Cahier des charges pour la réalisation d'un programme d'alignement de séquences

Jean-Michel Richer
L3 Informatique, Architecture des Ordinateurs

24 août 2009

1 Objectif

On vous demande d'écrire un programme en C et en assembleur de manière à réaliser l'alignement global de deux séquences d'ADN en utilisant un algorithme de programmation dynamique (Algorithme de Needleman et Wunsch).

2 Données et paramètres

L'algorithme de programmation dynamique consiste à calculer une matrice (un tableau à deux dimensions) dont chaque cellule comporte 4 valeurs : d, h, v, m. On représentera donc une cellule sous la forme :

```
typedef struct _Cell {
  float d, // diagonal
        h, // horizontal
        v, // vertical
        x; // maximum
} Cell;
```

La matrice est représentée sous la forme d'un tableau dynamique qu'il faudra allouer en fonction de la taille des séquences à aligner :

```
Cell **Matrix;
```

Le programme que vous devrez réaliser se nommera `align.exe` et comprendra 5 paramètres en entrée :

- n : un entier prenant la valeur 1 ou la valeur 2 : si $n = 1$ on exécutera la version C de l'algorithme 50000 fois, par contre si $n = 2$ on exécutera 50000 fois la version SSE que vous aurez implantée
- g : la pénalité liée à l'insertion d'un gap (gap) de type flottant simple précision, généralement négative
- s : la pénalité (ou score) de substitution (subst) de type flottant simple précision, généralement positive
- m : la pénalité (ou score) d'appariement (match) de type flottant simple précision, généralement positive
- f : un nom de fichier texte

Le fichier texte comprend les deux séquences à aligner avec la première séquence sur la première ligne et la seconde séquence sur la ligne suivante. Les séquences commencent par le caractère '-' qui est le caractère de gap, puis sont suivies par un nombre quelconque de A, C, G et T qui représentent les nucléotides qui composent une séquence d'ADN.

La première séquence sera stockée dans une chaîne S de longueur n, la seconde dans la chaîne T de longueur m. La longueur maximum des séquences est de 100 caractères.

```
int n, m;
char S[100];
char T[100];
```

Une fois les séquences lues, on allouera dynamiquement la matrice :

```
Matrix=(Cell **)calloc(n+1,sizeof(Cell *));
for (i=0;i<=n;++i) {
  Matrix[i]=(Cell *)calloc(m+1,sizeof(Cell));
}
```

Remarque : il est préférable d'allouer les données sur une adresse multiple de 16, notamment si on prévoit d'utiliser les instructions SSE.

3 Algorithme

Vous devrez alors implanter la fonction de remplissage de la matrice dont le code est donné ci-joint en utilisant les instructions SSE2.

```
// ============================================================
float MINI=-1000;

float weight(char a, char b) {
  return (a==b) ? match_penalty : subst_penalty;
}

void fill_matrix(Cell **M) {
  int i, j;

  M[0][0].d=0;
  M[0][0].h=0;
  M[0][0].v=0;
  M[0][0].x=0;

  // initialize first column
  for (i=1; i<=n; ++i) {
    M[i][0].d=MINI;
    M[i][0].h=MINI;
    M[i][0].v=M[i-1][0].v+gap_penalty;
    M[i][0].x=M[i][0].v;
  }

  // initialize first row
  for (j=1; j<=m; ++j) {
    M[0][j].d=MINI;
    M[0][j].h=M[0][j-1].h+gap_penalty;
    M[0][j].v=MINI;
    M[0][j].x=M[0][j].h;
  }

  // compute all cells from i=1..n, j=1..m
  for (i=1;i<=n;++i) {
    for (j=1;j<=m;++j) {
      float maxi;
      M[i][j].d=M[i-1][j-1].x + weight(S[i],T[j]);
      M[i][j].h=M[i][j-1].x   + gap_penalty;
      M[i][j].v=M[i-1][j].x   + gap_penalty;
      maxi=M[i][j].d;
```

```
      if (M[i][j].h>maxi) maxi=M[i][j].h;
      if (M[i][j].v>maxi) maxi=M[i][j].v;
      M[i][j].x=maxi;
    }
  }
}
```

4 Utilisation des registres SSE

On utilisera les registres SSE de la manière suivante : un registre SSE de 128 bits peut contenir 4 flottants en simple précision, la partie basse du registre sera occupée par d puis on trouvera h, v et x.

d	h	v	x
32 bits	32 bits	32 bits	32 bits

L'utilisation des registres SSE peut être réalisée pour les 3 traitements suivants :
- initialisation de la première ligne
- initialisation de la première colonne
- corps de la boucle (*compute all cells*)

4.1 initialisation de la première ligne

On peut utiliser par exemple $xmm0$ et $xmm1$ avec :
- $xmm0 = [MINI, 0, MINI, 0]$
- $xmm1 = [0, gap, 0, gap]$

puis ajouter $xmm1$ à $xmm0$ à chaque itération de la boucle j et stocker le résultat dans la cellule $M[0][j]$.

4.2 initialisation de la première colonne

On peut utiliser le même principe que précédemment.

