Conception d'un solveur GMRES parallèle creux pour cluster de GPUs

Lilia Ziane Khodja

Directeurs de thèse: Raphaël Couturier et Jacques M. Bahi
Equipe de recherche AND (Algorithmique Numérique et Distribuée)

14 février 2011







Objectifs

- Résolution de systèmes linéaires creux de très grandes dimensions;
- Matrices associées aux systèmes d'équations linéaires :
  - Grande dimension (plus de 10^5 inconnues) : une résolution gourmande en temps CPU et espace mémoire;
  - Nature creuse (moins de 5% valeurs non nulles) : accès irréguliers à la mémoire pour lire les valeurs non nulles.
- Méthode de résolution choisie: GMRES;
- Exploitation de la puissance des GPUs $\Rightarrow$ Cluster de GPUs.

GMRES

Generalized Minimal RESidual solver

- Développée par Y. Saad et M.H. Schultz en 1986 [Saa86];
- Généralisation de la méthode MINRES;
- Méthode itérative pour la résolution des systèmes linéaires creux asymétriques;
- Méthode générale : elle donne de très bonnes performances dans la plupart des cas.

Algorithme de GMRES avec redémarrage

1. Choisir la solution initiale x_0 et la tolérance d'erreur ε
2. $r_0 = M^{-1}(b - Ax_0)$
3. $\beta = \|r_0\|_2$
4. $v_1 = r_0/\beta$
5. Exécuter le processus d'Arnoldi:
   - $V_m = [v_1, \cdots, v_m]$ base orthonormale
   - $\overline{H}_m = (h_{i,j})$ matrice supérieure Hessenberg d'ordre $(m+1) \times m$
6. Résoudre le prob. des moindres carrés de taille m:
   $$min_{y \in \mathscr{R}^m} \left\| \beta e_1 - \overline{H}_m y \right\|_2$$
7. $x_m = x_0 + V_m y_m$
8. **Si** $\neg(\|b - Ax_m\|_2 < \varepsilon)$ **alors**
   Redémarrage avec x_m comme solution initiale
   Saut à la ligne 2
9. **fsi**

Carte graphique GPU

Graphic Processing Unit

- Initialement conçue pour la visualisation 3D en temps réel;
- Aujourd'hui, pour tout calcul intensif et parallèle;
- Architecture matérielle avantageuse :

