Performance Modeling of CVA6 with Cycle-Based Simulation

Côme Allart[1,2], Jean Roch Coulon[1], André Sintzoff[1], Olivier Potin[2] and Jean-Baptiste Rigaud[2]

[1]Thales DIS, Meyreuil, France
[2]Mines Saint-Étienne, CEA, Leti, Centre CMP, F-13541 Gardanne France

Abstract

This paper is about CVA6 performance. We introduce a cycle-based model of CVA6 core to predict improvement of architecture modifications without modifying RTL code nor running a design frontend flow. 99.2 % accuracy has been measured. It is exploited to forecast 4.54 CoreMark/MHz for a 2-wide superscalar CVA6. Before that, we measure CVA6 CoreMark/MHz score using a CVA6 parameter to take advantage of fast memory.

Introduction

CVA6 – formerly Ariane [1] – is a 32- or 64-bit in-order, single-issue, 6-stage pipeline RISC-V application processor. It has been developed at ETH Zurich and is now maintained by OpenHW Group [2].

We focus on evaluation of the RV32IMACB variant of CVA6, through RTL simulation without taking into account any considerations of area, power or frequency.

We proceed to an evaluation of CVA6 performance using a commonly-used metric, the CoreMark/MHz [3]. Then, we propose a cycle-based model of CVA6 pipeline, and we measure its accuracy. Furthermore, we use this model to estimate performance gain of architecture modifications, assisting design decisions.

CVA6 CoreMark/MHz score

We are integrating CoreMark into CVA6 repository for reproducibility. CoreMark is a program running in a loop; each iteration represents about 270,000 executed instructions. A lot of cache and branch misses occur during the first iteration; fewer occur in the subsequent ones as caches and branch predictors warm up. CoreMark usually processes the score as the mean score of hundreds of iterations so that the misses from the first ones have less impact on the score. We choose to only perform two iterations and process score only from the second one to remove the impact of the warm-up phase while reducing the resource consumption of simulation.

Previous work [4] evaluated CVA6 at 2.08 CoreMark/MHz using a revision from February 2020. Since then, both CVA6 and GCC have evolved[1], improving overall performance by 34 % to reach 2.79 CoreMark/MHz as shown on Figure 1.

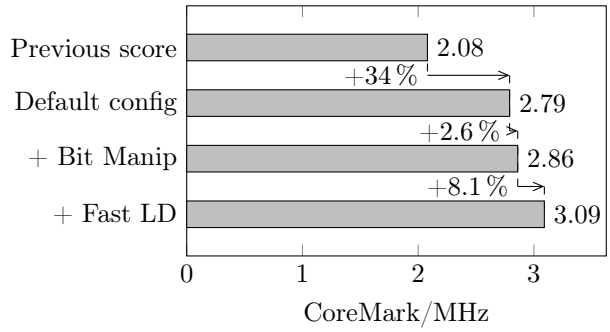

Figure 1: *Evolution of CVA6 performance*

Enabling bit manipulation extension raises performance to 2.86 CoreMark/MHz, as shown on Figure 1. Performance improvement of 2.6 % seems related to executed instructions count reduction of 3.6 %.

The insertion of a wait state during memory reads is not needed for all technologies. Disabling this insertion raises performance to 3.09 CoreMark/MHz, improving by 8.1 % the performance of CVA6.

Other complex modifications are possible to reach better performance. The model introduced in the following section would help to predict performance improvements of such modifications before starting any RTL development process.

Cycle-Based Model

We designed a cycle-based model of CVA6. The input of the model is a RISC-V Formal Interface (RVFI) trace from CVA6 simulation, which logs executed instructions. The output of the model is the annotation of predicted commit cycle number to the RVFI trace for each instruction.

