	7,9825	76,435
2048	9,2025	31,66	103,64	9,065	9,4	96,675	7,91	8,36	75,6025
4096	9,215	31,745	100,2125	9,3825	9,7575	95,5375	8,0725	8,45	75,0175

Conclusion

- Potentiel très intéressant pour l’extraction de caractéristiques sur des textures.
- Gain intéressant avec CPU et GPU sur de petites images.
- GPU surclasse le CPU sur de grandes images.
- Travaux futurs sur:
    - GTX 580
    - Réseau de neurones