CMPEN 431 Design Space Exploration

Samit Madatanapalli

Spring 2025

Dr. Sampson & Alexander Devic

Introduction

Different processors are architectured with different design goals, some optimize for performance, others aim for efficiency, costs, area, or even combinations of these factors. Achieving these set goals require exploring multidimensional spaces of core microarchitecture, memory hierarchies, and cache configurations. This paper delivers a heuristic proposal to find high performing configurations under four different optimizations: Energy savings (EDP), Energy efficiency per area ($EDAP$), Performance sensitive (ED^2P), and a balance of all(ED^2AP). With a total 18 different dimensions with up to 10 indices per dimension, finding the optimal configurations would be near impossible. Therefore I designed a search algorithm that uniquely adapts for each of the different optimizations. The following sections will explore more on the design space, development of heuristic strategies, results, and a final analysis.

Design Space

To make full use of the limited evaluations, the validation function is crucial for ensuring that only architecturally valid configurations are considered, allowing the algorithm to prioritize meaningful design points. To simplify the implementation, I first decoded the encoded index values into corresponding microarchitectural parameters, to enforce classes of constraints: cardinality, structural, and realism checks. Cardinality checks ensured each dimension was within its allowed minimum to maximum range. Structural constraints verified compatibility with dependent dimensions, for example $L2 < L1D + L1I$ and $L2\ block\ size < 2 \times L1\ block\ size$. Realism checks filtered out inefficient designs like mismatched cache latencies, bottleneck scenarios like $width > 2 \times fetch\ speed$, and other unrealistic HW constraints.

Now, after filtering out invalid configurations, the next step is to narrow down configurations toward individual optimizations. Since exhaustively iterating through each possible configuration is infeasible especially under a 1000 evaluation limit, I narrowed it down by targeting heuristics that exploit architectural characteristics for each optimization: EDP, $EDAP$, ED^2P, and ED^2AP.

EDP, which emphasizes efficiency, naturally led me to exploit simplicity (narrow, in order, minimal HW). As supported by lectures and energy tables in prompt, narrow in order processors consume significantly less energy per cycle. For example, the narrowest in order pipeline only consumes $8pJ$, whereas the widest out of order processors consume $27pJ$. Similarly smaller HW, like cache at $8KB$ only use $20pJ$ at $0.125mW$ while main memory consumes $20nJ$ at $512mW$ per access.

$EDAP$, which emphasizes area, led me to exploit low footprint configurations that still look at energy and delay. I primarily selected an in order narrow pipeline since this significantly reduces area overhead. Additionally to what was learnt in class, the in order pipelines scale at $\frac{width^2}{2} mm^2$, while out of order pipelines begin at a $4mm^2$and scales exponentially.

ED^2P, which emphasizes performance, led me to prioritize wide pipelines with fast fetch speeds and out of order execution. The performance table shows how in order processors take less time per clock cycle than dynamic with the same fetch width, but as learnt in class, out of order processors exploit ILP more aggressively and though there is a 5-10 ps increase, the processors can retire more instructions per cycle since it reduces pipeline stalls and hazards. Since ED^2P enforces delay exponentially, improving IPC is more impactful than only saving a few picoseconds of cycle time.

ED^2AP, which is a mix of all the previous objectives would require a hybrid approach. Since, all components dominate, exploring a wide range of configurations that balance performance, efficiency, and area without sacrificing any single metric.

Iterations of Heuristic

Iteration 1 - Greedy Search:

My first approach was a greedy, sequential approach where I initialized all dimensions to index 0, then incrementally and greedily selected space one dimension at a time. For example, in the first dimension (width), I tested all indices and locked in the index with better results. This continued for all dimensions, narrowing down to a "greedy" configuration. However, after testing, this was extremely exhaustive and I came to realize that I would be leaving many great configurations out simply due to a greedy early stage decision.

Iteration 2 - Neighbor Search

To fix the problem of bad early stage decisions and to explore more diversely, I switched to a more fair strategy that gave each dimension a chance. For each of the dimensions, I would increment its index and enclose it by modulating the valid choices:
$configuration[dimV] = (configuration[dimV] + 1) \% validChoices$

This generates a new "neighbor" of the previous iteration changing a single dimension, so over many configurations, it improves the diversity of the search with better results. However, I noticed when running, that there would be invalid configurations which would stall progress, therefore to fix this, iteration 3 introduced an embedded validator.

Iteration 3 - Retrieval Configuration

This iteration introduced an embedded validating checker to help reduce stalls. I would invoke the validateConfiguration() and if it returned 0, I would know that the configuration curated in iteration 2 failed and I would return a retreated configuration using a completely random configuration which I know would be less exhaustive on the system. While the 431projectUtils.cpp already rejects invalid configurations before finalizing them, my embedded validating checker would act earlier in the process by reducing redundant attempts within the proposal function.

