ÉCOLE NORMALE SUPÉRIEURE DE LYON

Formation CIRA : « Calcul Intensif en Rhône Alpes »

GPU par « intégration »

« Que l'on me donne six heures pour couper un arbre, j'en passerai quatre à préparer ma hache. »
Abraham Lincoln

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Développer ou Intégrer ?

- Qui programme :
  - Sa propre FFT ?
  - Sa propre multiplication matricielle ?
  - Ses propres librairies graphiques ?
  - Son propre langage de programmation ?
- Tout est affaire de « grain » !

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Développer ou intégrer ?

Développer (de zéro)

D'une grosse masse (projet)...
... par affinages successifs (code)...
... à un produit fini (application).

Intégrer

De composants (*framework*)...
... par assemblage...
... à un produit fini (application).

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Approches descendante ou ascendante ?

Descendante ou « top-down »
Une idée générale...
... se raffine en composants...
... de plus en plus petits...
... pour parvenir au grain souhaité.

Ascendante ou « bottom-up »
Chaque composant est connu...
...son intéraction est maîtrisée...
... il s'assemble à chaque autre...
... pour le former le projet global.

ÉCOLE NORMALE SUPÉRIEURE DE LYON

COTS : un acronyme (mé)connu ? « Commercial Off-The-Shelf »

Revue « Military & Aerospace Electronics »

COTS : 12 occurrences

Dans « ***Robots in combat missions*** » (page 26)

*« The DOD has mandated the use of **open standards** that **enable interoperability**, such as COM Express, VPX, and PC/104, because of the **cost savings** they deliver. »*

*« Commercial off-the-shelf (**COTS**) components also **allow** manufacturers to improve time-to-market, getting **new technologies deployed** in the field **faster**. »*

*« **Open-standard COTS** components offer even greater advantages, with their modularity **allowing designers** to **develop** more advanced systems **without increased project risk**. »*

ÉCOLE NORMALE SUPÉRIEURE DE LYON

L'approche « Intégration »

- COTS : quatre lettres très en vogue dans l'industrie
  - En développement : « Component Off-The-Shelf »
  - En production : « Commercial Off-The-Shelf »
- « *Off-The-Shelf* » : d'autres...
  - Ont déjà réfléchi à la question
  - Ont trouvé une solution adaptée à un problème
  - Produisent cette solution dans un contexte industriel
  - Assurent le maintien en condition opérationnelle
- « *Component* » ou « *Commercial* »
  - Je prends le « risque » de leur faire « confiance »...

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Entre développeur et intégrateur Que choisir ?

- 2 approches
- Une approche « intégrateur »
  - Le code utilise des librairies 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

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Des usages communs...

- De l'algèbre linéaire « basique » ou « avancée »
  - Basique avec BLAS
  - Avancée avec LAPACK
- De l'algèbre linéaire sur des systèmes « creux »
  - UMFPack
- De l'analyse spectrale avec des TF
  - FFTPack
- Des résolutions d'équations différentielles
  - PetSC
  - Trilinos

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Des adaptations logicielles aux évolutions technologiques

- En algèbre linéaire :
  - BLAS avec GotoBLAS, MKL (OpenMP), BLACS (MPI)
  - LAPACK avec ACML (OpenMP), Scalapack (MPI)
- En algèbre linéaire sur des systèmes creux
  - MUMPS (OpenMP, MPI) mais dépend de BLAS
- En analyse spectrale :
  - FFT avec FFTw3 (OpenMP), FFTw2 (MPI)

Cohérent de voir « apparaître » des versions GPU !

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Pour l'algèbre linéaire : la profusion

- Fonctions BLAS :
  - CuBlLAS : complètes (mais d'efficacité diverses)
  - AMDAPP/ClAmdBlas : incomplètes
  - Magma : incomplètes
- Fonctions « creuses »
  - CuSparse
- Fonctions LAPACK :
  - Cula : juste un recarrossage de LAPACK avec CuBLAS
  - Magma : incomplètes

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Et mon code « pur » ?

- Approche HMPP :
  - Une approche « pragma »tique à la OpenMP
  - OpenHMPP : une initiative (seulement une initiative...)
- Approche PGI Cuda Fortran :
  - LicenceS (pour PGI ET PGI/Cuda) : à la Matlab/Toolboxes
- Approche Par4All :
  - Un préprocesseur analyse et le code et le « transcrit »
  - Implémentation OpenMP, Cuda (et OpenCL)
  - Projet encore assez « vert »

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Pour la Transformée de Fourier

- CuFFT : pour le C, C++, Fortran
  - Float sur 32, Double sur 64, Complex en 32 et en 64
  - 1D, 2D, 3D
  - Mixte sur les sorties : S2C, D2Z,
- AMDAPP/ClAmdFft : pour le C, C++
  - Float sur 32, Double sur 64, Complex en 32 et en 64
  - 1D, 2D, 3D
- PyFFT (utilisant PyCUDA et PyOpenCL) !

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Autres...

- PetSc : support avec Cuda
- OpenMM : support avec Cuda et OpenCL
  - Socle pour des outils déjà existants
    - Gromacs
    - TkInter
    - Amber
  - API Python : ☺

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Utilisation de GPU sous Python : de Numpy vers Numpy/PyFFT

- Montage de Abbe & Porter : implémentation Matplotlib
- Diffraction de Fraunhoder : Double FFT
- Coeur de la compilation : FFT & FFTshift
- Numpy/FFT en temps réel impossile (grands espaces)
- Utilisation de PyFFT avec PyOpenCL & PyCUDA
  - Mille merci à Andreas Klöckner
- Vitesse dépendant de la carte !

ÉCOLE NORMALE SUPÉRIEURE DE LYON

PyFFT : Piece of c(ake|ode)...

- **Implémentation « standard » : def fraunhofer(source)**
  - return(factor*fftshift(fft2(fftshift(source))))
- **Implémentation OpenCL :**
  - gpuplan = Plan((dim_x, dim_y), normalize=True, fast_math=True, queue=queue)
  - gpu_data = cl_array.to_device(ctx,queue,fftshift(source
  - gpuplan.execute(gpu_data.data)
  - return(factor*fftshift(gpu_data.get()))
- **Implémentation CUDA :**
  - gpuplan = Plan((dim_x, dim_y), normalize=True, stream=stream)
  - gpu_data = gpuarray.to_gpu(fftshift(source))
  - gpuplan.execute(gpu_data)
  - return(factor*fftshift(gpu_data.get()))

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Numpy PyFFT : initialisation en PyCUDA ou PyOpenCL

Implémentation sous CUDA

```
from pyfft.cuda import Plan
import pycuda.driver as cuda
from pycuda.tools import make_default_context
import pycuda.gpuarray as gpuarray
cuda.init()
context = make_default_context()
stream = cuda.Stream()
```

Implémentation sous OpenCL :

