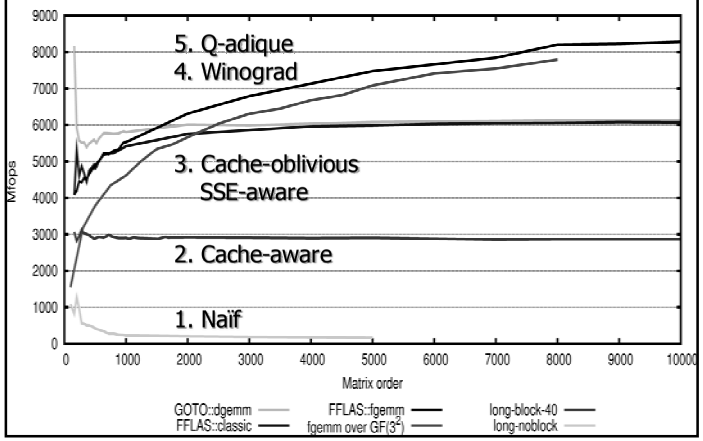
Compromis Opérations-Mémoire : algèbre linéaire dense et corps finis

Jean-Guillaume Dumas

Université de Grenoble – Laboratoire Jean Kuntzmann

FFLAS : 8.2 Gffops sur un XEON 3.6 GHz



Compromis Temps-Mémoire pour l'algèbre linéaire dense sur des corps finis

1. Applications et réductions
   - Cryptologie, Combinatoire, etc.
   - Matrices entières, rationnelles
     - Restes chinois, remontées, etc.
   - Matrices creuses : méthodes par blocs
2. FFLAS-FFPACK [2002]
   - Algèbre linéaire modulaire : architecture-oblivious
3. Gestion de la mémoire dans l'algorithme de Winograd
4. Utilisation de tables pour l'arithmétique modulaire

On a toujours besoin de plus petit que soit

- Factorisation d'entiers, logarithme discret
  - Algèbre linéaire modulo 2, modulo n
- Combinatoire
  - Formes normales entières, polynômes minimaux/caractéristiques entiers, etc : méthodes modulaires
- Algorithmes stables pour des problèmes numériques
  - Matrices rationnelles : méthodes modulaires
- Matrices creuses
  - Méthodes par blocs (Coppersmith-Wiedemann, Lanczos) :
    $\Rightarrow$Les blocs sont denses tels que MM est rapide
  - Méthodes probabilistes (réussite dépend de la taille du corps) :
    $\Rightarrow$Résolution dans une extension finie

Plan

1. Algèbre linéaire exacte
2. Une algèbre linéaire dense modulaire
   - Contraintes matérielles
   - Gestion de la division
   - Gestion de la mémoire
3. Accélération par la méthode de Strassen-Winograd
4. Arithmétique des corps finis

Calcul haute performance / Calcul exact ?

- Mémoire : optimiser les accès mémoire et en particulier l'utilisation des caches
  - cf BLAS (ATLAS, GOTO, etc.)
- Calcul exact
  - La division (ex: modulo p) est 10 à 100 fois plus lente qu'une multiplication/addition machine
  - SSE (registres de taille double) : en 2008, existent uniquement en flottant

FFLAS

[D, Gautier, Pernet 2002]

- Gestion de la division :
  - Plongement dans Z : faire tout un produit scalaire avant de faire une division
- Gestion de la localité :
  - Blocs
- Utilisation des SSE :
  - Laisser faire l'algèbre linéaire à un code numérique

FFLAS: algèbre linéaire dense exacte

Ex: Multiplication de matrices à coefficients entiers modulo un premier p

1. Convertir les matrices mod p vers des matrices numériques (*double*)
2. Utiliser les BLAS numériques (e.g. GOTO) pour multiplier en double
3. Reconvertir les *doubles* modulo p

$O(n^2)$ conversions versus $O(n^3)$ opérations arithmetiques rapides

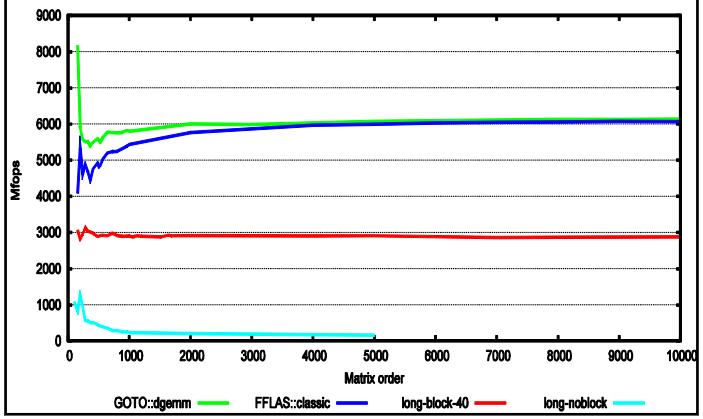
⇒ Exact tant que les opérations ne dépassent pas la taille du mot

