Modern Computer Architecture
Lab Assignment 1

Group 3: Aniket A Samant, Apoorva Arora, Snehal Jauhri

December 2018

1 Introduction

Given a parameterized VLIW processor, the aim of this assignment is to find the best parameter values, that is, to identify a resource configuration in order to optimize certain applications' execution in terms of metrics like performance (speed) and power consumption. Thus, the goal is to perform a design space exploration (DSE) for the VLIW processor to come up with a processor instance that is optimal for two given programs - 1) Matrix Multiplication, and 2) Convolution 3x3. For the purpose of this assignment we have considered a single cluster VLIW processor.

2 Benchmarks used

2.1 Matrix.c

matrix.c is a C program to test the result of matrix multiplication of two 64x64 matrices. Three matrices, a[64][64], b[64][64] and Result[64][64] are provided, and the matrix c[64][64] is computed as the product of the multiplication of a and b. c is compared with Result by checking the equality of the matrices, based on which "Matrix Test Passed" is printed (if all elements are equal), else "Matrix Test Failed". The two matrices are multiplied using three nested *for* loops which involve additions and multiplications of the elements of the a, b, and c matrices.

2.2 Convolution_3x3.c

convolution_3x3.c is a C program that runs a predefined 3x3 filter over a 64x64 image in order to perform a convolution operation on it. The convolution is carried out using two *for* loops, one for the width of the image and one for the height. The RGB values obtained by convolving the values of the filter and the image's pixels are stored in a new array of unsigned integers (the R,G, and B values are stored in the same location using left shift operators). On completion of the convolution operation, a string is printed to the terminal stating that the convolution is finished.

3 Assumptions about the environment

We assume the programs run on embedded devices, more specifically on devices meant for signal processing (for instance, sound cards and digital cameras), given the matrix and convolution operations. Thus, an application-specific architecture is considered. Hence, while we try to achieve faster execution time, **more focus is on area and power minimization**.

4 VEX Architecture Configuration Parameters

Here we discuss the processor resources and how they impact the performance of the benchmarks and our area model. We observe a trade-off between the execution speed and area utilized. This is because, in order

to increase the speed of execution, parallelism is increased (issue width), requiring more functional units. But increasing the number of these units - ALUs, multipliers, load/store units, branch units and registers - increases the area required as shown in Figure 1.

A_{1_ALU}	A_{1_Mult}	$A_{1\_LW/SW}$	A_{64_GPR}	A_{8_BR}	A_{misc}	$A_{1\_Connection}$
3273	40614	1500	26388	258	6739	1000

Figure 1: Area estimation for Default VEX Architecture

4.1 Matrix Multiplication

On analyzing the matrix.s assembly file, we observe that load, compare, addition and multiplication operations dominate for the common case which is Trace 1 (of the main() function) as indicated by the output of the 'pcntl' command. Thus, increasing these functional units that is, no. of ALUs, multipliers and load/store units would improve the performance. At the same time we need to keep the resource utilization(area) under check.

4.2 Convolution

Analyzing the convolution.s assembly file, we identify 'add', 'compare' and 'load' operations are dominating for the common case, that is, Trace 1 (main()). Thus, with increasing the number of ALUs or providing an ALU in each issue slot, the speed of execution is expected to increase.

5 The Design Space Exploration process

Space exploration was done by evaluating different configurations based on Area (calculated using the values given), Number of execution cycles (using the 'ta.log.000' output) and IPC for the common case (using the 'pcntl' tool provided in VEX).

5.1 Initial tests

Initially, we carried out tests on both the benchmarks using relatively expensive machine configurations (larger Area) i.e. using a large issue width (6 and above) and high number of ALUs, multipliers etc. and observed the following:

- Increasing the number of ALUs/Mutipliers/Load-Store Units/Connections for Load operations along with the Issue width does indeed lead to improved performance i.e. lower number of execution cycles and larger IPC. Meanwhile, increasing the number of connections for Stores has small to no improvement in the number of cycles.
- Increasing the number of General purpose registers leads to reduction in number of cycles for both benchmarks. However, even for an expensive configuration, both the benchmarks do not use more than 28 registers i.e. there is no improvement in performance by increasing the number of general purpose registers beyond 28.
- Increasing the number of Branch registers leads to reduction in number of cycles for convolution3x3. However, no significant further reduction in no. of execution cycles is observed after 4 Branch registers.