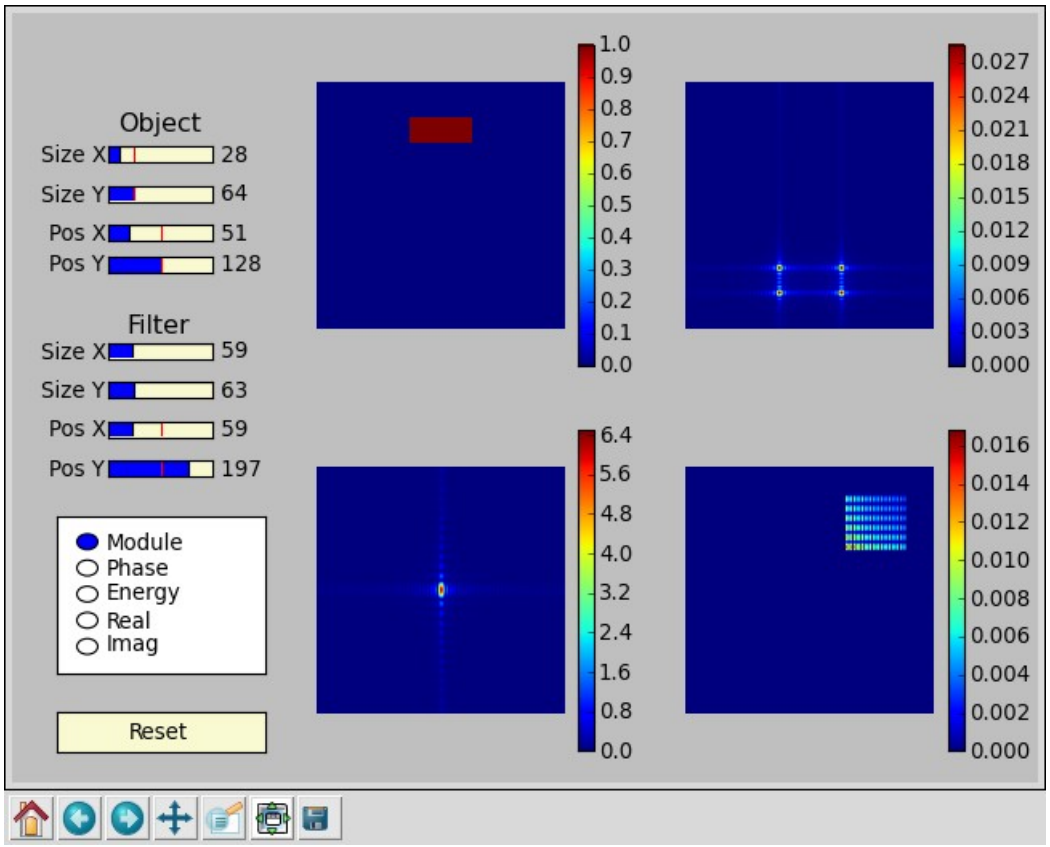
```
from pyfft.cl import Plan
import pyopencl as cl
import pyopencl.array as cl_array
ctx = cl.create_some_context(interactive=False)
queue = cl.CommandQueue(ctx)
```

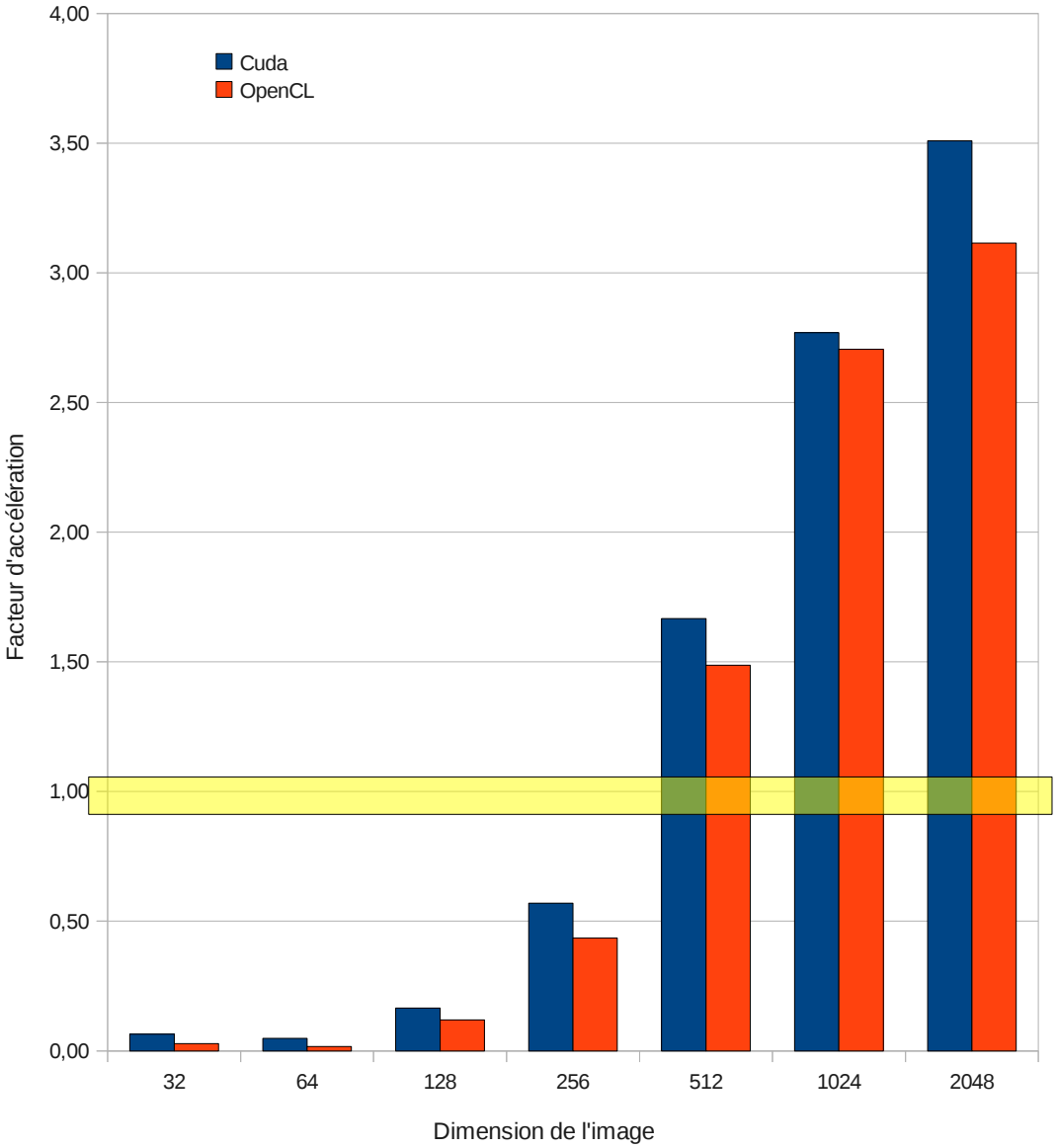
ÉCOLE NORMALE SUPÉRIEURE DE LYON

Numpy vers Numpy/PyFFT Interface & Résultats

Interface en Matplotlib

Dimensions de 32 à 2048





ÉCOLE NORMALE SUPÉRIEURE DE LYON

CuBLAS : approche descendante conversion de xHPL sous CuBLAS

« 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 : ca 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

ÉCOLE NORMALE SUPÉRIEURE DE LYON

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 ?

