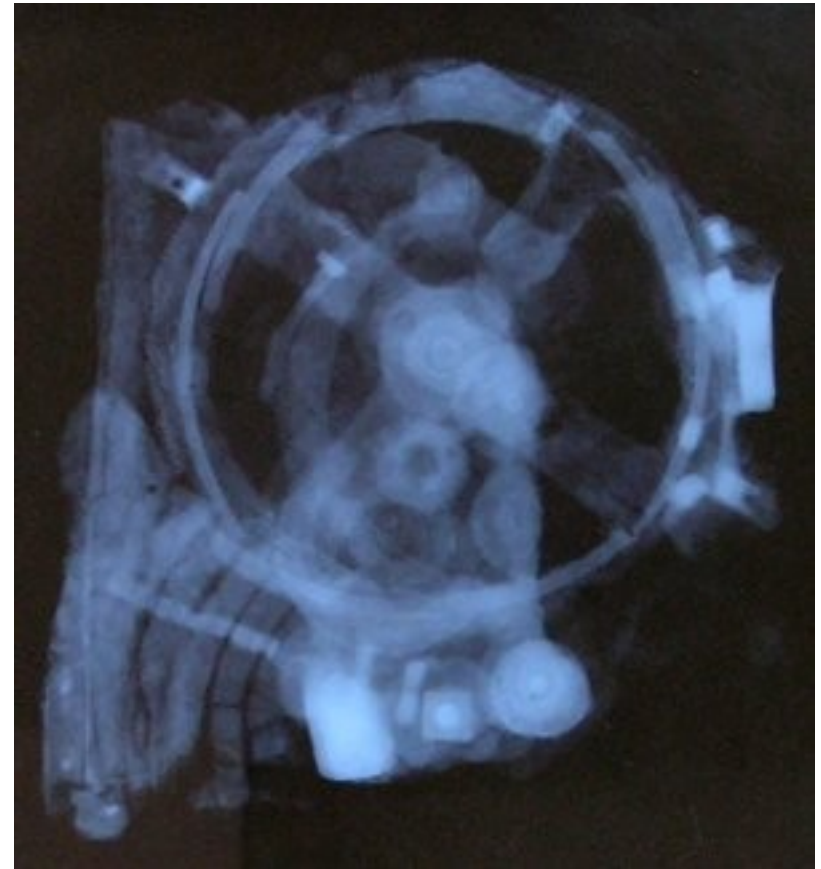


# Un GPU = un supercalculateur Du mythe à la réalité !



Tera10 au CEA



Machine d'Anticythère

# Avertissement !

- Ce que cette présentation n'est pas :
  - Une présentation exhaustive du GPU
    - GPU pour Graphics Processing Unit
  - Un cours de calcul scientifique
- Ce que cette présentation se veut :
  - Un rapide tour d'horizon de ce qui se dit (ne se dit pas)
  - Le retour d'une expérience « personnelle »
  - Quelques suggestions de « bonnes pratiques »

# Plan

- Légers retours en arrière
  - 1985 : *le petit film et le gros ordinateur*
  - 2000 : *la puissance est au bout du bus*
- Quelles différences entre (C/G)PU ? Un  $\pi$  ?
- 2 acteurs du jeu : Nvidia et ATI, la danse du Tflop
- Quelles approches pour quels coûts ?
- Une librairie comme arène : BLAS
  - Petits tests entre cartes...
- Conclusion (préliminaire)...

# Flash-Back 1985: « Starfighter »

- 25 minutes d'images de synthèse (3000x2000)
- Des scènes jusqu'à 600 000 polygones
- Pour en voir des exemples :
  - [http://www.break-in-studios.com/archive/TheLastStarfighter\\_CG.htm](http://www.break-in-studios.com/archive/TheLastStarfighter_CG.htm)
  - <http://firsthour.net/movie-review/the-last-starfighter>

# Flash-Back 1985 : « Starfighter »

- Quelles données ?
  - 25 minutes de film à 25 images par seconde
  - 22,5 milliards d'opérations pour une image
  - Un Cray XMP de 15 M\$ : 160 MFlops
- Quelle progression ? **x7000** !  
1984 : 2 mois de XMP      2009 : 13 m de GTX 285

