Programmation numérique

Camille Coti
camille.coti@lipn.univ-paris13.fr

École Doctorale, Institut Galilée, Université de Paris XIII

Plan du cours

1. Introduction
2. Exemples : les BLAS
3. Exemples : LAPACK
4. Calcul distribué
5. Obtenir des informations sur l'architecture matérielle

Plan du cours

1. Introduction
   - Exemple introductif
   - Bibliothèques de calcul numérique
   - Organisation des bibliothèques entre elles
   - Système de nommage des routines
   - Appel d'une fonction d'une bibliothèque
2. Exemples : les BLAS
3. Exemples : LAPACK
4. Calcul distribué
5. Obtenir des informations sur l'architecture matérielle

Exemple introductif

Dans un code scientifique, besoin d'une opération de calcul

- Par exemple : multiplication matrice-matrice

Comment faire ?

Exemple introductif

Dans un code scientifique, besoin d'une opération de calcul

- Par exemple : multiplication matrice-matrice

Comment faire ?

- Écrire une multiplication de matrices à la main :
  - Approche naïve : 3 boucles
  - Pas du tout optimal en calcul, en accès mémoire...
  - Implémenter un algorithme plus efficace
  - Très long !

Pourquoi faire ce travail soi-même ?

Exemple introductif

Dans un code scientifique, besoin d'une opération de calcul

- Par exemple : multiplication matrice-matrice

Comment faire ?

- Écrire une multiplication de matrices à la main :
  - Approche naïve : 3 boucles
  - Pas du tout optimal en calcul, en accès mémoire...
  - Implémenter un algorithme plus efficace
  - Très long !

Pourquoi faire ce travail soi-même ?

Utiliser des bibliothèques optimisées !

Bibliothèques de calcul numérique

Optimisées :

- Utilisation d'un algorithme efficace en nombre d'opérations de calculs (flops)
- Patterns d'accès mémoires efficaces sur les architectures à mémoire hiérarchique (cache blocking)
- Implémentées de façon efficace
- Éventuellement, optimisées pour une architecture particulière (fournies par le constructeur ou tunées)

Standardisées :

- Les besoins sont les mêmes un peu partout
- Fournissent un ensemble de routines "classiques" : multiplications, factorisations, résolution de systèmes...
- Conventions de nommage faciles : on déduit du nom d'une routine ce qu'elle fait, on peut deviner le nom d'une routine quand on sait ce qu'on veut

Exemples de bibliothèques de calcul numérique

Routines de base : les BLAS

- Basic Linear Algebra Subroutines
- Le plus bas niveau !
- Optimisées pour la machine, souvent fournis ou tunés

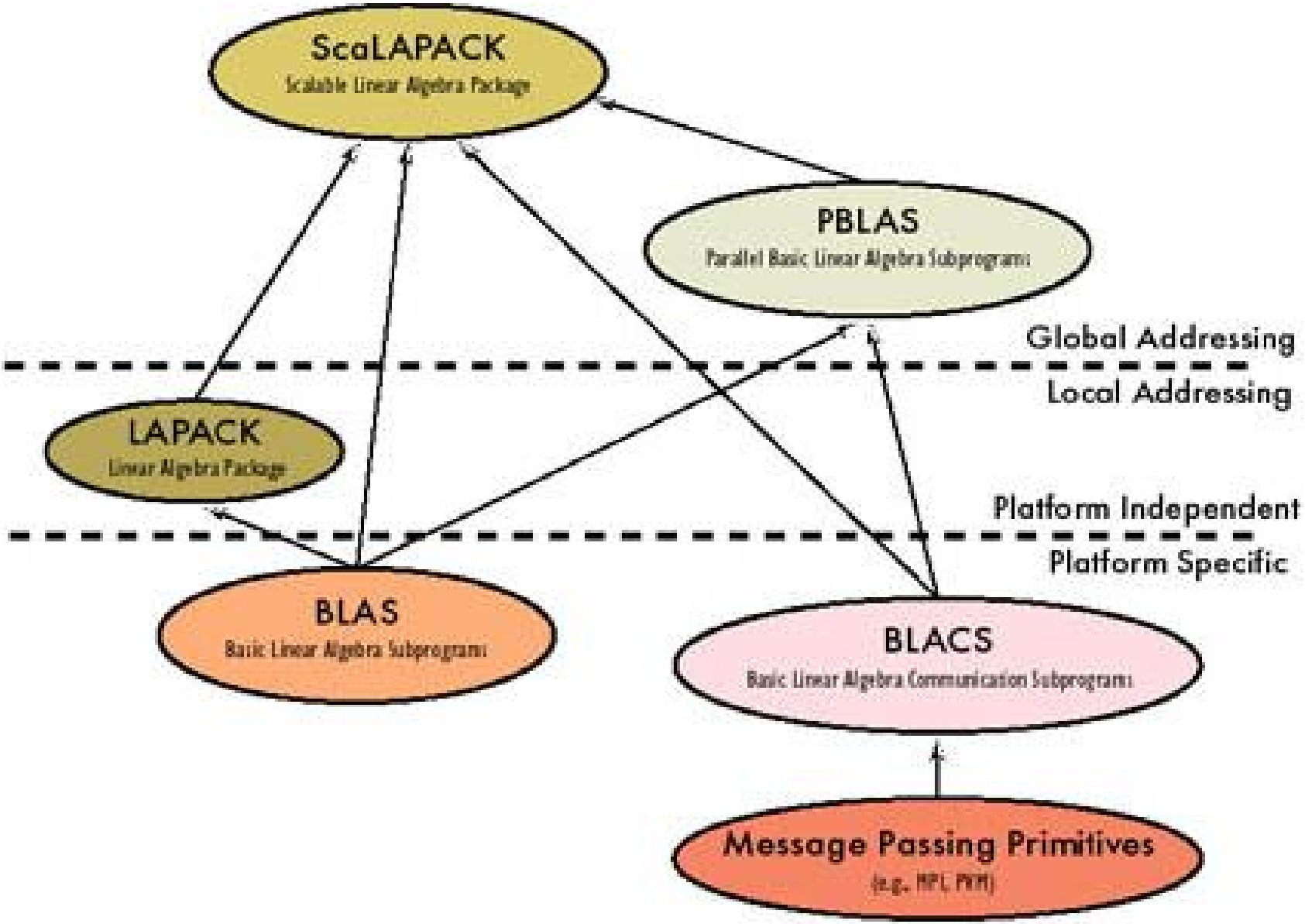
LAPACK : routines de plus haut niveau

MAGMA : pour les GPU

PLASMA : Parallel Linear Algebra for Shared Memory Architectures

ScaLAPACK : version distribuée de LAPACK

Dépendances entre les bibliothèques



Routines de base d'algèbre linéaire

Les BLAS

Ce sont des routines qui sont à la base de la plupart des calculs d'algèbre linéaire.

