Finding Optimal Hardware Configurations by Employing a Heuristics-Based Approach

Vyomakesh Puduru

CMPEN 431
12/15/2023

Table of Contents

1. Introduction

This project undertakes a comprehensive exploration of processor pipeline and memory hierarchy design spaces, crucial for enhancing performance and energy efficiency in the evolving field of computer architecture. Utilizing SimpleScalar, an advanced simulation tool, it navigates an 18-dimensional environment, employing a 5-benchmark suite to identify optimal processor configurations. The research goes beyond simulation, demanding a nuanced balance of architectural parameters. Central to this endeavor is the development and implementation of sophisticated heuristics targeting two optimization goals: performance maximization and energy efficiency. These goals are quantitatively assessed using the minimized geometric means of normalized energy-delay-squared product (ED^2P) and energy-delay product (EDP). The project aims to validate or challenge theoretical assumptions through a thorough yet targeted exploration of design space, providing valuable insights into the complexities and trade-offs inherent in computer architecture design.

2. Configuration Validation Function

Overview

The Configuration Validation Function is an essential part of the project. It serves to check the compatibility and feasibility of various parameter combinations within the defined 18-dimensional design space. The function evaluates each configuration against a set of constraints and filters out impractical or invalid configurations that do not conform to the requirements of the SimpleScalar tool and the logical boundaries of processor design.

Implementation

- Block Size and Cache Size: The function starts by validating the relationship between the cache parameters.
- Cache Latency Constraints: The function assesses the cache latencies to maintain a realistic balance between capacity and access time. It also accounts for the associativity of the cache and its impact on latency.
- Boundary Checks: It includes boundary checks for each dimension and logical checks that ensure coherence.

3. Heuristic Search Function Development

Approach for Performance Optimization(ED^2P)

Objective

The primary objective of this heuristic is to enhance the processor's performance by maximizing the Instruction Per Cycle (IPC). A high IPC indicates efficient utilization of the processor's resources, leading to improved performance. This section explains the reasoning behind the changes made to each configuration dimension to achieve this goal.

Dimension-Specific Strategies

1. Width [Dimension 0]:
   - Rationale: Having a higher fetch width will lead to increased throughput and increased performance.
   - Implementation: Narrowed the search space by excluding lower fetch width.
2. Fetchspeed [Dimension 1]:
   - Rationale: Since we are not bothered by energy efficiency, higher fetchspeed would increase performance.
   - Implementation: Assigned fecthspeed to 2.
3. Scheduling [Dimension 2]:
   - Rationale: As we learned from WR2, out-of-order scheduling provides a substantial boost to performance.
   - Implementation: Set Scheduling to out-of-order.
4. RUU Size [Dimension 3]:
   - Rationale: A larger RUU would be able to process more instructions, increasing efficiency.
   - Implementation: Narrowed the search space by excluding smaller RUU sizes.
5. LSQ Size [Dimension 4]:

- Rationale: Same as before. A larger LSQ would allow for more instructions to enter the pipeline bypassing stalled instructions.
- Implementation: Narrowed the search space by excluding smaller LSQ sizes.

6. Memports [Dimension 5]:
   - Rationale: As the focus is on performance, more memory ports would decrease the overall memory access time.
   - Implementation: Assigned memports to 2.

7-10. L1 Cache Sets and Ways [Dimension 6-9]:
   - Rationale: As just increasing the number of cache sets and ways has tradeoffs with performance, exploring different configurations of cache would be beneficial.
   - Implementation: Randomly select a configuration of L1 cache sets and ways.

8. Unified L2 Sets [Dimension 10]:
   - Rationale: As main memory accesses are time-consuming, having larger L2 sets would increase performance.
   - Implementation: Narrowed the search space by excluding lower UL2 sets.

9. Unifies L2 Blocksize [Dimension 11]:
   - Rationale: Due to the constraints applied to the width of the machine, certain UL2 blocksizes will automatically be rejected by the validation function. We can decrease the sample search space by removing these from consideration.
   - Implementation: Narrowed the search space by excluding invalid UL2 blocksizes.

10. Unified L2 Ways [Dimension 12]:
   - Rationale: For higher level cache, set associative offers higher performance than direct mapped.
   - Implementation: Excluded direct mapped from the search space.

11. TLB Sets [Dimension 13]:
   - Rationale: Due to the tradeoffs between overheads and performance, I choose not to apply any constraints on this dimension.
   - Implementation: Random Sampling within dimensional cardinality.

15-17. Latencies [Dimension 14-16]:
   - Rationale: As our generated cache configuration would have uniquely assigned latencies that will be validated later, we do not have to include these in our search space.
   - Implementation: Not included in search space.

18. Branch Predictor [Dimension 17]:

- Rationale: The perfect branch predictor is not feasible to implement, and any branch predictor is better for performance than NotTaken predictor.
- Implementation: Excluded the predictors mentioned above.

Approach for Energy Optimization (EDP)

Objective

The primary objective of this heuristic is to identify and evaluate configurations within the sample search space that are energy efficient while considering the balance between performance enhancements and associated overhead costs. This section explores the changes made to fulfill the objective.

