Energy Optimization of the CVA6: an FPGA-based RISC-V Processor for Deep Learning Applications

Justin Silver, Yara Al-Ahmad, Maxime Kahn, Jean-Baptiste Kammerer

***Abstract*—Within the context of a French national student competition, the RISC Makers team has optimized the energy consumption of an open-source RISC-V processor, known as Ariane CVA6, by targeting its cache subsystem. With the soft-core processor running the MNIST Convolutional Neural Network (CNN) deep learning classification program, an energy diminution of 2.03% was achieved. Execution time and FPGA resource was also slightly decreased.**

***Index Terms*—IEEE, RISC-V, FPGA, cache, Ariane, CVA6, low-energy, embedded**

I. Introduction

RISC-V is an Instruction Set Architecture (ISA) popular for its flexibility with regards to specific processor implementations, and its "open-source" nature. The OpenHW Group has leveraged this ISA to develop Ariane (CVA6): an application class 6-stage RISC-V CPU capable of booting Linux [1]. In a collaborative effort, Thales Group and others have modified the CVA6 core to be 32-bit compatible, and have also developed an FPGA implementation that targets Xilinx's Zybo-Z7 device. Leveraging the compact soft-core version of Ariane, Electrical Engineering graduate students all across France were invited to partake in a competition. The objective: reduce the energy consumption of the processor executing the MNIST deep learning classification program.

A. Baseline values

In order to properly compare the results between teams at the end of the competition, the following "baseline" values were obtained. These values relate to program performance in terms of execution time, energy consumption, and resource utilization. As per the competition's criteria, teams were constrained to not worsen execution time, and to not increase FPGA resource utilization by more than 10% for each logic category. Furthermore, students were encouraged to concentrate on CVA6's internal core, and to avoid modification to the processor's peripherals or main memory configuration.

At the end of the competition, results would be compared and the teams with the best energy improvements win.

B. Outline

This report details the work that the RISC Makers team from the University of Strasbourg has done. In Section II, a more in-depth analysis is made concerning the distribution of power consumption within the CVA6 core. In Section III and Section IV, the memory access patterns and cache subsystem are explored. With this knowledge, a plan to construct a new cache subsystem is presented in Section V. The results of this work are detailed in Section VI. Finally in Section VII, the rapport concludes with some closing thoughts regarding the competition and some ideas for future development that were not able to be completed within the competition's time frame.

J. Silver, Y. Al-Ahmad and M. Kahn are Electrical Engineering masters students and J. Kammerer is a professor at the University of Strasbourg in France

TABLE I
BASELINE ENERGY AND PERFORMANCE VALUES

Instructions	**Clock frequency (MHz)**
1725056	45
Total power consumption (mW)	**Total energy consumption (mJ**
307	14.32

TABLE II
DISTRIBUTION OF CVA6'S POWER CONSUMPTION (BASELINE)

Cache subsystem	**Frontend**	**Issue stage**	**Other**
42.9%	15.9%	30.2%	11%

TABLE III
BASELINE FPGA UTILIZATION VALUES

LUTs	**FFs**	**BRAMs**	**DSP blocks**
14675	9291	36	4

II. Energy analysis

Investigation on how to efficiently reduce the energy consumption of the CVA6 core began with a quantitative analysis, leveraging energy reports that were generated by Xilinx's FPGA tooling. These reports detailed the dynamic and static power consumption of each instantiated HDL module. Because the competition was only concerned with absolute energy diminution, it seemed judicious to target and optimize only the highest energy consuming blocks, since this is where the greatest energy gains could be achieved.

The cache subsystem stands out as an obvious optimisation target, being responsible for nearly half of the total power consumption[1]

Having identified an appropriate aspect of the CVA6 core to optimize, further analysis on the cache system was carried out.

[1]Values reference solely the instantiated Ariane CVA6 module (63 mW) and thus ignore peripheral and clock generation modules which are included in Table I

III. Cache analysis

As per the RISC-V ISA specifications [2], the CVA6 tracks certain events using hardware register "performance counters". By performing RTL simulation and logging the final values of these registers at the end of the execution of the MNIST application, the following data was gathered in Table IV, where I$ and D$ signifies the CPU's L1 instruction and data cache respectively.