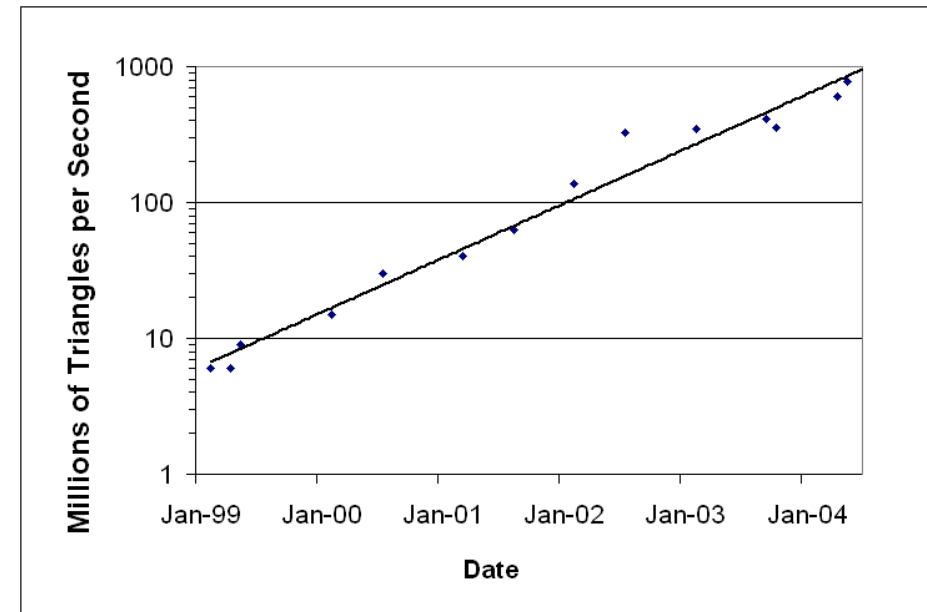


# Flash-Back : Méso-informatique 2006

- Journée à l'ENS-Cachan, fin février
  - Le MC (méso-calculateur), entre le PC et le HPC
- Pistes à explorer :
  - Python pour le calcul scientifique
  - Virtualisation pour l'apprentissage du //
  - Utilisation OpenGL pour le calcul scientifique

# La carte graphique et son langage

- 1999 – 2004 : puissance x100
- Un langage « standard » : l'OpenGL
- Des fonctions intéressantes
  - Transformation de coordonnées
  - Convolution
- Des limitations
  - Bus de communication (AGP)
  - Des calculs « rudimentaires »
    - Entiers, Virgule fixe, demi-flottante ou flottante
    - Des résultats peu contrôlés (ce n'était pas le but originel !)



# Progression CPU/GPU 10 ans

- CPU : *Nbench* **x15** (mémoire – entier – flottant)
  - 1999 : Pentium III 500 Mhz (2 – 1,7 – 3,2)
  - 2009 : Intel Core i7 (30 – 26,8 – 48,5)
- GPU : *Texture Fill Rate* **x368**
  - 1999 : Riva TNT2 250 Mpixels/s
  - 2009 : GTX 295 92160 Mpixels/s
- Croissance GPU >> Croissance CPU
  - Plus de 20x pour le processeur graphique
- Tout ça pour faire des « Starfighter » en temps réel



# « Nvidia Launches Tesla Personal Supercomputer »

- **Quand** : le 19/11/2008
- **Où** : sur Tom's Hardware
- **Quoi** : une carte C1060 PCIe avec 240 coeurs
- **Combien** : **933 Gflops SP** (et **78 Gflops DP**)

**Qui** : Nvidia



# 2 acteurs : Nvidia & ATI

- Nvidia : l'initiateur !
  - Cg Toolkit : un premier pas
  - CUDA : une vraie révolution
    - Coller aux besoins des scientifiques : CuBLAS, CuFFT
    - Disposer d'un maximum de wrapper : Python, Java, (Matlab)
    - Promouvoir le langage par la documentation : exhaustive !
  - OpenCL : un pas vers la standardisation...
- ATI : quelques trains de retard...
  - Le langage «CAL» pour commencer
  - ACML (venant d'AMD) pas intégré : seuls (S/D)GEMM
  - OpenCL : un pas vers la standardisation...

