CSCE 513: COMPUTER ARCHITECTURE

Protik Nag

1 Part A: Assembly Code

1.1 Pseudocode

Algorithm 1 Vector Multiplication

```
1:  function MULTIPLY(A, B, n)                ▷ Where A ∈ A^n, B ∈ B^n, n = Number of elements
2:      $s0 ← A
3:      $s1 ← B
4:      $t0 ← n
5:      $t1 ← 0                                ▷ Dot product accumulator
6:      $t2 ← 0                                ▷ Loop variable
7:      $t3 ← 0                                ▷ Offset value for traversing array
8:
9:      while true do
10:         $s2 ← 0($s0)                       ▷ First element of array A
11:         $s3 ← 0($s1)                       ▷ First element of array B
12:         $s4 ← 0                            ▷ Multiplication sum accumulator
13:
14:         while true do
15:             if $s3 == 0 then
16:                 Break                      ▷ Stop the loop if one of the variables goes to zero
17:             else
18:                 $s4 ← $s4 + $s2            ▷ One variable gets summed up to the $s4
19:                 $s3 ← $s3 − 1              ▷ Decrease other by 1
20:             end if
21:         end while
22:
23:         $t1 ← $t1 + $s4                    ▷ Update the dot product sum
24:
25:         $s0 ← $s0 + 4          ▷ Moving to the next element of the array A and set it to be the first element
26:         $s1 ← $s1 + 4                      ▷ Same goes for B
27:         $t2 ← $t2 + 1                      ▷ Increment loop variable
28:
29:         if $t2 == t0 then
30:             Break                          ▷ If loop counter equals to n then break
31:         end if
32:     end while
33:
34:     Print($t1)                             ▷ Result is in $t1
35:
36: end function
```

1.2 Samples

Table 1, 2 and 3 represent the sample sizes (32, 16 and 8 respectively), which have been used to test the code, expected output and obtained output. Screenshots of the MARS terminal will be found here [1].

Vector A	Vector B	n	Expected Output	Obtained Output
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]	[32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	32	5984	5984

Table 1: Sample of testing code with 32 sized arrays

Vector A	Vector B	n	Expected Output	Obtained Output
[15, 4, 19, 5, 2, 3, 4, 8, 5, 7, 11, 12, 3, 6, 5, 1]	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]	16	814	814

Table 2: Sample of testing code with 16 sized arrays

Vector A	Vector B	n	Expected Output	Obtained Output
[15, 4, 19, 5, 2, 3, 4, 5]	[4, 11, 8, 4, 1, 0, 6, 8]	8	342	342

Table 3: Sample of testing code with 8 sized arrays

2 Part B: Performance Analysis

2.1 Clock per Instruction (CPI)

$$Effective\ CPI = \Sigma_{i=1}^{n}(CPI_i * Proportion_i) \tag{1}$$

Here, CPI_i = Number of clock cycles required for the instruction class i and $Proportion_i$ = Percentage of the intruction class i among total number of instructions. Given that, R-type instructions require an average of 2 clocks, I-type instructions require an average of 3 clocks and J-type instructions require 4 clock.

Using above information, we can calculate the CPI for different lengths of vectors which is represented in the table 4.

Vector Length	**IC**	**R-Type**	**I-Type**	**J-Type**	**R-Type %**	**I-Type %**	**J-Type %**	**CPI**
8	346	94	210	42	27.17	60.69	12.14	2.85
16	990	290	564	136	29.29	56.97	13.74	2.84
32	3502	1090	1884	528	31.13	53.80	15.08	2.84

Table 4: Instruction Count and CPI Table

2.2 Energy Consumption

According to the given energy consumption rate for different instruction type we can calculate the total energy consumption in femto-joule, using the following formula:

$$Total\ Energy\ Consumed = \Sigma_{i=1}^{n}(IC_i * Energy_i) \tag{2}$$

Here, IC_i = Total number of instructions in i class and $Energy_i$ = Energy consumed to run per instruction of i class.

Length of Vectors	**IC**	**ALU**	**Jump**	**Branch**	**Memory**	**Other**	**Energy Consumption**
8	346	227	42	58	17	2	1258
16	990	651	136	168	33	2	3352
32	3502	2315	528	592	65	2	11008

Table 5: Energy Consumption

2.3 MIPS/mW

We can define MIPS/mW as follows:

$$\begin{aligned} MIPS/mW &= \frac{MIPS}{Power(mW)} \\ &= \frac{Instruction\ Count}{Execution\ Time * 10^6} * \frac{Execution\ Time}{Energy * 10^3} \\ &= \frac{Instruction\ Count}{Energy * 10^9} \\ &= \frac{Instruction\ Count}{Energy(in\ fj) * 10^{-15} * 10^9} \\ &= \frac{Instruction\ Count * 10^6}{Energy(in\ fj)} \end{aligned} \tag{3}$$

Following table 6 shows the calculation.