⇒ Chaque produit scalaire doit tenir dans la mantisse

Ex.: $n\,(p-1)^2/4 < 2^{52}$: pour $p \leq 2^{16}$, n=4 000 000 est OK
pour $n \leq 6000$, le premier peut avoir 20 bits

Pour des premiers ou des matrices plus grandes il est nécessaire de faire des étapes récursives de multiplications par blocs puis de convertir et de terminer par des additions modulaires (en kn^2).

Cache oblivious, SSE aware



FFPACK : réductions à la multiplication de matrices

- Algorithmes récursifs par blocs
  - LQUP [Ibarra et al. 1982] : élimination de Gauß en place
  - …
  - Polynôme caractéristique [Pernet, Storjohann 2007]
- Algorithmes en Cascade
  - Niveau récursif
  - Niveau adaptatif (à l'architecture, au modulo, …)
  - Niveaux de base
- Exemple : pseudo-inverse de Moore-Penrose

  MPINV(m;n) =LQUP(m;n) + AAT(r;m-r) + AAT(r;n-r) + 3TRSM(r,m-r) + 3TRSM(r,n-r) + MM(r) + LQUP(r) + 2UTUT(r) + 2INVT(r) + UTLT(r) + R(n-r;r;m-r)

Modélisation d'un algorithme adaptatif [AHA, 2006]

- Plusieurs algorithmes résolvent un même problème **f** :
  - Eg : ***algorithme*$_1$**, ***algorithme*$_2$**(block size), … ***algorithme*$_n$**
  - chaque *algorithme*$_k$ étant récursif :

algorithme$_k$ (n, …) {
….
f (n - 1, …) ;
….
f (n / 2, …) ;
…
}

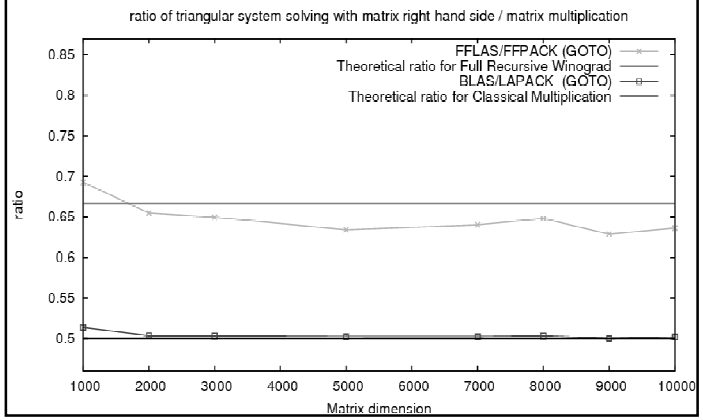
Adaptation
choisit *algorithme*$_i$ pour chaque appel à f

- Exemple : TRSM
  a) Poser T=UD, U unitaire puis résoudre UY=b, X=D^{-1}Y
  b) Appeler les BLAS : $n < \log_{(p+1)/2} (2^{m+1} / (p-1))$
  c) Récursif retardé : $n < 2^{m+1} / (p-1)^2$

☝ Selon n et l'architecture, une des variantes est plus rapide

⇒ L'algorithme adaptatif détermine automatiquement une bonne combinaison récursive d'algorithmes

Efficacité de la réduction : TRSM



Plan

1. Algèbre linéaire exacte
2. FFLAS-FFPACK
3. Gestion de la mémoire dans l'algorithme de Winograd
   - Multiplication de matrices rapide
   - Garanties de correction
   - Placements mémoire efficaces
     - C = A×B
     - C += A×B
4. Utilisation de tables pour l'arithmétique modulaire

