Intro

Pour commencer

Paralléliser

Améliorer le stockage

Application

Conclusion

MULTIPLICATION EXACTE MATRICE CREUSE – VECTEUR SUR ARCHITECTURES GPU ET CPU MULTICŒUR

Brice BOYER[1] Jean-Guillaume DUMAS[1]
Pascal GIORGI[2]

[1]LJK, Université de Grenoble, France
[2]LIRMM, Université Montpellier 2, France

JNCF '10

3 mai 2010

Intro
Pour commencer
Paralléliser
Améliorer le stockage
Application
Conclusion

Motivations

- Multiplication matrice creuse – vecteur (SpMV) : la base des algorithmes "boîte noire".

Motivations

- Multiplication matrice creuse – vecteur (SPMV) : la base des algorithmes “boîte noire”.
- GPU et CPU multicœurs se généralisent sur les ordinateurs personnels.

Motivations

- Multiplication matrice creuse – vecteur (SPMV) : la base des algorithmes “boîte noire”.
- GPU et CPU multicœurs se généralisent sur les ordinateurs personnels.
- Les GPU offrent une puissance de calcul à bas coût et de nouvelles manières de traîter les données.

Idées principales

– Utiliser des librairies numériques (cf FFLAS/FFPACK) ;

Idées principales

- Utiliser des librairies numériques (cf FFLAS/FFPACK) ;
- Découper A en sous-matrices, écrire $A = A_1 + \cdots + A_k$, tirant avantage d'être sur des corps finis ;

Idées principales

- Utiliser des librairies numériques (cf FFLAS/FFPACK) ;
- Découper A en sous-matrices, écrire $A = A_1 + \cdots + A_k$, tirant avantage d'être sur des corps finis ;
- Créer automatiquement des formats hybrides efficaces selon l'architecture utilisée.

Plan

1 Introduction

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides
5. Applications
   - Bien utiliser le GPU
   - Multivecteurs
   - Calcul du rang/algo par blocs de Wiedemann

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides
5. Applications
   - Bien utiliser le GPU
   - Multivecteurs
   - Calcul du rang/algo par blocs de Wiedemann
6. Conclusion

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides
5. Applications
   - Bien utiliser le GPU
   - Multivecteurs
   - Calcul du rang/algo par blocs de Wiedemann
6. Conclusion

Introduction

- But : une opération SPMV efficace sur de grandes matrices réutilisées abondamment.

Intro

Pour commencer

Paralléliser

Améliorer le stockage

Application

Conclusion

Introduction

- But : une opération SPMV efficace sur de grandes matrices réutilisées abondamment.
- Moyens : utiliser au mieux les architectures CPU multicœur et GPU :

Introduction

- But : une opération SPMV efficace sur de grandes matrices réutilisées abondamment.
- Moyens : utiliser au mieux les architectures CPU multicœur et GPU :
  - sur CPU, le standard OpenMP ;
  - sur GPU, ATI Stream, NVIDIA Cuda, OpenCL.

Introduction

- But : une opération SPMV efficace sur de grandes matrices réutilisées abondamment.
- Moyens : utiliser au mieux les architectures CPU multicœur et GPU :
  - sur CPU, le standard OpenMP ;
  - sur GPU, ATI Stream, NVIDIA Cuda, OpenCL.

Introduction

- But : une opération SpMV efficace sur de grandes matrices réutilisées abondamment.
- Moyens : utiliser au mieux les architectures CPU multicœur et GPU :
  - sur CPU, le standard OpenMP ;
  - sur GPU, ATI Stream, NVIDIA Cuda, OpenCL.
- Techniques :
  - utiliser des librairies numériques existantes,
  - adapter/optimiser le stockage des matrices selon les algorithmes de multiplication.

Introduction

- But : une opération SpMV efficace sur de grandes matrices réutilisées abondamment.
- Moyens : utiliser au mieux les architectures CPU multicœur et GPU :
  - sur CPU, le standard OpenMP ;
  - sur GPU, ATI Stream, NVIDIA Cuda, OpenCL.
- Techniques :
  - utiliser des librairies numériques existantes,
  - adapter/optimiser le stockage des matrices selon les algorithmes de multiplication.

Opération à implémenter

- $\mathbf{y} = \alpha A\mathbf{x} + \beta\mathbf{y}$.
- $\mathbf{y} = \alpha\, {}^{\top}\!A\mathbf{x} + \beta\mathbf{y}$.

