L'arithmétique sur GPU en 2009

Le bon, la brute et le truand

Sylvain Collange

Lyon, 5 février 2009

ELIAUS

UPVD
Université de Perpignan Via Domitia

Travaux en cours

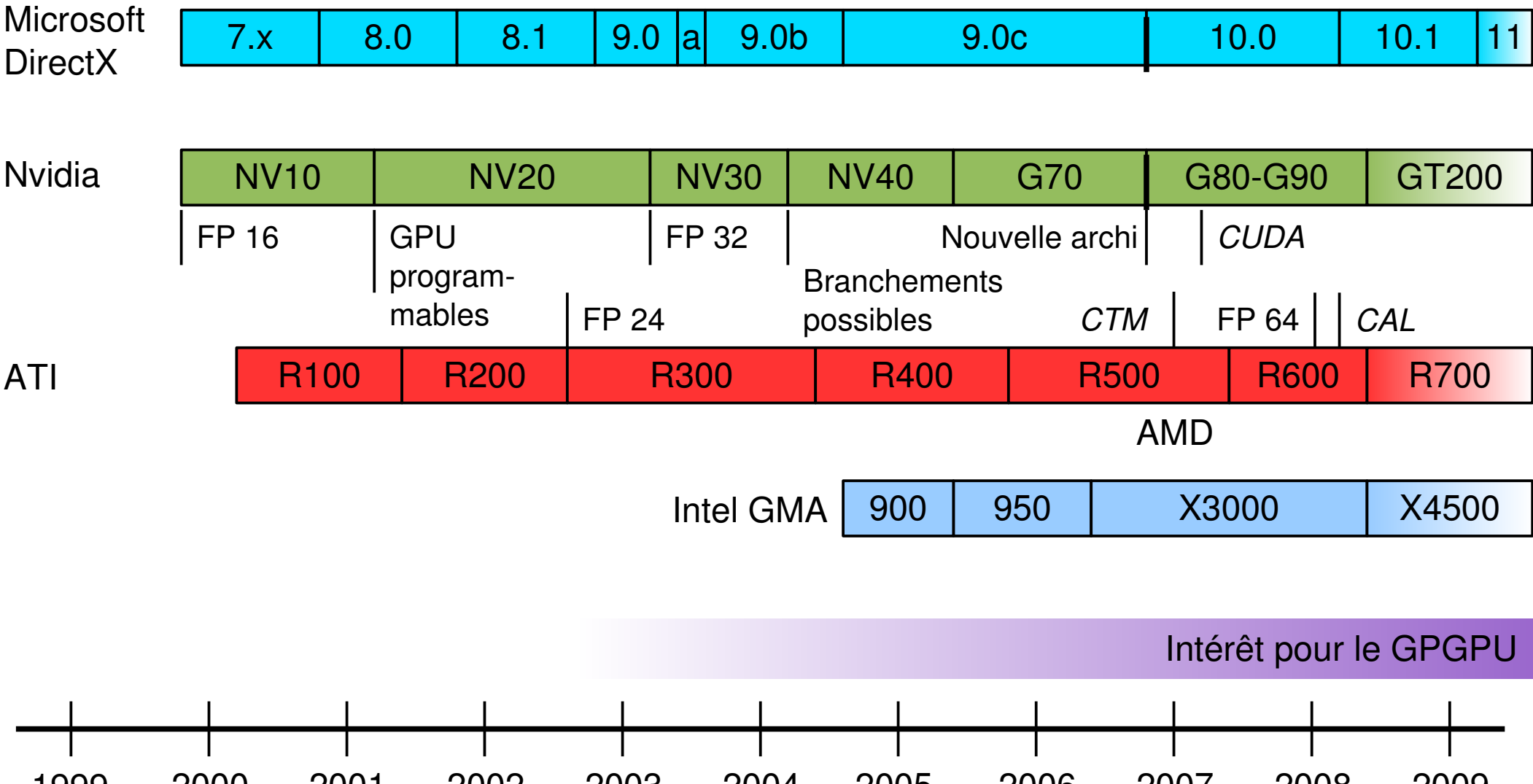
- Compréhension des architectures des GPU existants
  - Veille technologique
  - Tests
  - Algorithmes
  - Évolutions futures
- Développement d'un simulateur
  - Barra : un simulateur de GPU pour Cuda
- Propositions architecturales

