Application-Specific Reconfiguration of RISC-V BOOM Core

Rutvik Ajit Jere
The Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia, USA
rutvikj@vt.edu

Vinodh Kumar Reddy Kamasani
The Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia, USA
vinodh@vt.edu

Punitha Kamasani
The Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia, USA
punitha@vt.edu

***Abstract*— This project aims to explore optimized architecture configurations for specific applications on the RISC-V BOOM core architectures. By tuning various architectural parameters, we seek to improve performance for applications such as Machine Learning, Data Encryption, Graph Traversal, Finite Difference Method (Heat Equation Solver), Sorting Algorithm (Bubble Sort), and Fast Fourier Transform. This study will analyze a minimum of 7 different RISC-V small Boom core configuration parameters and evaluate the impact of each on the application's performance, measured primarily by Cycles per Instruction (CPI) and branch misprediction rate. We aim to document our changes to the baseline design and report and quantify why the changes made in the small BOOM core are superior to the baseline design.**

***Keywords*—RISC-V BOOM core, architecture optimization, application-specific configurations, pipeline depth, compute-intensive applications, memory-intensive applications, cycles per instruction (CPI), branch misprediction rate, branch prediction, performance evaluation.**

I. Introduction

Modern applications demand highly optimized hardware configurations tailored to specific workloads, especially given the rise of application-specific integrated circuits (ASICs). As processors like RISC-V offer flexibility in architectural configurations, our project focuses on identifying optimal RISC-V BOOM core configurations for applications such as Machine Learning, Data Encryption and Decryption, Graph Traversal, and Fast Fourier Transform (FFT). As a theme for our project, we envision our reconfigurations to be implemented in a low-power core employed in a remote application. Hence we decided to work with the small BOOM RISC-V core. The study's findings could enhance computing efficiency in fields like Machine Learning, Scientific Computing, Cybersecurity, and Embedded Systems.

II. Related Work

Research in architecture optimization often addresses parameters such as cache size, associativity, pipeline stages, and prefetching policies [2][3]. Previous studies indicate that parameter tuning can lead to substantial improvements in workload performance, particularly for scientific compute tasks. This project builds upon those findings to refine configurations as well as attempt to explore the memory-intensive applications on the RISC-V BOOM cores.

III. Proposed method

This project will involve the group exploring the baseline small BOOM core and subsequently configuring it to the aforementioned applications. We aim to divide the applications into "scientific computing" or "algorithm specific". For example, Machine Learning, Finite Difference Method (Heat Equation Solver), and FFT can be classified as scientific computing because we have used CPI as the evaluation metric before making changes to the small boom core's cache configuration and floating point values. Additionally, changes to these applications (such as the use of floating point values for math calculations, branch prediction, cache optimization for loops during integration, etc.) are also generally applicable in the field of scientific computing. This is because changing the ML algorithm would not force us to make wholesale changes to the proposed reconfiguration. On the other hand, applications like graph traversal and data encryption, which contain recursion, and various masking techniques, will require us to modify the core's branch prediction configuration. Furthermore, changing the graph traversal or the data encryption algorithm may force us to change the proposed reconfiguration as the functionality of those algorithms may change i.e., the reconfigurations proposed are specific to the algorithm specific application's program.

We have used custom programs as benchmarks to run on the small boom core and for design evaluation. We aim to study the changes and justify the new design over the baseline BOOM core design. We will vary at least 7 key parameters, including cache sizes (L1d, L1i), cache associativity, pipeline depth, branch prediction settings, and clock frequency.

IV. Evaluation Plan

A. Evaluation Metrics

The Primary metrics used for the evaluation are:

- **Cycles Per Instruction (CPI):** CPI was measured to quantify the efficiency of the execution after performing optimization. Lower CPI values indicate improved throughput.

- **Branch Misprediction Rate:** For applications involving loops and recursive calls, the efficiency of branch prediction was measured to assess the control flow.
- **Cache Performance:** Metrics such as L1 data cache hit/miss rates were analyzed to determine the impact of cache optimization on memory-intensive applications.