- Interface devenue un standard *de facto*
- Implémentations performantes, spécialisées (liées au matériel) ou généralistes

Calculs en précision simple, double, complexe et double complexe. Remarque : la première lettre du nom de la routine donne la précision (s, d, c, z).

Niveaux de BLAS

Rangées dans 3 catégories : les `niveaux`

- *Niveau 1* : opérations *vectorielles* de type $y = \alpha x + y$, produit scalaire, produit vectoriel, norme...
- *Niveau 2* : opérations *matrice-vecteur* de type $y = \alpha Ax + \beta y$, résolution de $Tx = y$ avec T triangulaire...
- *Niveau 3* : opérations *matrice-matrice* de type $C = \alpha AB + \beta C$, résolution de $C = \alpha T^{-1}C$...

Implémentations

- BLAS de référence (Netlib) : ancien, implémenté en Fortran, pas forcément les plus rapides
- GotoBLAS : implémenté en C, portable, rapide
- BLAS propriétaires : fourni avec la machine (MKL pour Intel, ACML pour AMD, ESSL pour IBM...), optimisé pour une plate-forme donnée
- Pour GPU : Cuda SDK contient une implémentation

Autres :

- xBLAS : BLAS fonctionnant un précision étendue

Système de nommage des routines

Système de nommage

Valable sur les BLAS, LAPACK, PLASMA, etc.

Lettre	Signification
1	Précision (s, d, c, z)
2-3	Type de matrice (GE = générale, OR = orthogonale...)
4-6	Algorithme (QRF, LU...)

Exemples :

- DGELUB = LU sur une matrice générale en double précision
- ZPOTRF = Cholesky sur une matrice définie positive (PO) en complexes double précision

Appeler une fonction d'une bibliothèque

Héritage de l'époque Fortran :

- Les bibliothèques étaient (voire sont toujours) écrites en Fortran, attention à l'interface
- Tout est passé par pointeurs
- Matrices à 1 dimension, ordre column-major
- Pas de valeur de retour : si la fonction retourne quelque chose, c'est le dernier paramètre

Si la lib est en Fortran et qu'on n'a pas de .h : il faut donner soi-même l'interface

```
void dgemm( char *transa, char *transb, int *m, int *n, int *k,
            double *alpha, double *a, int *lda, double *b,
            int *ldb, double *beta, double *c, int *ldc );
```

Exemple ici :

- d = double précision
- ge = matrices générales
- mm = multiplication matricielle

Passage de paramètres

Matrices

- Tableau à **une dimension**
- Ordre **column-major** (contrairement au C qui est row-major)
- Notion de **leading dimension**
  - Dans le cas où on utilise toute la matrice : nombre de lignes
  - Si on travaillesur une sous-matrice : distance entre les éléments de deux colonnes consécutives ayant le même indice sur les lignes

Lettres Beaucoup de paramètres comme "NoTranspose", "Lower", "Unit"...

- Passés comme des lettres
- Exemple : 'C' = Conjugate Transpose, 'N' = No Transpose, 'T' = Transpose
- On crée une variable constante de type `char` qui contient la lettre et on passe son adresse

Constantes

- Même chose : on passe l'adresse d'une variable qui contient la valeur de la constante

Exemple d'utilisation de LAPACK

DGESV : résolution de système linéaire

Interface C :

```
void dgesv( int *n, int *nrhs, double *a, int *lda,
            int *ipiv, double *b, int *ldb, int *info );
```

```
/* Définition des variables etc */
...

/* Allocation des matrices */
A = malloc( N * N * sizeof( double ) );
B = malloc( N * NRHS * sizeof( double ) );

/* Initialisation à des valeurs aléatoires */
dlarnv( &IONE, ISEED, &NN, A );
dlarnv( &IONE, ISEED, &NR, B );

/* Appel de DGESV */
dgesv( &N, &NRHS, A, &LDA, IPIV, B, &LDB, &info );
```

Où trouver de l'information

Sur Netlib

- Dépôt en ligne
- `http://www.netlib.org`
- Bibliothèques en téléchargement, documentation des routines
- Dans le code des routines elles-mêmes : au début du code

Les LAWN

- LAPACK Working Notes
- Très important : la LAWN 41 **Installation Guide for LAPACK, Susan Blackford et Jack Dongarra** : utilisation des routines, calculs effectués, complexité

Plan du cours

1 Introduction

2 Exemples : les BLAS
- Multiplication matrice-matrice
- Benchmarking
- Performances sur Magi

3 Exemples : LAPACK

4 Calcul distribué

5 Obtenir des informations sur l'architecture matérielle

Exemple d'utilisation de BLAS

DGEMM : multiplication matrice-matrice

Interface C :

```
void dgemm( char *transa, char *transb, int *m, int *n, int *k,
            double *alpha, double *a, int *lda, double *b,
            int *ldb, double *beta, double *c, int *ldc );
```

```
/* Définition des variables etc */
...

/* Allocation des matrices */
A = (double *)malloc(N*N*sizeof(double));
B = (double *)malloc(N*N*sizeof(double));
C = (double *)malloc(N*N*sizeof(double));

/* Initialisation à des valeurs aléatoires */
dlarnv(&IONE, ISEED, &NN, A);
dlarnv(&IONE, ISEED, &NN, B);
dlarnv(&IONE, ISEED, &NN, C);

/* Appel à DGEMM */
dgemm( &NoTranspose, &NoTranspose, &N, &N, &N, &c__1,
       A, &N, B, &N, &c__1, C, &N );
```

Remarques

- Indications passées à BLAS : NoTranspose...
- Passage des paramètres par adresse (interfaçage avec Fortran)
- Matrices déclarées en 1 dimension (Fortran)