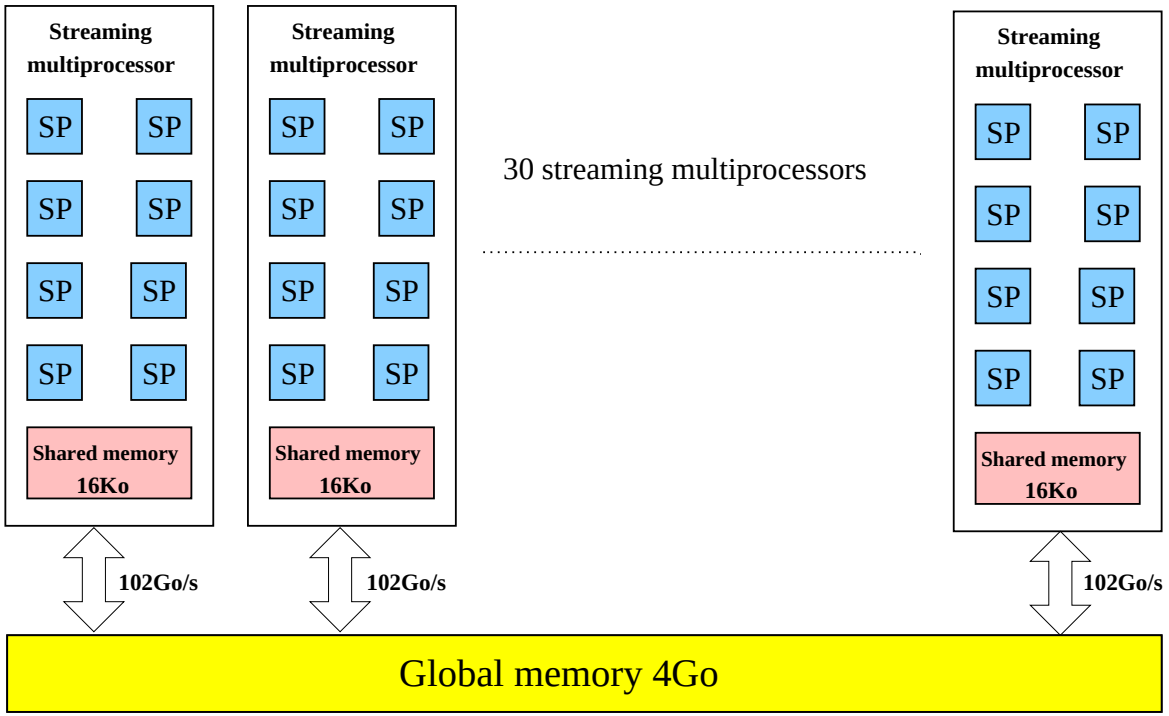


Figure: Exemple de GPU: Nvidia Tesla C1060 GPU

Programmation sur GPU

CUDA (Compute Unified Device Architecture programming)[Nvd07]

- Environnement de programmation Nvidia CUDA : extension de C;
- Dans un programme CUDA, deux types de codes :
  - Hôte (CPU) :
    - Contrôler le déroulement de l'application;
    - Exécuter le code séquentiel écrit en C pur;
    - Lancer les kernels sur le GPU.
  - Co-processeur (GPU) :
    - Exécuter les fonctions parallèles du code: kernels;
    - Calcul massivement parallèle SIMD (threads);
    - Rendre le résultat final d'un kernel au CPU.

Cluster de GPUs

- Un cluster de GPUs permet d'exploiter simultanément la capacité mémoire et la puissance de calcul de plusieurs cartes GPUs.