# Le pied à l'étrier : Nvidia et ATI

- 2 Implémentations : CUDA et Stream
- Ce qu'il faut (avoir) :
  - Une carte graphique compatible
  - Un pilote propriétaire compatible avec son OS
- Ce qu'il faut (faire) :
  - Installer le pilote propriétaire
  - Installer les bibliothèques
  - Installer le SDK pour lancer les exemples
- Ce qu'il faut (en penser) : à vous de voir !

# Implémentation sous Nvidia

- Une carte compatible :
  - Pas trop ancienne (en gros, moins de 3 ans)
  - <http://en.wikipedia.org/wiki/CUDA>
- Un OS compatible : *pour moi, c'est Debian Lenny*
- Installation :
  - Pilote *Developer Drivers version 195.36.15* : ok !
  - Librairies *CudaToolkit 3.0 Ubuntu 9.04* : ok !
  - Exemples *GPU Compatible SDK* : ok !
- Exécution des exemples CUDA et OpenCL : ok !

# Implémentation sous ATI

- Une carte compatible
  - Plutôt récente (au moins 4350)
  - <http://developer.amd.com/gpu/ATIStreamSDK/Pages/default.aspx>
- Un OS compatible : Debian Lenny (ou Squeeze)
- Installation
  - Pilote « normal » 10.4 ( Xorg 7.4 ET Xorg 7.5) : ok !
  - Librairies ati-stream-sdk-v2.1-lnx(32-64).tgz : ok !
    - Nécessaire : Lien librairies vers /usr/lib/OpenCL/vendors/
    - Indispensable : Installation de l'ICD de ATI (pour SDK 2.1)
- Exécution des exemples CAL et OpenCL : ok !

# Quelques soucis

- Attention !
  - Sur Nvidia :
    - Le double-précision pas encore généralisé : > GTX260
    - L'absence totale de retour d'erreur
  - Sur ATI :
    - Une disposition de la double-précision « bizarre »
      - Le 4360 à 50€ l'a mais pas la 5570 à 150€
    - Le lancement des exemples
      - Attention au X11 Forwarding en SSH : les exemples ne lancent plus
      - Utilisation du « ssh -x » pour le désactiver
  - Sur TOUS : un goulot d'étranglement, le **Lien...**
    - Et donc, nécessité de jouer avec les échanges mémoires

# Comment programmer en GPU ?

- **2 approches**

- Une approche « **intégrateur** »

- Le code utilise des bibliothèques génériques
- Le code n'est modifié que pour remplacer ces appels

- Une approche « **développeur** »

- Le GPU est un nouveau processeur
- Il exige un apprentissage comme tout nouveau matériel
- Le code doit être réécrit pour l'utiliser au mieux

- **1 contrainte mais 2 manifestations : le temps**

- Le temps de **programmation** : plutôt intégrateur
- Le temps **d'exécution** : plutôt développeur

# Le temps et les coûts

- Coûts d'entrée
  - Appréhender la technologie
  - Evaluer les gains en performance
- Coûts de sortie
  - Si technologie propriétaire, addiction...
- Coûts de MCO
  - Le Maintien en Conditions Opérationnelles
  - Des technologies spécifiques donc rares et chères...
- Une (sinon **LA**) solution : l'Open Source...



# Ma (timide) expérience : CuBLAS

- Qu'est-ce que BLAS ?
  - Basic Linear Algebra Subroutines
- Pourquoi faire ?
  - De l'algèbre linéaire (mais simple)
- Comment faire ? 3 types de fonctions
  - 1er type : manipulation de vecteurs
  - 2ème type : manipulation de matrices/vecteurs
  - 3ième type : manipulation de matrices/matrices
    - En plus, résolution de systèmes linéaires (simples...)

# De BLAS à CuBLAS

- Utilisation de BLAS
  - Résolution de systèmes linéaires complexes
  - Exploitation dans d'autres bibliothèques
    - MUMPS : résolution de matrices creuses
  - LINPACK : test utilisé pour le TOP 500
- Intégration/Implémentation
  - Propriétaires : MKL, ACML, MathKeisan, **CUDA**,
  - Open Source : CBLAS, FBLAS, GSL,
  - Autres : GotoBLAS

# CuBLAS : Approche descendante

- « Top-down » : conversion d'un programme
  - Une approche cependant naïve :
    - J'ai un programme qui utilise BLAS
    - Je trouve les appels de fonction BLAS
    - Je les remplace par des appels CuBLAS
    - Je recompile : ça marche !
    - J'exécute : ça NE marche PAS (enfin presque...) !
  - Parce que :
    - Des appels identiques mais pas les mêmes prototypes
    - Une base de conversion sur CBLAS et non FBLAS

