ET4074 Assignment 1 Report

Hang Ji,Srinidhi Srinivasan, Suryansh Sharma,
Delft University of Technology, The Netherlands
S.sharma-13@student.tudelft.nl
S.Srinivasan-2@student.tudelft.nl
H.Ji@student.tudelft.nl

I. Introduction

The aim of this assignment was to use a parameterizable VLIW soft-core processor (VEX) to determine a suitable architecture for running two benchmark C programs. The programs provided for this task were a 3x3 matrix multiplication and a 7x7 convolution which are a part of the powerstone benchmark suite used widely to test processor performance. The application domain which utilized both of the programs was identified to be Image Processing. This served as key in identifying the implementation environment for which the processor design was being developed. In order to customize the VLIW processor to application specific requirements the architecture configuration parameters provided in *configuration.mm* were tuned. This process involved an extensive design space exploration over a very large search space. The quality of the solution was evaluated based on the number of execution cycles and the total area of the design. The Design Space Exploration (DSE) conducted was focused on obtaining the most efficient processor design and involved a trade-off between performance and area utilization. While a high performance was the driving force behind the exploration, a strong emphasis was also given to restricting the area of the processor design to acceptable limits. In this report, we present our methodology for the design space exploration and discuss the results obtained. We also evaluate our solution design against the results obtained from Simulated Annealing which was applied to a subset of the design space. The results show that our solution is within the set of optimal design configurations and is a Pareto point on the Delay-Area Curve.

II. High-level Description of benchmarks

In this section a high-level description of the given benchmarks are provided. The analysis of the C code is done to understand the working and structure of the code.

A. Description of matrix.c

The program computes the product of two 64×64 matrices A and B. It implements a function *int matrix_mul* which computes the product by of a_{ij} and b_{jk} as well as of a_{ij+1} and b_{j+1k} in a single loop iteration. The program utilizes loop unrolling to reduce the program execution time. Compared to the naive implementation of matrix multiplication, the unrolling makes the program execute 131072 (64×32×64) iterations instead of 262144 (64×64×64). This implies only 50% of the jumps and conditional branches need to be taken. On the other hand, this means more registers have to be allocated to store variables. This loop unrolling is a trade-off between delay and area and needs to be factored in to the design of the architecture. For each product of c_{ik}, two operations are performed, multiplication and addition. The execution time can be improved by computing the two multiplication for each loop in parallel.

B. Description of convolution7×7.c

Convolution7×7.c is a C program which processes a given 64×64 image by applying a 7×7 filter. This is done to convert RGB images into colorful art. The *main* function implements image processing by applying the image filter to each pixel in the image. This generates an RGB value for each of the corresponding pixels by iterating over the entire image. The loops present in the *main* function which transform the pixel values mostly comprise of addition, comparison and multiplication operations.

C. Application Domain and Implementation Environment

The two benchmarks hinted towards many possible application where they can be utilized. While matrix multiplication is a fundamental program with a plethora of application scenarios, when combined with the 7×7 Convolution program, it is most suited for Image Processing based applications. The field of Image Processing requires extensive use of convolution filters and matrix multiplication and the high level analysis of the code in section II-B validates this claim.

The identification of Image Processing as the application makes the task of estimating the Implementation Environment and platform trivial. The domain is well suited for medium to large sized embedded systems and desktop computing which have the required resources to run such an application. Thus, the solution processor architecture is also aimed for being implemented on desktops. This served as a key factor for many design decisions of the exploration process.

III. Profiling

The effective exploration of the design space necessitates analysis of the performance bottlenecks present in the code.

The tool `pcntl` was used to identify the most execution cycle consuming sections of the code for each benchmark. The output of this tool gave key information about the various traces executed in each program along with the level of parallelism present. The traces which dominated each program in terms of the number of cycles were identified along with their IPC values. These are presented in Table [I]. Only Bottleneck traces are shown which contributed the most to the program's performance. The values showed that there existed great scope of introducing parallelism in the architecture to increase the IPC count of the bottlenecks. The tool was run on the reference architecture provided which used the default parameter values as mentioned in Table II.