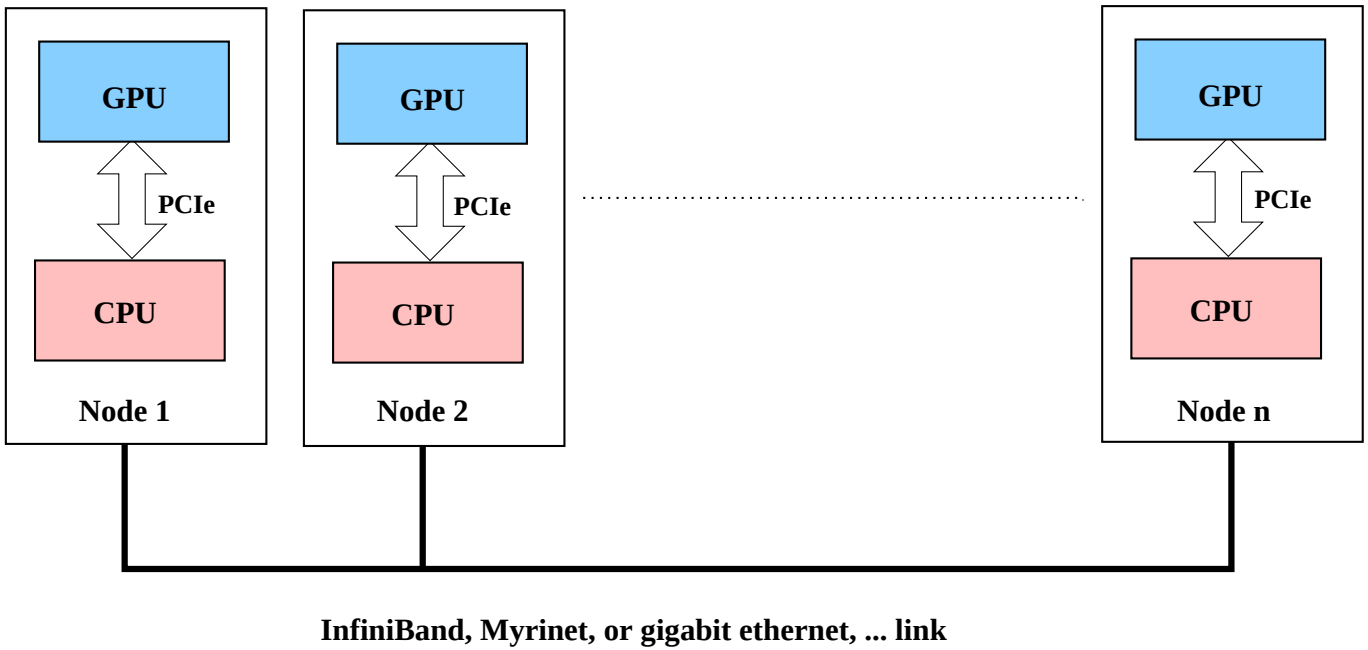


Figure: Exemple d'un cluster de GPUs

Points clés

Parallélisation sur un cluster de GPUs:

- Partitionnement des données;
- Définition des opérations à exécuter sur les GPUs;
- Détermination des points de synchronisation;
- Eviter au max les échanges CPU$\leftrightarrow$GPU.

Partitionnement des données

Soit le système de n équations linéaires suivant : $A \cdot x = b$

- Matrice creuse A en sous-matrices creuses A_i $(\lceil n/p \rceil \times n)$, $i = 1 \cdots p$;
- Vecteur résultant b en sous-vecteurs b_i de taille $\leq \lceil n/p \rceil$;
- Transfert des données des CPUs $\longrightarrow$ GPUs;
- Format de stockage des A_i sur les GPUs : HYB (ELL+COO);
- Vecteur solution x en sous-vecteurs x_i de taille $\leq \lceil n/p \rceil$ +dépendances.

Détermination des dépendances de données

Chaque CPU:

- Déterminer les voisins avec lesquels il partage les mêmes données;
- Calculer les dépendances;
- Echanger l'information avec les voisins.

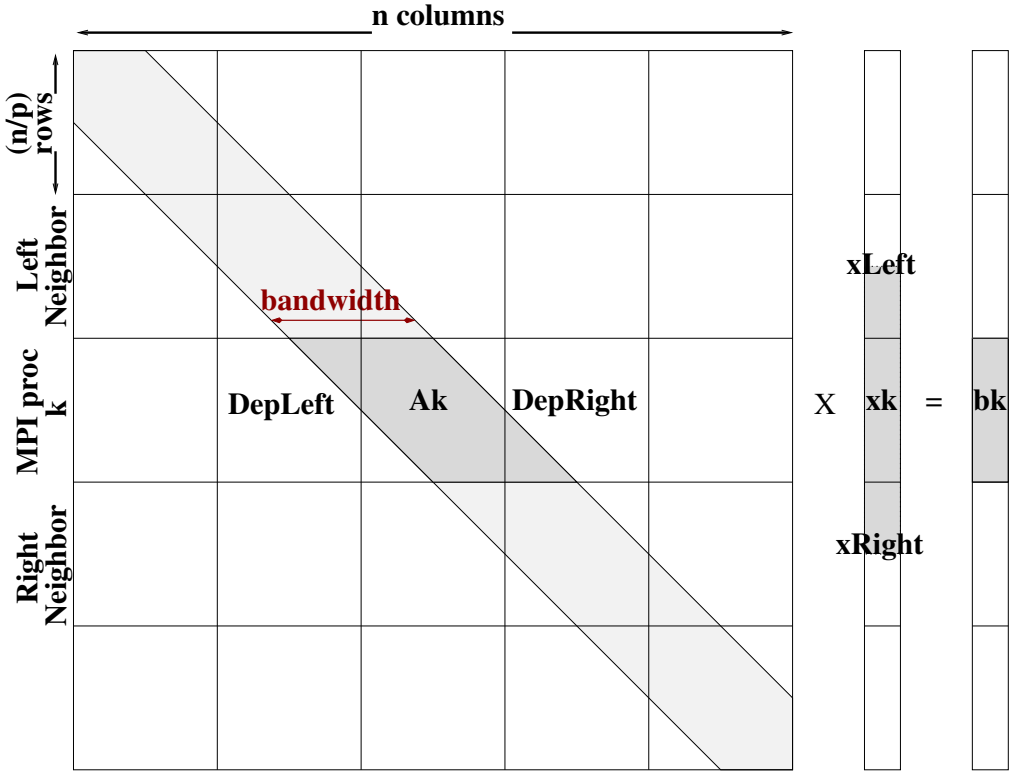


Figure: Partitionnement des données (matrice creuse bande)