# CuBLAS : Approche ascendante

- « Bottom-up » : apprentissage // des xBLAS
  - Une approche progressive
    - J'ai des fonctions élémentaires d'algèbre linéaire
    - Je les utilise pour établir un bench :
      - Je génère un vecteur X de dimension N
      - Je génère une matrice triangulaire A de dimension NxN
      - J'applique l'opération A.X qui donne B
      - Je résous le système pour trouver Y tel que :  $A.Y=B$
      - Je compare Y à X
    - Je programme en FBLAS
    - Je généralise en CBLAS et GSL en ajoutant des directives
    - Je programme en CuBLAS « use thunking »
    - Je programme en CuBLAS natif
    - Ca marche ! (pour des dimensions raisonnables)
  - Quels résultats ?

# Approche ascendante : (C/Cu)BLAS

## Avec CBLAS

```
cblas_dgemv(CblasRowMajor,CblasNoTrans,dim,dim,alpha,A,dim,X,incx,beta,Y,incx);  
cblas_dtrsv(CblasRowMajor,CblasUpper,CblasNoTrans,CblasNonUnit, dim,A,dim,Y,incx);  
cblas_daxpy(dim,beta2,Y,incx,X,incx);  
checksA[i]=(double)cblas_dnorm2(dim,X,incx);  
cblas_dswap(dim,X,incx,Y,incx);
```

## Avec CuBLAS version « use Thunking »

```
CUBLAS_DGEMV(&trans,&dim,&dim,&alpha,A,&dim,X,&incx,&beta,Y,&incx);  
CUBLAS_DTRSV(&uplo,&trans,&diag,&dim,A,&dim,Y,&incx);  
CUBLAS_DAXPY(&dim,&beta2,Y,&incx,X,&incx);  
checksA[i]=(double)CUBLAS_DNRM2(&dim,X,&incx);  
CUBLAS_DSWAP(&dim,X,&incx,Y,&incx);
```

# Approche ascendante : CuBLAS natif

## Déclaration, réservation et copie dans GPU

```
stat1=cublasAlloc(dim*dim,sizeof(devPtrA[0]),
(void**)&devPtrA);
stat2=cublasAlloc(dim,sizeof(devPtrX[0]),(void**)&devPtrX);
stat3=cublasAlloc(dim,sizeof(devPtrY[0]),(void**)&devPtrY);
if ((stat1 != CUBLAS_STATUS_SUCCESS) ||
    (stat2 != CUBLAS_STATUS_SUCCESS) ||
    (stat3 != CUBLAS_STATUS_SUCCESS)) {
    wrapperError ("Strsv",
CUBLAS_WRAPPER_ERROR_ALLOC);
    cublasFree (devPtrA);
    cublasFree (devPtrX);
    cublasFree (devPtrY);
    return;
}
stat1=cublasSetMatrix(dim,dim,sizeof(A[0]),A,dim,devPtrA,dim);
stat2=cublasSetVector(dim,sizeof(X[0]),X,incx,devPtrX,incx);
stat3=cublasSetVector(dim,sizeof(Y[0]),Y,incx,devPtrY,incx);
if ((stat1 != CUBLAS_STATUS_SUCCESS) ||
    (stat2 != CUBLAS_STATUS_SUCCESS) ||
    (stat3 != CUBLAS_STATUS_SUCCESS)) {
    wrapperError ("Strsv", CUBLAS_WRAPPER_ERROR_SET);
    cublasFree (devPtrA);
    cublasFree (devPtrX);
    cublasFree (devPtrY);
    return;
}
```