Multiplication de Matrice rapide

- Meilleure complexité théorique connue $O(n^{2.375})$
- On note $MM(n)=O(n^{w})$ la complexité du produit de matrices
- Algorithme Diviser pour Régner *[Winograd 1971]*
  - Complexité passe de $2n^3$ à $6n^{2.81}$
  - Augmentation des besoins en mémoire
  - Instable numériquement
    - Variante *[Strassen 1969]* $\approx 7n^{2.81}$: un peu plus stable
    - Variante *[Pan,Kaporin 2004]* $\approx 8.35\ n^{2.78}$: division en 48 blocs
    - Peu utilisées en numérique
  - En exact, aucun problème de stabilité
    - borne d'utilisation des FFLAS réduite *[D., Giorgi, Pernet 2008]*
    - de $n\,(p-1)^2/4 < 2^{52}$ à $\approx C^{4.62k}\,n\,(p-1)^2/4 < 2^{52}$
    - Suffisant néanmoins …

Winograd's algorithm computes the $m \times n$ matrix $C = A \times B$ with 22 block operations:

- 8 additions:

$$\begin{array}{ll} S_1 \leftarrow A_{21} + A_{22} & T_1 \leftarrow B_{12} - B_{11} \\ S_2 \leftarrow S_1 - A_{11} & T_2 \leftarrow B_{22} - T_1 \\ S_3 \leftarrow A_{11} - A_{21} & T_3 \leftarrow B_{22} - B_{12} \\ S_4 \leftarrow A_{12} - S_2 & T_4 \leftarrow T_2 - B_{21} \end{array}$$

[Strassen 1969]

- 7 recursive multiplications:

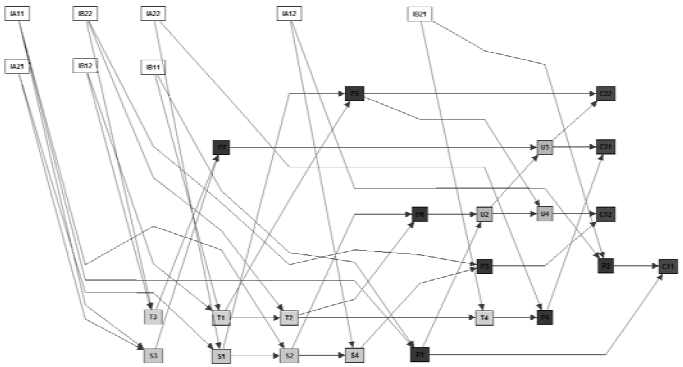
$$\begin{array}{ll} P_1 \leftarrow A_{11} \times B_{11} & P_5 \leftarrow S_1 \times T_1 \\ P_2 \leftarrow A_{12} \times B_{21} & P_6 \leftarrow S_2 \times T_2 \\ P_3 \leftarrow S_4 \times B_{22} & P_7 \leftarrow S_3 \times T_3 \\ P_4 \leftarrow A_{22} \times T_4 & \end{array}$$

[Winograd 1971]

- 7 final additions:

$$\begin{array}{ll} C_{11} \leftarrow P_1 + P_2 & C_{12} \leftarrow U_4 + P_3 \\ U_2 \leftarrow P_1 + P_6 & C_{21} \leftarrow U_3 - P_4 \\ U_3 \leftarrow U_2 + P_7 & C_{22} \leftarrow U_3 + P_5 \\ U_4 \leftarrow U_2 + P_5 & \end{array}$$

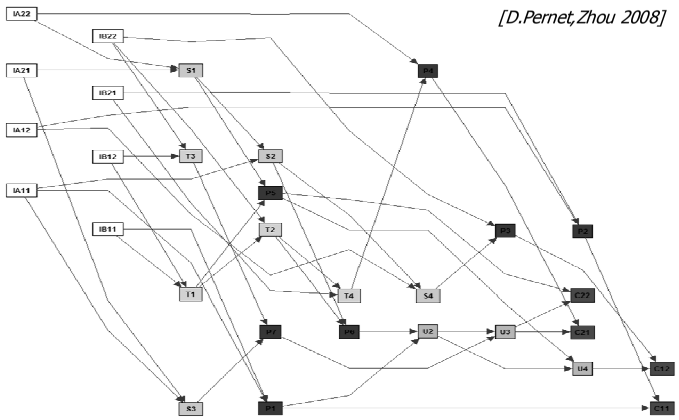
Ordonnancement et Placement : C = A×B



- Deux blocs mémoire temporaires supplémentaires : 2/3 n^2
- Optimal par jeux des galets *[Douglas & al. 1994]*