ÉCOLE NORMALE SUPÉRIEURE DE LYON

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 ?

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Des progrès sur 18 mois... xGEMM de CuBLAS 3.2 à 4.0

GFlops	FBLAS	CuBLAS	Thunking	CuBLAS/CBLAS
Mai 2010				
SP X=16000	12	350	327	**x27**
DP Y=10000	6	73	70	**x11**
SP $(X+1)^2$ $(X-1)^2$	12	97		**x8** 1/3,26
DP $(Y+1)^2$ $(Y-1)^2$	6	31		**x5** 1/2,35
Novembre 2011				
SP X=16000	13	367	336	**x28**
DP Y=10000	7	72	68	**x10**
SP $(X+1)^2$ $(X-1)^2$	13	291		**x22** 1/1,26
DP $(Y+1)^2$ $(Y-1)^2$	7	71		**x10** 1/1,01

ÉCOLE NORMALE SUPÉRIEURE DE LYON

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 ?

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Approche ascendante : (C/F/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_dnrm2(dim,X,incx);
cblas_dswap(dim,X,incx,Y,incx);
```

Avec FBLAS

```
dgemv_(&trans,&dim,&dim,&alpha,A,&dim,X,&incx,&beta,Y,&incx);
dtrsv_(&uplo,&trans,&diag,&dim,A,&dim,Y,&incx);
daxpy_(&dim,&beta2,Y,&incx,X,&incx);
dnrm2_(&dim,X,&incx,&checksA[i]);
dswap_(&dim,X,&incx,Y,&incx);
```

GPU !

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);
```