## Calcul

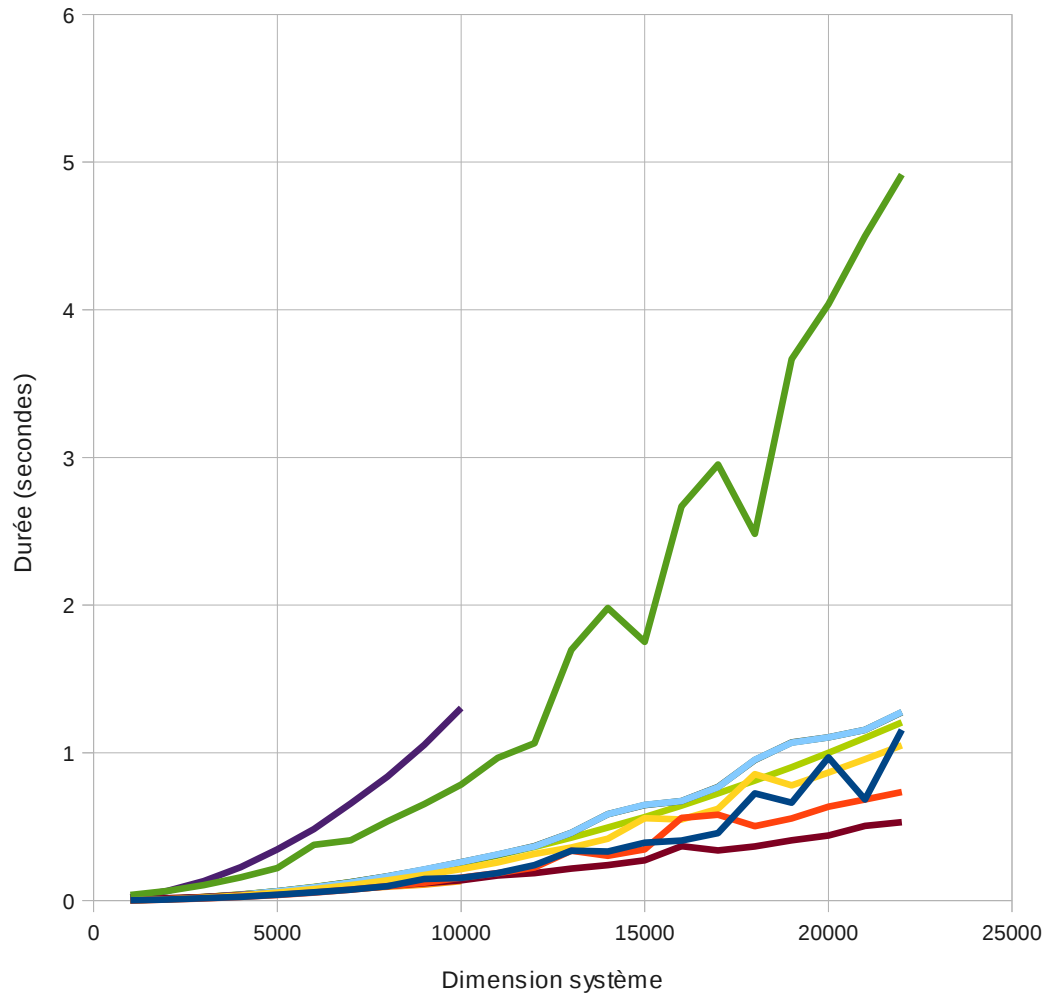
```
cublasDgemv(trans,dim,dim,alpha,devPtrA,dim,
devPtrX,incx,beta,devPtrY,incx);
cublasDtrsv(uplo,trans,diag,dim,devPtrA,dim,
devPtrY,incx);
cublasDaxpy(dim,beta2,devPtrY,incx,devPtrX,incx);
checksA[i]=(double)cublasDnrm2(dim,devPtrX,incx);
cublasDswap(dim,devPtrX,incx,devPtrY,incx);
```


## Copie résultats, libération GPU

```
stat1=cublasGetVector(dim,sizeof(X[0]),devPtrX,incx,
X,incx);
stat2=cublasGetVector(dim,sizeof(Y[0]),devPtrY,incx,
Y,incx);
cublasFree (devPtrA);
cublasFree (devPtrX);
cublasFree (devPtrY);
if ((stat1 != CUBLAS_STATUS_SUCCESS) ||
    (stat2 != CUBLAS_STATUS_SUCCESS)) {
    wrapperError ("Strsv",
CUBLAS_WRAPPER_ERROR_GET);
```

