Modern Computer Architectures – Lab Assignment 1

Group 18

Students: Saevar Steinn, Misael Hernandez, Tudor Voicu

Introduction

The goal of this assignment was to perform and document a Design Space Exploration process to determine a VLIW processor configuration optimized for two given applications according to specific metrics. The proposed programs for benchmarking of our configuration are two benchmarks related to similar encoding applications: v42 and POCSAG.

V42 is a modem encoding/decoding algorithm which extensively relies on recursive calls of a variable in loops. Regarding this aspect, the algorithm shows a sequential character because of iteration dependencies (current processing of a value relies on the previous value). The data cache in our configurations requires minimal size because the data is almost non-recurring in accesses – the characters to be encoded/decoded are random and do not depend on each other. The efficiency can be increased with the use of an accumulator (composing the 8bit message by adding and shifting a value) or a MAC block.

POCSAG is an asynchronous protocol used to transmit data to pagers. The implementation used in the PowerStone benchmark suite performs message synchronization, error correction and decodes both numeric and alphanumeric characters (4bit and 7bit). The processing flow of this algorithm is similar to v42 in terms of resources used: the application also relies heavily on shifts but also on complex arithmetic operations (at least for ECC and synchronization step).

Regarding the sequential character of the both algorithms, the parallelization is necessary (and mandatory) only when the "baud rate" exceeds the coding/decoding possibilities of the processor. E.g. if the baud of such a modem is set to 115200 bps, our configuration needs to perform 115200 full decodes per second. It is the same when talking about encodes. If this is the case (processor is powerful enough), for the case of decoding, the benefit of using a very wide issue CPU is minimal. Although the encoding can be parallelized (sequence of characters to be sent may not rely on each other, removing dependencies), the performance depends on the transmit buffer size (the write cache in our case – if it is too small, the CPU will stall until the messages are sent/flushed from the memory).

After doing extensive testing we settled to three different configurations based on single-cluster CPUs: a default configuration, the most powerful solution and our most efficient solution for gaining the most processing power. A multicore version has also been tested, but the parallel overhead proved to be very big, making the scenario unapproachable.

Configurations for V42

Default configuration

IssueWidth 1, MemLoad 1, MemStore 1, MemPft 1, Alu 2, Memory 1, Mpy, 1, CopySrc 1, CpyDst 1

The default and minimal configuration for running v42 shows a high percentage of stall cycles (12.3 %) with a very high hit rate for the cache (99.99 for instructions and 99.68 for the data). Therefore, some improvements are needed for the memory access and ILP – more processing blocks. The instruction memory operations take a considerable amount of time, 84.6 %, a figure which can be resolved by adding more issue paths.

```
Total Cycles:                           3080397 (6.16079 msec)
Stall Cycles:                            379312 ( 12.31%)
Executed operations:                    2544143

Instruction Memory Operations:
    Hits (Hit Rate):                    2608255 ( 99.99%)

Data Memory Operations:                   Cache
    Hits (Hit Rate):                     487504 ( 99.68%)

Percentage Bus Bandwidth Consumed:          2.50%
```

Result 1. Performance key-points for V42 using the Default Configuration.

Intermediate solution

IssueWidth 8, MemLoad 4, MemStore 4, MemPft 1, Alu 8, Memory 4, Mpy, 1, CopySrc 1, CpyDst 1

At this configuration level, the improvement is significant with a reduction of 30% in terms of computing time, although yielding a higher stall percentage due to big overhead in parallelism and sequential dependencies. Increasing the memory 4 times slightly increased the data hit rate (aprox. 0.1% improvement). The greatest improvement was seen in the memory access cycles (reduction of 36% vs the default scenario). Going beyond this level of hardware complexity, the improvement is 0, making this the most performant configuration. However, the area usage might be 8 or more times bigger and may not justify the 30 % improvement for this particular algorithm.

```
Total Cycles:                           2147878 (4.29576 msec)
Stall Cycles:                            378143 ( 17.61%)
Executed operations:                    2860932

Instruction Memory Operations:
    Hits (Hit Rate):                    1662552 ( 99.99%)

Data Memory Operations:                   Cache
    Hits (Hit Rate):                     574947 ( 99.73%)
```

Result 2 Performance key-points for V42 using the intermediate solution..

Final solution

IssueWidth 4, MemLoad 2, MemStore 1, MemPft 1, Alu 4, Memory 2, Mpy, 1, CopySrc 1, CpyDst 1

Due to the selected algorithm, the configuration for the ρVEX has been simplified very much where use of additional hardware was not necessary. Our goal was to have a little impact on performance with a lower hardware complexity comparing with the best configuration. The most efficient and powerful setting shows 1.4 % only in performance degradation vs the top configuration (presented before). We have set the memory load units to 4 and the memory store to 2 (writes are less frequent to memory than reads). Because of the extensive use of shifts and arithmetic operations, we needed 4 ALUs and 1 Multiply unit (increasing

the multipliers does not improve performance). This setting keeps the same number of stalls, but improves memory access and processing of operations.

```
Total Cycles:                          2176581 (4.35316 msec)
Stall Cycles:                           377728 ( 17.35%)
Executed operations:                   2832688

Instruction Memory Operations:
    Hits (Hit Rate):                   1678905 ( 99.99%)

Data Memory Operations:                  Cache
    Hits (Hit Rate):                   569365 ( 99.73%)
    Misses (Miss Rate):                  1553 (  0.27%)
```

Result 3 Performance key-points for V42 using our final solution..