Introduction

- **But** : une opération SpMV efficace sur de grandes matrices réutilisées abondamment.
- **Moyens** : utiliser au mieux les architectures CPU multicœur et GPU :
  - sur CPU, le standard OpenMP ;
  - sur GPU, ATI Stream, NVIDIA Cuda, OpenCL.
- **Techniques** :
  - utiliser des librairies numériques existantes,
  - adapter/optimiser le stockage des matrices selon les algorithmes de multiplication.

Opération à implémenter

- $\mathbf{y} = \alpha A\mathbf{x} + \beta \mathbf{y}$.
- $\mathbf{y} = \alpha {}^{\top}A\mathbf{x} + \beta \mathbf{y}$.

On se ramène à l'opération $\mathbf{y} = A\mathbf{x} + \mathbf{y}$.

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides
5. Applications
   - Bien utiliser le GPU
   - Multivecteurs
   - Calcul du rang/algo par blocs de Wiedemann
6. Conclusion

Matrices utilisées

nom	mat1916	bibd_81_3	EX5	GL7d15	mpolyout2
row	1916	3240	6545	460261	2410560
col	1916	85320	6545	171375	2086560
nbnz	195985	255960	295680	6080381	15707520
rang	1916	3240	4740	132043	1352011

TABLE: Matrices utilisées dans les tests

Stockages classiques

Exemples

- `COO` représente la matrice sous forme de triplets.
  `data`$[k]$ = `A[rowid`$[k]$`],colid`$[k]$`]` ;

Stockages classiques

Exemples

- COO représente la matrice sous forme de triplets.
  $\texttt{data}[k] = \mathrm{A}[\texttt{rowid}[k], \texttt{colid}[k]]$;
- CSR (version compressée de COO) :
  $\texttt{start}[i] \leq k < \texttt{start}[i+1] \Rightarrow \texttt{data}[k] = \mathrm{A}[i, \texttt{colid}[k]]$;

Stockages classiques

Exemples

- COO représente la matrice sous forme de triplets.
  $\texttt{data}[k] = \mathrm{A}[\texttt{rowid}[k], \texttt{colid}[k]]$;
- CSR (version compressée de COO) :
  $\texttt{start}[i] \leq k < \texttt{start}[i+1] \Rightarrow \texttt{data}[k] = \mathrm{A}[i, \texttt{colid}[k]]$;
- ELL est plus dense :
  $\texttt{data}[i, j_0] = \mathrm{A}[i, \texttt{colid}[i, j_0]]$.

Intro
Pour commencer
Stockages classiques
Les corps
Librairies
Paralléliser
Améliorer le stockage
Application
Conclusion

Stockages classiques

Exemples

- COO représente la matrice sous forme de triplets.
  $\mathtt{data}[k] = \mathrm{A}[\mathtt{rowid}[k], \mathtt{colid}[k]]$;
- CSR (version compressée de COO) :
  $\mathtt{start}[i] \leq k < \mathtt{start}[i+1] \Rightarrow \mathtt{data}[k] = \mathrm{A}[i, \mathtt{colid}[k]]$;
- ELL est plus dense :
  $\mathtt{data}[i, j_0] = \mathrm{A}[i, \mathtt{colid}[i, j_0]]$.

Autres exemples

- représentation dense ;
- représentation diagonale (DIA) ;
- représentation en colonnes (CSC) ;
- ...

Représentation de $\mathbf{Z}/m\mathbf{Z}$

Retarder la réduction.

Effectuer le moins possible d'opérations `fmod`.

Intro

Pour commencer

Stockages classiques

Les corps

Librairies

Paralléliser

Améliorer le stockage

Application

Conclusion

Représentation de $\mathbf{Z}/m\mathbf{Z}$

Retarder la réduction.

Effectuer le moins possible d'opérations `fmod`.

Représentations de $\mathbf{Z}/m\mathbf{Z}$

– classique : $[\![0, m-1]\!]$;

Intro
Pour commencer
Stockages classiques
Les corps
Librairies
Paralléliser
Améliorer le stockage
Application
Conclusion

Représentation de $\mathbf{Z}/m\mathbf{Z}$

Retarder la réduction.

Effectuer le moins possible d'opérations `fmod`.