B. Benchmark Programs

Custom benchmarks were used to measure the effectiveness of the modifications. Benchmarks are tailored to specific applications and were designed to capture the general characteristics of these workloads. These benchmark programs were written in C and compiled using the gcc compiler.

C. Experimental Setup

Each benchmark was performed on a baseline BOOM configuration, followed by the optimized configuration of each application. It must be considered that we removed the custom branch predictor and considered the standard branch predictor already available in the small boom core, used a random replacement policy, and disabled prefetching as the baseline configuration. The performance results were compared against the baseline BOOM configuration to quantify the improvements. After calculating the speedup factors, the bottlenecks were identified to validate the effectiveness of the proposed optimizations.

V. Applications and Optimizations

A. Machine Learning (DNN Training):

We have identified that the DNN training relies heavily on matrix multiplications and loop iterations. To optimize this application, additional floating-point registers were introduced, and the latency of Single and Double Precision Multiply Accumulate FPUs (SFMA and DFMA) was reduced. To address the control flow, the TAGE-L branch predictor (BPD) was implemented, improving branch prediction accuracy. The performance metric used for this application was CPI, since it is a fine composite metric to the several modifications of the small boom config and quantifies the overall speed-up of the reconfiguration. These modifications led to a CPI reduction and speedup, which is shown in Table. 1

The reconfigurations include: the number of floating point physical registers, integer physical registers, number of ROB entries, the latency of SFMA/DFMA FPUs, TAGE-L BPD, LRU replacement policy, dcache sets, and the number of load entries in the Load Store Unit (LSU). The trade-off for the reconfigurations is the fact that this could increase the overall area of the CPU, which we tried to avoid by decreasing the number of integer physical registers and increasing the associativity of the dcache. We could sacrifice some of the branch prediction capabilities by using a smaller BPD such as the Gshare or the SWPred. Furthermore, the PseudoLRU could replace the standard LRU as it is a lighter version of the latter. However, we must consider the impact of the reconfigurations for this application over the drawbacks.

Table 1. Evaluation Results of Machine Learning

	CPI	**Speedup**
Baseline	2.060	1
Optimized	1.049	1.96

B. Data Encryption (SHA-256):

Secure Hash Algorithm – 256 encryption mainly involves bitwise operations, modular arithmetic, and repeated loop executions. The small boom architecture was modified to utilize the Instruction Level Parallelism (ILP) by increasing the decode width, integer issue width, and the number of load/store buffer entries. Additionally, the number of floating point registers was reduced to reallocate the resources efficiently. A prefetching mechanism was introduced in the L1 cache to anticipate the memory access for hashing functions to further improve the efficiency. The performance metric used for this application is CPI since it can help assess the variations in ILP.

The changes include: number of fetch width, number of ROB entries, L1 prefetcher, number of integer and floating point physical registers, integer instruction issue width, and increasing the number of load and store entries in the LSU. The trade-offs include the fact that the area increases, however, the reconfigurations in floating point registers, and the advantage of lesser power consumption (through lesser CPI) partially mitigate these disadvantages. Furthermore, changes to the dcache may include a replacement policy or modifications to the dcache size or associativity rather than the L1 prefetecher as it may lead to unwanted data in the cache.

Table 2. Evaluation Results of Data Encryption

Metric	**Baseline**	**Optimized**	**% change**
CPI	1.33	0.99	-25.56%
Instruction cache Misses (I$ miss)	100	99	-1.0%
Data Cache Misses (D$ regular miss)	46	53	+15.22%

There is a significant reduction in the number of cycles because of the L1 prefetcher. This prefetcher's behavior has increased data cache missed by 15.22%, largely due to prefetching unnecessary blocks.

C. Graph Traversal (Depth-first Search):

The Depth-first search application involves recursive function calls and loops, making it heavily dependent on branch prediction for efficient stack navigation. To minimize mispredictions, a combination of advanced branch predictors like TAGE-L and BPD (Alpha 21264) were implemented. These optimizations have resulted in a good speedup and reduction in branch mispredictions compared to baseline design. We studied the change in the misprediction rate for this application as we have focused our efforts on optimizing the branches and jumps in the program code.