ÉCOLE NORMALE SUPÉRIEURE DE LYON

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

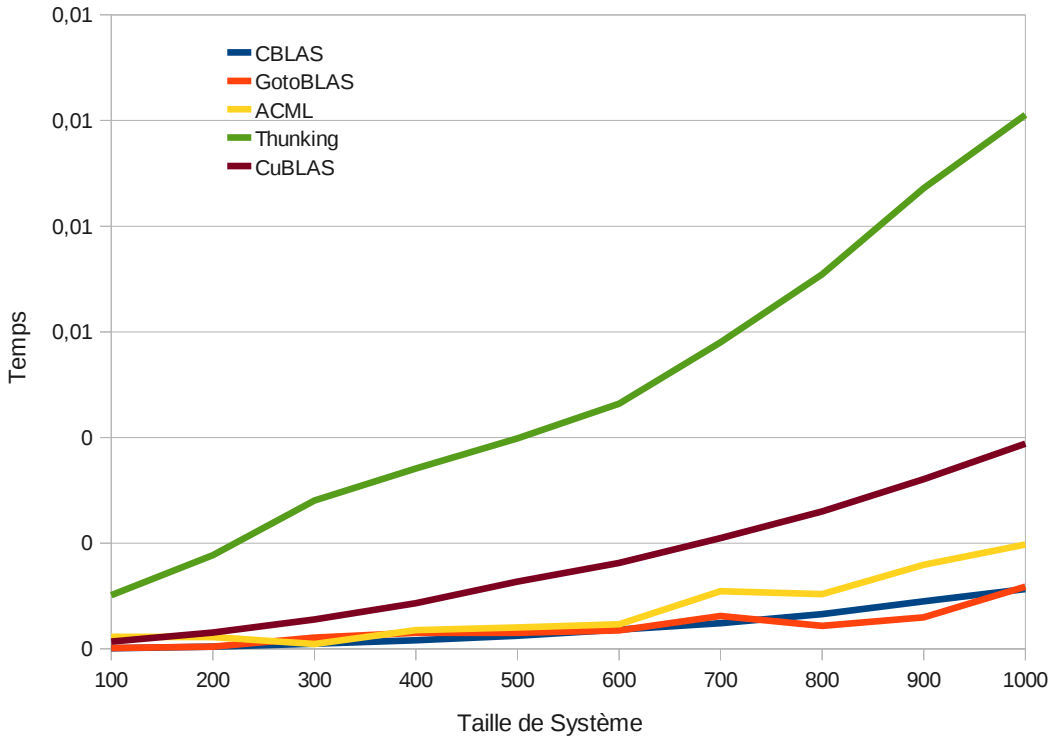
```
stat1=cublasGetVector(dim,sizeof(X[0]
),devPtrX,incx,X,incx);
stat2=cublasGetVector(dim,sizeof(Y[0]
),devPtrY,incx,Y,incx);
  cublasFree (devPtrA);
  cublasFree (devPtrX);
  cublasFree (devPtrY);
  if ((stat1 != CUBLAS_STATUS_SUCCESS) ||
      (stat2 != CUBLAS_STATUS_SUCCESS)) {
    wrapperError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

ÉCOLE NORMALE SUPÉRIEURE DE LYON

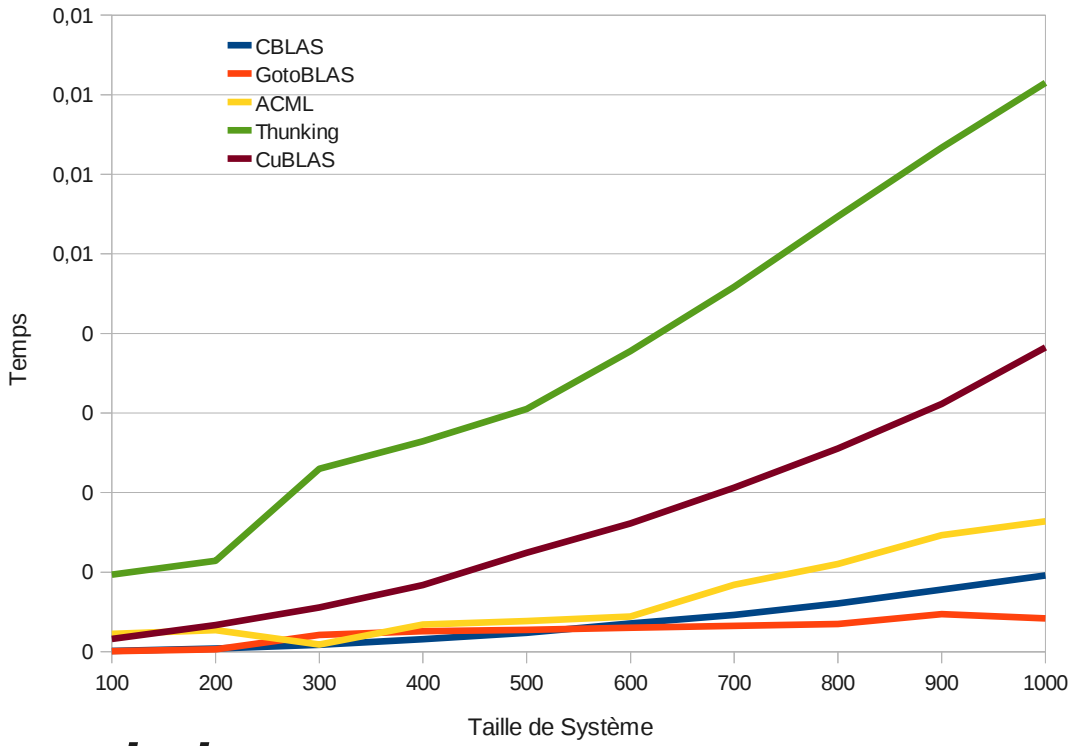
Approche ascendante : xTRSV Sur de « petites » dimensions

- GotoBLAS vainqueur, CBLAS honorable, ACML bof...
- Le mode CuBLAS entre 3 et 24 fois plus lent que GotoBLAS
- Le mode Thunking entre 8 et 147 fois plus lent que GotoBLAS

xTRSV en Double Précision



xTRSV en Simple Précision

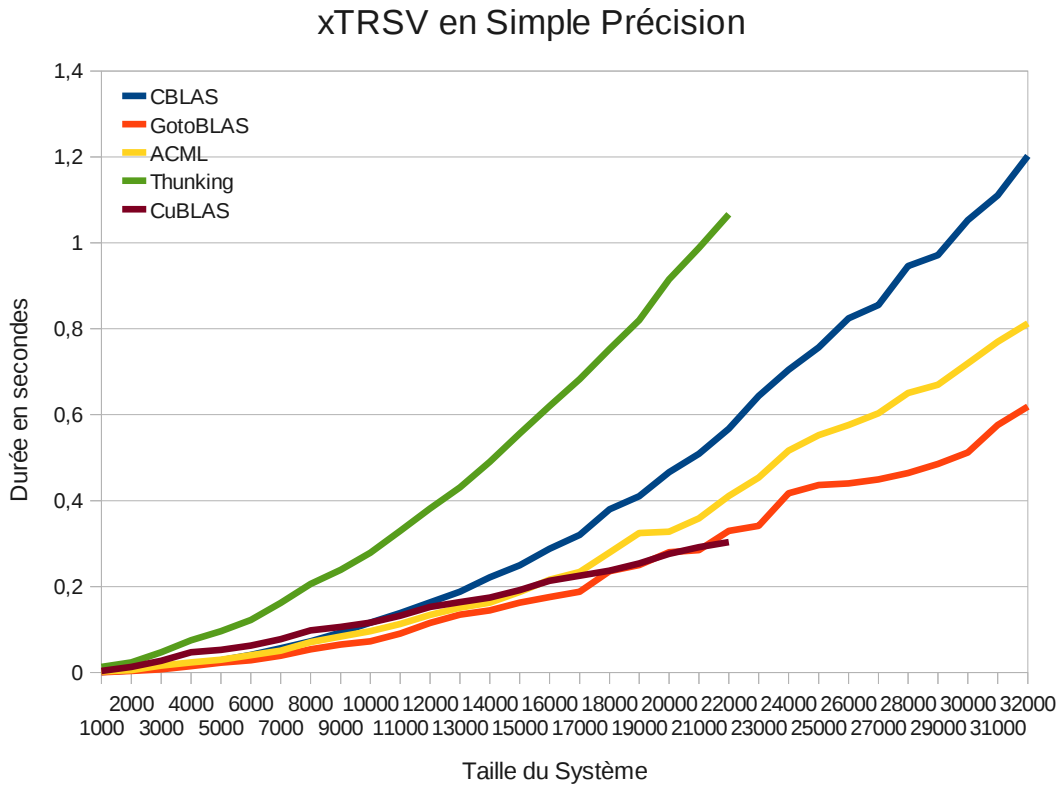


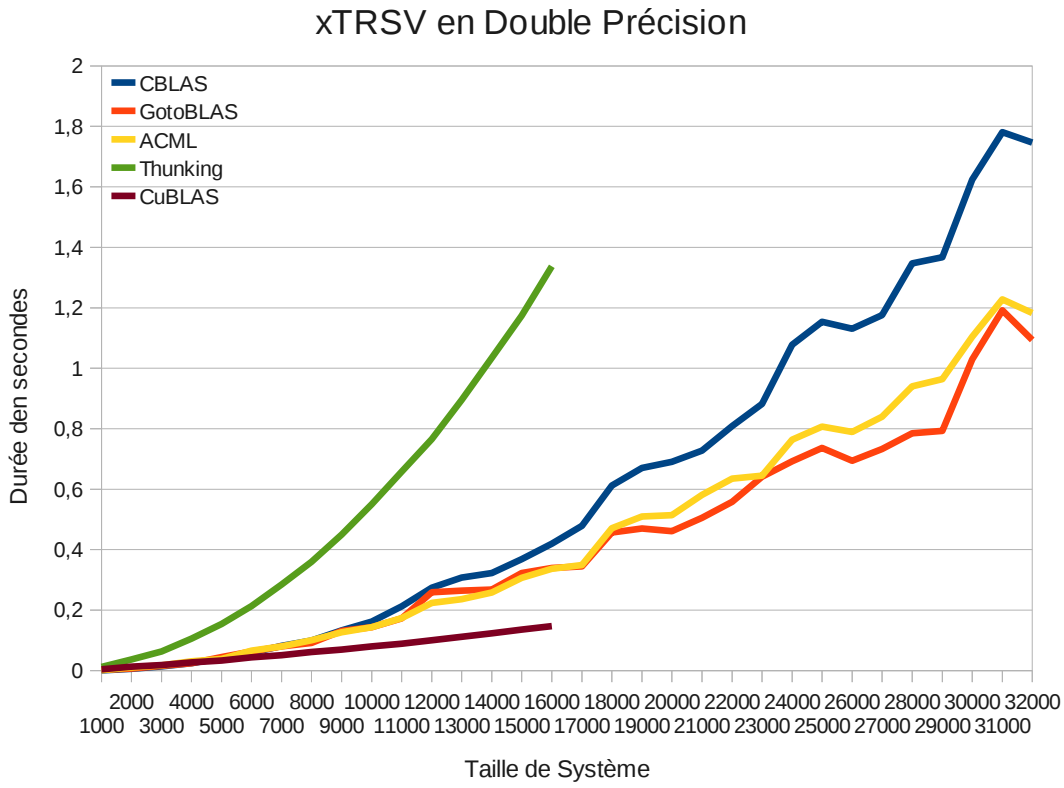
Résultats en « ***Less is better*** »