C=A×B en place : **1. écraser des entrées**

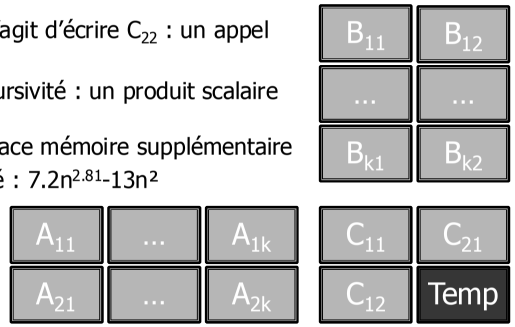
[D.Pernet,Zhou 2008]



Placements en place : **2. Compromis temps/mémoire**

C = A × B

- Presque toutes les étapes avec C_{22} comme bloc temporaire en utilisant l'algorithme de Winograd
- Quand il s'agit d'écrire C_{22} : un appel récursif
- Fin de récursivité : un produit scalaire
- Aucun espace mémoire supplémentaire
- Complexité : $7.2n^{2.81}-13n^2$



Produit avec accumulation : C += A×B

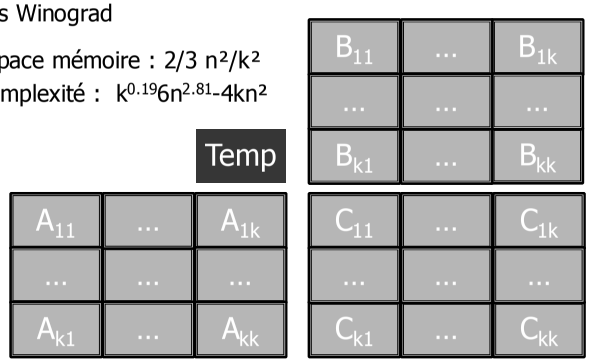
- *[Huss-Ledermann & al. 1996]* : 3 blocs supplémentaires, prouvé optimal par jeux des galets ; 4 appels récursifs et 3 appels A × B
  - Complexité : $6n^{2.81}-4n^2$ opérations, n^2 mémoire
- *[D., Pernet, Zhou 2008]* : Réduit la consommation mémoire à 2 blocs
  - 2 pré additions, compensées par 6 appels récursifs et 1 appel A × B
  - Complexité : $6n^{2.81}-4n^2+n^{2.58}-n^2$ opérations, 2/3 n^2 mémoire

#	operation	loc.	#	operation	loc.
1	$C_{22} = C_{22} - C_{12}$	C_{22}	12	$P_3 = \alpha S_4 B_{22} + C_{12}$	C_{12}
2	$C_{12} = C_{12} - C_{22}$	C_{12}	13	$P_1 = \alpha A_{11} B_{11}$	X_1
3	$S_1 = A_{21} + A_{22}$	X_1	14	$U_2 = P_6 + P_1$	C_{21}
4	$T_1 = B_{12} - B_{11}$	X_2	15	$P_2 = \alpha A_{12} B_{21} + \beta C_{11}$	C_{11}
5	$P_5 = \alpha S_1 T_1 + \beta C_{12}$	C_{12}	16	$U_1 = P_1 + P_2$	C_{11}
6	$S_2 = S_1 - A_{11}$	X_1	17	$U_5 = U_2 + C_{12}$	C_{12}
7	$T_2 = B_{22} - T_1$	X_2	18	$S_3 = A_{11} - A_{21}$	X_1
8	$P_6 = \alpha S_2 T_2 + \beta C_{21}$	C_{21}	19	$T_3 = B_{22} - B_{12}$	X_2
9	$S_4 = A_{12} - S_2$	X_1	20	$U_3 = P_7 + U_2 = \alpha S_3 T_3 + U_2$	C_{21}
10	$T_4 = T_2 - B_{21}$	X_2	21	$U_7 = U_3 + C_{22}$	C_{22}
11	$C_{22} = P_5 + \beta C_{22}$	C_{22}	22	$U_6 = U_3 - P_4 = -\alpha A_{22} T_4 + U_3$	C_{21}

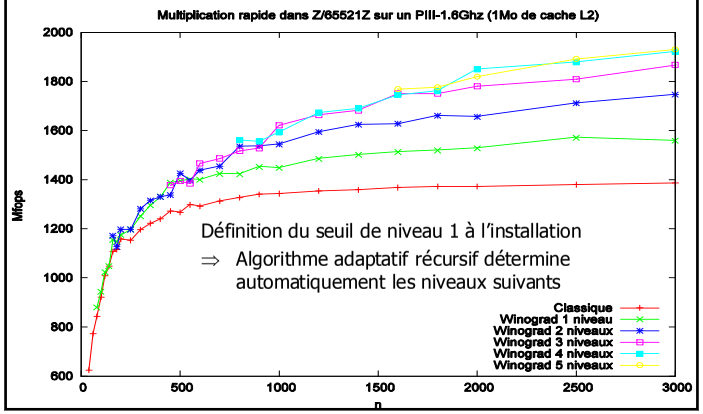
Placements en place : **2. Compromis temps/mémoire**