TABLE IV
Memory related performance counters

Loads	Stores	I$ misses	D$ misses
636316	32233	788	4558

Another remark that can be made concerns the percentage of memory related instructions compared to the total number of instructions. Namely, more a great deal of instructions are memory related, and most of these are "loads" (load byte, load unsigned byte, load word, etc.). With these findings, further investigation into the cache's structure with hopes to identify energy-related optimizations was done. For the interested reader, the exact architectural details of CVA6's default cache subsystem have been treated by [3] and as such will be ignored here for brevity's sake.

Above all, it seemed logical to minimize the number of cache misses in an effort to reduce the energy consumption of the cache subsystem. Not only were cache misses energetically costly due to main memory transactions, but these misses also had the potential to result in long CPU stalls and pipeline flushes. These were considered "indirect" energy costs, because other units of the processor would contribute more to the overall power consumption.

Of course, because the CVA6 is an out-of-order execution processor, not all cache misses are guaranteed to generate these costly pipeline stalls, granted there is other work to be done within the core. However, as mentioned in Section**??**, with memory-related instructions taking up a great number of the total instructions in the MNIST application, pipeline stalls resulting from cache misses seemed inevitable.

IV. Memory access analysis

In an effort to better understand the nature of the aforementioned cache misses, investigations on the memory access patterns were carried out. The hope was to spot obvious stride patterns, such as array looping, which would give a clue on how to better optimize the cache in terms of capacity and associativity.

As can be seen in Figure 1, there are obvious sequential memory access patterns as a result of the MNIST program. This should not come as a major surprise, since the CNN application performs convolution on images using weighted filters, and thus requires the CPU to loop through pixel arrays and perform multiply/accumulate operations at various moments during the program execution.

With these results in hand, attempts were made to modify the existing cache system, by first varying the size of the D$ as well as the width of cache lines brought in from main

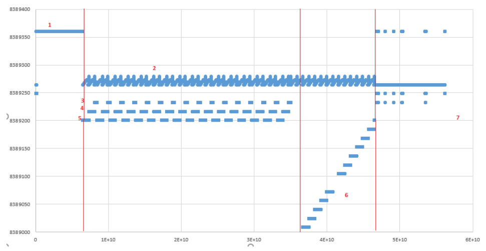

Fig. 1. Memory access address (hexadecimal) vs. time (s)

memory. In the latter case, doubling the cache line size resulted in nearly a 50% reduction of D$ misses. However, the price to pay for this was a doubling of the number of BRAMs needed to implement the cache.

Attempts were done made to modify set associativity, the replacement strategy, as well as the number of banking for the D$. However, the reconfiguration of the cache proved difficult. Even slight changes to the modules resulted in many simulation errors, as the interfaces between for example the memory request ports and the CPU request ports were tightly coupled.

At this point, despite little time remaining in the competition, it was decided to pursue the development of a new cache system. Since it would be designed from the ground up with the objective of reducing the energy consumption, certain decisions could be made that would accumulate the power saving and at the end, hopefully drastically reduce the consumption of the cache system.

V. RISC Makers data cache

One such example of improving upon the base cache subsystem concerned the way that tag comparisons were carried out. To reduce latency and because the original cache was 8-way set associative, all 8 ways of a single data bank were accessed in parallel to the tag comparison phase. Once a tag hit was determined, the appropriate way requested by the CPU would be mux selected. This approach has its advantages in terms of rapidity, but clearly results in higher energy consumption as compared to only accessing a single way, or even better yet, only accessing the data store if there was a tag hit. These considerations were kept in mind during the development of a new cache system.

To aid in this development, a finite state machine (FSM) diagram was constructed. Figure 2 illustrates the basic functioning of the system. Due to time constraints, it was impossible to fully implement this direct mapped, write-back, write-allocate data cache.

VI. Upstream integration changes and results

Although the new cache was not able to be integrated within the existing CVA6 system in time, during its development, several inconsistencies were found regarding the data and address bus sizes used to communicate between the CPU and main memory. Namely, there existed several 64 bit interconnects, whereas 32 bits would of sufficed (since the CVA6 had been reconfigured for 32 bits).