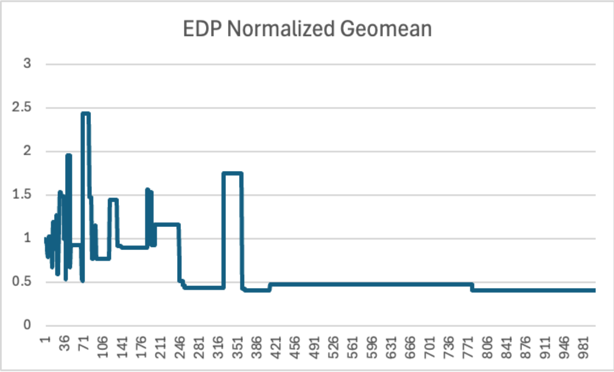
Iteration 4 - Optimization based filtering

To individually exploit each of the four optimizations instead of treating all dimensions equally, I tuned specific dimensions to take advantage of architectural characteristics that aligned with the selected optimized goal.

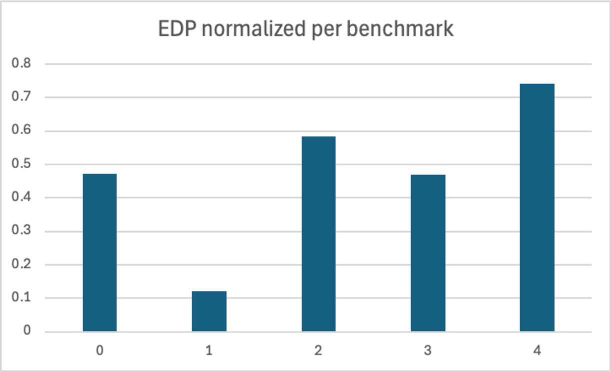
For ED^2P, I biased the configuration for high throughput, so large width, high fetch speed, and out of order execution. This aligned with the optimization with an emphasis on delay/performance rather than power. For EDP, I emphasized on the configuration to bias toward the lower half of the index space in each of the dimensions. This heuristic exploited simpler, more efficient hardware such as narrow in order processors with minimized HW. For $EDAP$, I emphasized on narrow in order processors and hardcoded dimensions like width, fetch speed, and scheduling. Unlike EDP, which randomly biases toward the lower half of each dimension, $EDAP$ selectively targets values that minimize physical footprint. For ED^2AP, I did not bias a single metric, instead I allowed all dimensions to be tunable which would allow a wide random exploration across all performance, area, and energy.

Results and Analysis

EDP



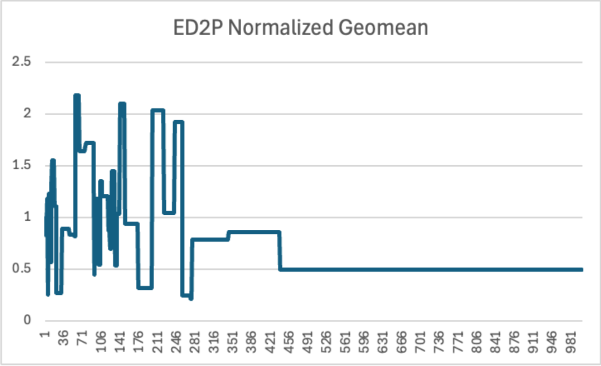
x-axis: iterations y-axis: normalized geomean



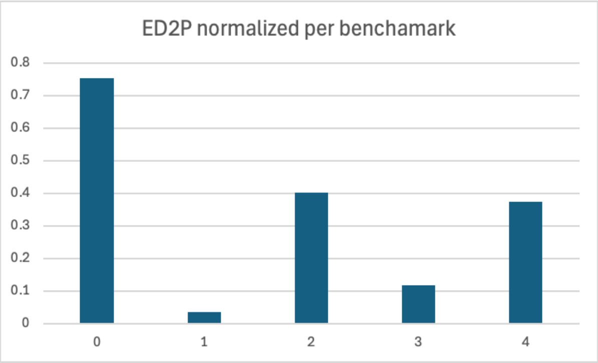
x-axis: benchmarks y-axis: normalized geomean

The early spike in the EDP plot shows the initial phase to explore inefficient configurations, but as more iterations were evaluated, the heuristic started to converge toward simpler and low energy efficient values. The best EDP configuration achieved the lowest value on benchmark 1 and the highest on benchmark 4.