Dimension-Specific Considerations

1. Width [Dimension 0]:
   - Rationale: Although higher width leads to higher performance, there is a point of diminishing returns for efficiency due to increasing overhead costs.
   - Implementation: Excluded the smallest and biggest width from the search space.
2. Fetchspeed [Dimension 1]:
   - Rationale: Since we are optimizing energy efficiency, we cannot say with certainty if one is better than the other.
   - Implementation: Random sampling within dimensional cardinality.
3. Scheduling [Dimension 2]:
   - Rationale: As we learned from WR2, out-of-order scheduling provides a substantial boost to performance and efficiency over in-order.
   - Implementation: Set Scheduling to out-of-order.
4. RUU Size [Dimension 3]:
   - Rationale: While a larger RUU would be able to process more instructions, there is a point after which there are diminishing returns concerning efficiency.
   - Implementation: Excluded the smallest, largest, and second largest RUU sizes from the search space.
5. LSQ Size [Dimension 4]:
   - Rationale: Same as previously mentioned, but since there aren't any large LSQ sizes, I didn't alter the search space.
   - Implementation: Random sampling within dimensional cardinality.
6. Memports [Dimension 5]:

- Rationale: As it is difficult to account for main memory access, I chose not to apply any constraints.
- Implementation: Random sampling within dimensional cardinality.

7-11. L1 Cache Sets and Ways [Dimension 6-9]:

- Rationale: As very large caches have high overhead costs, I chose to limit the number of sets. Set associativity also provides higher efficiency than direct mapped.
- Implementation: Excluded the largest set value and direct mapped way.

10. Unified L2 Sets [Dimension 10]:

- Rationale: As main memory accesses are time-consuming, having larger L2 sets would increase efficiency, but having large sets can be detrimental.
- Implementation: Narrowed the search space by excluding the 2 lowest and 2 highest UL2 sets.

11. Unifies L2 Blocksize [Dimension 11]:

- Rationale: Due to the constraints applied to the width of the machine, certain UL2 blocksizes will automatically be rejected by the validation function. We can decrease the sample search space by removing these from consideration.
- Implementation: Narrowed the search space by excluding invalid UL2 blocksizes.

12. Unified L2 Ways [Dimension 12]:

- Rationale: For higher level cache, set associative offers higher performance than direct mapped.
- Implementation: Excluded direct mapped from the search space.

13. TLB Sets [Dimension 13]:

- Rationale: Due to the tradeoffs between overheads and performance, I choose not to apply any constraints on this dimension.
- Implementation: Random Sampling.

15-18. Latencies [Dimension 14-16]:

- Rationale: As our generated cache configuration would have uniquely assigned latencies that will be validated later, we do not have to include these in our search space.
- Implementation: Not included in search space.

18. Branch Predictor [Dimension 17]:

- Rationale: The perfect branch predictor is not feasible to implement, and any branch predictor is better for performance than NotTaken predictor.
- Implementation: Excluded the predictors mentioned above.

While choosing dimensions, a certain amount of randomization is maintained to prevent local optima and investigate a wide range of configurations. Consequently, the heuristic may be able to find combinations that unexpectedly result in good performance thanks to this method.

4. Results and Analysis

Plot 1: [ED^2P]

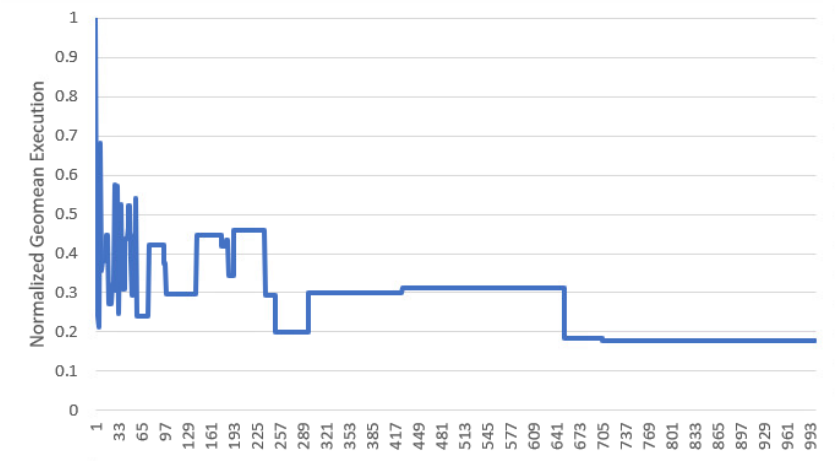

Figure 1 Normalized Geometric Mean of Execution Time

Plot 2: [EDP]

Figure 2 Normalized Geomean of Energy Delay Product

Plot 3:

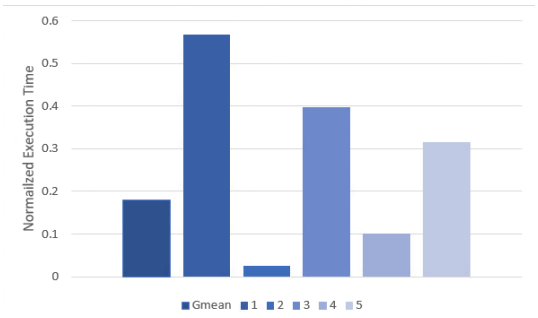

Figure 3 Normalized Execution Time of Best for performance configuration across all benchmarks.