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Approche ascendante : xTRSV Sur de « grandes » dimensions

- En SP, GotoBLAS vainqueur, CuBLAS talonne puis meurt
- En DP, le mode CuBLAS l'emporte si dimension > 4000
- Le mode Thunking toujours aussi catastrophique





Résultats en « ***Less is better*** »

ÉCOLE NORMALE SUPÉRIEURE DE LYON

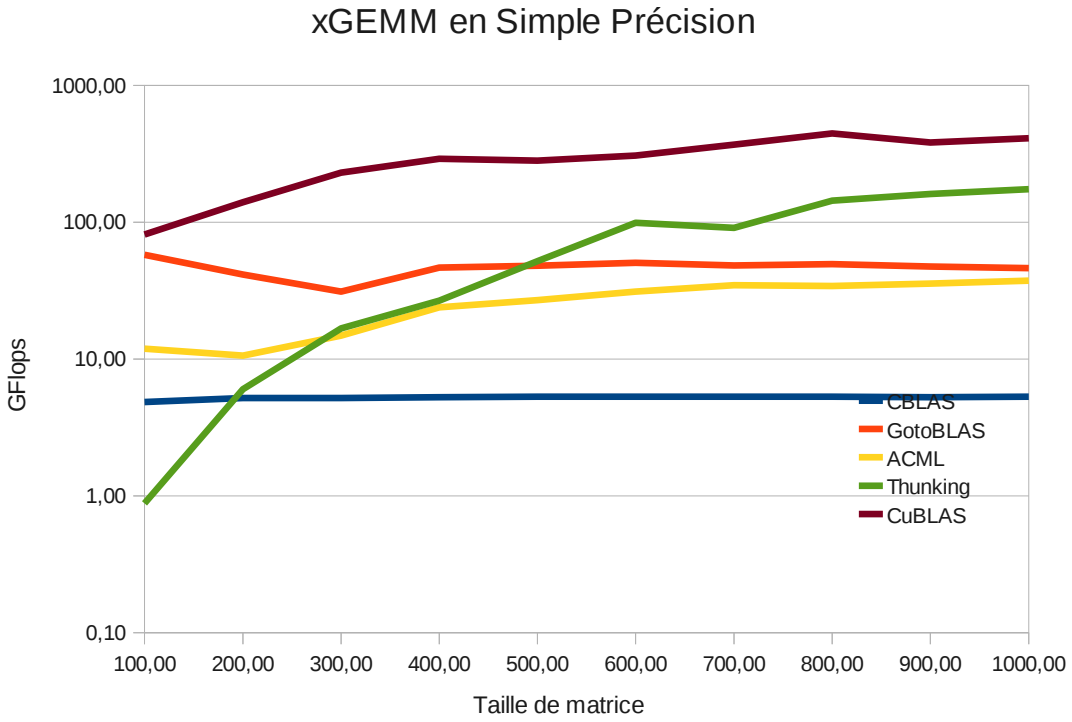
On reprend son souffle...

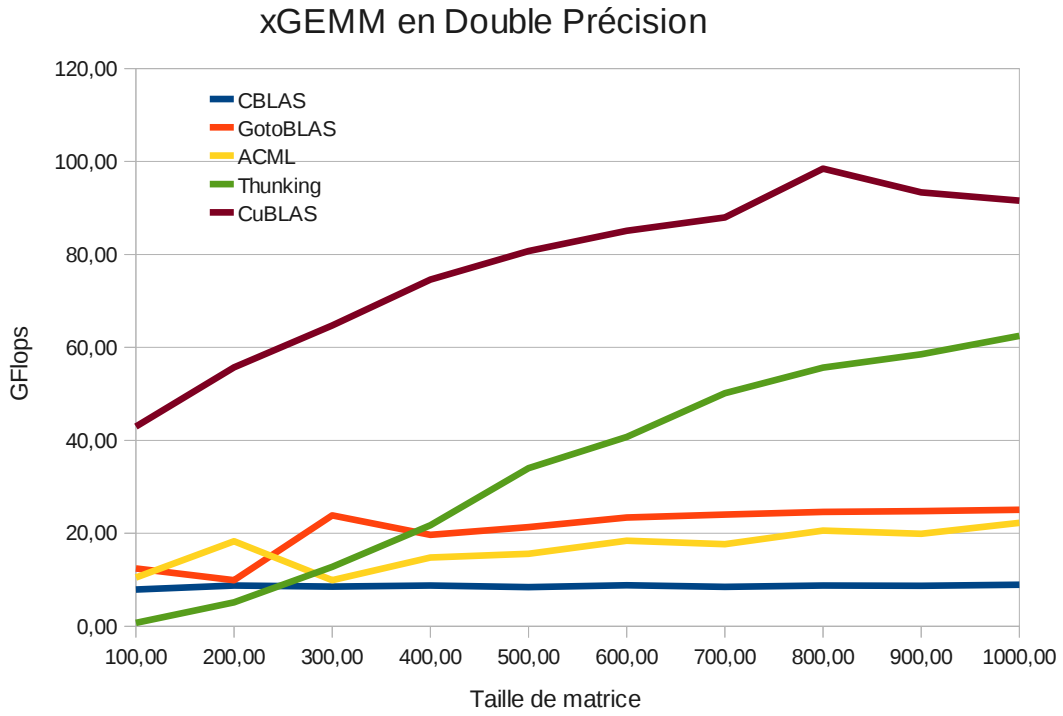
- Utilisation 5 commandes BLAS dans un banc d'essai
- Pour de petites dimensions, le GPU est mauvais
- Pour de grandes dimensions : gain nul ou
  - En Simple Précision, gain à partir de 22000
  - En Double Précision, gain à partir de 4000
  - En rapport à des BLAS OpenMP, indexation
- « Coupure » (taille de la RAM de la Carte)
- Alors, le GPU puissant en intégration, info ou intox ?
- Revenir à un seul et unique test : xGEMM