S. Collange, D. Defour, D. Parello. Barra, a modular functional GPU simulator for GPGPU. Soumis à SAMOS 08.

Plan

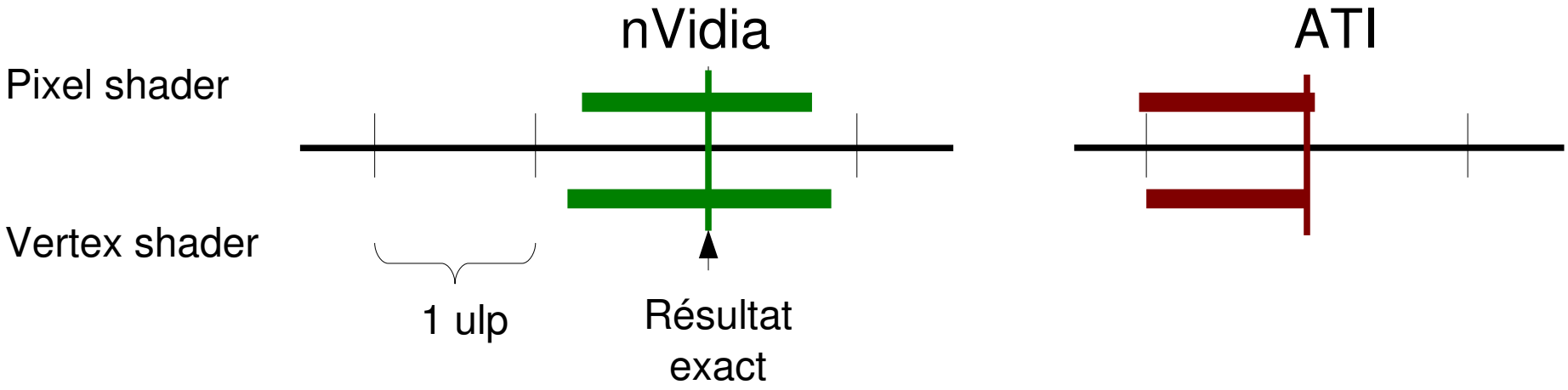
- Bref historique
- Enjeux
- Unités d'exécution
- Normes, outils
- Fonctions logicielles

Que s'est-il passé en 10 ans?



État de l'art en 2006

- Nvidia G70, ATI R500
- Arithmétique flottante “exotique”
  - Multiplieurs tronqués
  - Additionneurs avec 2 bits de garde sans sticky
  - int(14.0 / 7.0) = 1



S. Collange, M. Daumas, D. Defour. État de l'intégration de la virgule flottante dans les processeurs graphiques. *RSTI – TSI 27/2008, p. 719 – 733.* 2008

(R)évolutions en 2007

- Réécriture des couches logicielles
  - Microsoft DirectX 10
- Changements d'achitecture
  - Nvidia G80, AMD/ATI R600, Intel GMA X3100
- API spécifique GPGPU
  - Nvidia CUDA
- Régularisation de l'arithmétique
  - Arrondi au plus près IEEE, sans dénormaux
  - Calcul entier et logique

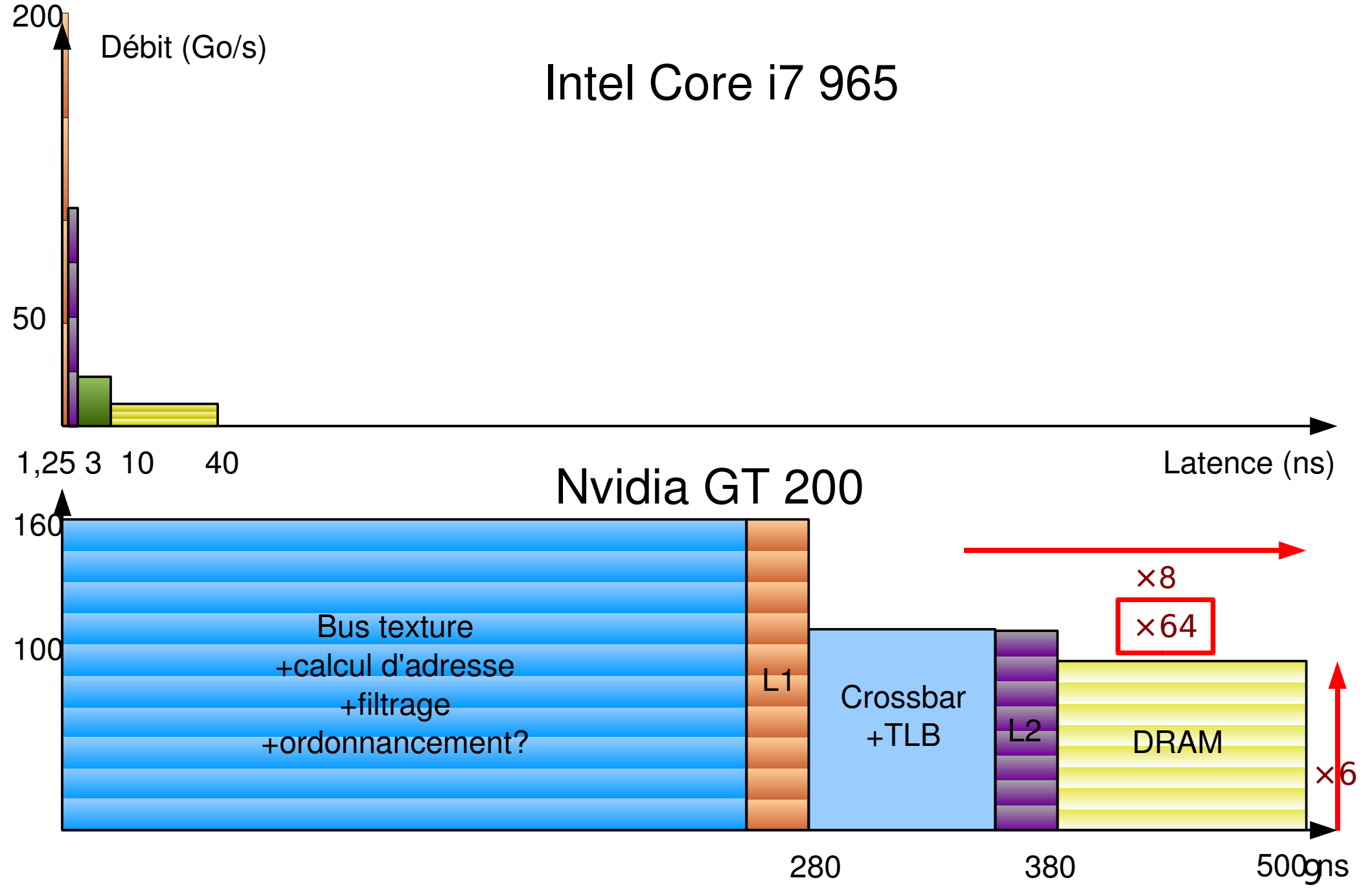
2008–2009

- Double precision
  - AMD RV670, RV770, Nvidia GT200
- API GPGPU standard
  - Kronos OpenCL 1.0
  - Microsoft Direct3D 11 (compute shader)

