M1 SESI 2017-2018
Architecture Multi-Processeurs
TP4 : Partage du Bus dans les architectures multi-processeurs

Kevin Mambu

March 26, 2018

Spécifications

Objectifs :

- Problèmes de performances posés par le partage du bus.
- Goulot d'étranglement lorsque plusieurs processeurs veulent accéder au bus car la bande passante du bus est bornée.
- Temps d'attente du bus proportionnel au nombre de processeurs dans l'architecture.

B) Architecture Matérielle

- Chaque processeur a un TTY attribué
  → N TTYs pour un contrôleur de TTY.
- $|seg\_tty| = NPROC \times 16$. Utilisation d'un framebuffer (256×256) sur 256 niveaux de gris.
- Spécifications du framebuffer
  - Un tampon de luminence, 64Ko
  - Un tampon de chrominance, 64Ko, pas utilisé ici
  - Fréquence de la lecture des tampons : 25 images par seconde
  - Fréquence d'affichage : $f = \frac{1}{40 \times 10^{-3}} = 250Hz$

Question B1

- name : nom de l'instance
- tgtid : index du signal du framebuffer vis-à-vis du PIBUS
- segtab : pointeur vers la table des segments
- latency : temps d'accès au framebuffer
- width : largeur de l'image (en pixels)
- height : hauteur de l'image (en pixels)
- subsampling : fréquence de sous'échantillonage de la chrominance (ici non-utilisé)

Question B2

Le segment associé au Framebuffer doit être aligné et sa taille doit être de $width \times height$ octets (ici $256 \times 256 = 64$Ko).

Dimensionnement des caches

- Caches Write-Through
- Correspondance directe
- 16 lignes de 8 mots

```
#define NPROCS      1 // number of processors
#define FB_NPIXEL  256 // Frame buffer width
#define FB_NLINE   256 // Frame buffer heigth
#define BLOCK_SIZE   512 // IOC block size
#define IOC_LATENCY 1000  // disk latency
#define RAM_LATENCY 0 // ram latency
#define ICACHE_WAYS 1        // instruction cache number of ways
#define ICACHE_SETS 16      // instruction cache number of sets
#define ICACHE_WORDS  8        // instruction cache number of words per line
#define DCACHE_WAYS 1        // data cache number of ways
#define DCACHE_SETS 16      // data cache number of sets
#define DCACHE_WORDS  8        // data cache number of words per line
#define WBUF_DEPTH  8        // cache write buffer depth
#define SNOOP    false // cache snoop activation
#define DMA_BURST 16  // number of words in a DMA burst
```

tp5_top.cpp

C) Compilation de l'application logicielle

Question C1

Le segment associé au contrôleur du Framebuffer fait partie de l'espace privilégié. Tout accès mémoire de l'application utilisateur avec le Framebuffer doit passer par un appel système pour être légal. À cause d'un manque de permissions de la part de l'application utilisateur, un accès direct avec le Framebuffer se traduirait par un Data Bus Error.

Question C2

Écrire ligne par ligne limite le nombre de transactions sur le PIBUS à une seule transaction rafale, une fois le registre intermédiaire plein. Plutôt qu'une transaction par octet, ce qui est plus coûteux.

Question C3

*unsigned int fb_sync_write(unsigned int offset, void *buffer,unsigned int length)*

- offset : le déplacement nécessaire où écrire dans le tampon du Framebuffer.
  *note : sachant qu'on écrit sur le Framebuffer ligne par ligne, l'offset doit être aligné (multiple de 256).
- buffer : addresse de base du tampon intermédiaire
- : longeur en octets de la donnée à écrire sur le framebuffer.

D) Caractérisation de l'application logicielle

Question D1

Execution à un processeur :

NB_CYCLES	NB_INST	CPI
5697418	4010805	1.42052

Question D2

WRITE_RATE	READ_RATE	IMISS_RATE	DMISS_RATE
0.142853	0.267784	0.008941	0.009121
IMISS_COST	DMISS_COST	WRITE_COST	
16.0537	14.3444	0.0000	

Les coûts sont des valeurs non-entières parce qu'elles sont moyennées.