Configurations for POCSAG

Default configuration

IssueWidth 1, MemLoad 1, MemStore 1, MemPft 1, Alu 2, Memory 1, Mpy, 1, CopySrc 1, CpyDst 1

The Pocsag algorithm of the Powerstone suite takes less compute time, in comparison with v42. As mentioned for the previous benchmark, this algorithm also has a sequential characteristic, making it less sensitive to a very wide issue architecture. However, the improvements in this case are more significant when going from the default (minimal) to the most powerful configuration. As noted before, an important fact can be seen by the very small (0.4%) fluctuation of the stall cycles (between the best and the worst case scenarios), suggesting the fact that the execution stalls are caused by memory accesses and, therefore, cache misses. This is due to the low characteristic of temporality of the input data.

The default configuration (*Result 4*) is the one provided as example. In this case the percentage of stall cycles is bigger than in v42, for the same configuration. The small size of the program doesn't put a big pressure on the cache. Due to high data locality feature for the decoding algorithm, like in the previous benchmark, the hit rate is 99 % for both data and instructions.

```
Total Cycles:                          55440 (0.11088 msec)
Stall Cycles:                          11186 ( 20.18%)
Executed operations:                   43083

Instruction Memory Operations:
Hits (Hit Rate):                       43252 ( 99.69%)

Data Memory Operations:                Cache
Hits (Hit Rate):                       6739 ( 98.99%)

Percentage Bus Bandwidth Consumed:     15.34%
```

Result 4 Performance key-points for POCSAG using the default configuration.

Intermediate solution

IssueWidth 4, MemLoad 2, MemStore 2, MemPft 2, Alu 6, Memory 2, Mpy, 1, CopySrc 1, CpyDst 1

The issue width has been increased to four and the number of ALUs to 6. Also, the memory load and store units have been increased accordingly in order not to starve the data path. This configuration shows an improvement of 88% comparing to the baseline and it represents the minimal configuration for which the lowest compute time has been obtained. The increase in ILP can also be observed by the doubling of the Percentage of Bus Bandwidth consumed (vs. de baseline), suggesting that this implementation of POCSAG is a more parallelizable algorithm than the V42.

```
Total Cycles:                              29164 (0.058328 msec)
Stall Cycles:                              11140 ( 38.20%)
Executed operations:                       43959

Instruction Memory Operations:
    Hits (Hit Rate):                       18766 ( 99.29%)

Data Memory Operations:                    Cache
    Hits (Hit Rate):                        6892 ( 99.01%)

Percentage Bus Bandwidth Consumed:         29.19%
```

Result 5 Performance key-points for POCSAG using the intermediate solution.

Final solution

IssueWidth 4, MemLoad 1, MemStore 1, MemPft 1, Alu 4, Memory 1, Mpy, 1, CopySrc 1, CpyDst 1

Our final solution shows a 1.7% decrease in performance versus the best performance and 81.4% increase over the baseline (default configuration). In comparison to the default configuration, only the issue-width and ALUs have been increased, all the other elements being kept to a minimum. Because the ALU takes 6 cycles/op, the improvement is justifiable. Also, using an ECC, the application has a bigger frequency of accesses to the memory.

```
Total Cycles:                              29669 (0.059338 msec)
Stall Cycles:                              11320 ( 38.15%)
Executed operations:                       44177

Instruction Memory Operations:
    Hits (Hit Rate):                       17613 ( 99.23%)
Data Memory Operations:
    Hits (Hit Rate):                        6888 ( 99.01%)

Percentage Bus Bandwidth Consumed:         29.00%
```

Result 6 Performance key-points for POCSAG using our final solution.

Reflections

Working for this lab assignment helped us gain valuable insights on the novel architecture of the ρVEX VLIW processor: we have learned about the advantages of a dynamic reconfiguration processor and about how its features can be managed in order to run programs more efficient. With the provided simulator, we have run our benchmarks and saw detailed reports on every run with different configuration settings. Being able to change the issue-width and number of functional units, we took into account the trade-offs of every scenario, converging to an optimal solution for the two studied algorithms.

Profiling the applications also facilitated our access to the low-level details of how hardware complexity influences the performance of a specific task and where increased hardware capabilities are necessary or not (e.g. how does cache size influence an encoding algorithm?). Furthermore, we have learned a valuable piece of knowledge: a multi-cluster architecture can actually add considerable overhead for sequential processing, making it less efficient in some scenarios then a single-cluster processor. By these means, after completing this assignment, we have a better idea on how to evaluate a good processor design as we gained important knowledge on how to do a match between a given application and a target processor, in terms of compatibility for satisfying efficiency and performance-related issues.