Benchmarking avec DGEMM

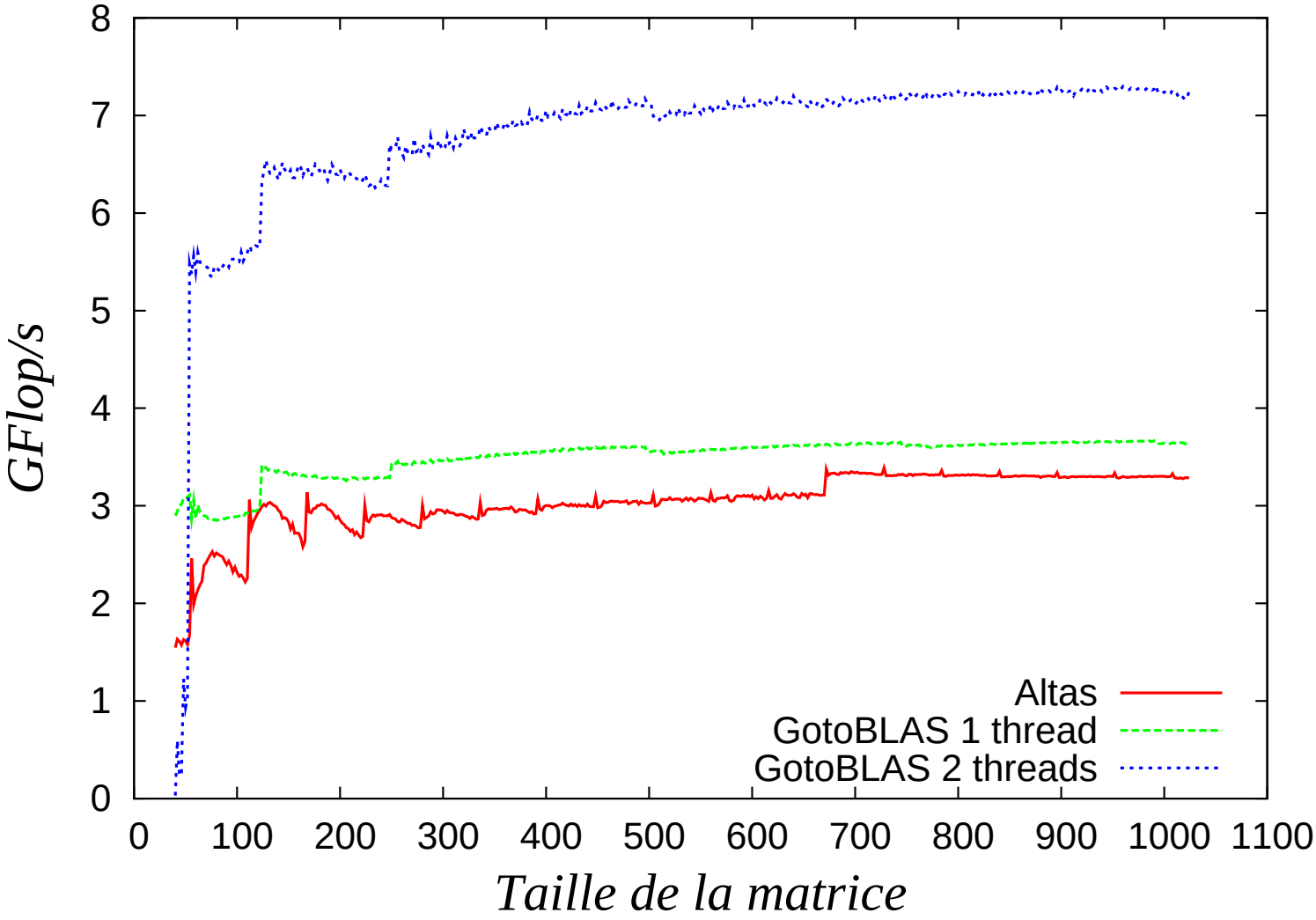
Principe : multiplication de deux matrices carrées aléatoires

- Calcul très CPU-intensif
- On augmente la taille des matrices
- Tant que les matrices tiennent en cache : très rapide !
- Quand les matrices ne tiennent plus en cache : déplacements de données

Estimation de l'occupation mémoire :

- Matrices 1000x1000, double précision
- 2 matrices de données + matrice résultat : 3×10^6 mots mémoire
- Double précision : un mot mémoire = 8 octets
- Total : $24 \times 10^6 \sim 23$ Mo $\rightarrow$ ne tient pas dans un cache de Core i7 !

Exemple de benchmarking de plusieurs implémentations de DGEMM



Est-on loin du pic théorique de la machine ?

Calcul du pic théorique de la machine

- Nombre de calculs effectués par cycle d'horloge
  - généralement 4
  - maintenant 8 sur certains processeurs
- Nombre de cycles d'horloge par seconde

Exemple :

- Intel Nehalem (Core i7) : 8 opérations SP ou 4 opérations DP par cycle
- En utilisant les instructions AVX : 16 opérations SP ou 8 opérations DP par cycle
- Core i7 3,5 GHz en double précision :
  - $3.5 \times 10^9 \times 4 = 14$ Gflops
  - Par coeur !!

Et en multi-coeur ?