Instructions are issued in order. Constraints are

[1] CVA6 commit 4f06aa6 and GCC 13.1.0 with -O3 -fno-tree-loop-distribute-patterns -nostdlib -nostartfiles -funroll-all-loops -falign-jumps=4 -falign-functions=16

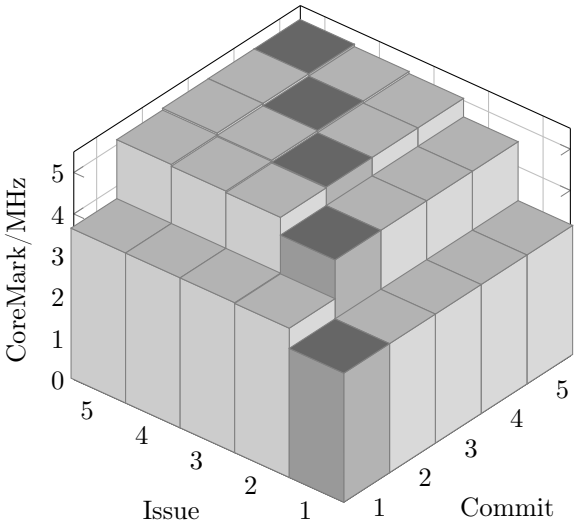

Figure 2: *Prediction provided by the model for an N-wide issue, M-wide commit CVA6.*

implemented to make issuing impossible in certain situations such as data hazards or branch misses. No cache misses are considered, so loads always have their best execution time.

To evaluate the accuracy of the model, we run it on the whole trace of a CoreMark program and perform matching measurement on the second CoreMark iteration. We measure the delay between each instruction commit and the previous one $\Delta t_i = t_i - t_{i-1}$ in the cycle-annotated output RVFI trace. For each instruction i, this delay is compared between the model and the RTL simulation – using a modified version of the RVFI tracer to annotate cycles. The accuracy is the proportion of matches, as Equation 1 shows.

$$\text{Accuracy} = \frac{\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}}{\#\{i\}} \qquad (1)$$

At the time of writing, the model accuracy is 99.2 %. It is measured on the second iteration of CoreMark.

Model exploitation

The model is applied to the second iteration of the CoreMark with different maximum numbers of issues and commits per cycle. The results of this experiment are provided on Figure 2. We observe:

- The performance seems to have a limit when the core becomes highly superscalar. It must be related to the in-order constraint; an out-of-order core would reduce the impact of RAW data hazards with dynamic scheduling [5].
- Unbalanced N-wide issue and M-wide commit core has almost the same performance as the balanced $\min(N, M)$-wide core; which likely has better area, power consumption and timing.

The model yields a performance of 4.54 CoreMark/MHz for a 2-wide superscalar CVA6.

Discussion

The memory technology is a condition to get the score of 3.09 CoreMark/MHz. The model is wrong in 0.8 % of executed instructions, including cache misses. A superscalar processor could reveal structural hazards that do not happen in a scalar core; therefore, the prediction of the model could differ with reality.

Conclusion

We performed CVA6 performance evaluation and reported 2.86 CoreMark/MHz with one latency cycle for memory reads and 3.09 CoreMark/MHz without it. A cycle-based model of CVA6 has been introduced, yielding accurate results for 99.2 % of executed instructions. It was used to evaluate the performance improvement of a 2-wide superscalar core, resulting in a performance of 4.54 CoreMark/MHz.

We plan to try several concrete superscalar architectures using the model to evaluate their performance according to the added structural hazards.

We might also add caches to the model and integrate it into Spike, the RISC-V ISA Simulator. It would help software developers, to measure performance of a given program running on CVA6 without RTL simulation.

Acknowledgements

These activities are supported by the TRISTAN project funded by the Key Digital Technologies Joint Undertaking (KDT JU) under grant agreements 101095947. The present action reflects only the authors' view; the European Commission and the JU are not responsible for any use that may be made of the information it contains.

References

[1] F. Zaruba et al. "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology". In: *IEEE Transactions on VLSI Systems* (2019).

[2] OpenHW Group. *CVA6 RISC-V CPU*. `https://github.com/openhwgroup/cva6`.

[3] EEMBC. *CoreMark*. `https://www.eembc.org/coremark/`.

[4] Alexander Dörflinger et al. "A comparative survey of open-source application-class RISC-V processor implementations". In: *18th ACM ICCF*. 2021.

[5] John L. Hennessy and David A. Patterson. *Computer architecture: a quantitative approach*. Sixth edition. Cambridge, MA: Morgan Kaufmann Publishers, 2019. ISBN: 9780128119051.