Lab Guide 1

Instruction Level Parallelism

Objectives:
- Review of the basic processor performance equation
  - Execution time = Instructions x Cycles Per Instruction x Clock Cycle Time (Texe = #I x CPI x Tcc)
- Identify instruction level parallelism (ILP) and its limitations on the matrix multiplication case study

Introduction

This lab session uses the `perf` tool to profile the execution of the Matrix Multiplication (MM) of type double case study (both were introduced in *Lab Guide 0[1]*). The main goal is to collect performance events counts of the basic performance equation, as well as other performance counters in order to understand ILP and its limitations.

The students are encouraged to use their own code, developed in lab session 0, but a simple MM code can be downloaded from the course eLearning page (see the Annex 1 to use the Search cluster).

To get a better profile and more accuracy, the code (C program) should be compiled with `-g -fno-omit-frame-pointer`.

Exercise 1 - The basic processor performance equation

a) <u>Performance estimation</u>: What is the complexity of the MM (i.e., in big O notation, where N is the problem size and NxN is the matrix size)? What increase in execution time is expected when the N doubles? Which component of the performance equation is affected (e.g., #I, CPI or Tcc)?

b) <u>Instructions estimation</u>: Look at the MM assembly code that is generated (e. g., `gcc -O2 -S ...`) with and without optimisation (-O2 versus -O0). Can you estimate the number of instructions executed for a NxN matrix, on each case? What is the expected gain from the compiler optimised version?

c) <u>Measure</u>: The number of instructions executed (#I) and clock cycles (#CC) can be directly measured with `perf stat -e cycles,instructions ./a.out` (note: this takes overall metrics, not metrics by function as `perf record`, but it is enough for this case. Why?). Fill the following table for matrix sizes of 128x128, 256x256 and 512x512 using the -O2 optimisation level. Measure the time without optimisation just for a 512x512 matrix size. Also, fill the table by calculating the average CPI column for the 512x512 matrix size (see lecture notes for CPI definition).

	size	Texe	#CC	#I	#I Estimated	Average CPI (calculated)
-O2	128					
	256					
	512					
-O0	512					

d) <u>Accuracy of the #I estimation</u>: Compare the measured #I against the estimative from 1b). Comment the difference. Does it increases at the expected rate when the N value doubles?

---

[1] The resolution of this guide assumes the resolution of the previous guide.

e) <u>#I vs CPI tradeoff</u>: What is the gain obtained with the O2 optimisation level on a 512x512 matrix? What component of the performance equation is responsible for this gain? Why it is lower than expected (estimated in 1b)?

Exercise 2 - Instruction level parallelism and its limits

a) Look again at the previous table. Why is the CPI less that one in some cases? What is the ideal CPI on this machine?

b) Explain why the CPI value is lower (i.e., better!) with the -O0 optimisation level (there are two important factors; one of them is in the answer to the next question).

c) Look at the assembly of the inner loop of the code generated with -O2 level. In this code identify the Read After Write (RAW) data dependencies. (note: your generated assembly code should be similar to the following assembly code for double data type: SD is single double):

```
.L12:
    movsd (%rdx), %xmm0  ;move SD from memory to register %xmm0
    mulsd (%rax), %xmm0  ;move SD from memory and mul with %xmm0
    addq  $4096, %rax    ; add 4096 to %rax
    addq  $8, %rdx
    addsd %xmm0, %xmm1
    movsd %xmm1, (%rcx)
    cmpq  %rax, %rsi
    jne   .L12
```

d) Draw an instruction dependency graph (i.e., task dependency graph, each instruction is a task) where each box is an assembly instruction (e.g., primitive operation) and each arrow represents a data dependency (one of these instructions should be represented by two boxes, why?).

e) Look at the final graph, without further improvements, what is the effective CPI of this code on an ideal machine (e.g., a processing unit with infinite resources, and 1 cycle operations: memory load, add, multiplication, …) (suggestion: focus on the longest dependency chain).

f) (*) Suggest code improvements to increase the ILP available on this code and compute the potential gain of that code on an ideal machine by drawing a new dependency graph.

g) (*) Implement the optimised code and test it on small matrices (e.g., 128x128). Compare the performance with and without this optimisation.

Annex 1: (simple) Instructions for using the search cluster:

a) **Copy local file to remote machine (don't forget the two points and point at the end) :**

   `scp <local file name> <student_id>@s7edu.di.uminho.pt:.`

b) **Login:** `ssh <student_id>@s7edu.di.uminho.pt`

c) **Load the gcc environment:** `module load gcc/9.3.0`

d) **Compile:** `gcc -g -fno-omit-frame-pointer -O2 …`

e) **Run:** `srun --partition=cpar perf stat -r 5 -e instructions,cycles <<full_path>>/a.out`