Question D3

Nombres de transacations :

IMISS	DMISS	UNC	WRITE
35860	36582	5266	572955

Si le nombre de transactions de lectures est relativement faible, le nombre de transactions d'écritures lui est bien plus conséquent. Cela doit être dû au fait que les écritures sont prioritaires.

E) Exécution sur architecture multi-processeur

Question E1

```
__attribute__ ((constructor)) void main()
{
    unsigned char   buf[NPIXEL];
    int     n = procid();
    int     nprocs = NB_PROCS;
    unsigned int  line;
    unsigned int  pixel;

    for(line = n ; line < NLINE ; line = line + nprocs)
    {
        for(pixel = 0 ; pixel < NPIXEL ; pixel += 1)
        {
            buf[pixel] = build(pixel, line, 5);
        }
        if (fb_sync_write(256 * line, buf, NPIXEL))
            tty_printf(" !!! wrong transfer to frame buffer for line %d\n", line);
        else
            tty_printf(" - building line %d\n", line);
    }

    tty_printf("\ncycles = %d\n", proctime() );
    exit();

} // end main
```

main.c

Question E2

```
#############################################################################
# File : reset.s
# Author : Alain Greiner
# Date : 15/12/2011
#############################################################################
# - It initializes the Status Register (SR)
# - It defines the stack size  and initializes the stack pointer ($29)
# - It initializes the EPC register, and jumps to user code.
#############################################################################

   .section .reset,"ax",@progbits

   .extern seg_stack_base
   .extern seg_data_base

   .func reset
   .type   reset, %function

reset:
        .set noreorder

# initializes stack pointer
   la   $29,  seg_stack_base
   li   $8, 0x10000     # stack size = 64Kbytes
   mfc0  $27,  $15,  1
   addiu $27,  $27,  1
   mult  $27,  $8
   mflo  $8
   addu  $29,  $29,  $8    # seg_stack_base += 64K

# initializes SR register
        li   $26,  0x0000FF13
        mtc0  $26,  $12     # SR <= 0x0000FF13

# jump to main in user mode
   la   $26,  seg_data_base
        lw   $26,  0($26)      # get the user code entry point
   mtc0  $26,  $14     # write it in EPC register
   eret

   .set reorder

   .endfunc
   .size reset, .-reset
```