Table 3. Evaluation Results of Graph Traversal

Config	CPI	Misprediction rate
Baseline	2.938	68.48%
TAGE-L	1.398	4.43%
2 BPD	1.413	4.72%
ALPHA 2126	1.607	8.58%

By implementing the TAGE-L BPD, the speed is increased by x2.102 and the misprediction rate was decreased by 91.81% compared to the baseline. However, the use of a smaller BPD may allow us to implement more reconfigurations to the BPD. However, due to the way the program is written, we argue that the best BPD can yield the most optimal results.

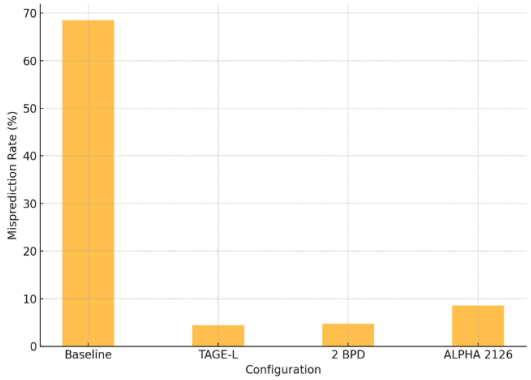

Figure 1. Misprediction Rate (%) Comparison

D. Finite Difference Method (Heat Equation Solver):

For the finite difference method, we noticed a multitude of floating point values due to the heavy use of differential equations. Prompting the use of more floating point physical registers, reducing the latency of the SFMA FLPUs, and increasing the issue width of the floating point instructions. Furthermore, due to various loops, function calls, and if conditions, an excellent branch predictor and dcache replacement policy would be necessary to jump from several branches and ensure that the relevant data is placed in the cache. For this application, we implemented the Double BPD and the PseudoLRU cache replacement policy. Additionally, we tried changing the MSHR of the dcache with no noticeable change in the performance metric. The performance metric used for this application was CPI, as it summarized the various assortment of changes to the small boom config and quantified the overall speed-up of the reconfiguration.

Table 4. Evaluation Results of Finite Difference Method

Config	CPI	Speedup
Baseline	2.353	1
Optimized	1.215	1.936

The reconfigurations include: the number of floating point physical registers, integer physical registers, number of ROB entries, the latency of SFMA/DFMA FPUs, Double BPD, PseudoLRU replacement policy, and the issue width of the floating point registers. The trade-offs for the reconfigurations is the fact that this could increase the overall area of the CPU, which we tried to avoid by decreasing the number of integer physical registers and the issue width of the integer instructions. We could sacrifice some of the branch prediction capabilities by using a smaller BPD such as the Gshare or the SWPred. Furthermore, the PseudoLRU despite having lesser time complexity than the standard LRU as it is a lighter version of the latter, does not perform as well. Since we aimed to improve the CPI of the benchmark, we decided to stick with the PseudoLRU to further reduce the number of cycles. However, we must consider the significant upgrade in the performance metric for this application over the drawbacks.

E. Fast Fourier Transform (FFT):

FFT workloads involve extensive trigonometric calculations, vector operations, and loop iterations. To optimize the performance, the TAGE-L branch predictor was integrated to improve branch prediction accuracy, reducing pipeline stalls. Floating-point register usage was also enhanced by increasing the number of available registers. The performance metric used for this application was CPI and the number of dcache misses, as it summarized the various assortment of changes to the small boom config and quantified the overall speed-up of the reconfiguration. The number of dcache misses will further justify the reconfigurations of the small boom's cache.