TABLE I: `pcntl` Output of Benchmarks (Bottlenecks)

Benchmark	Procedure	Trace no.	Cycles	Operation	IPC
matrix.c	matrix_mul	1	19	55	2.89
	main	1	20	38	1.90
convolution_7x7.c	max	1	3	3	1.00
	min	1	3	3	1.00
	main	1	3.59	37	133

From the output pf the `pcntl` tool, it was observed that the traces that required the most number of cycles belonged to the nested loops present in both the programs. These traces became the focus of the future Design Space Exploration which was performed on the assembly language code of the two benchmarks. Analysis of the assembly language code ascertained that the high level program of the two benchmarks depended heavily on arithmetic functions like addition and multiplication for which ALUs and multipliers were required. The constructs which used ALUs and multipliers were mainly found within loops, hence, suggesting the extensive use of these components for each iteration. The use of Load-Store units was relatively less in comparison to the other functional units.

A design decision was made to improve the performance of this **common case** by reducing the number of cycles present in these traces and thereby increasing their IPC. This approach was justified by Amdahl's Law which proved to be a guiding principle in the optimization process. This implied that any enhancement of the performance of the application was not to be focused on a specific benchmark which would incur an unwarranted gain in area while only improving the performance of a single benchmark. The improvements made to the performance should focus on reducing the area increase penalty by considering steps which sped up both the benchmarks. The law which explains much like the law of diminishing returns prohibited our exploration from making small trivial changes which lead to a large area increase.

IV. Evaluation Metric

The processor architecture had to be well suited to run both of the benchmarks. This meant that the selection of an appropriate evaluation metric for the design space exploration process was non-trivial. This section delineates the evaluation metric chosen for the exploration as well as its justification for the task.

A. Normalization and Geometric Mean

The metric that was most vital for measuring performance was the number of Execution Cycles for each benchmark. However, the performance evaluation of the processor architecture could not have been done independently for each benchmark. This method would not yield a single measure, a common fundamental base, to compare both the benchmark's performance for varying processor configurations. It was, therefore, decided that the number of execution cycles should be converted into a single number which evaluated the performance of the architecture relative to the provided reference configuration.
The first step was to normalize the number of execution cycles for each benchmark against the reference. The next step was to calculate the Geometric Mean of the two normalized values to arrive at a single aggregate measure of performance. This was done to make the performance independent of the reference chosen and give a generalized summary metric. This metric was called delay and was calculated using the formula:

$$Delay = \sqrt[2]{\prod_{i=1}^{2} \frac{ExecutionCycles\_i}{Reference\_ExecutionCycles}} \tag{1}$$

B. Total Area of Processor

The total area of the architecture implemented including all of the register files, cache and functional units was to be found as a function of the configuration parameters. This was done using the formula below:

$$TotalArea = (\#ALU * A_{ref\_ALU}) + (\#Mult * A_{ref\_Mult}) \\ +(\#LW/SW * A_{ref\_LW/SW}) + (A_{GR}) + (A_{BR}) + (A_{Cache}) \tag{2}$$

where A_{GR}, A_{BR} and A_{Cache} are

$$A_{GR} = \frac{A_{ref\_GR}}{64} * (\#GR) * \left(\frac{IssueWidth}{4}\right)^2 \tag{3}$$

$$A_{BR} = \frac{A_{ref\_BR}}{8} * (\#BR) * \left(\frac{IssueWidth}{4}\right)^2 \tag{4}$$

$$A_{Cache} = (\#MemStore + \#MemLoad + \#MemPft) \\ *A_{ref\_LW/SW}) \tag{5}$$

V. VEX Instance Design Space

The design space provided in terms of the various VEX configuration parameters that could be modified to generate different architectures consisted of the following

- Issue Width (IW)
- Number of of ALUs (ALU)
- Number of Multipliers (Mpy)
- Number of Load/Store Units (LD/SR)
- Number of Memory Load Connections (MemStore)
- Number of Memory Store Connections (MemLoad)
- Number of Memory Pre-fetch Connections (MemPft)
- Number of General-purpose Registers (GR)
- Number of Branch Registers (BR)

The value of all of these parameters for the provided reference configuration is tabulated in Table II.