reset.s

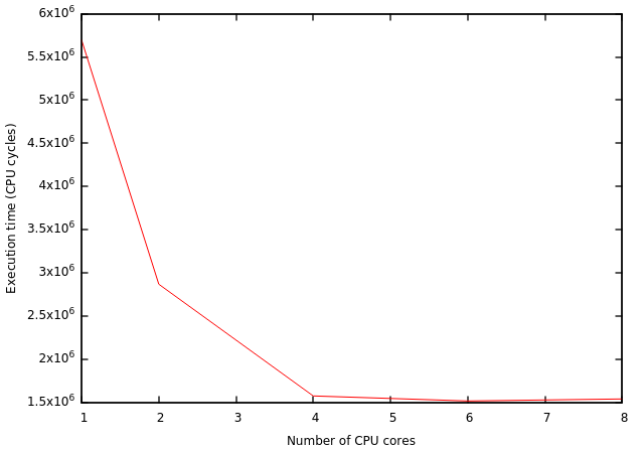
Question E3

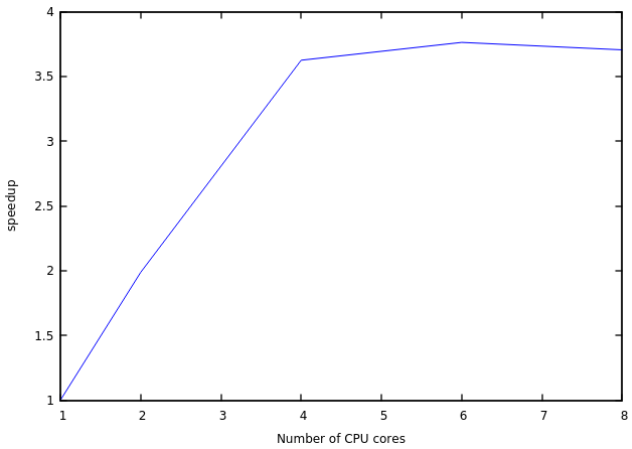
Le fichier config.h spécifie la constante `NB_PROC`, le nombre de processeurs. De ce fait, à chaque fois que l'on modifie le nombre de processeurs il faut re-génerer le binaire du programme.

Question E4

```
# nbprocs speedup nbcycles
  1      1.000 5697419
  2      1.987   2867065
  4      3.625   1571860
  6      3.763   1514141
  8      3.705   1537767
```

QuestionE4.plot





Posons l'équation suivante :

$$\begin{aligned} speedup &= \frac{execution\_time(1)}{execution\_time(N)} \\ &= \frac{execution\_time(1)}{N \times execution\_time(\frac{1}{N})} \end{aligned} \tag{1}$$

Théoriquement, plus le nombre de processeurs augmente, plus le temps d'execution par CPU diminue en fonction du nombre de processeurs, car ces derniers se partagent la charge de l'exécution d'instructions et par extension du nombre de MISS. Mais cette formule ne prend pas en compte la bande passante maximale du BUS, et on peut le constater sur l'évolution des courbes ci-dessus : à partir de 4 coeurs, le speedup ralentit considérablement et passé 6 coeurs le speedup diminue. Parce que le bus est devenu un goulôt d'étranglement.