5.2 Obtaining the Pareto Optimal Front

Putting greater emphasis on obtaining a configuration with a small area, we ran the tests first using the configuration with the smallest area possible and then incrementally added more functional units. The idea was to obtain a significant reduction in Execution cycles and higher IPC while still keeping the Area minimal.

Figure 3 (Appendix) shows the processor resources and how their configuration affects the area. Figure 4 (Appendix) shows how the resource configuration affects the performance for both the benchmarks. Several points were kept in mind while configuring the parameters, such as:

- Functional units were added based on insight from analyzing the assembly files.
- Issue Width was gradually increased and, in tandem, functional units were gradually added to take advantage of this larger Issue Width.
- We stopped adding functional units of a particular type once little to no improvement was seen from adding more functional units of that type. (For the same Issue Width)
- The number of multipliers was limited to a maximum of 2, since they lead to a large increase in Area.
- Since the addition of Load operations/connections and ALUs has a relatively lower cost in terms of Area, these were preferred to be added. Moreover, we kept the number of Loads and ALUs equal/almost equal to the Issue width to obtain a larger IPC.
- The number of general purpose registers was limited to 28, since we know from the initial tests that a value larger than this is not required. Occasionally we tested with 32 registers to re-evaluate this result.

The plot of Area vs the Execution cycles (summed for both our benchmarks) for each tested configuration can be seen in **Figure 2**. The green line represents the Pareto optimal front obtained.

5.3 Automation involved

The process of simulating various architecture configurations based on parameters provided by the user was automated in the following way:

- An Excel spreadsheet consisting of row entries corresponding to various architecture parameters was used as the source for generating the "configuration.mm" file for the configurations provided by the user (each row corresponding to a particular configuration, and the columns corresponding to the different parameters).
- A python script (located in the same directory as the benchmark files') was used to iterate over the aforementioned Excel sheet row by row, and perform the following operations for each row:
    1. Based on the values provided by the user for the configuration parameters (issue width, ALUs, etc.) under the corresponding columns, a configuration.mm file was created in the same directory.
    2. For each of the benchmarks, a shell script was called that created their output directories with the simulation results for the configuration.
    3. Through regex matches, the values for "Execution cycles" and trace 1's ILP (since trace 1 is the main trace in both benchmarks) for each benchmark were extracted by the py script (from the two output directories) and added to a row in a different Excel spreadsheet consisting of 4 columns corresponding to the two values for each of the two benchmarks.
- The above steps were repeated for each row in the configurations Excel sheet, and hence finally, we had an output Excel sheet consisting of the simulation result values for each configuration.

6 Final solution and results

As seen from **Figure 2**, for the first few points (top left points), we get almost a 'free lunch' as we add functional units and increase the issue width in our configuration. The total number of execution cycles (both benchmarks summed up) reduces significantly with only a small increase in Area. From configuration index 9 onward (red point in the plot), we start noticing a larger increase in Area as the number of execution cycles reduces. This shows the 'law of diminishing returns' from Area 60000 onward as a relatively small

improvement in total execution cycles is seen, even with addition of more functional units and a larger issue width. (bottom right points in the plot)

Since our overall plot already places a larger emphasis on reducing area (our configurations chosen assuming an embedded domain application), we chose **Configuration ID 9 (red point in Figure 2)** as the best configuration. This configuration uses the least area while still achieving a reasonable level of performance (i.e. execution cycles and IPC) as compared to the top left points in our plot:

Configuration:
RES: **IssueWidth 3**, RES: **MemLoad 2**, RES: **MemStore 1**, RES: **MemPft 1**, RES: **Alu 3**, RES: **Mpy 1**, RES: **Memory 2**, REG: **$r0 16**, REG: **$b0 3**