C += A × B

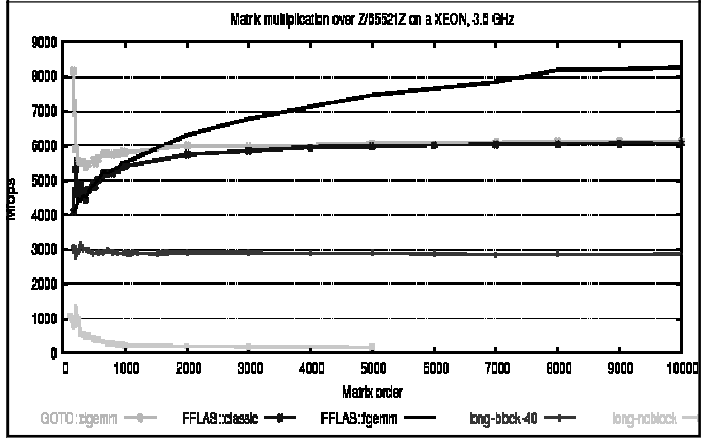
- Comme pour C=A×B : un premier niveau classique appelle des Winograd
- Espace mémoire : 2/3 n^2/k^2
- Complexité : $k^{0.19}6n^{2.81}-4kn^2$



Multiplication avec Strassen-Winograd : gain de 38% sur un PIII 1.6 GHz



FFLAS : 8.2 Gffops sur un XEON 3.6 GHz



Plan

1. Algèbre linéaire exacte
2. FFLAS-FFPACK
3. Gestion de la mémoire dans l'algorithme de Winograd
4. Utilisation de tables pour l'arithmétique modulaire
   - DQT : Transformée q-adique
   - REDQ : multiplication/réduction de polynômes modulo
   - FQT : compromis temps-mémoire pour les extensions

Arithmétique compressée

- Modulo 2 : utiliser les bits
- Dans GF(5^3) ? → utiliser 7 bits par élément
- Plus généralement dans Z/5Z[X] utiliser au moins 3 bits par coefficient
  - En utiliser un peu plus permet de retarder la division par p
  - *[Saunders'01]* en utiliser encore un peu plus permet de faire n AXPY tout en restant exact et donc de faire un seul produit scalaire pour tous les coefficients qui tiennent dans la mantisse en une seule fois
  - *[D,G,P'02]* Application aux corps finis : convertir les éléments de GF(pk) en q-adique avec q une puissance de nombre premier suffisamment grande.

Transformée Q-adique (DQT)

- Comment utiliser les BLAS numériques dans une extension de corps fini ?
  - Éléments représentés par soit :
    - Polynôme à coefficients modulo la caractéristique
    - Exposant d'un générateur du groupe des inversibles
- Changer la représentation → remplacer l'indéterminée par un entier q suffisamment grand :
  - $X^4+2X^3+3X^2+4X+5$ modulo 7
  - $64^4+2.64^3+3.64^2+4.64+5 = 17314053$
  - $100^4+2.100^3+3.100^2+4.100+5=102030405$
- Les calculs polynomiaux deviennent des calculs entiers faisables numériquement.

REDQ

- Comment faire k réductions modulaires simultanées ?

Réduction flottante [Shoup] :
  - $r = r - \lfloor r/p \rfloor \times p$

⇒ Sur le format compressé DQT :
  - Calculer 1 seule division : $d = \lfloor r/p \rfloor$
  - Décaler/multiplier k fois : $u_i = r/q^i - \lfloor d/q^i \rfloor \times p$
  - Rajuster k fois : $r_i = (u_i - qu_{i+1}) \bmod p$

REDQ rapide : compromis temps/mémoire

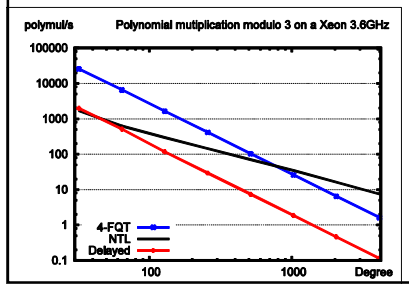
- Le rajustement est lent
- mais tous les ui sont plus petits que p