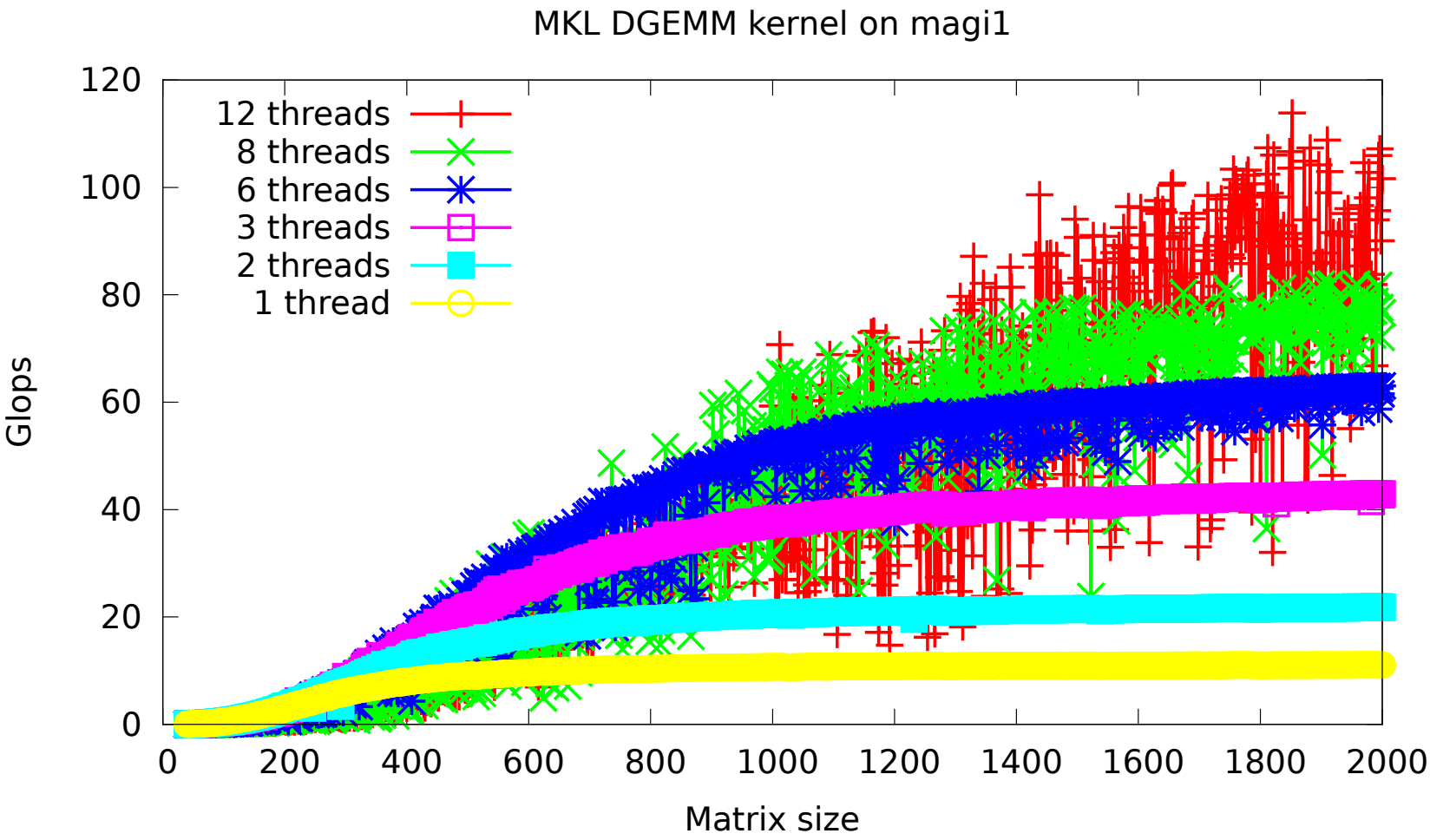
Attention : les BLAS multithreadés ont tendance à utiliser **tous les coeurs** qu'ils trouvent

- Un processus sur un octo-coeur : 8 threads
- Deux processus sur un octo-coeur : 8 threads **par processus** donc 16 threads en tout !

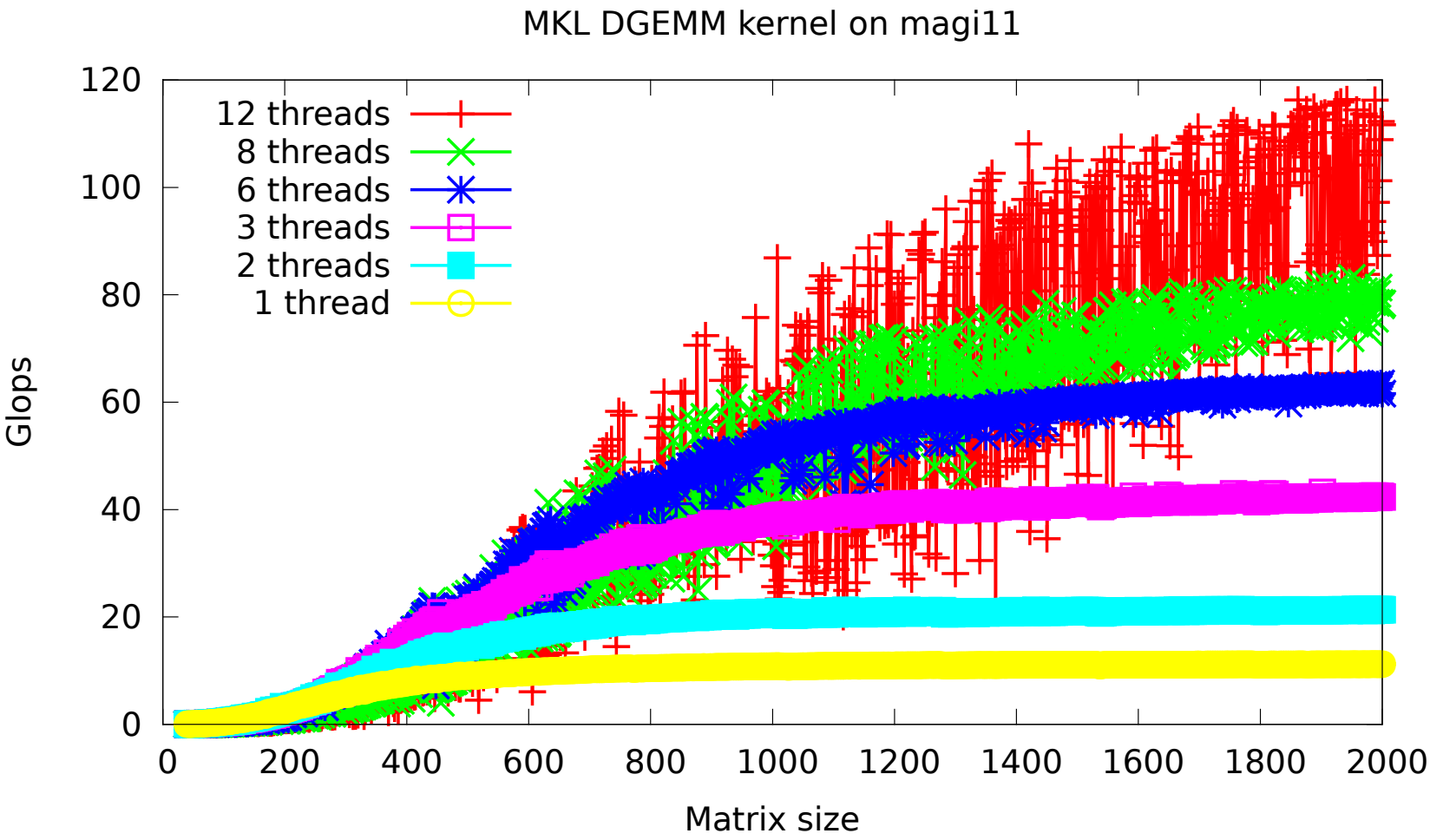
Dangereux pour les performances !