Plan

- Bref historique
- Enjeux
- Unités d'exécution
- Normes, outils
- Fonctions logicielles

Une architecture optimisée débit



Conséquences

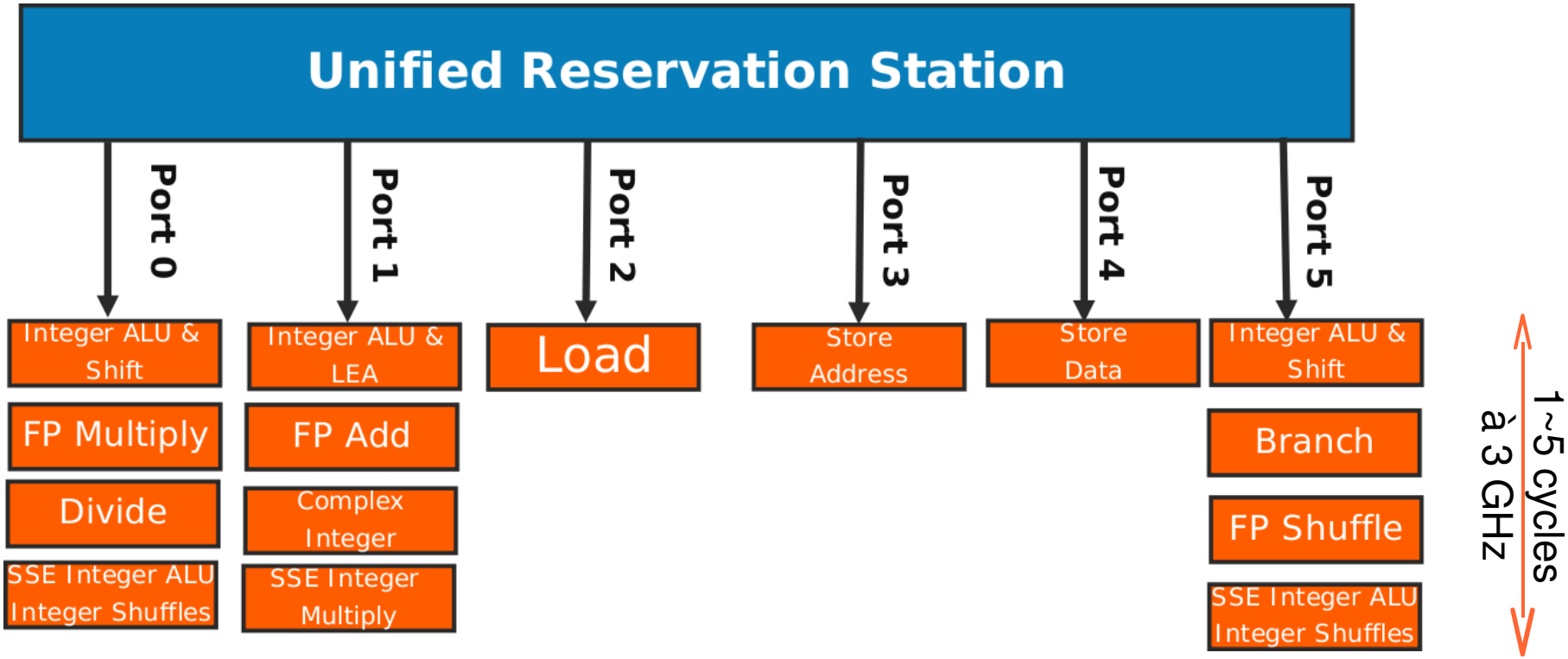
- GTX280 / Core i7 965
  - Puissance de calcul ×5
  - Débit mémoire×6
  - Latence ×8
  - Données en vol = parallélisme nécessaire ×64
- Exploitation du parallélisme de données par SIMD et multithreading matériel
- Nombre de threads en vol
  - Nvidia GT200 30 720 threads
  - AMD RV770 81 280 threads
- Permet de masquer les latences

Plan

- Bref historique
- Enjeux
- Unités d'exécution
- Normes, outils
- Fonctions logicielles

Un CPU : Intel Core i7

- Optimisé pour latence, consommation
- Grand nombre d'unités d'exécution séparées