4.3 corps de la boucle

On peut utiliser les registres $xmm0$, $xmm1$ et $xmm2$:
- $xmm0$ représente $M[i-1][j-1]$
- $xmm1$ représente $M[i-1][j]$
- $xmm2$ représente $M[i][j-1]$

on calcule ensuite les pénalités à ajouter :
- $xmm0 + xmm3$ où $xmm3 = [0, 0, 0, subst$ ou $match]$
- $xmm1 + xmm4$ où $xmm4 = [0, 0, 0, gap]$
- $xmm2 + xmm4$

puis à partir de $xmm0$, $xmm1$ et $xmm2$ grâce à des décalages, des ET logique et OU logique on obtient les valeurs qui composeront $M[i][j]$. Enfin, en utilisant l'instruction MAXPS et des décalages on peut obtenir le maximum des 3 valeurs (d, h, v).

4.4 instructions SSE

Les instructions que vous serez probablement amené à utiliser sont les suivantes (se référer aux guides INTEL, vous pouvez également utiliser d'autres instructions SSE) :
- **MOVD xmm,r32** déplace le contenu du registre 32 bits vers la partie basse de xmm
- **MOVD r32,xmm** déplace la partie basse de xmm vers le registre 32 bits
- **MOVAPS xmm,[adr32]** déplace les 128 bits pointés par l'adresse 32 bits vers le registre xmm

– **MOVAPS [adr32],xmm** déplace les 128 bits du registre xmm vers l'adresse 32 bits
– **PAND xmm1, xmm2** réalise un ET logique entre les deux registres xmm
– **POR xmm1, xmm2** réalise un OU logique entre les deux registres xmm
– **PSLLDQ xmm,imm8** : décalage à gauche du registre de imm8 octets
– **PSRLDQ xmm,imm8** : décalage à droite du registre de imm8 octets
– **MAXPS xmm1, xmm2** calcule et stocke la valeur maximale de chacune des 4 valeurs flottantes entre xmm1 et xmm2 dans le registre xmm1

5 Exemple

Un exemple des valeurs à obtenir est donné Table 1.

	-	A	T
-	$d = 0.00 \quad v = 0.00$ $h = 0.00 \quad x = 0.00$	$d = -5000.00 \quad v = -5000.00$ $h = -1.00 \quad x = -1.00$	$d = -5000.00 \quad v = -5000.00$ $h = -2.00 \quad x = -2.00$
A	$d = -5000.00 \quad v = -1.00$ $h = -5000.00 \quad x = -1.00$	$d = 4.00 \quad v = -2.00$ $h = -2.00 \quad x = 4.00$	$d = 1.00 \quad v = -3.00$ $h = 3.00 \quad x = 3.00$
C	$d = -5000.00 \quad v = -2.00$ $h = -5000.00 \quad x = -2.00$	$d = 1.00 \quad v = 3.00$ $h = -3.00 \quad x = 3.00$	$d = 6.00 \quad v = 2.00$ $h = 2.00 \quad x = 6.00$
G	$d = -5000.00 \quad v = -3.00$ $h = -5000.00 \quad x = -3.00$	$d = 0.00 \quad v = 2.00$ $h = -4.00 \quad x = 2.00$	$d = 5.00 \quad v = 5.00$ $h = 1.00 \quad x = 5.00$
T	$d = -5000.00 \quad v = -4.00$ $h = -5000.00 \quad x = -4.00$	$d = -1.00 \quad v = 1.00$ $h = -5.00 \quad x = 1.00$	$d = 6.00 \quad v = 4.00$ $h = 0.00 \quad x = 6.00$

TAB. 1 – alignement de S ="-ACGT" avec T ="-AT", $gap = -1$, $subst = 2$, $match = 4$

6 Tests

Pour tester l'efficacité de votre implantation par rapport à la version C vous exécuterez les commandes suivantes :

```
time ./align.exe 1 -1 2 4 sequences1.txt
time ./align.exe 2 -1 2 4 sequences1.txt
```

où `sequences1.txt` est un fichier de référence avec 2 séquences (à récupérer sur mon site web).

7 Points divers

1. le programme doit bien évidemment être écrit en partie en C et en assembler x86 + SSE2 (pour le sous-programme de remplissage de la matrice),
2. vous devrez travailler seul ou en binôme,
3. le programme est à rendre pour le 18 Décembre 2009 dernier délai en m'envoyant un mail contenant l'ensemble des sources sous forme d'un fichier archive de type TAR. Vous n'oublierez pas d'indiquer vos nom et prénom dans un fichier readme.txt.
4. vous fournirez un script `compile.sh` qui permet d'obtenir l'exécutable

`jean-michel.richer@info.univ-angers.fr`

8 Notation

Les points suivants seront pris en compte lors de la notation :
- lisibilité du code
- présence de commentaires
- utilisation des instructions SSE
- erreurs à la compilation
- fonctionnement global du programme
- efficacité de la version SSE par rapport à la version C compilée avec l'option `-O2`

May the force of the Core 2 Quad be with you !