- Attention à ne pas utiliser plus d'unités de calcul (processus ou threads) que de coeurs disponibles
- Variable d'environnement pour fixer le nombre de threads :
  - `GOTO_NUM_THREADS` ou `OMP_NUM_THREADS` pour GotoBLAS
  - `MKL_NUM_THREADS` ou `OMP_NUM_THREADS` pour MKL
  - Fonctions comme `omp_set_num_threads()` ou `mkl_set_num_threads()`

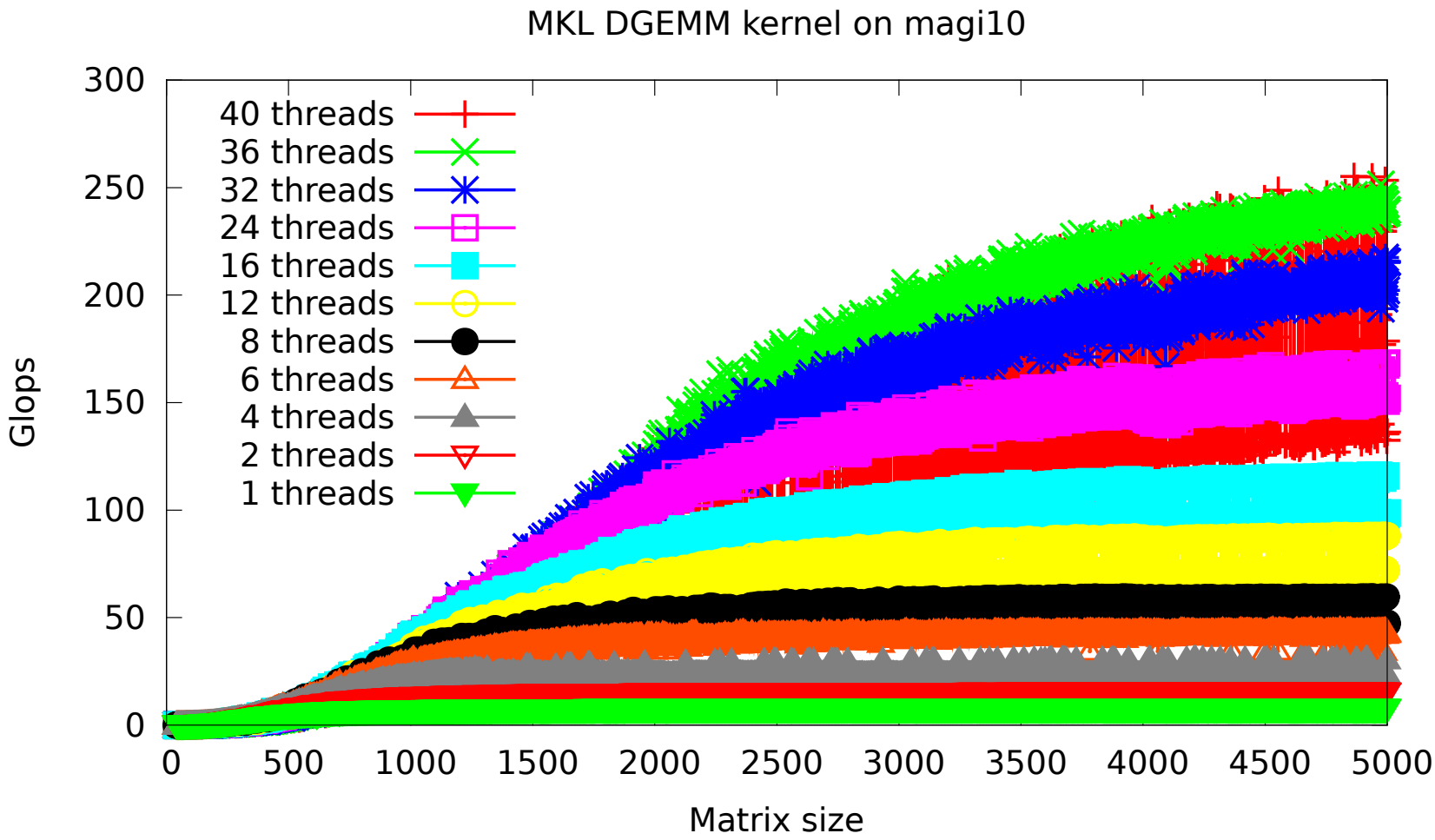
Benchmarking sur Magi : DGEMM sur magi1



Benchmarking sur Magi : DGEMM sur magi11



Benchmarking sur Magi : DGEMM sur magi10



Plan du cours

1 Introduction

2 Exemples : les BLAS

3 Exemples : LAPACK
- Routines de LAPACK
- Factorisations : Los Tres Amigos
- Un point sur la précision de calcul
- Performances comparées

4 Calcul distribué

5 Obtenir des informations sur l'architecture matérielle

Routines fournies par LAPACK

Routines de **résolution de systèmes** :

- Résolution de systèmes d'équations linéaires (successeur de LINPACK)

Mais aussi

- Calcul de valeurs propres (successeur de EISPACK)
- Factorisations : Los Tres Amigos (LU, QR, Cholesky), factorisation de Schur

Utilisation :

- Même convention de nommage que pour les BLAS
- Dépend des BLAS : nécessité d'avoir des BLAS performants
- Écrit en Fortran : mêmes contraintes d'interfaçage
- Compilation : utiliser un linkeur Fortran pour l'édition des liens ou linker avec une lib comme `libgfortran`
- Avec certains compilateurs : ajouter un underscore _ à la fin des noms des routines : `dlarnv` devient `dlarnv_`

Implémentations :

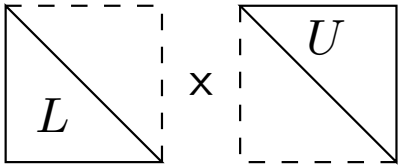
- ATLAS : installateur qui auto-tune la bibliothèque. Long mais nécessaire
- Sinon, fourni dans ESSL, MKL et ACML

Décomposition LU

Definition

La décomposition LU d'une matrice inversible A est sa décomposition en deux matrices L et U avec :

- $A = LU$
- L est une matrice triangulaire inférieure
- U est une matrice triangulaire supérieure



Calcul de la décomposition LU

On la calcule par élimination gaussienne

- On calcule les colonnes une par une
- On détermine un pivot de Gauss
- On élimine les termes sous la diagonale par combinaison linéaire

Theorem (Complexité de la factorisation LU)

La complexité du calcul de la factorisation LU d'une matrice de taille $M \times N$ est en $O(\frac{2}{3}MN^2)$

Décomposition LU : applications

Applications : résolutions de systèmes linéaires, inversion de matrices, calcul de déterminant...