Calcul local

A chaque itération de GMRES :

- Même algorithme GMRES exécuté sur chaque CPU, mais sur des données différentes: $A_i \cdot x_i = b_i$;
- Chaque CPU :
  - détient le contrôle de l'exécution de GMRES;
  - lance les calculs parallèles et intensifs sur le GPU (kernels).
- Calculs sur le GPU :
  - Adaptation du code CPU au GPU: fonctions écrites en CUDA;
  - Produit matrice-vecteur: kernel de la méthode hybride HYB;
  - Opérations sur les vecteurs: produits scalaires, normes euclidiennes, AXPY, ...$\Longrightarrow$cublas.
- Transfert des résultats calculés localement GPU$\rightarrow$CPU.

Calcul global

A chaque itération de GMRES :

- Pour chaque produit matrice-vecteur creux : $A \cdot x$
  - Construire le vecteur x global (x_i+dépendances) : MPI_Alltoallv;
- Calcul des résultats globaux (normes et produits scalaires): MPI_Allreduce.

Communications

Dans chaque itération de GMRES :

- Entre les GPUs $\leftrightarrow$ CPUs (fonctions cublas) :
  - CPU $\rightarrow$ GPU (cublasSetVector);
  - GPU $\rightarrow$ CPU (cublasGetVector).
- Entre les CPUs (fonctions MPI) :
  - Construction des vecteurs x : MPI_Alltoallv;
  - Calcul des résultats globaux : MPI_Allreduce;
  - Echange d'informations : MPI_Send, MPI_Recv, ...

Plate-forme de tests

- Cluster de GPUs:
  - Infiniband;
  - Six Quad-Cores Xeon E5530 Nehalem;
  - Deux cartes GPUs Tesla C1060 par CPU;
  - Linux cluster 2.6.18 OS;
  - $\Rightarrow$cluster de 12 GPUs.
- Mesures: Facteur d'accélération $\frac{T_{CPU}}{T_{GPU}}$ du cluster de 12 GPUs comparé à :
  - Un cluster de 12 CPUs;
  - Un cluster de 24 CPUs.

Données de tests

- Génération automatique de matrices creuses bandes;
- Matrices réelles de l'université de Floride [Florida];
- La matrice de chaque CPU est une copie multiple de la même matrice réelle;
- L'ensemble des copies sur le cluster construit une matrice de taille $2 \cdot 10^7$, $4 \cdot 10^7$ et $6 \cdot 10^7$ inconnues.