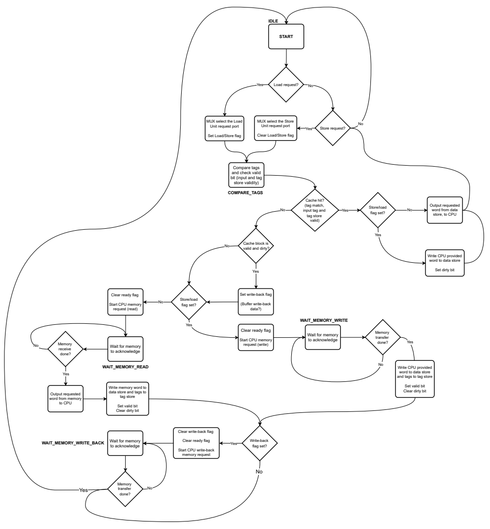
IDLE
START
Load request?
Yes
No
MUX select the Load Unit request port
Set Load/Store flag
MUX select the Store Unit request port
Clear Load/Store flag
Yes
Store request?
No
Compare tags and check valid bit (input and tag store validity)
COMPARE_TAGS
Cache hit? (tag match, input tag and tag store valid)
No
Yes
Store/load flag set?
No
Output requested word from data store, to CPU
Yes
Write CPU provided word to data store
Set dirty bit
Cache block is valid and dirty?
No
Yes
Set write-back flag
(Buffer write-back data?)
Store/load flag set?
No
Clear ready flag
Start CPU memory request (read)
Yes
Clear ready flag
Start CPU memory request (write)
WAIT_MEMORY_WRITE
Wait for memory to acknowledge
Memory transfer done?
Yes
No
Write CPU provided word to data store and tags to tag store
Set valid bit
Clear dirty bit
Wait for memory to acknowledge
WAIT_MEMORY_READ
Memory receive done?
No
Yes
Output requested word from memory to CPU
Write memory word to data store and tags to tag store
Set valid bit
Clear dirty bit
Write-back flag set?
Yes
No
Clear write-back flag
Clear ready flag
Start CPU write-back memory request
WAIT_MEMORY_WRITE_BACK
Wait for memory to acknowledge
Memory transfer done?
No
Yes

Fig. 2. FSM controller diagram for the RISC Makers data cache

At the same time, an exploration was done on the OpenHW Group GitHub repository that housed the CVA6 source code, which was originally forked for the present competition. As it turned out, there had been a fairly lengthy development gap between the two (roughly 2 years). As time was running out in the competition, it seemed reasonable to try and integrate the upstream changes in an attempt to improve upon the energy consumption of the Ariane core.

After successively integrating these upstream changes, an energy improvement of 2% was obtained, as well as various improvements regarding the execution time and FPGA resource utilization. As part of some of the upstream changes made over the time since the original fork, the 32 bit nature of the core was improved upon, pruning some of the unnecessary 64 bit buses and data structures.

TABLE V
ENERGY AND PERFORMANCE VALUES AFTER UPSTREAM CHANGES INTEGRATION

Instructions	**Clock frequency (MHz)**
1725056	45
Total power consumption (mW)	**Total energy consumption (mJ**
301	14.03

VII. CONCLUSION

A great deal of learning was done during this competition. Not only on the technical front due to limited documentation regarding the CVA6 core, but also concerning relational and communication aspects of working in a team. It proved quite difficult to juggle final year internships, end of year exams, and personal obligations while simultaneously maintaining an open line of communication within the team. This was exceptionally more difficult since all the team members were only able to work together remotely.

With that being said, although a significant decrease in the energy consumption of the CVA6 core was not achieved during the course of the competition, the authors are confident that one key to significantly reducing the CPU's energy consumption is by concentrating on its cache system. Once a configurable and modifiable architecture is put into place, more exotic changes could be made. Such examples include cache prefetching techniques [4] and/or non-blocking cache architectures [5].

ACKNOWLEDGMENT

REFERENCES

[1] F. Zaruba, S. Member, and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-ready 1.7GHz 64bit RISC-V Core in 22nm FDSOI Technology," Tech. Rep. [Online]. Available: https://github.com/pulp-platform/ariane
[2] A. Waterman and K. Asanovic, "The RISC-V Instruction Set Manual Volume I: Unprivileged ISA - Document Version 20191213," *RISC-V Foundation*, vol. I, 2019. [Online]. Available: https://riscv.org/technical/specifications/
[3] V. Martinoli, Y. Teglia, B. Abdellah, and R. Leveugle, "CVA6's Data cache: Structure and Behavior," Tech. Rep. [Online]. Available: https://arxiv.org/abs/2202.03749
[4] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," Tech. Rep.
[5] D. Kroft, "LOCKUP-FREE INSTRUCTION FETCH/PREFETCH CACHE ORGANIZATION," Tech. Rep.