Représentations de $\mathbf{Z}/m\mathbf{Z}$

- classique : $[\![0, m-1]\!]$;
- centrée : $[\![-\lfloor \frac{m-1}{2} \rfloor, \lceil \frac{m-1}{2} \rceil]\!]$.

Représentation de $\mathbf{Z}/m\mathbf{Z}$

Retarder la réduction.

Effectuer le moins possible d'opérations `fmod`.

Représentations de $\mathbf{Z}/m\mathbf{Z}$

- classique : $[\![0, m-1]\!]$;
- centrée : $[\![-\lfloor \frac{m-1}{2} \rfloor, \lceil \frac{m-1}{2} \rceil]\!]$.

Compromis

- coûts différents des `float` ou des `double` ;
- représentation centrée : plus d'accumulations mais réductions plus coûteuse ;
- certains GPU n'ont pas de `double` .

Exemples (CPU)

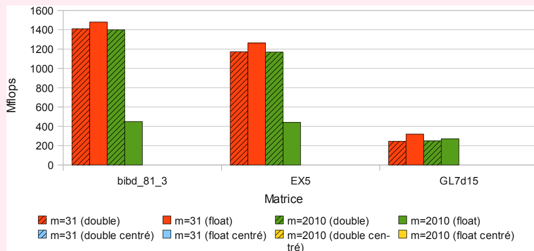


FIGURE: Matrice bibd_81_3 sur Xeon 3.2GHz au format ELL_R

Exemples (CPU)

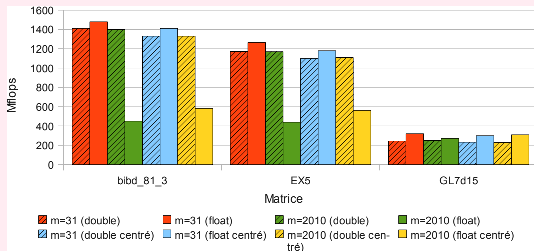


FIGURE: Matrice `bibd_81_3` sur Xeon 3.2GHz au format `ELL_R`

Exemples (GPU)

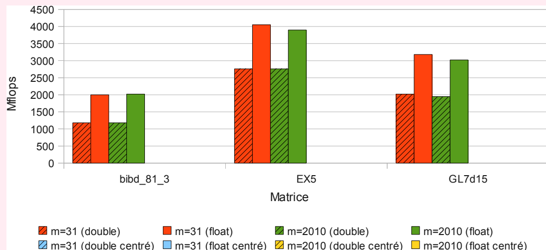


FIGURE: Matrice `bibd_81_3` sur GTX280 au format `ELL_R`

Exemples (GPU)

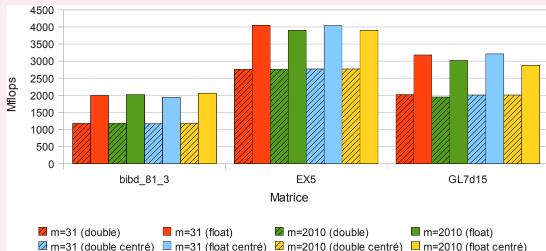


FIGURE: Matrice `bibd_81_3` sur GTX280 au format `ELL_R`

Intro
Pour commencer
Stockages classiques
Les corps
Librairies
Paralléliser
Améliorer le stockage
Application
Conclusion

Quelques librairies numériques

Peu de librairies performantes

- Sur le CPU
  - OSKI
  - Sparse BLAS

Quelques librairies numériques

Peu de librairies performantes

- Sur le CPU
  - OSKI
  - Sparse BLAS
- Sur le GPU
  - Cusp

Utiliser les librairies numériques

Difficultés par rapport au cas dense

extraction de sous-matrices de matrices creuses non triviale.

Utiliser les librairies numériques

Difficultés par rapport au cas dense

extraction de sous-matrices de matrices creuses non triviale.

Solution

Création de sous-matrices A_i de A avec au plus r éléments non nuls par ligne.
Si $m = 2010$ et le plus grand flottant est $M = 16777215$, alors on peut effectuer $M/m^2 = 4$ accumulations avant réduction.
On choisit $r = 4$.

Utiliser les librairies numériques

Difficultés par rapport au cas dense

extraction de sous-matrices de matrices creuses non triviale.