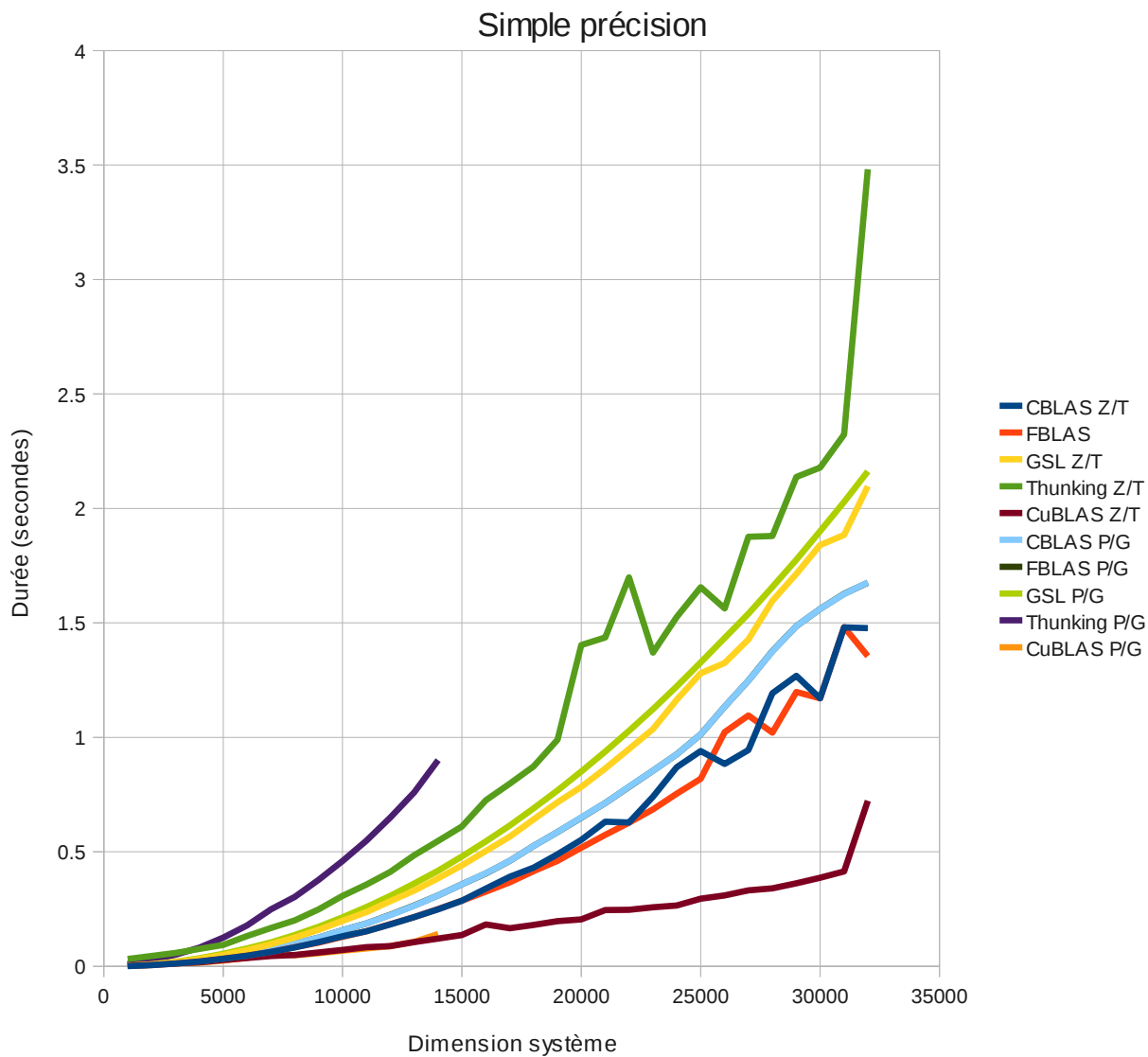
# Approche ascendante : résultats ?

Double précision



- **Double Précision**
- 2 bancs :
  - Z800+Tesla
  - P390+GTX260
- 10 jeux de tests
- CBLAS ~ FBLAS
- Thinking : 
- CuBLAS : x2
- Coupure : RAM !

# Approche ascendante : résultats ?



- **Simple Précision**
- 2 bancs :
  - Z800+Tesla
  - P390+GTX260
- 10 jeux de tests
- CBLAS ~ FBLAS
- Thinking : 😞 😞
- CuBLAS : x3
- Coupure : RAM !