The reconfigurations include: the number of floating point physical registers, integer physical registers, number of ROB entries, the latency of SFMA/DFMA FPUs, TAGE-L BPD, LRU replacement policy, dcache sets. The trade-offs for the reconfigurations is the fact that this could increase the overall area of the CPU, which we tried to avoid by decreasing the number of integer physical registers and increasing the associativity of the dcache. We could sacrifice some of the branch prediction capabilities by using a smaller BPD such as the Gshare or the SWPred. Furthermore, the PseudoLRU could replace the standard LRU as it is a lighter version of the latter. However, we must consider the impact of the reconfigurations for this application over the drawbacks.

Table 5. Evaluation Results of Fast Fourier Transform

FFT Size	Metric	Before Optimization	After Optimization	Observations
16	CPI	1.479	1.393	-5.8%
	D$ Miss	15	15	No change
32	CPI	1.387	1.277	-7.9%
	D$ Miss	24	24	No change
64	CPI	1.341	1.211	-9.7%
	D$ Miss	45	43	-4.4%
128	CPI	1.336	1.184	-11.4%
	D$ Miss	111	83	-25.20%

After the optimization, there is a steady improvement in CPI is observed across all data sizes. A significant reduction in cache misses is observed for FFT size 128, dropping from 111 (before optimization) to 83 (after optimization).

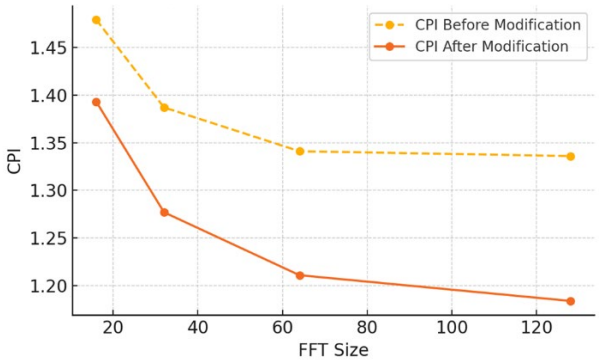

Figure 2. CPI Comparison Before and After Modification

F. Sorting Algorithm (Bubble Sort):

The bubble Sort Algorithm is highly iterative and memory–access intensive. So, the performance can be optimized by improving cache usage and branch prediction. TAGE-L Branch Predictor was enabled to improve control flow and minimize stalls caused due to loop iterations. We have added an inclusive L2 cache with a capacity of 64KB, 8 nways, and 1 nbank to reduce the instruction and data cache missed. The number of sets and Miss Status Holding Registers (MSHRs) were also increased. Finally, L1 cache prefetching was enabled, to minimize the iterations by anticipating and loading the data blocks.

The trade-offs for implementing the reconfigurations will include an increase in area owning to the inclusion of an extra layer of cache. Furthermore, we noticed that the including the prefetecher may result in a few cache mispredictions. Overall, despite the improvement in CPI, the dcache misses did not increase as expected which does not justify the reconfigurations of the cache. Although the benchmark program's code warranted it.

Table 6. Evaluation Results of Sorting Algorithm

Metric	Before Optimization	After Optimization	Observations
CPI	3.84	1.29	-66.41%
D$ Miss	12	11	-8.33%

VI. TIMELINE

i. **Nov. 1 - Nov. 7 :** Literature review and setup baseline RISC-V BOOM configuration.
- Status: completed on time.
- This phase was straight forward as planned.

ii. **Nov. 8 - Nov. 14 :** Test and tune core parameters on the baseline setup.
- Status: Delayed.
- The initial setup was done successfully, we got errors in the make files when trying to create multiple FFT sizes.

iii. **Nov. 15 - Nov. 21 :** Tune configurations for the scientific compute applications (Machine Learning, FFT, Heat Equation Solver).
- Status: Delayed by a few days due to the changes made in the CSR.scala and core.scala files to include number of branches, mispredictions, loads and stores.
- This fine-tuning took more time than expected due to additional iterations of the branch predictor configuration.

iv. **Nov. 22 - Nov. 28:** Tune configurations for algorithm specific applications (Graph Traversal, Encryption, Bubble Sort).
- Status: Delayed by a few days.
- Encryption was completed on time. However, we encountered challenges in Optimizing the Bubble sort algorithm and Graph Traversal.

v. **Nov. 29 - Dec. 5:** Run custom benchmarks and gather final performance data, prepare the presentation.
- Status: Completed on time.
- As the tuning was done successfully, the benchmarks were run as planned. We once again made some optimizations after running the benchmarks to further improve the performance.

vi. **Dec. 6 - Dec. 14:** Analyze results and draft findings, Final modifications.
- Status: Completed on time.
- The data collected during benchmarks made it easier to create the report.