Area: **61240.8625** units

IPC of Common Case (Trace 1) for matrix.c: **2.5**
IPC of Common Case (Trace 1) for convolution3x3.c: **2.49**

Sum of **Execution cycles** = 754913 + 518563 = **1273476** units

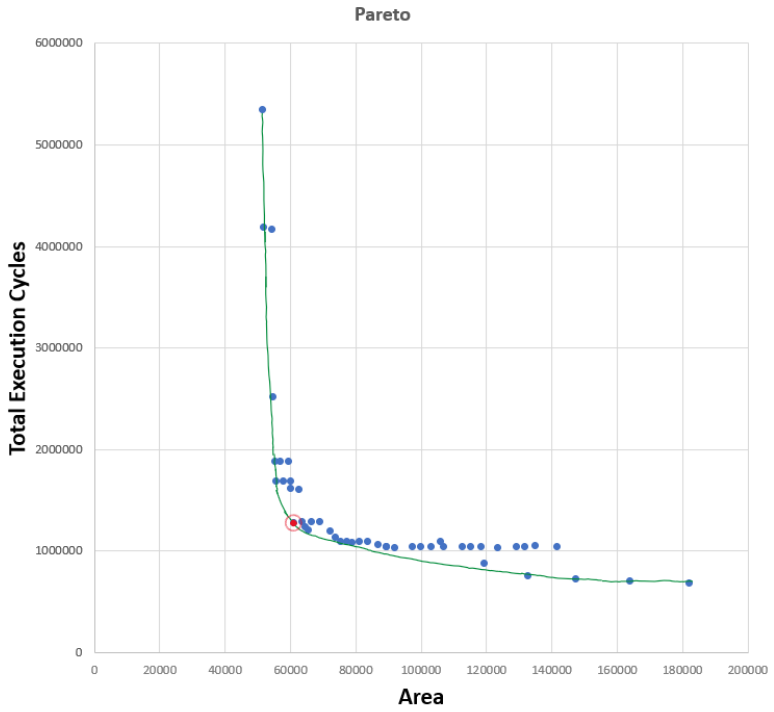

Figure 2: Area vs Total Execution Cycles

7 Conclusion & Reflection

The assignment involved modifying the resource configuration of the processor and analyzing its effects on the performance. Working on the assignment, we learned how to derive relevant information from the configuration files, code(.c) files, assembly files and output of tools like 'pcntl' and ta.log. Based on the information from above sources we were able to relate the configuration parameters such as issue width, no. of functional units, no. of registers, no. of load/store units and no. of loads with performance metrics like the execution time for each trace, IPC, avgerage ILP and area. We could identify the bottlenecks in the program and glean relevant information on which our design decisions were based.

Appendix

This section consists of the Excel configuration data used for automating the simulation process, and the corresponding results obtained. (See the "Automation Involved" section for further details.)