# CuBLAS : retour sur xHPL

- 1 mois après l'échec, reprise de xHPL
  - Primitives de CuBLAS basées sur FBLAS, pas CBLAS
  - Expérience sur « USE\_THUNKING » pratique...
- Conversion par « copier/coller/remplacer »
- Succès !
- Alors, quel gain ?
  - Une carte : 28 Gflops (au lieu de 78...)
  - Des bizarreries dans les dépassements de capacité...
- Tout ça pour ça ?

# xHPL : une routine « transposée »

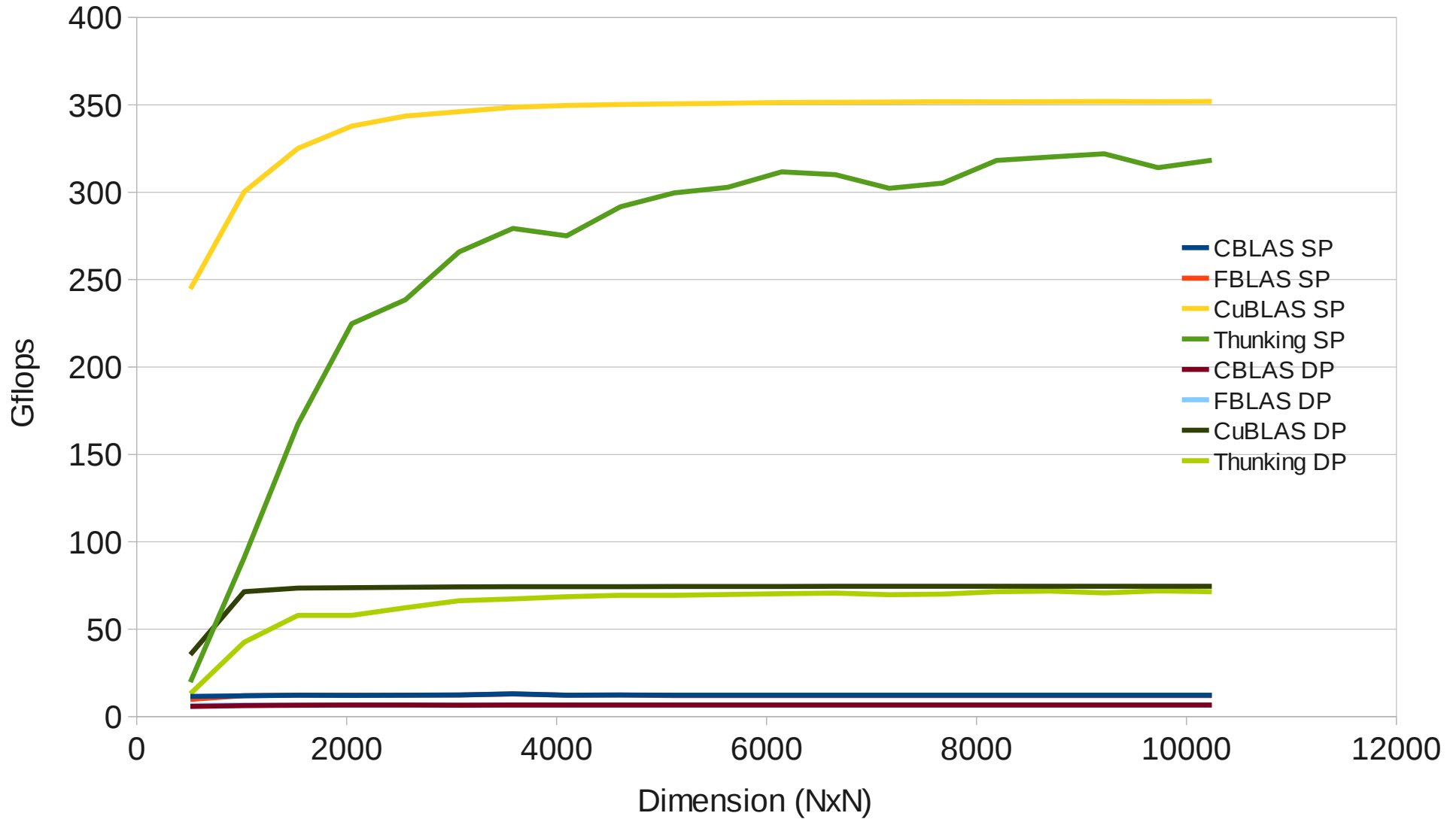
```
FBLAS
#ifdef HPL_CALL_FBLAS
    double          alpha = ALPHA;
#ifdef HPL_USE_F77_INTEGER_DEF
    const F77_INTEGER      F77N = N, F77incx = INCX, F77incy = INCY;
#else
#define F77N          N
#define F77incx      INCX
#define F77incy      INCY
#endif
    F77daxpy( &F77N, &alpha, X, &F77incx, Y, &F77incy );
#endif

CuBLAS
#ifdef HPL_CALL_CUBLAS
    double          alpha = ALPHA;
#define CUBLASN          N
#define CUBLASincx      INCX
#define CUBLASincy      INCY
    CUBLAS_DAXPY( &CUBLASN, &alpha, X, &CUBLASincx, Y, &CUBLASincy );
#endif
```