Un GPU : Nvidia GT200

- Simple précision
- Arithmétique entière
  - Multiplieur 24x24 -> 32

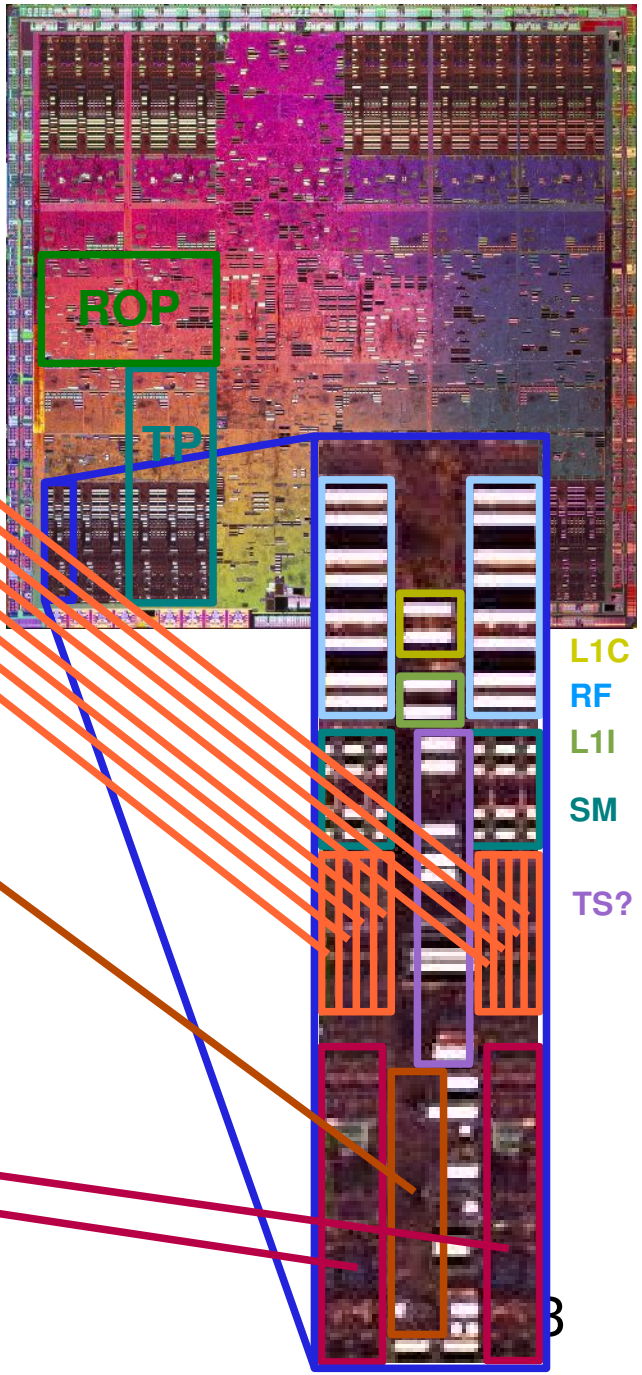
~11 cycles à 1,5 GHz

- Double précision
  - FMA

~19 cycles

- Interpolation + correction perspective
- Fonctions élémentaires SP
- Multiplication SP