⇒ Tabulation du rajustement : r = T u

Exemple pour $GF(p^k)$:
- Arithmétique avec logarithme de Zech (chaque élément est représenté par son exposant par rapport à un générateur du groupe des inversibles) nécessite déjà des tables en $O(p^k)$
- Multiplication DQT + REDQ tabulée == FQT
  - Polynômes de degré k-1, produit de degré 2k-2
  - Rajustement avec deux tables de taille également p^k
- qadic.mws

Multiplication de polynômes modulaires

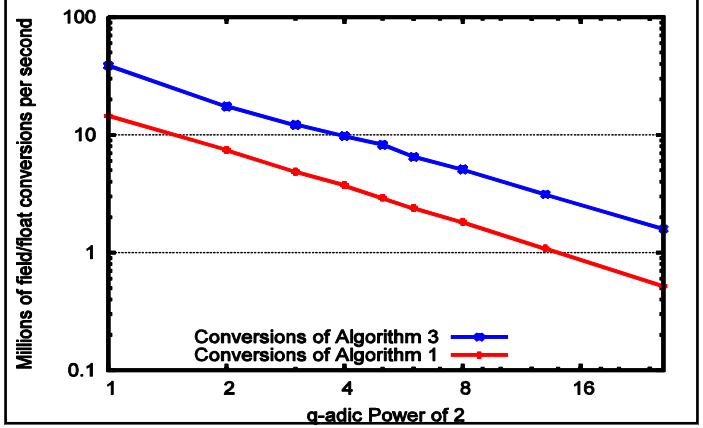
- De manière générale, si la taille du modulo est plus petite que la mantisse disponible :
  - Représentation compressée



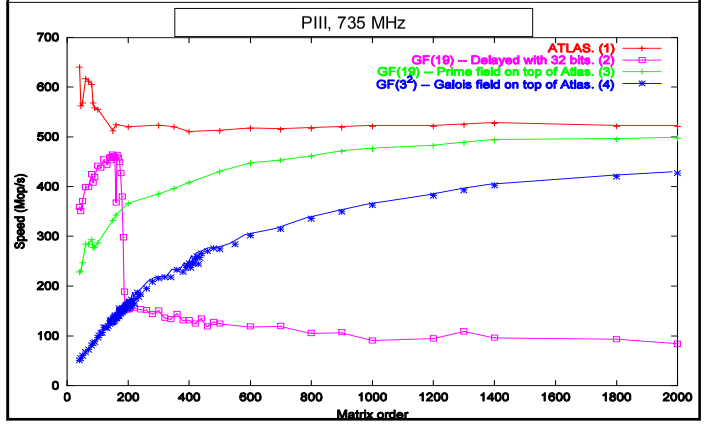
$X^5+2X^4+3X^3$ | $4X^2+5X+6$

→ Calculs regroupés
→ Accélération de la multiplication de base

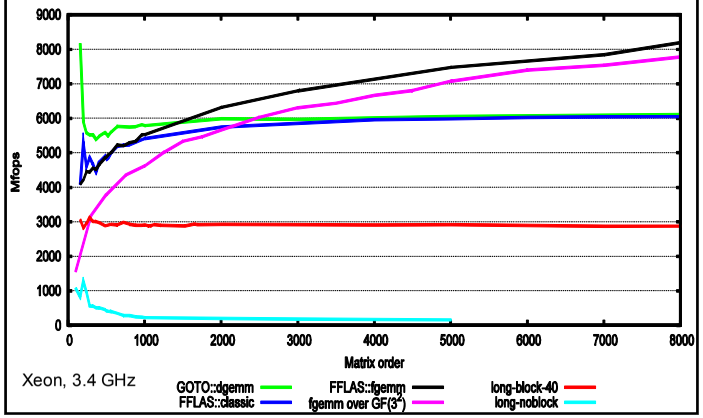
FQT vs conversions retardées



FFLAS en 2002



fgemm aujourd'hui
Hz x 4.6 ; BLAS x 12 ; GF(9) x 19.2



Perspectives

- Améliorations
  - Petites dimensions
  - Autres découpes récursives, multiplications rapides de matrices m × n
- Extensions
  - Corps finis plus grands : plus grandes mantisses ?
  - Parallélisme : pBLAS ; plus haut niveau → adaptive pattern ?
  - Sur Z : résolution de système, déterminant, échelon *[LinBox]* MM ?
  - Sur Q ?