Joshua Ferguson

Homework 1

1.3 [2] <§1.3> Describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.

1.A compiler turns a high level language into assembly code instructions

2.The assembly code instructions are translated into machine language

3.the machine language instructions are carried out by the processor

1.5 [4] <§1.6> Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.

a. Which processor has the highest performance expressed in instructions per second?

Processor 2 wins with 2.5 billion IPS

b. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

the clock rate is a measure of cycles per second so the number of cyles is the clock rate * 10.

P1_cycles = $3.0 * 10^{10}$

P2_cycles = $2.5 * 10^{10}$

P3_cycles = $4.0 * 10^{10}$

the number of instructions executed is the number of cycles divided by the CPI, therefore:

P1_instructions = $P1\_cycles/1.5 = 2 * 10^{10}$

P2_instructions = $P2\_cycles/1 = 2.5 * 10^{10}$

P3_instructions = $P3\_cycles/2.2 = 1.82 * 10^{10}$

c. We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

time = ((instructions * CPI) / (clock rate))

time * 7 = ((instructions * (CPI * 1.2)) / (clock rate))*7

new clock rate = clockrate * (1.2/.7)

new clock rate = clockrate * (1.71)

P1 new clock rate = 3*1.71 = 5.13 Ghz

p2 new clock rate = 2.5 *1.71 = 4.27 Ghz

p3 new clock rate = 4 * 1.71 = 6.84 Ghz

1.7 [15] <§1.6> Compilers can have a profound impact on the performance of an application. Assume that for a program, compiler A results in a dynamic instruction count of 1.0E9 and has an execution time of 1.1 s, while compiler B results in a dynamic instruction count of 1.2E9 and an execution time of 1.5 s.

a. Find the average CPI for each program given that the processor has a clock cycle time of 1 ns.

i_count_a=1.0E9

t_a=1.1

i_count_b=1.2E9

t_b=1.5

t=i_count*CPI*clock_cycle

CPI=t/(i_count*clock_cycle)

CPI_a= 1.1

CPI_b= 1.25

b. Assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?

$performance_b y_t ime = \frac{CPI_a * i_c ount_a * clock_c ycle_a}{CPI_b * i_c ount_b * clock_c ycle_b}$

$\frac{clock_c ycle_b}{clock_c ycle_a} = \frac{CPI_b * i_c ount_b}{CPI_a * i_c ount_a}$ with $performance_t ime$ being constant the clock is 1.36 times faster

c. A new compiler is developed that uses only 6.0E8 instructions and has an average CPI of 1.1. What is the speedup of using this new compiler versus using compiler A or B on the original processor?

with A: $performance = \frac{1}{execution\_time}$

$relative\_performance = \frac{execution\_time\_old}{execution\_time\_new}$

$relative\_performance = \frac{i\_count\_new}{i\_count\_old}$ as both CPI and clock_cycle are the same for both

the performance is now 1.67 times faster

1.9 Assume for arithmetic, load/store, and branch instructions, a processor has CPIs of 1, 12, and 5, respectively. Also assume that on a single processor a program requires the execution of 2.56E9 arithmetic instructions, 1.28E9 load/store instructions, and 256 million branch instructions. Assume that each processor has a 2 GHz clock frequency.

Assume that, as the program is parallelized to run over multiple cores, the number of arithmetic and load/store instructions per processor is divided by 0.7 x p (where p is the number of processors) but the number of branch instructions per processor remains the same.