Vector Length	IC	Energy Consumption	MIPS/mW
8	346	1258	275039.75
16	990	3352	295346.06
32	3502	11008	318132.27

Table 6: MIPS/mW

2.4 Performance Comparison

Energy consumption rises as we gradually increase the number of elements in the vectors. The following graph captures this upward trend.

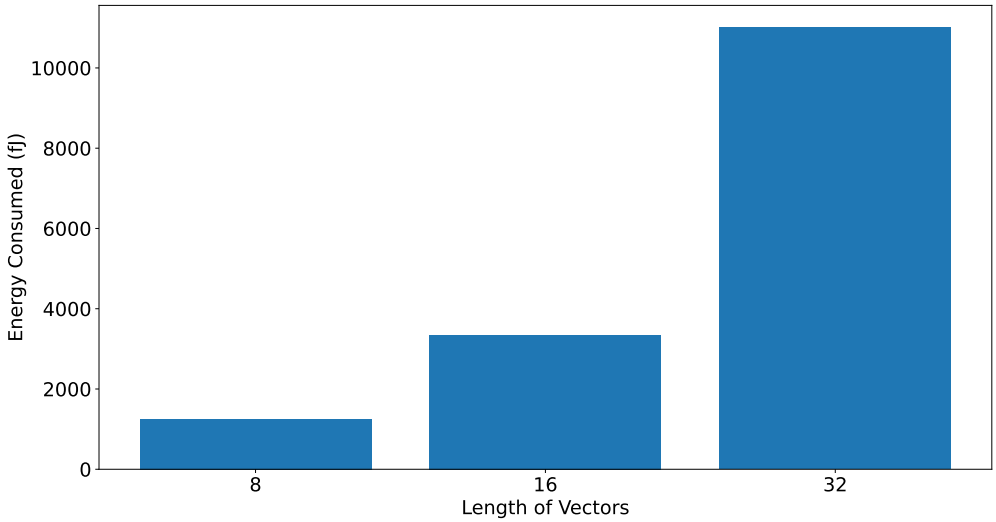

Figure 1: Energy Consumption vs Length of Vectors

Similarly, execution time will step up with the increasing number of elements. We know,

$$Execution\ time = IC * CPI * CycleTime \tag{4}$$

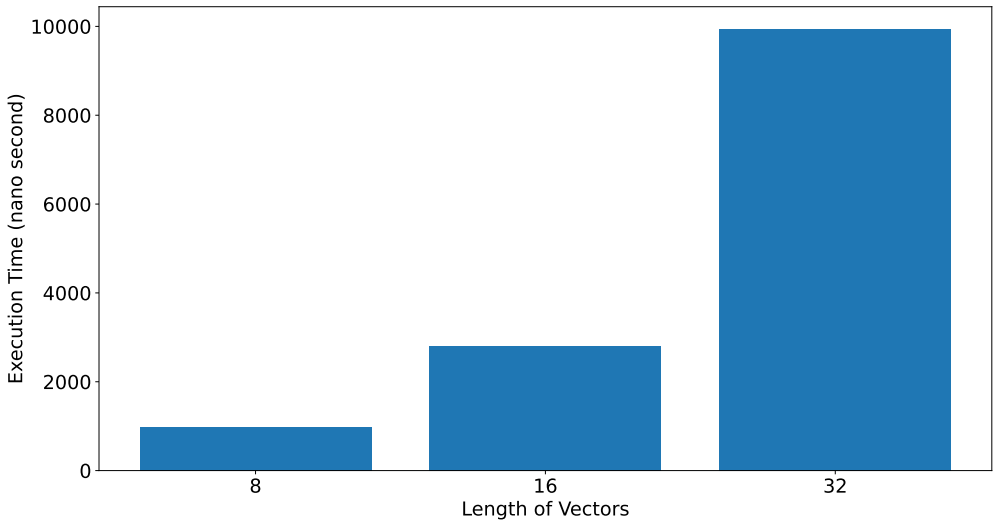

Figure 2: Execution Time vs Length of Vectors

As CPI and Cycle Time are identical, we can relate execution time with the number of instructions. Performance is inversely proportional to the execution time.

References

[1] Protik Nag. Screenshots from mars terminal. `https://github.com/ProtikNag/Computer-Architecture-Homeworks/tree/main/Images`, Oct 1, 2023.