TD/TP8 : Mémoire cache

Nga Nguyen, EISTI

Exo 1 : Localité

Classez les trois fonctions suivantes en fonction de leur localité spatiale.

```
#define N 1000
typedef struct {
        int vel[3];
        int acc[3];
} point;
point p[N];

void clear1(point *p, int n) {
        int i, j;
        for (i = 0; i < n; i++) {
                for (j = 0; j < 3; j++)
                        p[i].vel[j] = 0;
                for (j = 0; j < 3; j++)
                        p[i].acc[j] = 0;
        }
}

void clear2(point *p, int n) {
        int i, j;
        for (i = 0; i < n; i++) {
                for (j = 0; j < 3; j++) {
                        p[i].vel[j] = 0;
                        p[i].acc[j] = 0;
                }
        }
}

void clear3(point *p, int n) {
        int i, j;
        for (j = 0; j < 3; j++) {
                for (i = 0; i < n; i++)
                        p[i].vel[j] = 0;
                for (i = 0; i < n; i++)
                        p[i].acc[j] = 0;
        }
}
```

Exo 2 : Cache

Supposons que nous avons un système avec les propriétés suivantes :

- Mots d'un octet
- Adresses de 12 bits (m = 12)
- Cache de type "2-way set associative" (E = 2) avec des blocs de 4 octets (B = 4) et 4 ensembles (S = 4)

Le contenu du cache est le suivant (notation hexadécimale) :

Set index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0				
2	00	1	48	49	4A	4B
	40	0				
3	FF	1	9A	C0	03	FF
	00	0				

1. Indiquez sur les 12 bits d'adresse, quels bits sont utilisés pour block offset, set index et tag.
2. Indiquez dans la séquence suivante s'il s'agit d'un hit ou miss, ainsi que la valeur récupérée.

Opération	Adresse	Hit ?	Valeur
Read	0x834		
Write	0x836		
Read	OxFFD		

Exo 3 : Analyse de miss rate

Vous écrivez un jeu 3D qui va vous ramener beaucoup de succès et fortune. Vous travaillez actuellement sur une fonction pour effacer l'écran avant de dessiner le frame suivant. L'écran est un tableau de 480 x 640 pixels. L'ordinateur que vous avez a un « direct-mapped cache » de 64 KB, avec des lignes de 4 octets. Voici les déclarations des variables :

```
struct pixel {
        char r;
        char g;
        char b;
        char a;
};
struct pixel buffer[480][640];
register int i, j;
register char *cptr;
register int *iptr;
```

Assumons que :

- sizeof(char) = 1
- sizeof(int) = 4
- le buffer commence à l'adresse mémoire 0
- le cache est initialement vide
- les seuls accès mémoires sont ceux au tableau buffer. Les variables i, j, cptr, et iptr sont stockées en registre.

A. Quel est le pourcentage des écritures qui provoquent du "cache miss" dans le code suivant :

```
for (j=0; j < 640; j++) {
        for (i=0; i < 480; i++){
                buffer[i][j].r = 0;
                buffer[i][j].g = 0;
                buffer[i][j].b = 0;
                buffer[i][j].a = 0;
        }
}
```

B. Quel est le pourcentage des écritures qui provoquent du "cache miss" dans le code suivant :

```
char *cptr;
cptr = (char *) buffer;
for (; cptr < (((char *) buffer) + 640 * 480 * 4); cptr++)
        *cptr = 0;
```

C. Quel est le pourcentage des écritures qui provoquent du "cache miss" dans le code suivant :

```
int *iptr;
iptr = (int *) buffer;
for (; iptr < (buffer + 640 * 480); iptr++)
        *iptr = 0;
```

D. Quel code (A, B, ou C) est le plus rapide ?