CONFIG INDEX NO	ISSUE WIDTH	MEM LOAD	MEM STORE	MEM PFT	NUM ALU	NUM MPY	NUM MEMORY	NUM GPR	NUM BR	AREA
0	1	1	1	1	2	1	1	4	1	51795.42813
1	2	1	1	1	2	1	1	4	2	52137.0125
2	2	2	1	1	2	1	2	4	2	54637.0125
3	2	2	1	1	2	1	2	8	2	55049.325
4	2	2	1	1	2	1	2	12	2	55461.6375
5	2	2	1	1	2	1	2	16	2	55873.95
6	3	2	1	1	2	1	2	12	3	57040.15938
7	3	2	1	1	2	1	2	16	3	57967.8625
8	3	2	1	1	3	1	2	12	3	60313.15938
9	3	2	1	1	3	1	2	16	3	61240.8625
10	3	3	1	1	2	1	3	12	3	59540.15938
11	3	3	1	1	2	1	3	16	3	60467.8625
12	3	3	1	1	3	1	3	12	3	62813.15938
13	3	3	1	1	3	1	3	16	3	63740.8625
14	3	3	1	1	3	1	3	20	3	64668.56563
15	3	3	1	1	3	1	3	24	3	65596.26875
16	3	3	1	1	3	2	3	24	3	106210.2688
17	4	3	1	1	3	1	3	16	4	66659.4
18	4	4	1	1	3	1	4	16	4	69159.4
19	4	4	1	1	4	1	4	16	4	72432.4
20	4	4	1	1	4	1	4	20	4	74081.65
21	4	4	1	1	4	1	4	24	4	75730.9
22	4	4	1	1	4	1	4	28	4	77380.15
23	4	4	1	1	4	1	4	32	4	79029.4
24	4	4	1	1	4	2	4	32	4	119643.4
25	5	4	1	1	4	1	4	24	5	81329.46875
26	5	5	1	1	4	1	5	24	5	83829.46875
27	5	5	1	1	5	1	5	24	5	87102.46875
28	5	5	1	1	5	1	5	28	5	89679.42188
29	5	5	1	1	5	1	5	28	5	89679.42188
30	5	5	1	1	5	1	5	32	5	92256.375
31	5	5	1	1	5	2	5	32	5	132870.375
32	6	5	1	1	5	1	5	28	6	97648.7875
33	6	6	1	1	5	1	6	28	6	100148.7875
34	6	6	1	1	6	1	6	28	6	103421.7875
35	6	6	1	1	6	1	6	32	6	107132.6
36	6	6	1	1	6	2	6	32	6	147746.6
37	7	6	1	1	6	1	6	28	7	112834.2469
38	7	7	1	1	6	1	7	28	7	115334.2469
39	7	7	1	1	7	1	7	28	7	118607.2469
40	7	7	1	1	7	1	7	32	7	123658.075
41	7	7	1	1	7	2	7	32	7	164272.075
42	8	7	1	1	7	1	7	28	8	129462.8
43	8	8	1	1	7	1	8	28	8	131962.8
44	8	8	1	1	8	1	8	28	8	135235.8
45	8	8	1	1	8	1	8	32	8	141832.8
46	8	8	1	1	8	2	8	32	8	182446.8

Figure 3: Resource Configuration

CONFIG_INDEX_NO	MATRIX_EXEC_CYCLES	MATRIX_TRACE1	CONV_EXEC_CYCLES	CONV_TRACE1
0	3585905	0.88	1747225	0.91
1	2933801	1.09	1245928	1.28
2	2892582	1.11	1265447	1.26
3	1638502	1.48	874157	1.66
4	1062886	1.77	811723	1.67
5	962534	1.96	715870	1.76
6	1062886	1.77	811723	1.67
7	962534	1.96	715870	1.76
8	1017057	1.9	592344	2.33
9	754913	2.5	518563	2.49
10	1062886	1.77	811723	1.67
11	962534	1.96	715870	1.76
12	1019104	1.9	580440	2.37
13	756960	2.5	518563	2.5
14	756960	2.5	476709	2.56
15	754912	2.5	443415	2.71
16	654432	2.89	431945	2.75
17	756960	2.5	518563	2.5
18	756960	2.5	518563	2.5
19	751647	2.5	434058	3.05
20	751647	2.5	372487	3.34
21	747551	2.5	342105	3.53
22	745439	2.5	337951	3.54
23	745439	2.5	330139	3.55
24	548831	3.44	322389	3.57
25	747551	2.5	342105	3.53
26	747551	2.5	342105	3.53
27	745501	2.5	307260	4
28	743325	2.5	292008	4.09
29	743325	2.5	292008	4.09
30	743325	2.5	280600	4.24
31	481181	3.93	267642	4.33
32	743325	2.5	292008	4.09
33	743325	2.5	292008	4.09
34	743197	2.5	288039	4.29
35	743197	2.5	288039	4.22
36	481053	3.93	232611	5.09
37	743197	2.5	288039	4.29
38	743197	2.5	288039	4.29
39	741148	2.5	291574	4.31
40	741148	2.5	287853	4.36
41	479004	3.93	214755	5.5
42	741148	2.5	291574	4.31
43	741148	2.5	291574	4.31
44	739100	2.5	310670	4.16
45	739100	2.5	299261	4.14
46	476956	3.93	199317	5.94

Figure 4: Resource Configuration and Performance