Université Grenoble Alpes, Grenoble INP, UFR IM^{2}AG

Master 1 Informatique and Master 1 MOSIG

UE Parallel Algorithms and Programming

TD # 2

Frédéric Desprez (Inria), Jean-Francois Méhaut (UGA/Polytech), Thomas Ropars (UGA/IM^{2}AG)

February 14, 2017

In this set of exercises, we will work on computing the number of cycles required to execute some C programs. We will run this analysis assuming a 40-MHz processor. The cost associated with each instruction is summarized in Table 1.

Instruction type	**Cycles**
Integer arithmetics	1
Data transfer	2
Floating point operation	2
Control	2
printf	20

Table 1: Cost per instruction

Exercise 1

We consider the C code snippet described in Figure 1.

```
p = 0 ;
while (b > 0)
{
  p = p + a ;
  b = b - 1 ;
}
printf ("%d\n", p) ;
```

Figure 1: Simple code snippet

Questions

1. What is computed by this code snippet? Compute the output for inputs $a = 2$ and $b = 6$.

2. Each instruction requires that its operands are loaded into registers. The only exception is in case the operand is a constant. Loading an instruction into the control unit of a processor also costs 1 cycle. Give the number of cycles required for executing each line of Figure 1.
3. Which variable of this program impacts its execution time?
4. Express the execution time of the program as a function of this variable.
5. Assuming that the value of this variable is 1023, compute the execution time of the program in seconds.

Exercise 2

To access one element in a vector, the index should first be loaded into a register. One memory load is then required to access the element of the vector. The `register` keyword in C tells the compiler to store the variable being declared in a CPU register (if possible), to optimize access.

Questions

1. Assuming that the 3 vectors fit into the fast memory of the processor, compute the number of cycles required to execute the loop in Figure 2.

```
#define N 1024

register int i ;

for (i = 0 ; i < N; i = i +1)
{
  X [i] = Y [i] + Z [i] ;
}
```

Figure 2: Vector sum ($X = Y + Z$)

We now apply loop unrolling to the previous code, as described in Figure 3.

```
#define N 1024

register int i ;

for (i = 0 ; i < N; i = i +4)
{
  X [i] = Y [i] + Z [i] ;
  X [i+1] = Y [i+1] + Z [i+1] ;
  X [i+2] = Y [i+2] + Z [i+2] ;
  X [i+3] = Y [i+3] + Z [i+3] ;
}
```

Figure 3: Vector sum – Loop unrolling

Questions

2. Compute the number of cycles required to execute the loop in Figure 3.
3. Explain why loop unrolling improves performance.

Exercise 3 (bonus)

It is possible to observe the assembly code generated by gcc when compiling C programs. In this exercise, we suggest you to observe the assembly code that would be generated for the programs we studied during this session, and to compare it against the number of cycles we computed.

There are different ways of obtaining the assembly code. We suggest you to use one of the two following solutions:

- Generating a file containing the C code and the corresponding assembly code.

  ```
  $ gcc -c -g -Wa,-a,-ad ex1.c > ex1.lst
  ```

- *Disassembling* the code using gdb.
  - Compile your program with debug symbols enabled (`-g` option)
  - Load the program into gdb and ask for disassembling the `main`. (command: `disassemble /m main`)