Solution

Création de sous-matrices A_i de A avec au plus r éléments non nuls par ligne.
Si $m = 2010$ et le plus grand flottant est $M = 16777215$, alors on peut effectuer $M/m^2 = 4$ accumulations avant réduction.
On choisit $r = 4$.

L'algorithme est alors :

```
spmv(y,A,x){                        //y += A x
    foreach submatrix Ai in A do{
        spmv_num(y,Ai,x);         //y += Ai x
        reduce(y,m);
    }
}
```

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides
5. Applications
   - Bien utiliser le GPU
   - Multivecteurs
   - Calcul du rang/algo par blocs de Wiedemann
6. Conclusion

OpenMP

Exemple de parallélisation avec OpenMP

```
void spmv_csr(float *data, int *colid,int *start,
              int lig, float *x,  float *y){

    for(int i = 0; i < lig; ++i) {
        for(int j = start[i]; j < start[i+1]; ++j) {
            y[i] += data[j] * x[colid[j]];
        }
    }
}
```

OpenMP

Exemple de parallélisation avec OpenMP

```
void spmv_csr(float *data, int *colid,int *start,
              int lig, float *x,  float *y){
#pragma omp parallel for
    for(int i = 0; i < lig; ++i) {
        for(int j = start[i]; j < start[i+1]; ++j) {
            y[i] += data[j] * x[colid[j]];
        }
    }
}
```

OpenMP : performances

Les processeurs utilisés sont un Intel Xeon 8 cœurs à 3.2GHz et un Core2Duo à 3GHz.

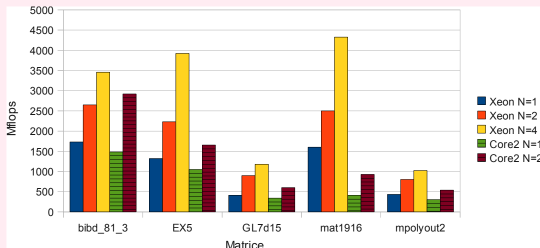


FIGURE: Performances sur un format CSR avec N cœurs utilisés

Cuda

Parallélisation avec Cuda

```
#define BLOCK 256
__global__ void
spmv_csr_ker(float *data, int *colid,int *start,
             int lig, float *x, float *y){
   int i = blockDim.x * blockIdx.x + threadIdx.x;
   if(i<lig) {
      for(int j = start[i]; j < start[i+1]; ++j) {
         y[i] += data[j] * x[colid[j]];
      }
   }
}
void
spmv_csr_dev(float *data, int *colid,int *start,
             int lig, float *x, float *y){
   int grid = ceil(lig,BLOCK);
   spmv_csr_ker<<<grid,BLOCK>>>(data,colid,start,lig,x,y);
}
```

Cuda

Parallélisation avec Cuda

```
#define BLOCK 256
__global__ void
spmv_csr_ker(float *data, int *colid,int *start,
             int lig, float *x, float *y){
   int i = blockDim.x * blockIdx.x + threadIdx.x;
   if(i<lig) {
      for(int j = start[i]; j < start[i+1]; ++j) {
         y[i] += data[j] * x[colid[j]];
      }
   }
}
void
spmv_csr_dev(float *data, int *colid,int *start,
             int lig, float *x, float *y){
   int grid = ceil(lig,BLOCK);
   spmv_csr_ker<<<grid,BLOCK>>>(data,colid,start,lig,x,y);
}
```

OpenMP

Exemple de parallélisation avec OpenMP

```
void spmv_csr(float *data, int *colid,int *start,
              int lig, float *x,  float *y){
#pragma omp parallel for
    for(int i = 0; i < lig; ++i) {
        for(int j = start[i]; j < start[i+1]; ++j) {
            y[i] += data[j] * x[colid[j]];
        }
    }
}
```

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides
5. Applications
   - Bien utiliser le GPU
   - Multivecteurs
   - Calcul du rang/algo par blocs de Wiedemann
6. Conclusion

Précompiler la matrice

Idée

Consiste à compiler un fichier qui contient l'opération $y{+}{=}Ax$.

Précompiler la matrice

Idée

Consiste à compiler un fichier qui contient l'opération $y{+}{=}Ax$.