VII. CONTRIBUTIONS OF INDIVIDUALS

For this project, we have distributed the applications to ensure a balanced workload. We used Custom Code for every application.

A. Rutvik Ajit Jere

- **Machine learning (DNN):** Configured floating-point registers and branch predictors for matrix-heavy workloads.
- **Graph Traversal (Depth First Search):** Modified the BOOM to utilize the branch prediction mechanisms to handle the recursive calls efficiently.
- Proposed the initial project idea and led efforts in refining the final project report.

B. Vinodh Kumar Reddy Kamasani

- **Data Encryption (SHA-256):** Tuned ILP, L1 prefetching, and resource reallocation (FPUs) for efficient hashing.

- **Fast Fourier Transform (FFT):** Reduced the floating-point unit latency and improved the cache performance for vector operations.
- Presented the project and created the initial project report draft.

C. *Punitha Kamasani*

- **Finite Difference Method (Heat Equation Solver):** Optimized pipeline depth and cache settings for iterative workloads.
- **Sorting Algorithm (Depth-First Search):** Modified the Branch Prediction mechanisms to handle recursive calls efficiently.
- Designed the presentation slides and generated the graphs for the visual representation of obtained results.

VIII. Design Goals and Challenges

The project aims to achieve a fine balance between performance improvement and resource efficiency, keeping in mind the limitations of power, area, and thermal management. Key challenges include identifying optimal configurations for varying workloads, avoiding resource contention, and minimizing latency in data-intensive applications.

IX. Impact on Broader Applications

Optimizing BOOM core configurations for specific applications may have broader implications, extending to areas like edge computing and real-time systems. By enhancing core efficiency, this project could contribute to making RISC-V cores more competitive for specific application domains, particularly in resource-constrained environments.

As we mentioned earlier, the theme of this project was to implement a low power reconfiguration of the small boom config to enable a lightweight execution of the application. However, we could use the large boom config to enhance the overall performance (while sacrificing area and power consumption). Furthermore, accelerators such as the Hwacha and SHA accelerators are readily available to enhance applications such as ML, FFT, and SHA. These accelerators are bolted along with the SoC to improve the vectorization (in the case of Hwacha) which can run independently of the large boom config. Upon implementing this, we noticed a speedup of 2.814 (when compared to the baseline small boom config). Additionally, we can use power consumption based performance metrics to quantify the reconfigurations rather than just CPI or branch misprediction rate.

References

[1] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. F. Wong, "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration," ACM Transactions on Design Automation of Electronic Systems, vol. 29, no. 1, Art. 20, pp. 1–23, Jan. 2024. doi: https://doi.org/10.1145/3630013.

[2] Z. Azad, M. S. Louis, L. Delshadtehrani, A. P. Ducimo, S. Gupta, P. Warden, V. J. Reddi, and A. M. Joshi, "An end-to-end RISC-V solution for ML on the edge using in-pipeline support," 2020.

[3] J. Ma et al., "Construction of RISC-V Lightweight Trusted Execution Environment Based on Hardware Extension," 2021 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), Shenyang, China, 2021, pp. 237-242 https://doi.org/10.1109/ICPICS52425.2021.9524261.

[4] S. Kalapothas, M. Galetakis, G. Flamis, F. Plessas, and P. Kitsos, "A Survey on RISC-V-Based Machine Learning Ecosystem," Information, vol. 14, no. 2, Art. 64, 2023. doi: https://doi.org/10.3390/info14020064.

[5] Wikipedia contributors, "RISC-V," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/RISC-V. [Accessed: Nov. 18, 2024]