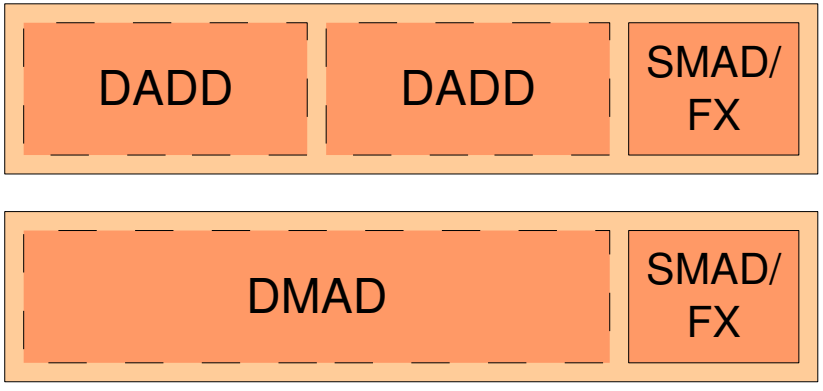
~21 cycles

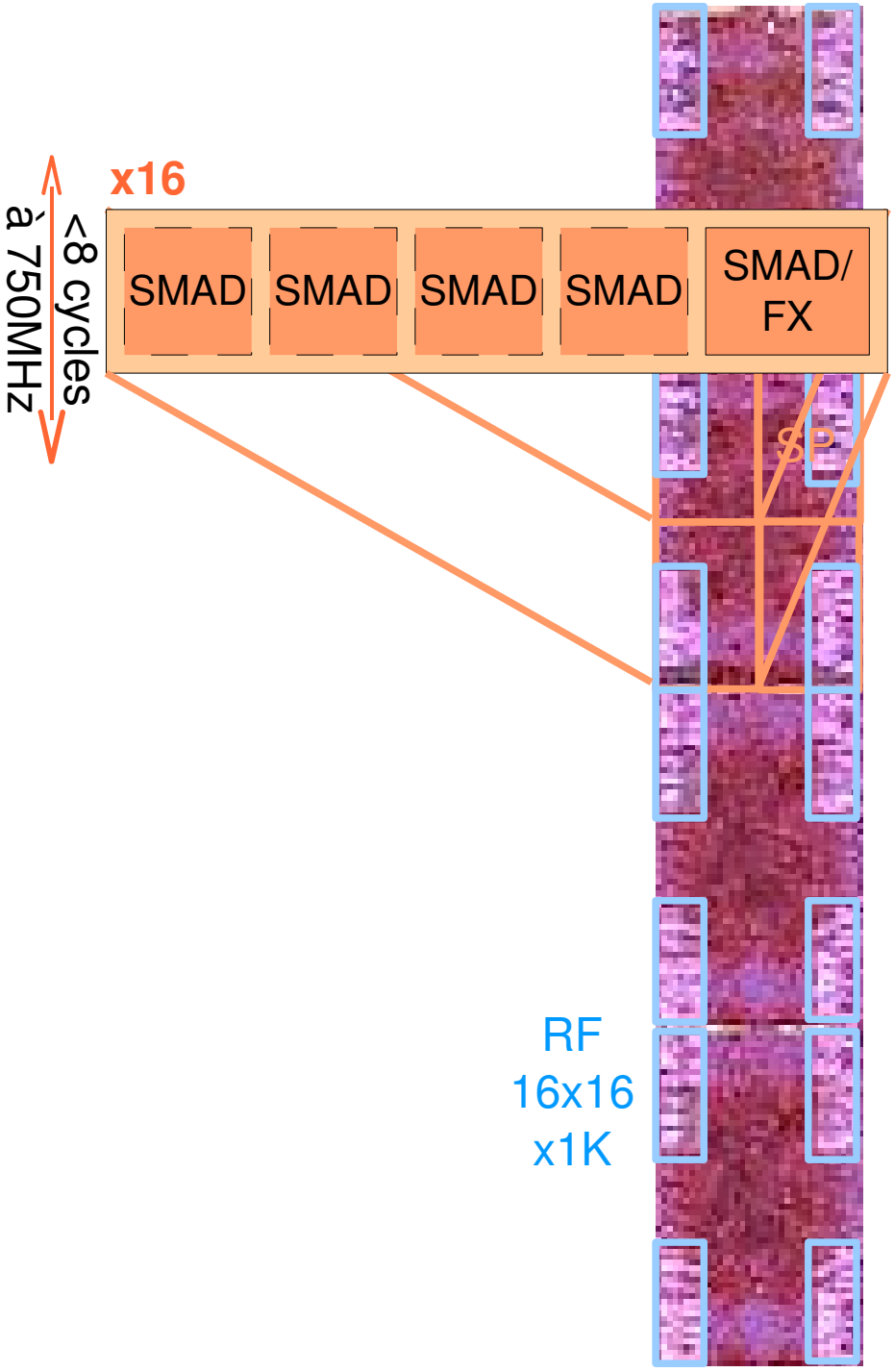


M. Siu, S. Oberman. US Patent 7225323, 2007.

Un autre GPU : AMD RV770

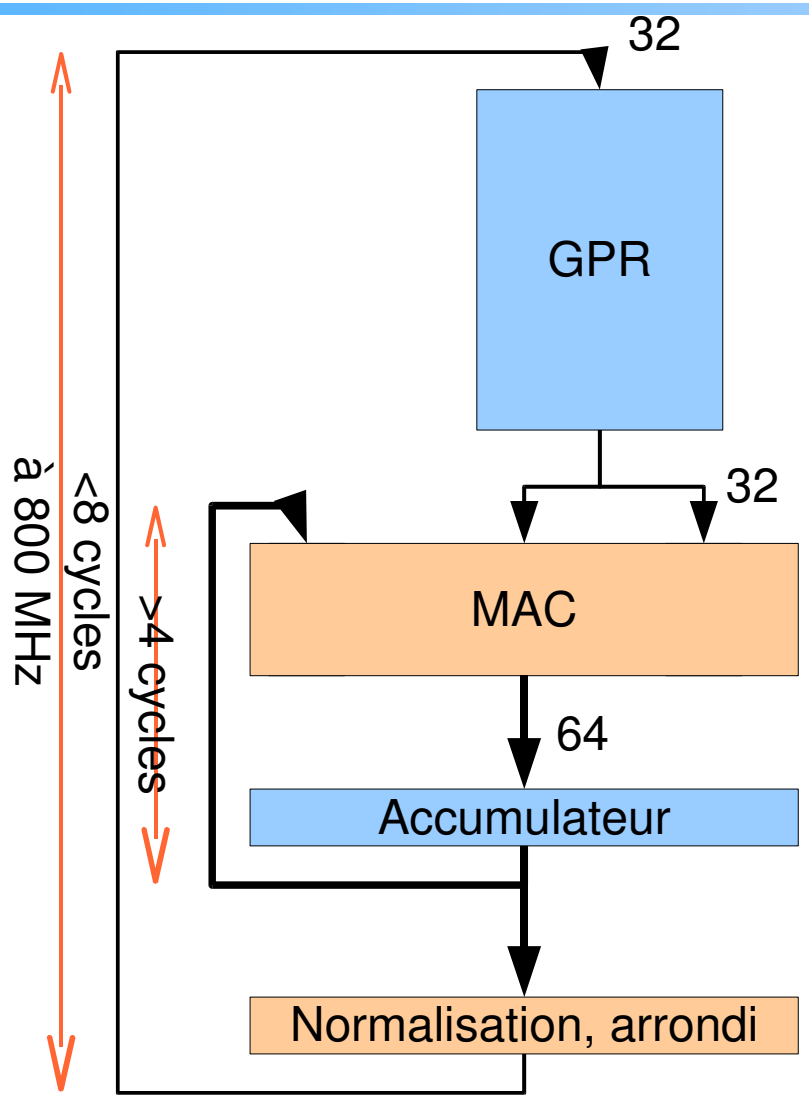
- Simple précision, calcul entier
- Double précision
  - Utilisation du même pipeline





