

Master professionnel Informatique

Optimisations pour graphique et multimédia

Examen Mars 2005
1H ; tous documents autorisés

Question 1

Pour les boucles suivantes, indiquer celles qui sont peuvent utiliser les instructions SIMD (« vectorisables ») et celles qui ne le peuvent pas en précisant brièvement pourquoi.

```
int i, j ;  
float X[N], Y[N] ;
```

Boucle 1

```
for (i=0 ; i<N ; i++)  
    for (j=0 ; j<N ; j++)  
        Y[i][j] = X[i][j] + X[i][j-1];
```

Boucle 2

```
for (i=0 ; i<N ; i++)  
    for (j=0 ; j<N ; j++)  
        Y[i][j] = Y[i][j] + Y[i][j-1];
```

Boucle 3

```
for (i=0 ; i<N ; i++)  
    for (j=0 ; j<N ; j++)  
        Y[j][i] = Y[j][i] + X[j-1][i];
```

Boucle 4

```
for (i=0 ; i<N ; i++)  
    for (j=N-1 ; j>=0 ; j--)  
        Y[i][j] = Y[i][j] + Y[i][j-1] ;
```

Question 2

Transformer la boucle suivante pour la rendre vectorisable.

```
float A[400], B[400], Z;  
*p_a = &A[0] ;  
*p_b = &B[0] ;  
*p_z = &Z ;  
int f ;  
  
for (f=0;f<400;f++)  
    *p_z += *p_a++ * *p_b++ ;
```

Question 3

Le jeu d'instruction IA-32 contient des instructions SIMD de conversion d'entiers 32 bits (int) en flottants 32 bits simple précision (float) : CVTQ2PS .

On considère des variables 128 bits (`_mm128i`) `v0`, `v1`, `v2`, `v3`, `v4`, `v5`, `v6`, `v7` pour les entiers et des variables 128 bits (`_mm128`) `f0`, `f1`, `f2` et `f3` pour les flottants simple précision.

En supposant que la variable `v0` contient 16 octets (`unsigned char`) correspondant à des pixels (niveaux de gris), donner la suite des instructions SIMD nécessaires pour convertir les 16 données 8 bits en 16 données de type `float` dans les variables `v4` à `v7`.

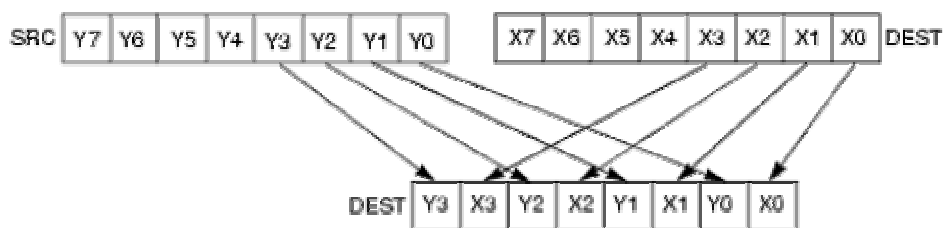
Quelle opération faut-il alors effectuer pour avoir des données flottantes comprises entre 0 et 1 ?

On pourra fournir la réponse sous forme d'un programme avec des intrinsics, soit sous forme de schémas explicitant le travail de chacune des instructions successives.

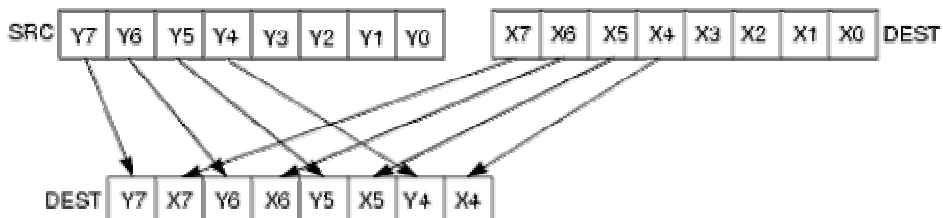
Quel est le nombre total d'instructions nécessaires pour faire la conversion ?

ANNEXE : Instructions SIMD IA-32 utilisables

CVTDQ2PS	<code>_mm_cvtepi32_ps (a)</code>	Convertit 4 entiers 32 bits signés en 32 bits flottants
DIVPS	<code>_mm_div_ps (a,b)</code>	Quatre divisions flottantes 32 bits
MOVDQA	<code>_mm_load_si128 (*p)</code>	Chargement aligné de 128 bits
MOVDQA	<code>_mm_store_si128 (*p,a)</code>	Rangement aligné de 128 bits
MULPS	<code>_mm_mul_ps (a, b)</code>	Quatre multiplications flottantes 32 bits
PACKUSWB	<code>_mm_packs_epu16 (m1, m2)</code>	Compacte avec saturation shorts en octets non signés
PMAXUB	<code>_mm_max_epu8 (a, b)</code>	Max des octets non signés source et destination
PMINUB	<code>_mm_min_epu8 (a, b)</code>	Min des octets non signés source et destination
POR	<code>_mm_or_si128 (a, b)</code>	Ou logique parallèle
PSRLDQ	<code>_mm_srli_si128 (a, imm)</code>	Décalage logique gauche de « imm » octets
PSSLDQ	<code>_mm_slli_si128 (a, imm)</code>	Décalage logique droite de « imm » octets
PUNPCKHBW	<code>_mm_unpacklo_epi8 (m1, m2)</code>	Entrelace les octets (haut) de la destination et la source
PUNPCKHWD	<code>_mm_unpacklo_epi16 (m1, m2)</code>	Entrelace les shorts (haut) de la destination et la source
PUNPCKLBW	<code>_mm_unpacklo_epi8 (m1, m2)</code>	Entrelace les octets (bas) de la destination et la source
PUNPCKLWD	<code>_mm_unpacklo_epi16 (m1, m2)</code>	Entrelace les shorts (bas) de la destination et la source
PXOR	<code>_mm_xor_si128 (a, b)</code>	Ou exclusif parallèle



PUNPCKLBW



PUNPCKHBW