Plot 4:

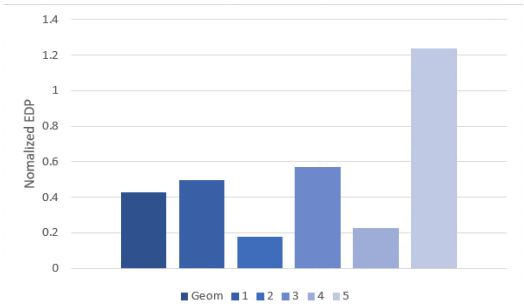

Figure 4 Normalized EDP

The graphical analysis of Plot1 and Plot2 indicates that configurations designed to optimize performance consistently surpass the baseline in terms of efficiency. Plot 1 shows that introducing stochastic elements in the search function helps mitigate local optima convergence, suggesting that further strategic adjustments, like relaxing initial constraints, could enhance optimization. Contrastingly, Plot 2 reveals a more extensive exploration of the sample space, despite also encountering issues with local optima. This implies that, unlike in Plot 1, modifying constraints might be more beneficial than merely relaxing them for a more effective search optimization process.

5. Best Design Points

Optimized for Performance

Dimension	Index	Value
Width	2	4
Fetchspeed	1	2
Scheduling	1	Out-of-order
RUU Size	4	64
LSQ Size	3	32
Memports	1	2
L1 D$ Sets	2	128
L1 D$ Ways	1	2
L1 I$ Sets	3	256
L1 I$ Ways	1	2
Unified L2 Sets	4	4096
Unified L2 Blocksize	2	64
Unified L2 Ways	1	2
TLB Sets	4	64
L1 D$ Latency	1	2
L1 I$ Latency	2	3
Unified L2 Latency	3	8
Branch Predictor	5	Combined

In alignment with the strategy aimed at optimizing for performance, the most effective configuration identified possesses substantial sizes for the Reorder Buffer Unit (RUU), Load Store Queue (LSQ), and cache dimensions. The configuration was meticulously crafted with a primary focus on achieving a higher Instructions Per Cycle (IPC) rate. This emphasis on IPC, coupled with the strategic decision not to impose an upper limit on the processor width, has been instrumental in shaping a configuration that excels in performance metrics. These insights into the configuration parameters — notably the RUU, LSQ, and cache sizes, as well as the processor width — underscore the nuanced understanding of system architecture that guided the optimization process. This comprehensive approach not only aligns with performance optimization goals but also offers a template for future endeavors in configuration tuning.

Optimized for Efficiency

Dimension	Index	Value
Width	2	4
Fetchspeed	1	2
Scheduling	1	Out-of-order
RUU Size	3	32
LSQ Size	3	32
Memports	1	2
L1 D$ Sets	2	128
L1 D$ Ways	1	2
L1 I$ Sets	3	256
L1 I$ Ways	1	2
Unified L2 Sets	3	2048
Unified L2 Blocksize	3	128
Unified L2 Ways	2	4
TLB Sets	4	64
L1 D$ Latency	1	2
L1 I$ Latency	2	3
Unified L2 Latency	5	10
Branch Predictor	4	2Level PAg

The configuration derived through the heuristic function can be aptly characterized as optimal, as it adeptly balances the trade-offs between hardware implementation overhead and the benefits derived from enhanced performance efficiency. This equilibrium is achieved through the adoption of a conservative approach towards the sizing of the Reorder Buffer Unit (RUU), Load Store Queue (LSQ), and Level 1 (L1) caches, effectively reducing system complexity. Concurrently, the configuration employs a sizeable Unified Level 2 (UL2) cache. This strategic choice significantly diminishes the frequency of main memory accesses, thereby yielding time and energy efficiencies. This configuration exemplifies a thoughtful integration of system components, prioritizing a harmony between hardware simplicity and performance optimization, illustrating a comprehensive and nuanced approach to system design.

6. Conclusion

In conclusion, the Configuration Validation Function, and the Heuristic Search Function, as expounded in this report, have been instrumental in effectively navigating the sample design space. The Heuristic Search Function, by judiciously adjusting parameters to better reflect the established criteria, substantially reduces the breadth of the potential search area. This focused strategy enables the formulation of designs that markedly surpass the baseline configuration in terms of optimization. Additionally, the Configuration Validation Function plays an indispensable role in this process, executing a critical sanity check on each proposed configuration. Weeding out configurations that fail to meet essential validity standards further sharpens the focus of the search space.

Future enhancements to the optimization process could involve refining the Configuration Validation Function to iteratively adjust and evaluate single parameters based on the most optimized configuration to date. This approach promises to incrementally refine the configuration, thereby improving search efficiency and achieving a more nuanced level of optimization.

7. References

- SimpleScalarSlides.pdf
- Provide code framework