High Performance Computing

Application à la cryptographie

RAPPORT

08 juin 2009

Olivier Jais-Nielsen

High Performance Computing

Application à la cryptographie

Algorithme initial

Cryptage

On parallélise un chiffreur auto synchronisant par inclusion (message embedded system) obéissant au système linéaire sur le tore de dimension n suivant :

$$\begin{cases} x_{k+1} = A.x_k + B.m_k \ \ mod(m) \\ y_k = C.x_k + D.m_k \ \ mod(m) \end{cases}$$

En pratique, on prend comme dimension n = 3.

On choisit comme clé $\theta = \begin{pmatrix} 1 \\ 7 \\ 4 \end{pmatrix}$ et on définit ainsi : $A = \begin{pmatrix} \theta_0 & 1 & 0 \\ \theta_1 & 0 & 1 \\ \theta_2 & 0 & 0 \end{pmatrix}$ la matrice compagnon de la séquence θ ; $B = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $C = (1 \quad 0 \quad 0)$, D = 0 et m = 256.

(m_k) désigne l'image à crypter en tant que séquence.

Le programme effectue une boucle permettant de déterminer les valeurs de (x_k) et (y_k) puis sauvegarde l'image cryptée, c'est-à-dire la séauence (y_k).

Décryptage

Le système de décryptage correspondant au précédent système de cryptage est régit par le système suivant :

$$\begin{cases} \hat{x}_{k+r+1} = P.\hat{x}_{k+r} + B.(C.B)^{-1}.y_{k+r} \\ \hat{m}_{k+r} = -(C.B)^{-1}.C.A.\hat{x}_{k+r} + (C.B)^{-1}.y_{k+r} \end{cases}$$

Avec $P = \begin{pmatrix} 0 & 0 & 0 \\ \theta_1 & 0 & 1 \\ \theta_0 & 0 & 0 \end{pmatrix}$.

Comme précedemment, le programme effectue une boucle permettant de déterminer les valeurs de $(\hat{x}_k)$ et de $(\hat{m}_k)$, puis sauvegrader l'image décryptée, c'est-à-dire la séquence $(\hat{m}_k)$.

Algorithme Parallélisé

On pourrait envisager de paralléliser les opérations d'algèbre linéaire. Néanmoins, elle concerne des matrices de dimensions au maximum 3x3. La granularité serait donc très mauvaises.

Cryptage

On considère le système $\begin{cases} x_{k+1} = A.x_k + B.m_k \mod(m) \\ y_k = C.x_k + D.m_k \mod(m) \end{cases}$.

On remarque que ce système est de la forme : $\begin{cases} x_{k+1} = f(x_k, k) \\ y_k = g(x_k, k) \end{cases}$.

On peut en déduire :

- que la séquence (x_k) peut être calculé indépendamment de la séquence (y_k)
- Pour k donné, seuls les valeurs de x_k et de k sont nécessaires au calcul de y_k

On peut donc paralléliser le programme de la façon suivante : un processeur calcul les valeur de (x_k) et envoie, immédiatement après calcul chaque valeur au second processeur. Le second processeur reçoit chaque valeur, dans l'ordre d'envoie, donc des k croissants, et en déduit ainsi les valeurs de (y_k).

Décryptage

Le système de décryptage présente exactement les mêmes dépendances. On peut donc appliquer la même stratégie en prenant $(\hat{x}_k)$ au lieu de (x_k) et $(\hat{m}_k)$ au lieu de (y_k).

Implémentation

On utilise MPICH2 avec C++. Pour la gestion des images et l'algèbre linéaire, on utilise CImg (qui exclue donc la parallélisation de l'algèbre linéaire).

Etant donné que les séquences considérées représentent des images (cryptées ou non), on travail avec des indices bidimensionnelles (k_1, k_2) plutôt que k, grâce au changement de variable :

$$\begin{cases} k_1 = \lfloor k/w \rfloor \\ k_2 = k \bmod(w) \end{cases}$$ où w est la largeur de l'image.