# Retour approche ascendante...

- Simplifier le banc (mais conserver le « test »)
  - Uniquement résolution de système : xTRSV
    - ☹️ – Tous les tests sont comparables (aucun gain...)
  - Uniquement produit matriciel : xGEMM
    - Propriété : Transposé (A \* B) = Transposé(B) \* Transposé(A)
    - Résultats (en Gflops) : **Yesss !**
      - SP : FBLAS/CBLAS : **12**, CuBLAS : **350/327** : x27 !!!
      - DP : FBLAS/CBLAS : **6**, CuBLAS : **73/70** : x11 !
    - Surprise : **Ah !!! CuBLAS préfère les x16 !**
      - SP : 16000<sup>2</sup>, **350**, mais 15999<sup>2</sup> ou 16001<sup>2</sup>, **97** : x3,6 !
      - DP : 10000<sup>2</sup>, **73**, mais 9999<sup>2</sup> ou 10001<sup>2</sup>, **31** : x2,35
- De réels gains, mais à certaines conditions...

# Retour approche ascendante



# Alors, que faire ?

-  Attendre Larrabee d'Intel ?
  - 12/2009, Intel annonce son retrait
-  Se lancer sur d'autres processeurs, comme Cell ?
  - 11/2009, IBM annonce l'arrêt de développement
-  Utiliser CUDA avec CuBLAS et CuFFT et Nvidia
  - Une solution rapide à court terme, moindre coût
-  Utiliser Streams
  - Aucune facilité d'utiliser les anciennes librairies (ACML)
-  Se lancer dans OpenCL ?
  - Une référence, un standard, quelques docs, mais tout à refaire
-  Pas encore d'implémentation Open Source
-  Sombrier dans l'addiction : HMPP ou compilateur PGI

# Conclusion préliminaire

- 3 activités du calcul scientifique et quelle utilisation des GPU ?
  - **Visualisation** : un acteur incontournable
  - **Traitement de données** : une alternative aux DSP
  - **Simulation numérique** : une utilisation sous haute surveillance
- Et si finalement le GPU n'était qu'un
  - Calculateur spécialisé, avec une puissance colossale, mais...
    - Des contraintes d'exploitation
    - Des domaines d'application spécifiques
  - Calculateur « analogique »
    - Un retour aux « phénomènes » physiques, mais lesquels ?
- Mais GPU devient GP GPU
  - (GP comme *General Purpose*, bientôt *JeePU*)

# Épilogue

- Des bonnes pratiques
  - Du code « générique »
    - des bibliothèques éprouvées pour pérenniser l'ensemble
    - des directives pour permettre de rajouter d'autres technologies
  - Des tests (permanents, et ce n'est pas plus mal !)
- Une période charnière
  - 2(3 ?) acteurs et 1 standard (OpenCL) en émergence
  - Une re«fusion» du coprocesseur dans le processeur ?
- A suivre (par OpenGPU)

# Sites

- [http://www.gpureview.com/show\\_cards.php?card1=](http://www.gpureview.com/show_cards.php?card1=)
- [http://developer.nvidia.com/object/cuda\\_3\\_0\\_downloads.html](http://developer.nvidia.com/object/cuda_3_0_downloads.html)
- [http://developer.amd.com/gpu/ATIStreamSDK/assets/ATI\\_Stream\\_SDK\\_Getting\\_Started\\_0](http://developer.amd.com/gpu/ATIStreamSDK/assets/ATI_Stream_SDK_Getting_Started_0)
- <http://www.generation-nt.com/intel-abandon-projet-gpu-larrabee-carte-graphique-actualite-9>