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Approche ascendante : xGEMM Sur de « petites » dimensions

- CuBLAS, toujours leader entre 1,5x et 9x plus rapide
- Thunking entre 65x plus lent et 4x plus rapide
- Barres des 90 GflopsDP et 400 Gflops SP franchies.



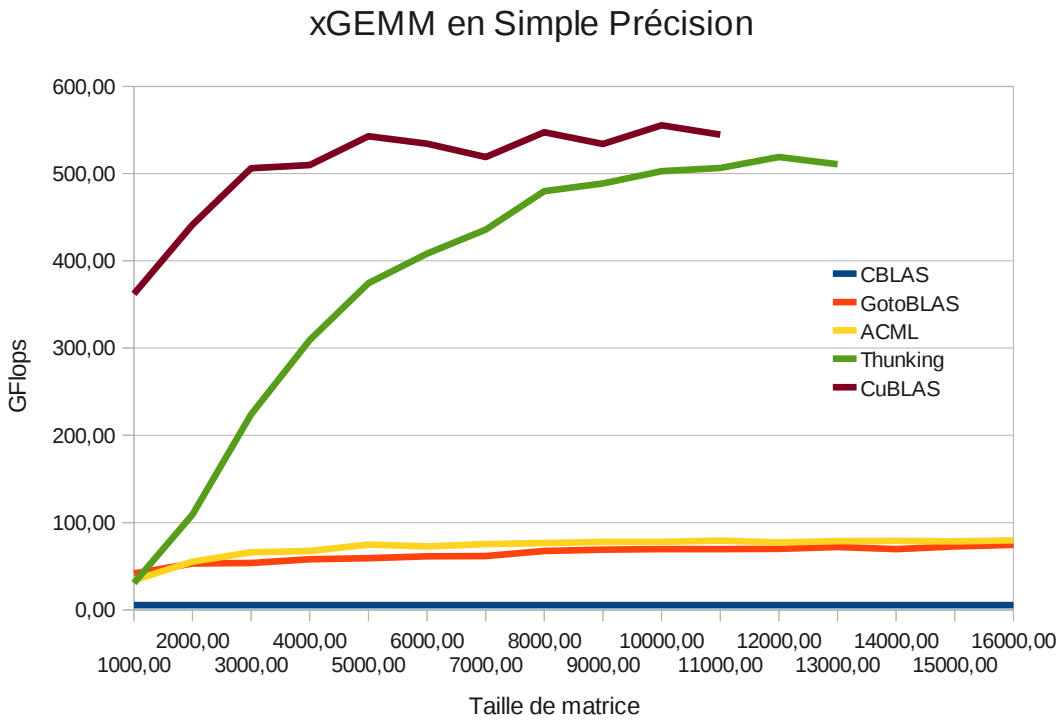


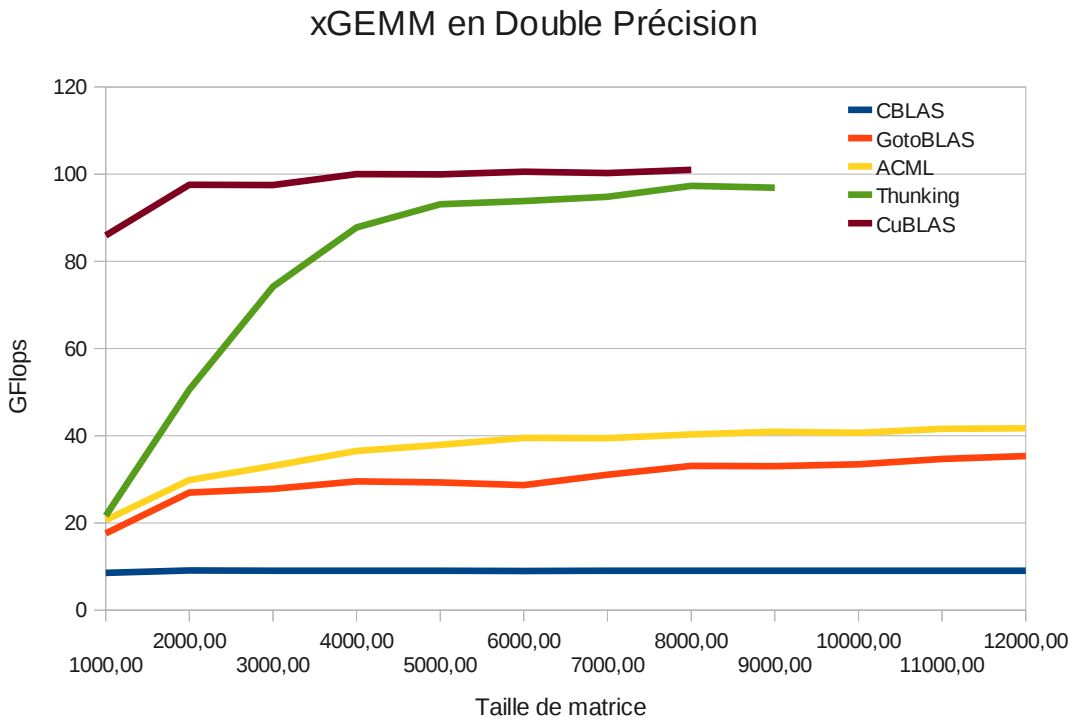
Résultats « ***More is better*** »

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Approche ascendante : xGEMM Sur de «grandes» dimensions

- CuBLAS poursuit son envolée
- Thunking résiste...
- Barres des 100 Gflops et 500 Gflops franchies





Résultats « ***More is better*** »

ÉCOLE NORMALE SUPÉRIEURE DE LYON

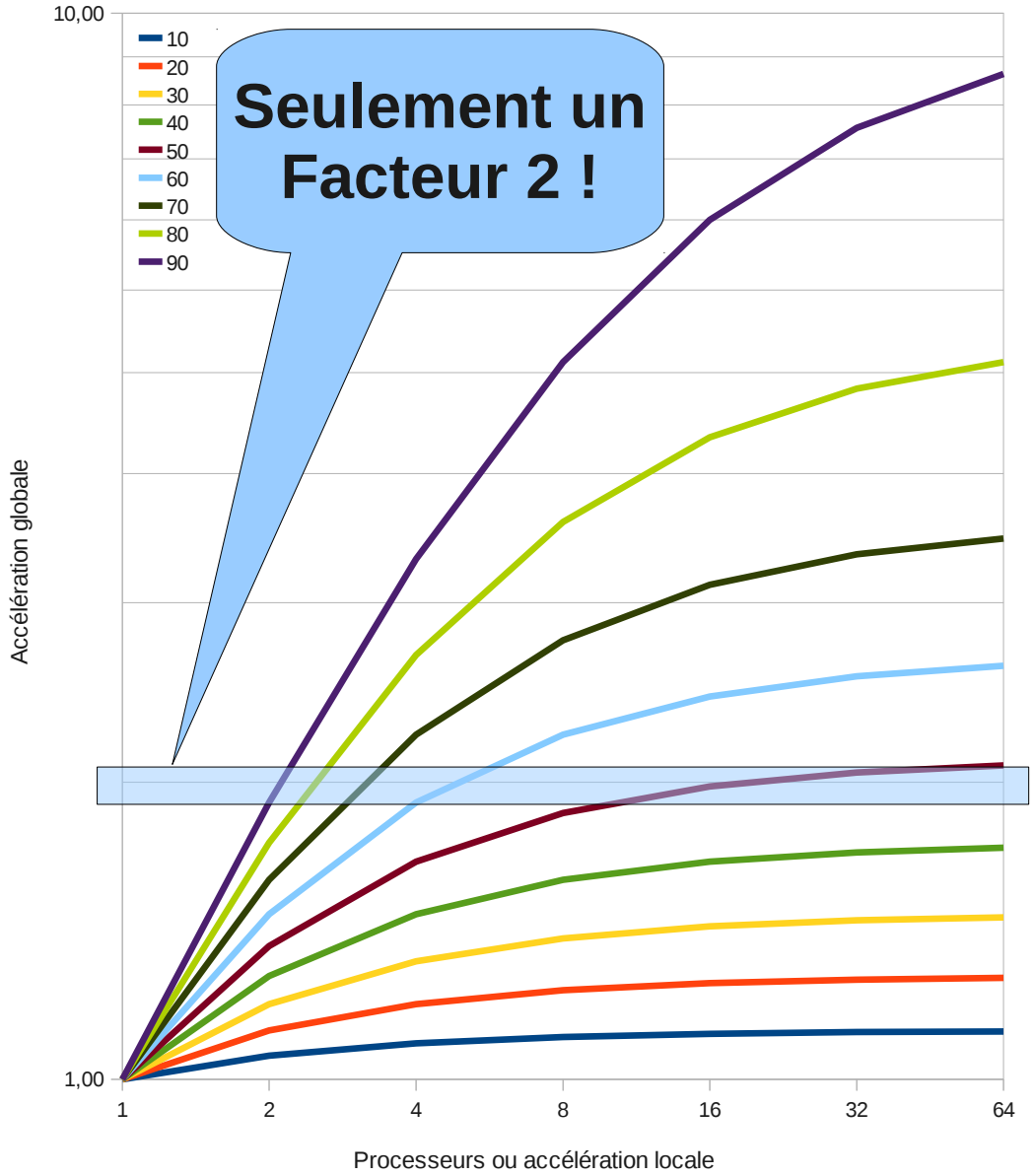
Conclusion sur l'intégration pure

- GPU ne rime que parfois avec performances trapues
- GPU performant si utilisé dans ce qu'il sait « faire »
  - Ce pour quoi il a été conçu : produit matriciel
- GPU performant si utilisé sur de gros volumes
  - Mais pas trop gros pour tenir dans sa mémoire