Intel GMA X3100

- GPU+processeur vidéo
- Architecture type DSP
- Accumulateur long
  - Mantisse en complément à 2
  - Non normalisé
  - 54 bits = 2 * 24 + 5 + 1 signe
- Arithmétique entière
  - 4 x 32-bit ou 8 x 16-bit
- Multiplieur entier 16x32 -> 48



Largeur SIMD et unités vectorielles

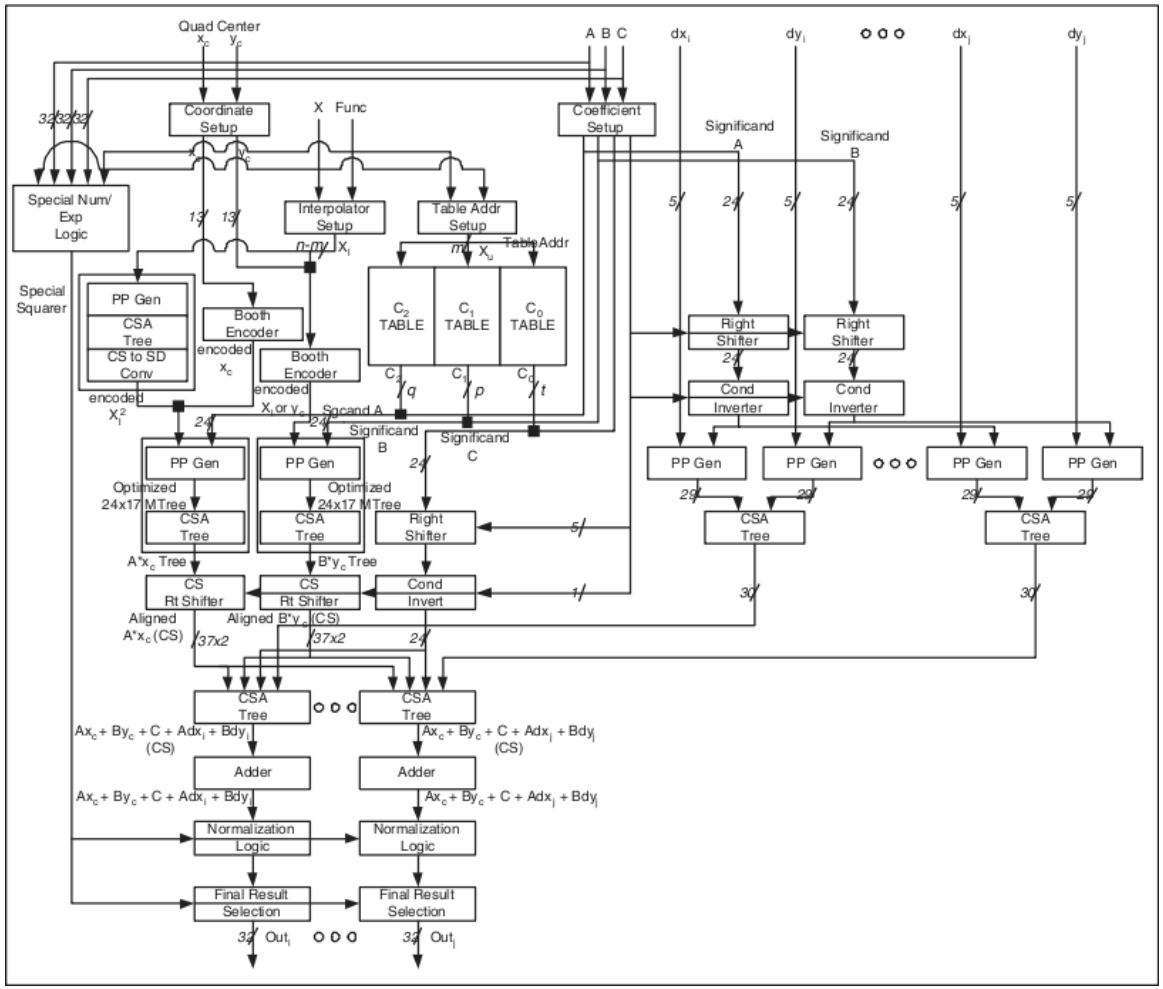
- Largeur d'un vecteur SIMD : donnée architecturale
  - Indépendante de l'implémentation
- Vecteur SIMD plus large que les unités
  - Sur CPU, conservation de la compatibilité
  - Sur GPU
    - Diminution de la pression sur les registres et le décodage
    - Réduction des dépendances entre instructions

	Largeur SIMD	Registres	Unités	Nb U	IPC
Intel Pentium 3, AMD K7	128-bit	64-bit	64-bit	2	1
Intel Pentium 4	128-bit	128-bit	64-bit	2	2
Intel Core 2, AMD K8	128-bit	128-bit	128-bit	3	3
Nvidia G80	1024-bit/512-bit	512-bit?	256-bit	2	½
AMD R600	2048-bit	512-bit	512-bit	5	1.25 (¼)
Intel GMA X3100	512-bit/256-bit	256-bit	128-bit	1	½

Fonctions élémentaires matérielles

- Sur GPU grand-public depuis DirectX 7.0 (1999)
  - Avant que les GPU ne soient programmables
- rcp(x)=1/x, rsq(x)=1/sqrt(x)
  - Normalisation de vecteurs
  - Projection, correction perspective
- $\log_2$, 2^x
  - Calculs d'éclairement
- sin, cos
  - Calculs géométriques
- Précision : 10 bits → 24 bits