1.9.1 [5] <§1.7> Find the total execution time for this program on 1, 2, 4, and 8 processors, and show the relative speedup of the 2, 4, and 8 processor result relative to the single processor result.

execution_time=clock_cycle/clock_rate

I'm getting the clock cycle with this:

```
def get_clock_cycles(cpi_a,cpi_m,cpi_b,i_count_a,i_count_m,i_count_b):
     clock_cycle=cpi_a*i_count_a+cpi_m*i_count_m+cpi_b*i_count_b
     return clock_cycle
```

clock cycle for 1 I'm assuming doesn't suffer from the same constaint as the multicore, otherwise this number is off:

clock_cycle for 1= 19200000000.0

execution_time_1=9.6 seconds

for multicores with m being .7 * number of cores:

```
get_clock_cycles(1,12,5,2.56E9/m_prop,1.28E9/m_prop,256E6)/2E9
```

relative_speedup_n=execution_time_1/execution_time_n

execution_time_2=7.04 seconds

2 cores are 1.36 times faster

execution_time_4=3.84 seconds

4 cores are 2.5 times faster

execution_time_8=2.24 seconds

8 cores are 4.28 times faster

1.9.2 [10] <§§1.6, 1.8> If the CPI of the arithmetic instructions was doubled, what would the impact be on the execution time of the program on 1, 2, 4, or 8 processors?

execution_time_1=10.88 seconds

execution_time_2=7.95 seconds

execution_time_4=4.29 seconds

execution_time_8=2.47 seconds

1.9.3 [10] <§§1.6, 1.8> To what should the CPI of load/store instructions be reduced in order for a single processor to match the performance of four processors using the original CPI values?

since as demonstrated by the last code snippet that its the same equation with the value just multiplied by a value equal to .7 * the number of cores, just take the values for i_count_a

relative_speedup_4=execution_time_1/execution_time_4

execution_time_4=execution_time_1/relative_speedup_4

1.13 Another pitfall cited in Section 1.10 is expecting to improve the overall performance of a computer by improving only one aspect of the computer. Consider a computer running a program that requires 250 s, with 70 s spent executing FP 1instructions, 85 s executed L/S instructions, and 40 s spent executing branch instructions.

total time = 250s

fp execution time = 70s

L/S instruction execution time = 85s

Br execution time = 40s

1.13 .1 [5] <§1.10> By how much is the total time reduced if the time for FP operations is reduced by 20%?

total time = 250s

fp execution time = 70s

L/S instruction execution time = 85s

Br execution time = 40s

time_reduced=total_time-new_time

=250-(70*.8+(250-70))

time_reduced=14s

1.13 .2 [5] <§1.10> By how much is the time for INT operations reduced if the total time is reduced by 20%?

a_time_original=250-70-40-85

a_time_original=55

a_time_new=55*.8

a_time_new=44

1.13 .3 [5] <§1.10> Can the total time can be reduced by 20% by reducing only the time for branch instructions?

yes

20% reduciton means 200 seconds

250=55+80+70+40

250-50=5+80+70+40

5/55=.0909 percent = 1-.0909 90.909%

1.14 Assume a program requires the execution of 50 × 106 FP instructions, 110 × 106 INT instructions, 80 × 106 L/S instructions, and 16 × 106 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

1.14 .1 [10] <§1.10> By how much must we improve the CPI of FP instructions if we want the program to run two times faster?

```
def
get_clock_cycles(cpi_a,cpi_f,cpi_m,cpi_b,i_count_a,i_count_f,i_count_m,i_co
unt_b):
     clock_cycle=(cpi_a*i_count_a)+(cpi_f*i_count_f)+(cpi_m*i_count_m)+
(cpi_b*i_count_b)
     return clock_cycle
```

I'm assuming that 50*106 is not meant to be $50 * 10^6$

original_time=clock_cycle/2Ghz

original_time=2.71s

desired_time=1.355s

new_fp_cpi=(clock_cycle/2-(clock_cycle_without_fp))/fp_instructions

the result is -4.12

in other words, it is impossible to get this performance boost from just from tweaking the floating point CPI.

1.14 .2 [10] <§1.10> By how much must we improve the CPI of L/S instructions if we want the program to run two times faster?

.8 performance boost 4/.8=5 it needs to run 5 times faster

1.14 .3 [5] <§1.10> By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30%?

new time = 1.81E-05 s

old time =2.71E-05 s

the new load is 1.5 times faster