Résolution de système avec LU

On cherche à résoudre le système $Ax = b$.

- On décompose A en $A = LU$
- Le système devient alors = $LUx = b$
- On résout en deux étapes :
  - $Ly = b$
  - $Ux = y$

Stabilité numérique

L'utilisation d'un pivot apporte un risque de propagation des erreurs d'arrondi

- L'utilisation d'un pivot global améliore la stabilité numérique
- Mais la recherche du pivot global augmente le nombre d'opérations à effectuer

Décomposition de Cholesky

Definition

La décomposition de Cholesky d'une matrice A définie positive et symétrique est sa décomposition en une matrice L triangulaire inférieure telle que $A = LL^T$.

Applications : chimie quantique, inversion de matrice
Exemple :

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 5 & 5 & 5 \\ 1 & 5 & 14 & 14 \\ 1 & 5 & 14 & 15 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Theorem (Existence, unicité et complexité)

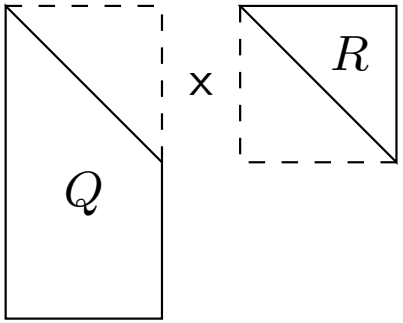
Si A est une matrice carrée symétrique définie positive, alors il existe au moins une matrice réelle triangulaire inférieure L telle que $A = LL^T$. (existence)
Si on impose que tous les éléments diagonaux de la matrice L sont positifs, alors cette matrice L est unique. (unicité)
La complexité du calcul de la décomposition de Cholesky d'une matrice carrée de taille N est en $O(\frac{N^3}{3})$. (complexité)

Factorisation QR

Definition

La décomposition QR (ou factorisation QR) d'une matrice A est sa décomposition en deux matrices Q et R avec :

- $A = QR$
- R est une matrice triangulaire supérieure
- Q est une orthogonale



Applications : résolution de systèmes linéaires non carrés, décomposition en éléments propres, moindres carrés...

Theorem (Existence et unicité)

Si A est une matrice inversible, alors il existe au moins une décomposition QR de A. (existence)
Si on impose que tous les éléments diagonaux de la matrice R sont positifs, alors cette décomposition est unique. (unicité)

Décomposition QR : méthode de Householder (LAPACK)

Méthode de Householder

On applique des réflecteurs H_k successifs pour projeter un à un les vecteurs de la matrice A sur un hyperplan.

Exemple :

$$A = \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \qquad H_1 A = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & x & x \end{bmatrix} \qquad H_2(H_1 A) = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \end{bmatrix}$$

On obtient $R = H_2 H_1 A$, d'où on déduit $Q = H_2 H_1$

Theorem

Le calcul de la factorisation QR d'une matrice de taille $M \times N$ en utilisant l'algorithme de Householder nécessite $2MN^2 - 2/3N^3$ opérations en calculant uniquement la matrice R, et $4MN^2 - 4/3N^3$ opérations en reconstituant également la matrice Q. Sa complexité est donc en $O(MN^2)$.

Calcul en précision mixte

Exemple avec *GESV : *General Matrix Factorization and Multiple Right-Hand Side Solve*

- Résolution de système de type $A \times X = B$, avec A une matrice, X les inconnues et B des colonnes
- Calcul itératif en deux phases :
  1. Approximation d'une solution : complexité en N^3
  2. Raffinement de la solution : complexité en N^2

Observation :

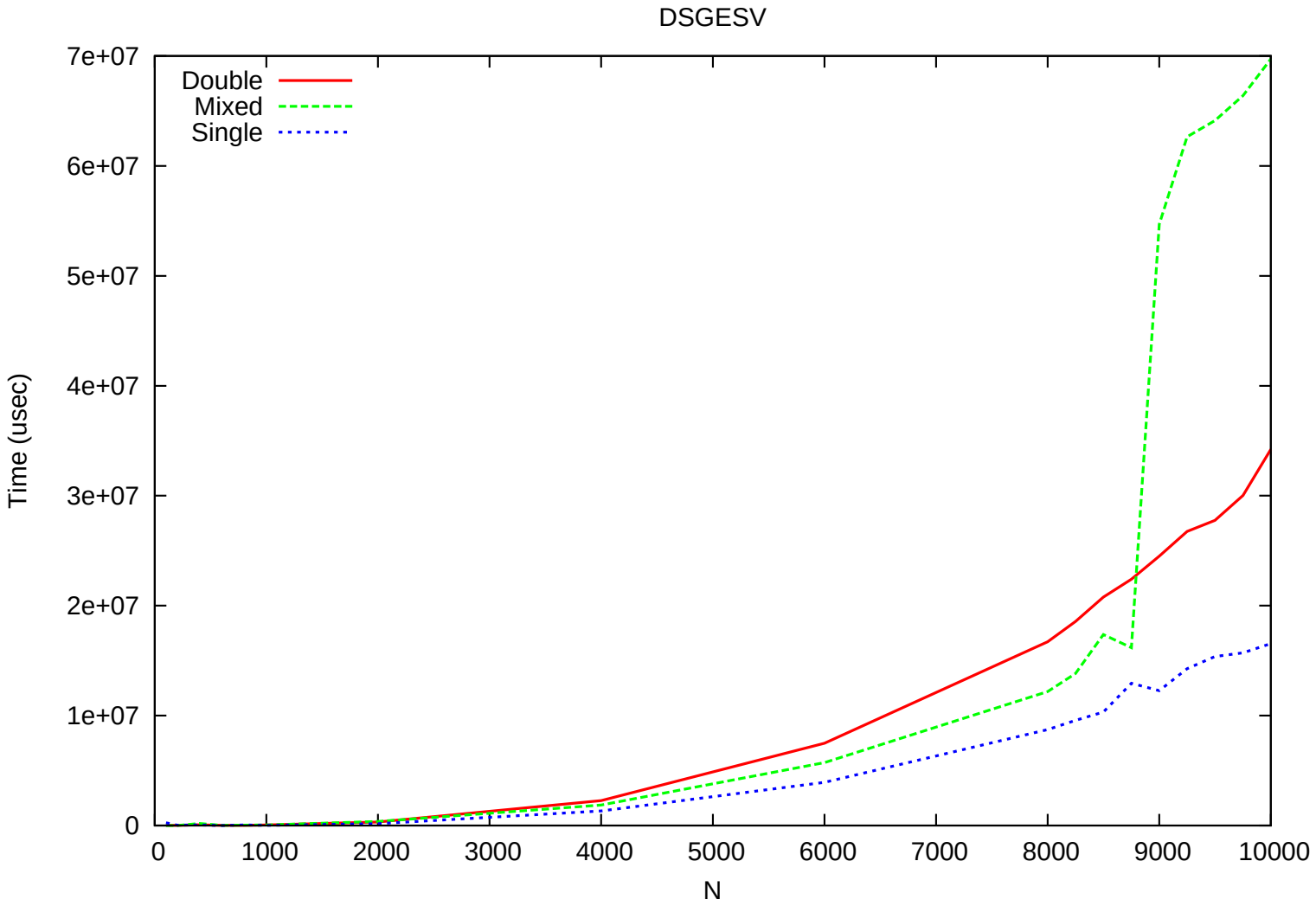
- Les calculs en simple précision sont plus rapides
- Les calculs en double précision donnent des résultats plus précis