Exemple

Si $A = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}$, on créerait ($m = 27$) :

```
void spmv_jit (float * y, const float * x){
    y[0] += 2*x[0];
    y[0] += x[1];
    y[0] = fmod(y[0],27);
    y[1] += 3*x[1];
    y[1] = fmod(y[1],27);
}
```

Précompiler la matrice

Idée

Consiste à compiler un fichier qui contient l'opération $y{+}{=}Ax$.

Exemple

Si $A = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}$, on créerait ($m = 27$) :

```
void spmv_jit (float * y, const float * x){
    y[0] += 2*x[0];
    y[0] += x[1];
    y[0] = fmod(y[0],27);
    y[1] += 3*x[1];
    y[1] = fmod(y[1],27);
}
```

Optimisations possibles

Problèmes

Les grosses matrices ne compilent pas.

Problèmes

Les grosses matrices ne compilent pas.

Solution

Découper en sous-matrices que `gcc` peut compiler.

Problèmes

Les grosses matrices ne compilent pas.

Solution

Découper en sous-matrices que `gcc` peut compiler.

Avec `nvcc`, pas de bonnes solutions pour les grosses matrices.

Problèmes

Les grosses matrices ne compilent pas.

Solution

Découper en sous-matrices que `gcc` peut compiler.

Avec `nvcc`, pas de bonnes solutions pour les grosses matrices.

Performances

`bibd_81_3`	
compiler	61s
SpMV	620Mflops

Prendre en compte les ± 1

Remarque.

Nombreuses applications donnant des matrices contenant beaucoup de 1 et/ou -1.
Pour les petits m, probablement beaucoup de ± 1.

Prendre en compte les ± 1

Remarque.

Nombreuses applications donnant des matrices contenant beaucoup de 1 et/ou -1.
Pour les petits m, probablement beaucoup de ± 1.

Une matrice creuse est codée

```
template<class Field>
struct SparseMatrix{
    SparseMulMatrix<Field> Mult;
    SparseAddMatrix<Field> Plus;
    SparseAddMatrix<Field> Moin;
};
```

Prendre en compte les ± 1

Remarque.

Nombreuses applications donnant des matrices contenant beaucoup de 1 et/ou -1.
Pour les petits m, probablement beaucoup de ± 1.

Une matrice creuse est codée

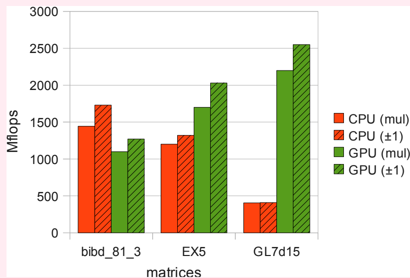
```
template<class Field>
struct SparseMatrix{
    SparseMulMatrix<Field> Mult;
    SparseAddMatrix<Field> Plus;
    SparseAddMatrix<Field> Moin;
};
```

Bénéfices

- pas besoin de stocker les 1 ou les -1 ;
- sauve des multiplications ;
- retarde la réduction.

Prendre en compte les ± 1

Comparaison des performances en fonction de la prise en compte des ± 1.



Intro
Pour commencer
Paralléliser
Améliorer le stockage
JIT
± 1
Formats Hybrides
Application
Conclusion

Différences entre les formats de stockage

Différences entre les formats de stockage

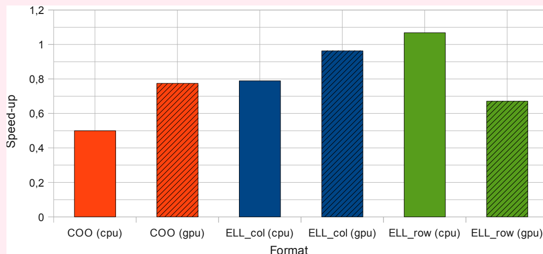


FIGURE: Rapport des performances sur un Xeon 3.2GHz et une GTX280 de divers formats par rapport au format `CSR` (matrice `bibd_81_3`).

Format Hybrides

Idée

Découper la matrice A selon des matrices $A_0 + A_1$ encodées différemment.

Format Hybrides

Idée

Découper la matrice A selon des matrices $\mathrm{A}_0 + \mathrm{A}_1$ encodées différemment. Par exemple $\mathtt{ELL} + \mathtt{CSR}$.