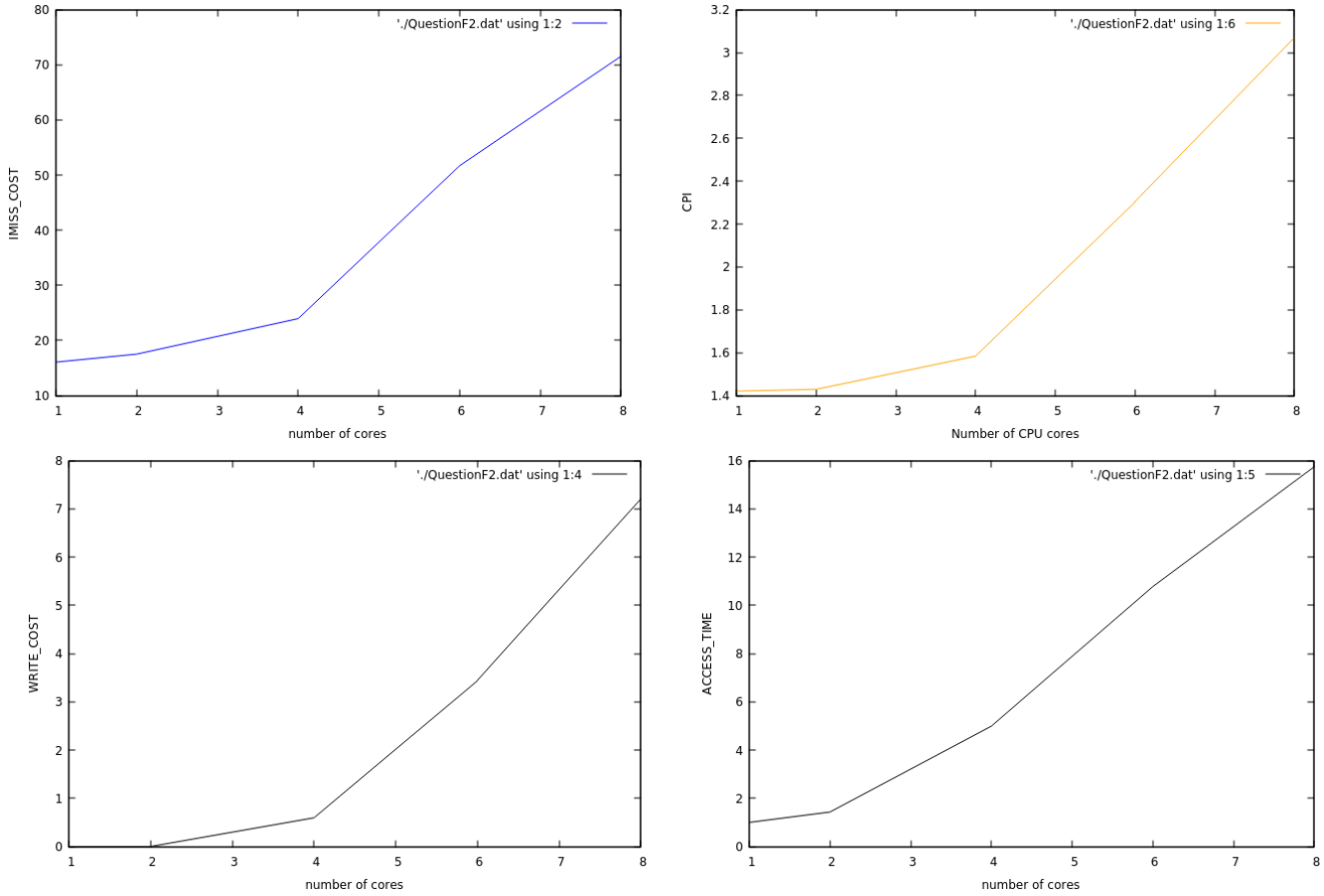
Question F1

Au delà de l'exécution du programme, d'autres instructions continuent à être exécutées et les comptabiliser fausserait notre instrumentation.

Quesion F2

```
# QuestionF2.dat
# NPROCS   IMISS\_COST    DMISS\_COST  WRITE\_COST  ACCESS\_TIME  CPI
   1  16.0541    14.3447    0           1          1.42061
   2  17.5069    16.1298    0           1.42839    1.42908
   4  23.9456    21.1079    0.595562    4.99196    1.58367
   6  51.6789    41.5059    3.43646     10.7797    2.3027
   8  71.5392    58.0357    7.19707     15.7462    3.06756
```

QuestionF2.dat



Question F3

La dégradation de la valeur du CPI est liée à l'augmentation du coût des MISS mais également à l'augmentation du coût des écritures : en effet, parce que les écritures sont prioritaires, les processeurs doivent supporter lors de leur attente la congestion passive du bus ainsi qu'un supplément dû aux tampons d'écritures postées.

Question G1

IMISS	DMISS	UNC	WRITE
9	9	2	2

Question G2

Soit F la fréquence, n le nombre d'instructions de la transaction et N le temps total d'execution :

$$F = \frac{n}{N}$$

	Temps_occupation	Fréquence
IMISS	9	0.006294
DMISS	9	0.006245
UNC	2	0.000924
WRITE	2	0.100564

Question G3

$$BP = \sum Temps\_occupation \times F$$

À partir de cette formule, on peut démontrer que la bande passante utilisée par un processeur est $BP = 0.315827$, soit 31.58%, théoriquement 3.1666 coeurs est la limite supportée par le PIBUS avant saturation.

Bonus : `atomic_increment`

```
# The caller should loop on this
atomic_increment :
  ll r2,  0(r4)
  add r2, r2, r5
  sc  r2, 0(r4)
  jr r31

# With internal loop
atomic_increment2 :
  ll r2,  0   (r4)
  add r2, r2, r5
  sc  r2, 0   (r4)
  beq r2, r0, atomic_increment
  jr r31
```

atomic_increment.s