Principe des calculs en précision mixte :

- Faire la première phrase (en N^3) en précision simple
- Affiner le résultat (en N^2) en précision double

But : **obtenir un résultat en double précision avec une rapidité proche de la simple précision**

Comparaison de DGESV, SGESV, DSGESV



Plan du cours

1 Introduction

2 Exemples : les BLAS

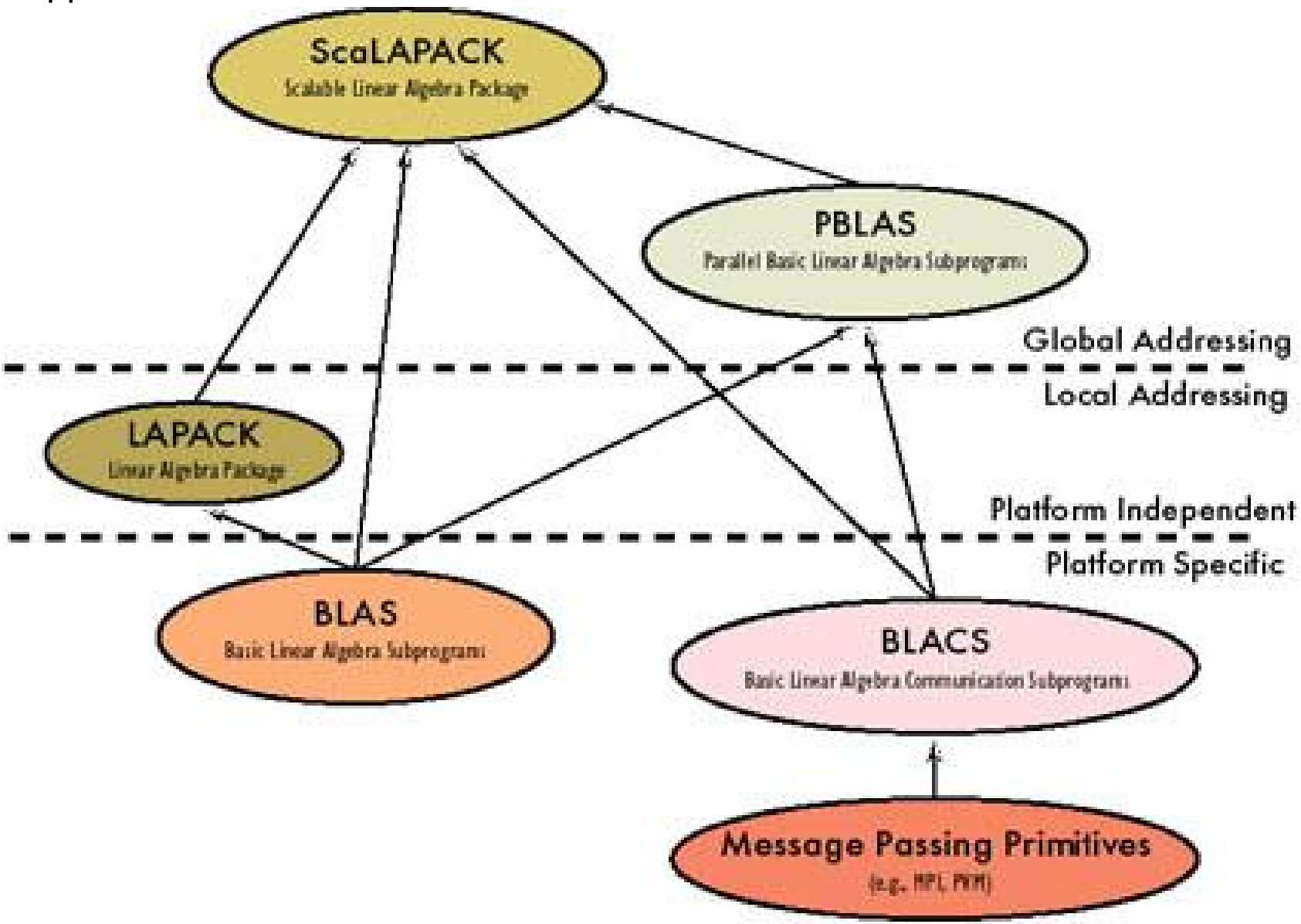
3 Exemples : LAPACK

4 Calcul distribué
- Les BLACS
- ScaLAPACK
- Utilisation de MPI

5 Obtenir des informations sur l'architecture matérielle

ScaLAPACK

Rappel de l'architecture :



Utilisation des BLACS

BLACS : **Basic Linear Algebra Communication Subprograms**

- Routines utilisées par ScaLAPACK, PBLAS
- Routines de communication de haut niveau
- Surcouche au-dessus du middleware de communication : MPI, PVM...
- Fournissent une abstraction : les routines de calcul au-dessus n'ont pas à savoir comment on communique

Permettent notamment de mettre en place une **grille de processus**

- Distribution de la matrice sur ces processus
- Routines de communication sur cette topologie

Avec les CBLACS (interface C qui appelle les BLACS) :

```
Cblacs_pinfo( &iam, &nprocs ) ;
Cblacs_get( -1, 0, &ictxt );
Cblacs_gridinit( &ictxt, "Row", nprow, npcol );
Cblacs_gridinfo( ictxt, &nprow, &npcol, &myrow, &mycol );
Cblacs_gridexit( 0 );
```

Routines ScaLAPACK

Même convention de nommage que pour LAPACK

- Avec un `p` au début : parallel
- Exemple : `pdgeqrf` : parallel, double precision, matrices générales, facto QR

Doivent être appelées **après initialisation de la grille BLACS** !

