GPGPU pour la cryptographie (et la sécurité)

Antoine Joux

SSTIC, 5 Juin 2009

Utilisation comme accélérateur cryptographique

- Cryptographie à clef secrète
  - CryptoGraphics (Cook et al. 2006): Implementation (lente) en OpenGl de l'AES
  - CHES 2007 (Harrison, Waldron): Implementation AES
  - Asiacrypt 2007 (Yang, Goodman): Implementation DES et AES en DirectX et CTM, gain GPU/CPU jusqu'à 16 pour AES, 60 pour le DES
  - USENIX 2008 (Harrison, Waldron):
- Cryptographie à clef publique
  - Cryptography and Coding (Moss, Page, Smart 2007): RSA en OpenGL
  - CHES 2008 (Szerwinski, Güneysu): RSA, DSA et ECC en CUDA: gain de 20-30% pour RSA/DSA

Applications à la cryptanalyse

Généralement recherche exhaustive:

- Application “embarrasingly parallel”
- Peu de transfert de données en CPU et GPU
- Exemple: factorisation par la méthode ECM (Berstein et al. 2009)
  - Factorisation de nombres de taille moyenne
  - Utilisation des courbes elliptiques (forme d’Edwards)

Algorithmique appliquée à la cryptographie

Deux exemples:

- Algèbre linéaire (multiplication de matrices sur $\mathbb{F}_2$)
- (Recherche de multicollisions)

Algèbre linéaire en cryptographie

- Sur corps fini (souvent $\mathbb{F}_2$)
- Algèbre lineaire avec matrices pleines ou creuses
- Beaucoup d'applications complexes:
  - Calcul de base de Gröbner
  - Réduction de réseau
  - Étape de factorisation/log.discret

Un problème exploratoire simple

- Multiplication de matrices sur $\mathbb{F}_2$
- Avec des mots de 32 bits, matrices $32n \times 32n$
- Code élémentaire pour matrice 32×32 ($\texttt{DIM} = 32$):

```
void Mul(int res[DIM][DIM],
 int mat1[DIM][DIM], int mat2[DIM][DIM])
{
  int l,c,k;
  for (l=0;l<DIM;l++) {
    for (c=0;c<DIM;c++) {
      res[l][c]=0;
      for (k=0;k<DIM;k++) {
        res[l][c]+=mat1[l][k]*mat2[k][c];
      }
      res[l][c]%=2; } } }
```

Représentation efficace en mémoire

- Compacter 32 bits par entier:

```
#define WORD unsigned int
#define bit(M,l,c) ((M[l]>>c)&1)
#define flipbit(M,l,c) if (1) {M[l]^=(1UL<<c);}

void Mul(WORD res[DIM],WORD mat1[DIM],
         WORD mat2[DIM])
{ int l,c,k,val;
  for (l=0;l<DIM;l++) {
    res[l]=0;
    for (c=0;c<DIM;c++) {
      val=0;
      for (k=0;k<DIM;k++) {
        val^=bit(mat1,l,k)&bit(mat2,k,c); }
      if (val) flipbit(res,l,c); } } }
```

Accélération sur CPU

- Exploiter le parallélisme interne (32 opérations/mot)
- Facile pour les multiplications des produits scalaires
- Plus dur pour l'extraction des colonnes:
  - Transposition rapide
- Et pour les sommes:
  - Repliement successifs (5/mot)
  - Partage des repliements (2/mot)
- Timings (gcc 4.3.3, 10^5 mult.):

basic	7.6 s
par bits	13.2 s
optim	0.12 s

Intégration dans grande multiplication

- Produit par bloc
- Légère optimisation possible (partage des transposées)
- Timings (gcc 4.3.3):

Dim	Normal	Optim	gcc 4.2
4096	2.4s	2.3s	6.9 s
6144	7.8s	7.4s	23.7 s
8192	18.6s	17.5s	57.4 s

Portage sur GPU en CUDA

- Transposée complexe $\Rightarrow$ pas assez de mémoire locale (utilisation registres)
- Génération automatique du code
- Partage des repliements très difficile
- Resultats sur Carte Tesla

Dim	GPU	CPU (gcc 4.1.2)
4096	0.17s	4.7s
6144	0.59s	15.8s
8192	1.38s	37.7s

- Sur Mac (moins de registres):

Dim	Normal	Optim	gcc 4.2	GPU
4096	2.4s	2.3s	6.9 s	4.5 s
6144	7.8s	7.4s	23.7 s	13.8 s
8192	18.6s	17.5s	57.4 s	$--$

Mais . . .

- Bien que 32×32 optimisée $\Rightarrow$ Peut mieux faire sur $32n \times 32n$
- Gains possibles:
  - Méthode des 4 Russes (compromis temps-mémoire)

4096	$0.12s$
6144	$0.41s$
8192	$0.88s$
16384	$6.3s$

  - Utilisation de Strassen
- Sur CUDA ?

Une autre application cryptographique: la recherche de multicollisions

- Une k-collision pour F est un k-uplet tel que:

$$F(a_1) = F(a_2) = \cdots = F(a_k)$$

- En général, dur à trouver, nécessite:

$$k!\, N^{(k-1)/k}$$

évaluations de F.

Algorithme simple pour les k-collisions

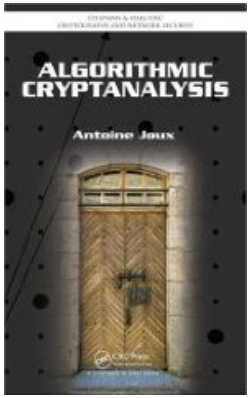
- Évaluer F en $k!\, N^{(k-1)/k}$ points aléatoires
- Trier
- Recherche k égalité successives (non-triviales)

Très couteux en termes de mémoire.
Pas vraiment parallélisable.

Quid de la sécurité ?

- Cracker de mots de passe . . .
- Reconnaissance de signatures de virus *(Vasiliadis et al.)*

- Hôte pour virus et autres parasites ?



Conclusion