Intro
Pour commencer
Paralléliser
Améliorer le stockage
JIT
± 1
Formats Hybrides
Application
Conclusion

Format Hybrides

Idée

Découper la matrice A selon des matrices $\mathrm{A}_0 + \mathrm{A}_1$ encodées différemment. Par exemple $\mathtt{ELL} + \mathtt{CSR}$.

Exemple

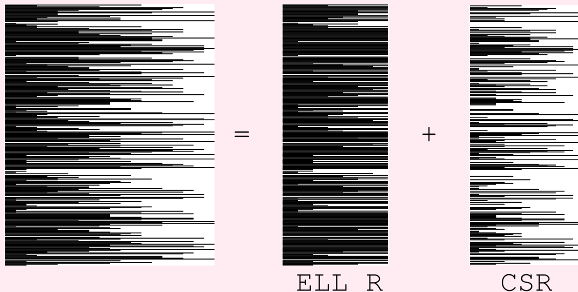
(diagramme de remplissage des lignes)

Format Hybrides

Idée

Découper la matrice A selon des matrices $A_0 + A_1$ encodées différemment. Par exemple $\mathtt{ELL} + \mathtt{CSR}$.

Exemple



(diagramme de remplissage des lignes)

Format Hybrides

Algorithme

```
choose_format(SparseMatrix A) {
    SparseAddMatrix A_p = extrait_pun(A);
    bool opt_p = essaie_optim(A_p);
    SparseAddMatrix A_m = extrait_mun(A);
    bool opt_m = essaie_optim(A_m);
    SparseMulMatrix A_g = extrait(A,opt_p,opt_m);
    optim(A_g);
    retourne(A_p,A_m,A_g);
}
```

Format Hybrides

Algorithme

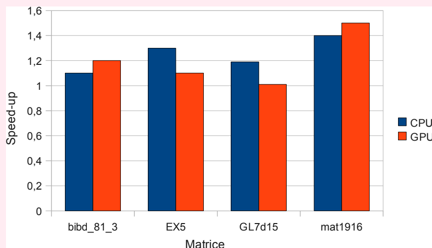
```
choose_format(SparseMatrix A) {
    SparseAddMatrix A_p = extrait_pun(A);
    bool opt_p = essaie_optim(A_p);
    SparseAddMatrix A_m = extrait_mun(A);
    bool opt_m = essaie_optim(A_m);
    SparseMulMatrix A_g = extrait(A,opt_p,opt_m);
    optim(A_g);
    retourne(A_p,A_m,A_g);
}
```

optim :

- choisit un format principal (ELL, ELL_R, CSR).
- choisit un format secondaire (si besoin) (CSR, COO_S, COO).

Format Hybrides

Accélération sur diverses matrices



(par rapport au format `CSR`)

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides
5. Applications
   - Bien utiliser le GPU
   - Multivecteurs
   - Calcul du rang/algo par blocs de Wiedemann
6. Conclusion

Problèmes liés à l'implémentation

Comparer les deux approches :

```
spmv_n(y,A,x,n){
   A_d = copy_on_gpu(A);
   y_d = copy_on_gpu(y);
   x_d = copy_on_gpu(x);
   for (i=0; i<n; ++i) {
      spmv_dev(y_d, A_d, x_d); // spmv on GPU
      x_d = y_d;
   }
   y = copy_on_cpu(y_d);
}
```

Problèmes liés à l'implémentation

Comparer les deux approches :

```
spmv_n(y,A,x,n){
   A_d = copy_on_gpu(A);
   y_d = copy_on_gpu(y);
   x_d = copy_on_gpu(x);
   for (i=0; i<n; ++i) {
      spmv_dev(y_d, A_d, x_d); // spmv on GPU
      x_d = y_d;
   }
   y = copy_on_cpu(y_d);
}
```

Problèmes liés à l'implémentation

Comparer les deux approches :

```
spmv_n(y,A,x,n){
    A_d = copy_on_gpu(A);
    y_d = copy_on_gpu(y);
    x_d = copy_on_gpu(x);
    for (i=0; i<n; ++i) {
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        x_d = y_d;
    }
    y = copy_on_cpu(y_d);
}
```

Problèmes liés à l'implémentation

Comparer les deux approches :

```
spmv_n(y,A,x,n){
    A_d = copy_on_gpu(A);
    y_d = copy_on_gpu(y);
    x_d = copy_on_gpu(x);
    for (i=0; i<n; ++i) {
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        x_d = y_d;
    }
    y = copy_on_cpu(y_d);
}
```