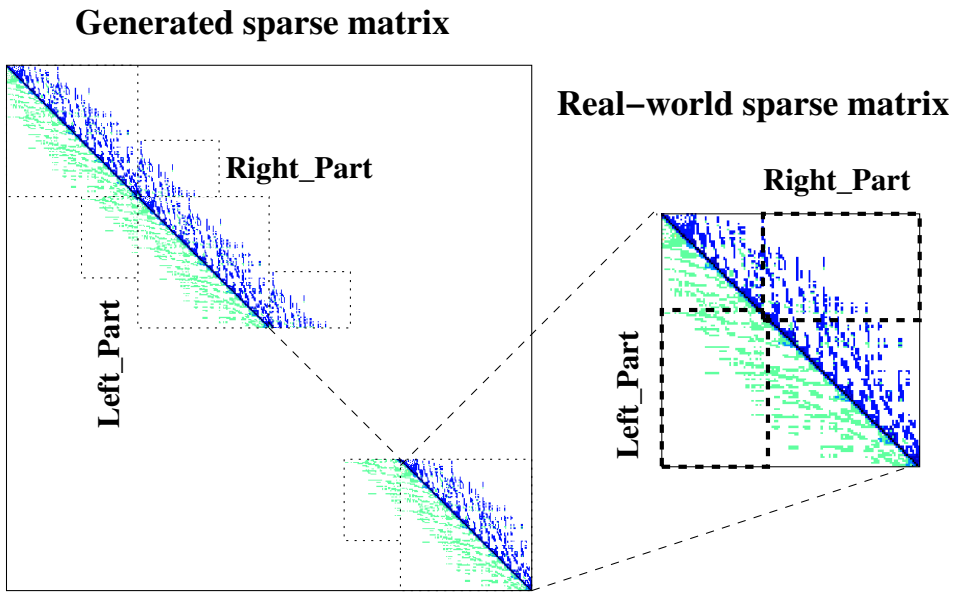


Figure: Génération de grandes matrices

Matrices de tests

- Matrices creuses de Florida:

Matrix	Nb. rows	Nb. nonzeros	Bandwidth
af_0_k101	503,625	9,027,150	1,715
bcsstk18	11,948	80,519	1,649
BenElechi1	245,874	6,698,185	1,170
cage14	1,505,785	27,130,349	1,492,442
ecology2	999,999	2,997,995	2,001
FEM_3D_thermal2	147,900	3,489,300	206,125
G3_circuit	1,585,478	4,623,152	1,584,983
shallow_water2	81,920	204,800	61,440
hood	220,542	5,494,489	220,542
thermal2	1,228,045	4,904,179	1,228,045

12 CPUs vs. 12 GPUs

- Facteur d'accélération $\frac{T_{12CPUs}}{T_{12GPUs}}$:
  - Précision: 3.56e-7 à 1.33e-14
  - Erreur: 2.41e-9 à 1.47e-17

Matrix	$2 \cdot 10^7$	$4 \cdot 10^7$	$6 \cdot 10^7$
af_0_k101	7.74	7.78	7.87
bcsstk18	7.88	8.14	8.23
BenElechi1	7.90	8.14	8.14
cage14	6.80	6.49	6.82
ecology2	8.59	9.14	8.86
FEM_3D_thermal2	7.09	7.30	7.55
G3_circuit	6.59	7.34	7.72
shallow_water2	9.06	9.60	9.75
hood	6.30	6.72	-
thermal2	4.94	5.58	6.81

24 CPUs vs. 12 GPUs

- Facteur d'accélération $\frac{T_{24\,CPUs}}{T_{12\,GPUs}}$:
  - Précision: 3.56e-7 à 1.33e-14
  - Erreur: 2.41e-9 à 1.47e-17

Matrix	$2{\cdot}10^7$	$4{\cdot}10^7$	$6{\cdot}10^7$
af_0_k101	5.29	5.23	5.40
bcsstk18	5.39	5.26	5.49
BenElechi1	5.90	5.70	5.72
cage14	6.14	5.84	4.81
ecology2	6.04	6.32	6.13
FEM_3D_thermal2	4.83	5.08	5.23
G3_circuit	5.99	6.51	5.32
shallow_water2	5.99	6.50	6.38
hood	4.57	4.68	-
thermal2	3.28	3.75	5.29

Conclusion et perspectives

- Bilan:
  - Premier solveur GMRES pour cluster de GPUs;
  - Résolution de grands systèmes linéaire creux:
    - est environ 8 fois plus rapide sur 12 GPUs que 12 CPUs;
    - est environ 5 fois plus rapide sur 12 GPUs que 24 CPUs.
  - Facteurs d'accélérations sensibles à la taille et la largeur de bande de la matrice.
- Perspectives:
  - Autres structures de matrices creuses;
  - Modes de partitionnement selon la structure de la matrice;
  - Conception de solveurs non linéaires pour cluster de GPU;
  - Prise en compte de l'asynchronisme des clusters de GPU distants.

Références

- N. Bell and M. Garland: "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors", Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009.
- The University of Florida Sparse Matrix Collection : http://www.cise.ufl.edu/research/sparse/matrices/
- NVIDIA: NVIDIA CUDA Compute Unified Device Architecture: Programming Guide", version 1.0, 2007.
- Y. Saad and M.H. Schultz: "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems", SIAM J. Sci. Stat. Comput., 7:856-869, 1986.

Fin

Merci pour votre attention!