- GPU à toujours comparer au CPU
  - Pour comparer les résultats : cohérence
  - Pour comparer les performances : référence
- Approche intégrative : bien connaître sa « hache »

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Loi d'Amdahl
N processeurs, Accélération A

- De 1 à N processeurs
  - S : code séquentiel
  - P : code parallèle
  - Speed UP : 1/(1-P+P/N))
- Une accélération A
  - S : code non « boosté »
  - B : code « boosté »
  - Speed UP : 1/(1-B+B/A))
- Intéressant si (B|P) > 90%



ÉCOLE NORMALE SUPÉRIEURE DE LYON

Par4all ou « le parallèle pour les nuls »

- Petit exemple : pour tous les (i,j) : C[i,j]=A[i,k]B[k,j]
- Le code :

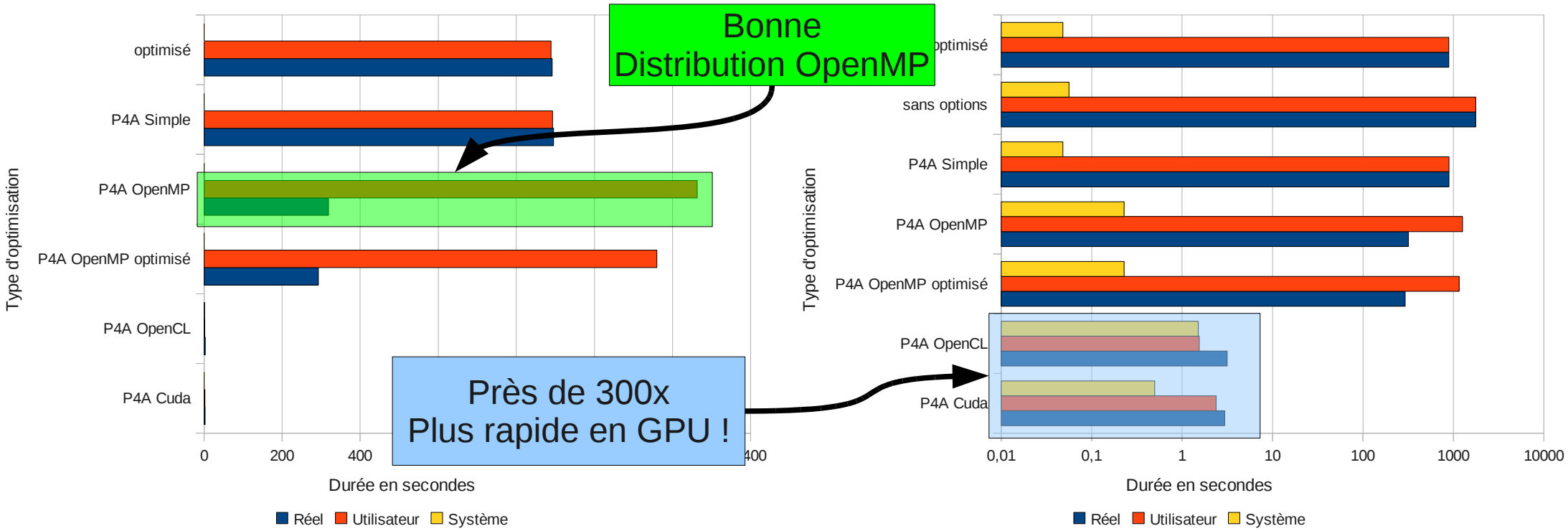
```
#define size 2048;
int main(void) {
    float a[size][size]; float b[size][size]; float c[size][size];
    int i,j,k;
    for (i = 0; i < size; ++i) for ( j = 0; j < size; ++j)
        a[i][j]=(float)i+j; b[i][j]=(float)i-j; c[i][j]=0.0f;