Problèmes liés à l'implémentation

Comparer les deux approches :

```
spmv_n(y,A,x,n){
    A_d = copy_on_gpu(A);
    y_d = copy_on_gpu(y);
    x_d = copy_on_gpu(x);
    for (i=0; i<n; ++i) {
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        x_d = y_d;
    }
    y = copy_on_cpu(y_d);
}
```

Problèmes liés à l'implémentation

Comparer les deux approches :

```
spmv_n(y,A,x,n){
    A_d = copy_on_gpu(A);
    y_d = copy_on_gpu(y);
    x_d = copy_on_gpu(x);
    for (i=0; i<n; ++i) {
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        x_d = y_d;
    }
    y = copy_on_cpu(y_d);
}


n_spmv(A,n){
    randomize(y);
    randomize(x);
    A_d = copy_on_gpu(A);
    for (i=0; i<n; ++i) {
        y_d = copy_on_gpu(y);
        x_d = copy_on_gpu(x);
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        y = copy_on_cpu(y_d);
    }
}
```

Problèmes liés à l'implémentation

Comparer les deux approches :

```
spmv_n(y,A,x,n){
    A_d = copy_on_gpu(A);
    y_d = copy_on_gpu(y);
    x_d = copy_on_gpu(x);
    for (i=0; i<n; ++i) {
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        x_d = y_d;
    }
    y = copy_on_cpu(y_d);
}


n_spmv(A,n){
    randomize(y);
    randomize(x);
    A_d = copy_on_gpu(A);
    for (i=0; i<n; ++i) {
        y_d = copy_on_gpu(y);
        x_d = copy_on_gpu(x);
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        y = copy_on_cpu(y_d);
    }
}
```

Problèmes liés à l'implémentation

Comparer les deux approches :

```
spmv_n(y,A,x,n){
    A_d = copy_on_gpu(A);
    y_d = copy_on_gpu(y);
    x_d = copy_on_gpu(x);
    for (i=0; i<n; ++i) {
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        x_d = y_d;
    }
    y = copy_on_cpu(y_d);
}


n_spmv(A,n){
    randomize(y);
    randomize(x);
    A_d = copy_on_gpu(A);
    for (i=0; i<n; ++i) {
        y_d = copy_on_gpu(y);
        x_d = copy_on_gpu(x);
        spmv_dev(y_d, A_d, x_d); // spmv on GPU
        y = copy_on_cpu(y_d);
    }
}
```

Comparaison

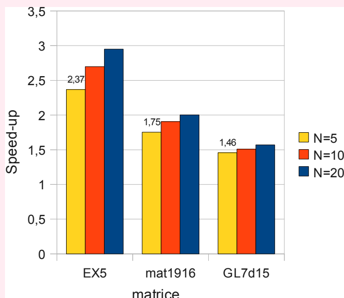


FIGURE: Accélération de $y \leftarrow A^N x$ par rapport à $N = 1$ pour $N = 5, 10, 20$ sur GTX280.

Algorithmes par bloc

Multivecteurs

Algorithmes par bloc

Multivecteurs

Pour des algorithmes par blocs ;

Algorithmes par bloc

Multivecteurs

Pour des algorithmes par blocs ;
implémentation particulière.

Algorithmes par bloc

Multivecteurs

Pour des algorithmes par blocs ;
implémentation particulière.

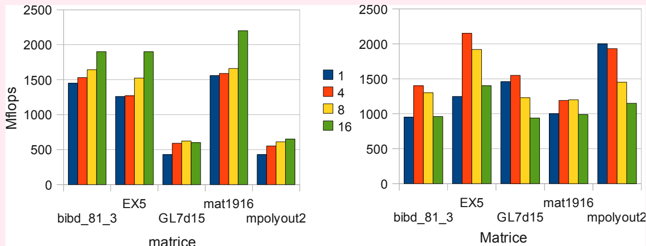


FIGURE: Performances de SPMV sur des multivecteurs de taille $1, 4, 8$ et 16 (CPU à gauche/GPU à droite).