Nvidia : tables + interpolation



Évaluation de fonction / Interpolation d'attributs

~21 cycles

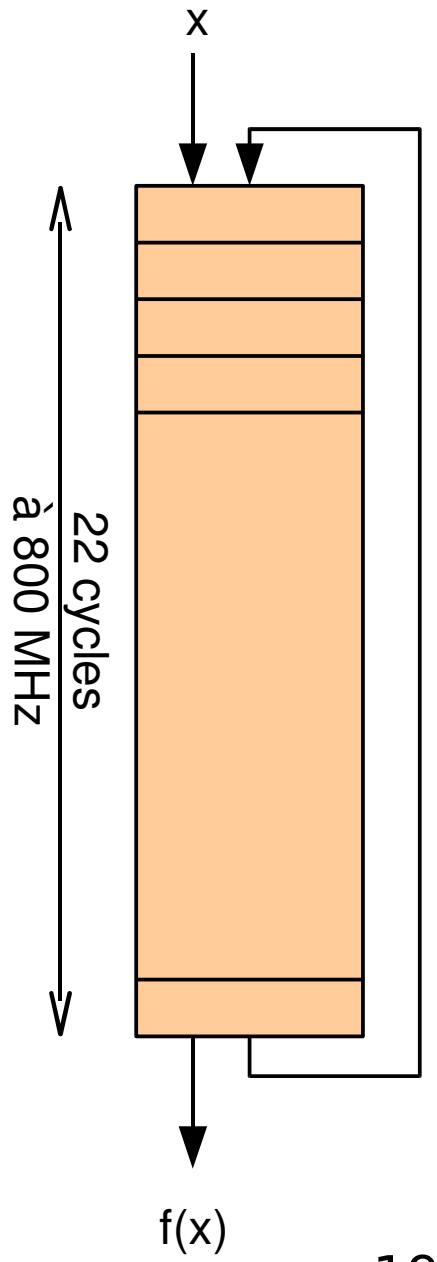
Mul | Mul | Mul | Mul

Multiplication / Correction perspective

S. Oberman, M. Siu. A High-Performance Area-Efficient Multifunction Interpolator. *Arith17*. 2005

Intel GMA

- Une unité pipelinée
- Plusieurs “tours” suivant la fonction, la précision voulue, l'argument
- Opérations en un tour
  - Inverse
  - Multiplication - addition?
- Autres fonctions : approx. polynomiale, fractions rationnelles?
- Réduction d'argument modulaire pour sin, cos
  - Avec arrêt anticipé



Plan

- Bref historique
- Enjeux
- Unités d'exécution
- Normes, outils
- Fonctions logicielles

DirectX 10.1

- Requiert arrondi correct au plus près pour 4 opérations
  - +, -, ×, /
- Nvidia : pas de support existant ni prévu
- AMD : R670 et R770 compatibles
  - D'après AMD
  - Nos tests sur la division :
    - Le compilateur génère toujours une multiplication par l'inverse en simple précision
    - Erreur jusqu'à 2 ulps
    - Encore int(14.0 / 7.0) = 1

Morpion arithmétique

	Format	FMA	Arrondis	Dénormaux	Inf, NaN	Flags	Exceptions
Intel X86	80 bits	✗	4 Dynamiques	Microcode	✓	✓	✓
IBM PowerPC	64 bits	✓	4 Dyn.	Microcode	✓	✓	✓
Intel IA-64	82 bits	✓	4 Dyn. + Stat.	Microcode	✓	✓	✓
IBM Cell SPU	32 bits	✓	RZ	✗	✗	✓	✗
	64 bits	✓	4 Dyn.	En sortie	✓	✓	✗
Nvidia GT200	32 bits	✗	2 Statiques	✗	✓	ND, Bug?	✗
	64 bits	✓	4 Statiques	✓	✓	Non docu.	✗
AMD RV770	32 bits	✗	RN	✗	✓	✗	✗
	64 bits	✗	RN	✗	✓	✗	✗
OpenCL	32 bits	Opt	RN	Opt	✓	✗	✗
	64 bits	✓	4 Statiques	✓	✓	✗	✗

Généralisation du FMA?

- AMD : SSE5
  - Dans Llano (32nm) vers 2011?
  - http://developer.amd.com/CPU/SSE5
- Intel : AVX 2e génération
  - Dans Haswell (22nm) vers 2012
  - http://software.intel.com/sites/avx
- VIA/Centaur : en matériel dans le Nano
  - Uniquement utilisé par le microcode?
- ICT/ST Microelectronics : Loongson 2E/2F (2007)

Nvidia : prédication et drapeaux

- Chaque instruction peut mettre à jour un registre de drapeaux
  - 4 registres différents, 4 bits par registre
- Prédication par un registre de drapeaux et une condition
- Émulation des drapeaux IEEE-754 :

```
           fmul p1|r3, r2, r1
 @!p1.nan  fadd p1|r2, r0, r2
 @!p1.nan  fmad p1|r4, r3, r2, r1  <- nan
 @!p1.nan  fmax p1|r1, r3, r4
 @p1.nan   call erreur_nan
           ...
```

- Non documenté, non supporté, non utilisé
- +inf détecté comme normal : bug matériel?