TABLE II: Intermediate VEX Instances

Parameter Name	Reference	Step 1(max)	Step 2	Step 3	Step 4	Step 5(min)
Issue Width	4	20	20	8	8	2
No. of ALU	4	40	14	6	6	4
No. of Multipliers	2	20	14	6	4	2
No. of Load/Store	1	20	10	4	2	1
No. of General-purpose Registers	64	40	40	40	30	8
No. of Branch Registers	8	4	4	4	4	1
Delay	1	0.4553008703	0.4553008703	0.6739740839	0.7392400742	2.7501227769
Area	125444	1429400	1065618	344754	240046	98820

VI. Design Space Exploration

A. Preliminary Exploration

The design space consisted of countably infinite set of possible configurations. This was constrained by finding an upper bound for each parameter. The most important parameter for this task was the issue width. It was determined from theoretical observation that for a given issue width, the maximum number of ALUs which could be *utilized* were twice the issue width while the maximum number of multipliers and Load-Store units were equal to the value of the issue width. This was verified experimentally by changing functional and load-store units above this limit. This was done for different values of issue widths. All of these cases resulted in an increase in total area without any change in performance. This was due to the fact that the additional functional and load-store units could not be accommodated in any of the parallel pipelines of the VLIW processor.

While the upper bound for functional and load-store units depended on the issue width, the number of 32-bit general-purpose registers was based on the memory requirements of the two benchmark programs. The number of 1-bit branch registers depended on the number of branching statements present in the assembly level code of the programs. The upper bounds for both of these were subject to the nature of the programs and hence, needed an experimental approach to discover.

B. Optimization

The first goal of design space exploration was to reach the right end of the Area-Delay curve where the delay value saturated to a lower bound and any configuration change aimed at reducing delay simply resulted in an increase in the total area. It was observed that the largest number of parallel instructions possible was 20 and any issue-width higher than 20 did not yield in an increase in the number of parallel instructions in a cycle and therefore, did not reduce delay. The only difference between VEX Instances with higher issue-width compared to 20 was in the total area. This confirmed that an Issue-width of 20 was a saturation point.

When the assembly file was assessed again for the 20 issue width VEX Instance, it was observed that in each block, the maximum number of ALU operations processed were 14, Multiply operations were 14 and Load-Store operations were 10. It was also observed that in the entire program the maximum number of 32-bit registers used was 40 and the maximum number of 1-bit registers were 4. The other configuration parameters were set to reference values(default) and the number of general purpose registers GR was increased in multiples of 2 starting from 8 up to 256. Analysis of the matrix.s showed that no additional registers were being utilized after 32. This limit was 40 for the convolution7$\times$7.s program. The same experiment was run for other issue widths and functional unit combinations all of which pointed to the same conclusion. A similar approach was adopted to find the upper-bound for Branch registers which were determined to be 3 for the matrix benchmark and 4 for convolution7$\times$7.s. It should be noted that the Number of Connections of Memory Store and Memory Load were always kept the equal to that of the number of Load-Store units and Memory Pre-fetch was kept at a constant 1 for each configuration throughout the design space exploration process.

The second step (Table II) of the exploration process comprised of changing the configuration file to the values determined above. As expected no change was noticed in the delay value proving that the values implied the saturation points for each component. The next step was to bring down the large total area by reducing the issue-width. The values of number of ALUs, multipliers and the Load-Store units were maintained in the same proportion as established before. This significantly reduced the area and brought it closer to fit the environment the benchmarks would need to perform in. In order to reduce the area further , in the fourth step , the number of functional units was scaled down. Hence the values of the multipliers, Load-Store an 32-bit registers have been reduced to give us an optimal solution that could be obtained through manual design space exploration.

The last step in manual exploration was to find the other extreme of the Pareto curve. This was the point which denoted maximum delay but the smallest area possible in a configuration. The values obtained for this condition, was the lowest possible values that could be run without either of the benchmarks crashing. The configuration that was successfully ran both the benchmarks without crashing is mentioned in Table II along with the configurations and delay-area values of the steps mentioned.