Algorithme

3 phases :

1. Calcul de la suite $S_i = {}^{\top}YA^{i}Y$;

Algorithme

3 phases :

1. Calcul de la suite $S_i = {}^{\top}YA^iY$;
2. Calcul (σ–base) du polynôme minimal P_A^x de $\sum_i S_i x^i$;

Algorithme

3 phases :

1. Calcul de la suite $S_i = {}^{\top}YA^iY$;
2. Calcul (σ–base) du polynôme minimal P_A^x de $\sum_i S_i x^i$;
3. Interpolation du déterminant dans $\operatorname{rg}(A) = \deg\left(\det P_A^x\right) - \operatorname{codeg}\left(\det P_A^x\right)$.

Parallélisation

1. $S_i = {}^{\top}YA^iY$: on utilise SPMV.

Parallélisation

1. $S_i = {}^{\top}YA^iY$: on utilise SPMV.
2. σ−base : réduction à la multiplication de matrices polynomiales
   calcul en parallèle des DFT et des multiplications point à point.

Intro
Pour commencer
Paralléliser
Améliorer le stockage
Application
Bien utiliser le GPU
Multivecteurs
Rang/Wiedemann
Conclusion

Parallélisation

1. $S_i = {}^{\top}YA^iY$: on utilise SpMV.
2. σ–base : réduction à la multiplication de matrices polynomiales
   calcul en parallèle des DFT et des multiplications point à point.
3. interpolation : évaluations de $\det P_A^x$ en parallèle.

Résultats

Matrice	mat1916		bibd_81_3		EX5	
nb cœurs	1	8	1	8	1	8
Seq-LB	15.09	3.08	47.73	12.41	84.21	20.22
Seq-SPMV	5.02	0.91	41.28	7.56	49.66	7.36
σ-base	9.02	1.64	18.45	3.63	37.45	8.39
Interpolation	0.37	0.29	1.07	0.82	2.29	1.75
Total-LB	24.48	5.01	67.25	16.86	123.95	30.36
Total-SPMV	14.41	2.84	60.80	12.01	89.40	17.50

TABLE: Rang $\mod 65521$ avec OpenMP sur un Xeon E5345, 8 × 2.33GHz

Plan

1. Introduction
2. Formats de stockage, librairies et techniques existantes
   - Stockages classiques
   - Représentation des corps finis
   - Utilisation des librairies numériques
3. Parallélisation
   - CPU
   - GPU
4. Améliorer les formats de stockage
   - Juste à temps
   - Prendre en compte les ± 1
   - Formats Hybrides
5. Applications
   - Bien utiliser le GPU
   - Multivecteurs
   - Calcul du rang/algo par blocs de Wiedemann
6. Conclusion

Conclusion

- Nous obtenons de très bonnes performances avec OpenMP et Cuda ;
- la création de formats hybrides dépendant du type de parallélisme et de la répartition des éléments non nuls par ligne permet de meilleures performances ;
- un algorithme heuristique permet de choisir efficacement des formats rapides ;
- la partie CPU de cette librairie s'est intégrée facilement dans LinBox.

Conclusion/Perspectives

Pour continuer...

- SPMV
  - implémentation sur des corps finis généraux, sur $\mathbb{Z}/2\mathbb{Z}$;
  - parallélisation sur multi-GPU ;
  - méthodes hybrides CPUs/GPUs ;
  - utilisation de METIS pour partitionner/réordonner la matrice ;
  - utilisation de Sparse BLAS.
  - intégrer la librairie dans LINBOX ;

Conclusion/Perspectives

Pour continuer…

- SpMV
  - implémentation sur des corps finis généraux, sur $\mathbb{Z}/2\mathbb{Z}$;
  - parallélisation sur multi-GPU ;
  - méthodes hybrides CPUs/GPUs ;
  - utilisation de Metis pour partitionner/réordonner la matrice ;
  - utilisation de Sparse BLAS.
  - intégrer la librairie dans LinBox ;
- calcul du rang
  - Utilisation des GPU pour SpMV et les $\sigma-$bases ;
  - Se rapprocher des performances optimales sur multi-cœurs ;
  - Redessiner l'algorithme des $\sigma-$bases en vue d'un meilleur parallélisme.
  - Étendre l'algorithme aux corps finis généraux.

Merci !

Multiplication exacte matrice creuse – vecteur sur architectures GPU et CPU multicœur

Brice Boyer[1] Jean-Guillaume Dumas[1]
Pascal Giorgi[2]

[1]LJK, Université de Grenoble, France
[2]LIRMM, Université Montpellier 2, France

JNCF '10

3 mai 2010