     for (i=0; i<size; ++i) for (j=0; j<size; ++j)
         for (k=0; k<size; ++k)
              c[i][j] += a[i][k] * b[k][j];
    return 0;
}
```

- La commande :
  - Pour OpenMP : `p4a --openmp matrix.c -o matrix-omp`
  - Pour Cuda : `p4a --cuda matrix.c -o matrix-cuda`
  - Pour OpenCL (du dev') : `p4a --opencl matrix.c -o matrix-ocl`

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Par4all : et les perfs'

- Alors, ça « scotche », hein ?
  - Une très bonne distribution OpenMP
  - Une très bonne accélélation OpenCL et Cuda (presque suspecte)

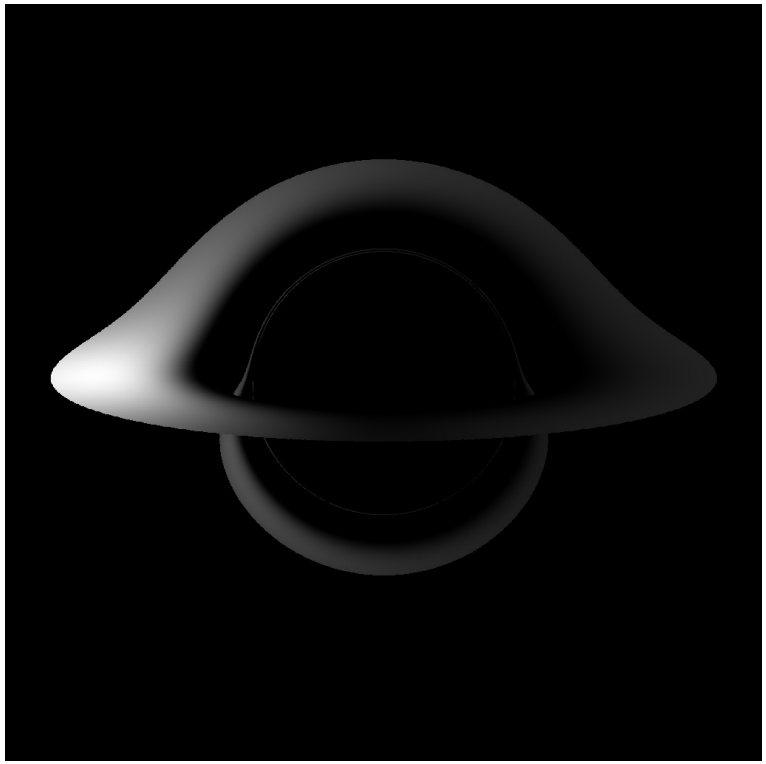


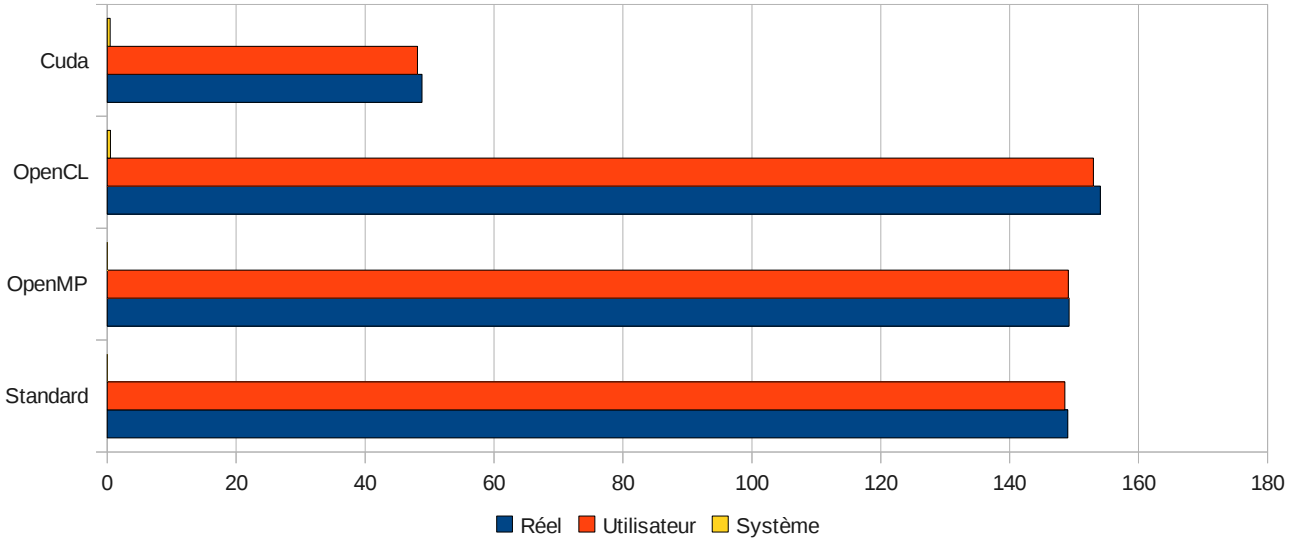
Bon OK, c'est le meilleur des scénarios !

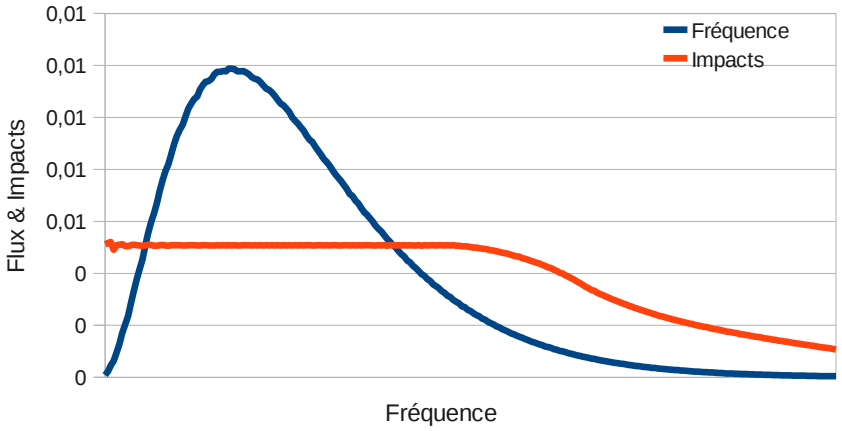
ÉCOLE NORMALE SUPÉRIEURE DE LYON

Par4all dans la vraie vie...

- Accrétion autour d'un corps noir
- Simulation de Luminet en 1979
- Apparence et spectre
- Utilisation de Par4All
- Facteur 3 en Cuda...







ÉCOLE NORMALE SUPÉRIEURE DE LYON

Quelques conseils généraux

- Toujours penser aux 3 coûts
  - D'entrée : apprentissage (formation versus enseignement)
  - D'exploitation : des années à maintenir
  - De sortie : une technologie n'est pas éternelle
- Un conseil :
  - Appel générique
  - Utilisation de directive pour différencier les appels
  - Toujours « caster » les variables !

ÉCOLE NORMALE SUPÉRIEURE DE LYON

GPU par intégration : Quelle conclusion

- Des librairies existent : de + en + complètes, mais
  - Elles sont exigeantes :
  - Elles (ne) sont efficaces que dans certains cas
- Des outils se facilitant le travail existent :
  - Ils sont simples d'usage mais encore un peu « verts »
- Comme toujours, quelques préalables :
  - Une introspection du code : parallélisation & propreté
  - Un banc de tests le plus atomique possible
  - Un staff d'ingénieurs motivés ☺