C. Simulated Annealing

Simulated annealing was used as the method of choice for obtaining the optimized design. The algorithm was well suited to explore the extremely large (but finite) set of potential solutions. The upper bounds discovered VI-A and the insights gained VI-B helped shape the algorithm's implementation. The reduction in the design space made it possible to run the algorithm in reasonable time. Since the design space was bound by the constraints mentioned above, 50000 steps were chosen each with a varying configuration parameter. This led to the algorithm exploring 50000 different VEX Instances bound between the limits mentioned in Table II. The plot of this exploration was generated to obtain Pareto optimal design point which suited the application domain and implementation platform. The algorithm was made to optimize the product of Delay and Area as calculated from Equations 1 and 2.

VII. Results and Discussion

Combining results from manual exploration and the simulated algorithm we propose the VEX Instance in Table III. The implementation has a **delay of 0.75 x Reference** and an **area of 0.95 x Reference**.

TABLE III: Proposed VEX Instance (solution)

Parameter	Name	Value
Issue Width	IW	7
No. of ALU	ALU	7
No. of Multipliers	Mpy	2
No. of Load/Store	LD/SR	2
No. of Memory Load Connections	MemLoad	2
No. of Memory Store Connections	MemStore	2
No. of Memory Pre-fetch Connections	MemPft	1
No. of General-purpose Registers	GR	26
No. of Branch Registers	BR	4

The results obtained from the Simulated Annealing algorithm are plotted in 1. This plot shows the position of the proposed VEX Instance marked with a red cross in the design space. The reference is marked with a black triangle. The curve validates the proposed solution as a Pareto optimal point. The application domain of Image Processing is not restricted to desktop based platforms alone and also finds wide implementation in medium sized embedded processing systems like cameras. This means that our design should not have a large area footprint. While the Image Processing

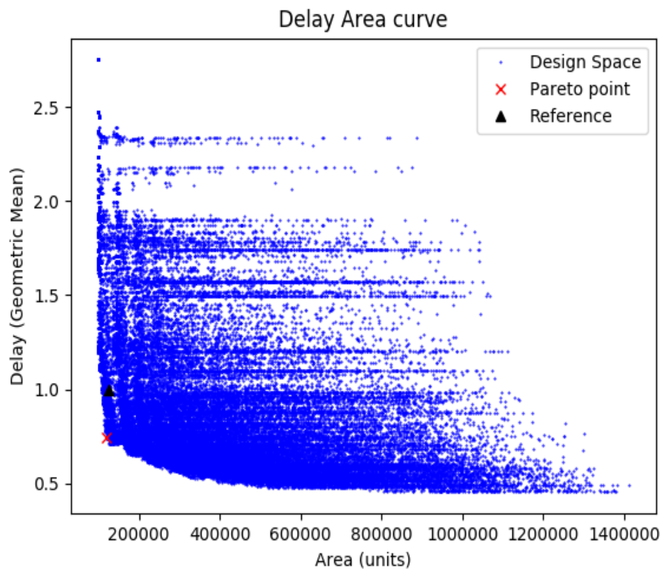

Fig. 1: Delay-Area Curve for Design Space (Pareto Curve)

Application necessitates high performance, the area penalty incurred for achieving this higher performance would render this architecture impractical for these small footprint devices and hence, our focus on low total area. Our solution finds this optimal point in the trade off by having a balanced number of memory and functional units.

VIII. Conclusion

The proposed architecture with the configuration parameters describe an optimum VEX Instance well placed in the Delay-Area trade off for both of the benchmarks. It also considers the area restrictions provided by the implementation platform. A single solution satisfying all of these criteria has been proposed.

IX. Reflection

This assignment served as a gateway to understanding the techniques of Design Space Exploration process for a VLIW processor. It gave us insights into the methodology utilized by computer architects to optimize hardware to accelerate performance while respecting the provided resource constraints. This strengthened our concepts of Instruction Level Parallelism in a processor and the fundamental limit that exists to such parallelism. It also made us validate our solution against a Pareto point curve.