Les données doivent être **distribuées sur les processus**

Appel :

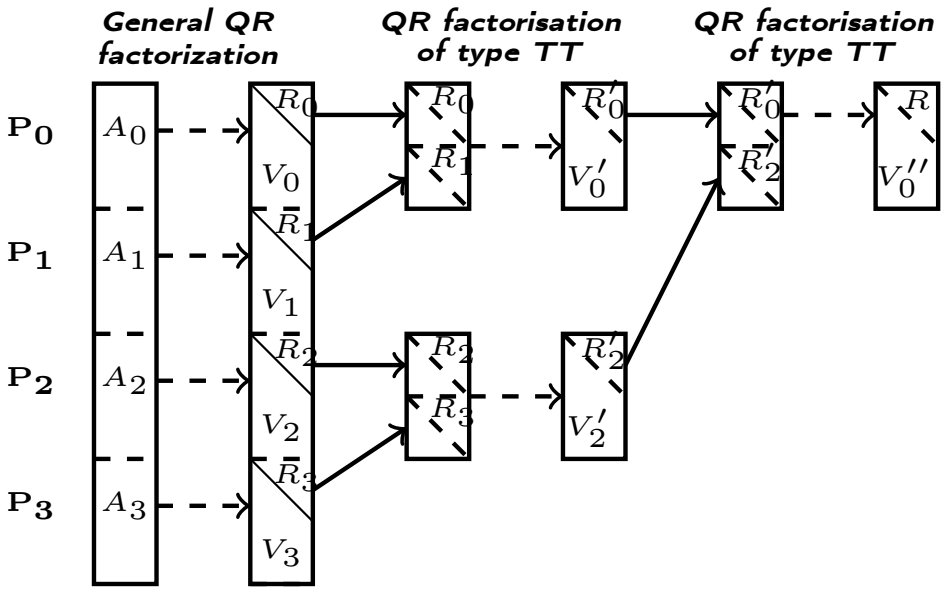
```
pdgeqrf_(&m, &n, A, &ia,&ja,descA, tau, work, &lwork, &info);
```

MPI

Possibilité d'implémenter d'autres algorithmes parallèles utilisant LAPACK et les BLAS

- Par exemple en communiquant via MPI

Exemple avec TSQR : facto QR pour matrices "tall and skinny" (M » N) à évitement de communications



Communications le long d'un arbre binaire

- QR sur ma sous-matrice
- Communication avec un voisin :
  - Si mon rang est pair : recevoir depuis $r + 1$
  - Si mon rang est impair : envoyer à $r - 1$
- Si mon rang est impair : fin

Implémentation en MPI

2 possibilités :

- En utilisant une communication collective : réduction
  - L'opération effectuée (facto QR des matrices constituées des triangles) est binaire, associative et commutative
  - On appelle la fonction `MPI_Reduce`
  - Avec un type construit
  - Et un pointeur vers une fonction qui fait une facto QR
- "À la main" : en écrivant la remontée de l'arbre par des communications point à point, les appels aux fonctions de calcul à la réception

Performances de TSQR

Performance en MFlops/sec/proc sur une matrice de dimensions m = 1.000.000 et n = 50 (scalabilité forte).

# of processors	32	64	128	256
1. TSQR	690	666	662	610
2. TSQR with MPI_Reduce	420	411	414	392
3. ScaLAPACK Householder QR	193	190	206	184

Référence : *Computing the R of the QR factorization of tall and skinny matrices using MPI_Reduce, Julien Langou, arXiv:1002.4250v1 [math.NA]*

Plan du cours

1 Introduction

2 Exemples : les BLAS

3 Exemples : LAPACK

4 Calcul distribué

5 Obtenir des informations sur l'architecture matérielle

Importance de bien connaitre son architecture

Pour programmer des codes **efficaces** , il est important de bien connaitre son architecture :

- Tunning de son code en réglant des paramètres :
  - Taille des blocs pour le cache-blocking
  - Paramètres de récursion...
- Nombre exact de coeurs : pas si évident que ça !

Exemple avec le nombre de coeurs :

- Sur un Core i7 i7-2600 :

```
coti@maximum:~$ cat /proc/cpuinfo | grep processor | wc -l
8
```

- En réalité : 4 coeurs + hyperthreading
- $\rightarrow$ 4 coeurs physiques, 8 coeurs logiques
- On peut vouloir n'utiliser que les 4 coeurs (utilisation des caches, optimisation des chemins vers les pipelines d'instructions...)
- Comment savoir combien on a de coeurs physiques ?

Utilitaire hwloc

Utilisation de l'utilitaire **hwloc**

- Découverte de la topologie locale de la machine : CPU, coeurs, niveaux de mémoire hiérarchique

```
coti@maximum:~$ hwloc-ls
Machine (7962MB) + Socket L#0 + L3 L#0 (8192KB)
  L2 L#0 (256KB) + L1 L#0 (32KB) + Core L#0
    PU L#0 (P#0)
    PU L#1 (P#4)
  L2 L#1 (256KB) + L1 L#1 (32KB) + Core L#1
    PU L#2 (P#1)
    PU L#3 (P#5)
  L2 L#2 (256KB) + L1 L#2 (32KB) + Core L#2
    PU L#4 (P#2)
    PU L#5 (P#6)
  L2 L#3 (256KB) + L1 L#3 (32KB) + Core L#3
    PU L#6 (P#3)
    PU L#7 (P#7)
```

On a donc :

- 8 Go de RAM sur la machine
- Une socket (un CPU)
- 8 Mo de cache L3 partagé entre les coeurs du CPU
- 4 coeurs physiques
- 256 Ko de cache L2 par coeur physique
- 32 Ko de cache L1 par coeur physique

- 2 unités de calcul (coeurs logiques) par coeur physique
- les coeurs logiques se partagent les caches L1 et L2