Plan

- Bref historique
- Enjeux
- Unités d'exécution
- Normes, outils
- Fonctions logicielles

Nvidia, AMD : racine carrée

- sqrt(x) = rcp(rsq(x))
- Pourquoi pas x * rsq(x) ?
  - Comportement correct pour x=0 et x=inf
- Pourquoi pas une autre instruction mul?
  - Le R600 a déjà plein de variantes du mul (compatible, DX9, DX10, IEEE-754...)
- Comme sur Itanium : prédicat
  - Faisable sur Nvidia?

```
          frsq p1|r2, r1
@!p1.le   fmul    r1, r1, r2   // Condition : (Normal ou NaN)
                               // Résultat dans r1
```

  - Bug du +inf non détecté
    - sqrt(-0) = -0 mais sqrt(+0) = NaN

Division entière

- Intel : en matériel
- Nvidia : algorithme “école primaire”
- AMD : Newton-Raphson en entier
  - Approximation initiale en virgule fixe, 23 bits

T. Rodeheffer. Software Integer Division. *Microsoft Research TR-2008-141.* 2008

Nvidia : division, racine carrée en double

- Déjà en matériel en simple précision
  - Approximation ~23 bits justes
  - Itérations de Newton-Raphson pour atteindre la double
- On a un FMA en double précision
  - Itérations de Markstein
- Conditions idéales pour du Newton-Markstein?

Cas délicat

- $1/ (1,11...1)_b$
  - Arrondi correct = 0,100...01 = 0,5 + 2^{-53}
- Si 1/b ≈ y = 0,5
  - Itération de Newton-Markstein

    $b = 2 - 2^{-52}$
    $e = 1-by \rightarrow e = 2^{-53}$
    $y' = y+ey \rightarrow y' = 0,5+2^{-54} = (1+2^{-53})2^{-1}$
  - Arrondi pair : y' arrondi à 0,5
  - Pas de convergence
- PowerPC, Itanium : choix d'une approximation initiale garantissant la convergence
- GT200 : approximation initiale non conçue pour Newton à l'origine

Algorithme Nvidia

```
                                                    a = 1, b = 2 – 2^(-52)

y0 = (double)(rcp(double2float_rz(b)))      y0 =  0.500000596046448
                     // y0 ≥ y

e1 = 1 – b * y0      // Markstein           e1 = -0.000000119209289
y1 = y0 + y0 * e1    // y1 ≤ y              y1 =  0.499999999999930

e2 = e1 * e1         // Goldschmidt         e2 =  0.000000000000142
y2 = y1 + y1 * e2                           y2 =  0.500000000000000

e3 = 1 – b * y2      // Markstein           e3 =  0.000000000000001
y3 = y2 + y2 * e3                           y3 =  0.500000000000000

q0 = a * y0          // q0 ≥ q              q0 =  0.500000596046448
r0 = a – b * q0                             r0 = -0.000000119209289
q1 = q0 + r0 * y2                           q1 =  0.500000000000001

r1 = a – b * q1                             r1 = -0.000000000000001
q2 = q1 + r1 * y3                           q2 =  0.500000000000001
```

AMD

- Comme Nvidia : 2 itérations de Newton-Markstein

```
y0 = (double)(rcp(float(b)))

e1 = 1 – b * y0      // M
y1 = y0 + y0 * e1

e2 = 1 – b * y1      // M
y2 = y1 + y1 * e2

q0 = a * y2
r0 = a – b * q           // M
q1 = q0 + r0 * y2
```

- Mais... sans FMA!
  - Arrondi fidèle pour le prix d'un arrondi correct
- Pas de racine carrée

Fonctions élémentaires

- Opérateurs matériels limités à quelques fonctions, faible précision (erreur relative, grands arguments...)
- AMD : fonctions matérielles et fonctions de HLSL (Microsoft Direct3D)
- Nvidia : libm dans CUDA
  - En simple et double précision
  - Reflète l'état de l'art, sans tables
  - Raisonnablement précise
    - 1 ulp pour logf, 2 ulps pour pow...
    - Trigo : réduction d'argument par Cody-Waite (SP proche), Boldo-Daumas-Li (DP proche) ou Payne-Hanek (lointain)
    - sinf(10^{22}) à 2 ulps

Conclusion

- Nombreuses applications à l'arithmétique des ordinateurs
- Constructeurs : exploitent la littérature existante

M. Schulte, E. Swartzlander. *Truncated multiplication with correction constant.* **VLSISP6.** 1993

U. Kulisch. *Circuitry for generating scalar products and sums of floating point numbers with maximum accuracy.* **US Patent 4622650.** 1986

M. Daumas, C. Mazenc, X. Merrheim, J.-M. Muller. *Modular Range Reduction: a new algorithm for fast and accurate computation of the elementary functions.* **JUSC.** 1995

P. Markstein. *IA-64 and elementary functions.* **HP prof. Books.** 2000

S. Oberman, M. Siu. *A High-Performance Area-Efficient Multifunction Interpolator.* **Arith17.** 2005

E. Schwartz et al. *Hardware implementations of denormalized numbers.* **Arith 16.** 2003

T. Rodeheffer. *Software Integer Division.* **Microsoft Research TR-2008-141.** 2008

W. Cody, W. Waite. *Software manual for elementary functions.* **Prentice Hall.** 1980

M. Payne, R. Hanek. *Radian reduction for trigonometric functions.* **SIGNUM.** 1983

S. Boldo, M. Daumas, R-C. Li. *Formally Verified Argument Reduction with a Fused-Multiply-Add.* **IEEE TC.** À paraître.

- Algorithmes / opérateurs spécifiquement adaptés aux contraintes des GPU?