ED^2P



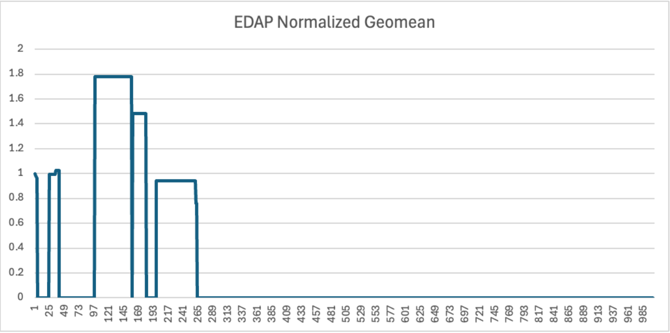
x-axis: iterations y-axis: normalized geomean



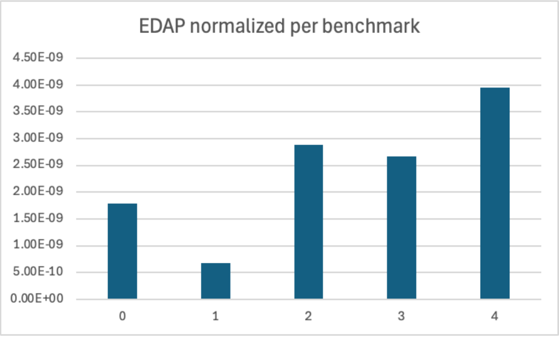
x-axis: benchmarks y-axis: normalized geomean

The early spike in the ED^2P plot shows the initial phase to explore inefficient configuration, but as more iterations were evaluated, the heuristic started to converge toward high throughput designs. The best ED^2P configuration prioritized a wider out of order processor achieving the lowest value on benchmark 1 and the highest on benchmark 0.

$EDAP$



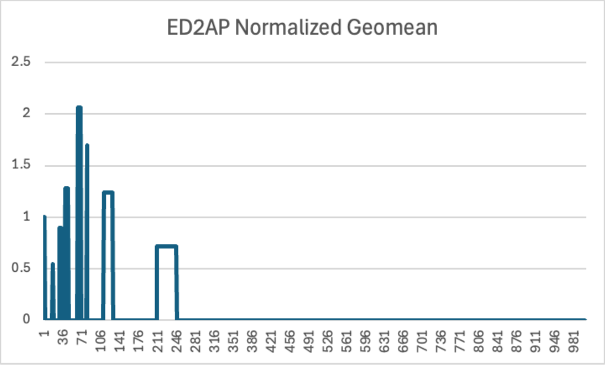
x-axis: iterations y-axis: normalized geomean



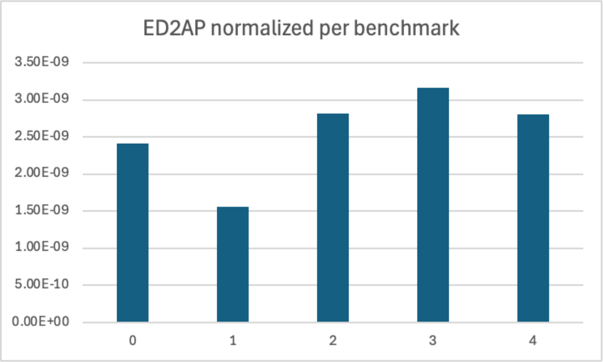
x-axis: benchmarks y-axis: normalized geomean

$EDAP$ normalized geomean had a spiky start but as more iterations were evaluated, the heuristic started to converge toward a minimized $EDAP$ around $1.07\ \times\ 10^{-8}\ Js\ mm^{2}$. The best $EDAP$ configuration used a narrow in order processor and low indexed dimensions to minimize a physical footprint without a huge loss in performance. This configuration achieved the lowest value on benchmark 1 and the highest on benchmark 4.

$ED^{2}AP$



x-axis: iterations y-axis: normalized geomean



x-axis: benchmarks y-axis: normalized geomean

Similar to the other optimizations $ED^{2}AP$, the normalized geomean had a spiky start but as more iterations were evaluated, the heuristic started to converge toward a minimized $ED^{2}AP$ around $1.87\ \times\ 10^{-8}\ Js^{2}\ mm^{2}$. The best $ED^{2}AP$ configuration which factors in energy, performance, and area resulted in a narrow in order processor with scattered HW configurations. It achieved the lowest value on benchmark 1 and the highest on benchmark 3.

All of the optimizations were normalized to the baseline (0, 0, 0, 0, 0, 0, 5, 0, 5, 0, 2, 2, 2, 3, 0, 0, 3, 0).

	EDP	ED^2P	$EDAP$	ED^2AP	Baseline
0: Width	2	4	1	1	1
1: Fetch speed	2	2	2	2	1
2: Scheduling	out of order	out of order	in order	in order	in order
3: RUU Size	128	64	16	8	4
4: LSQ Size	32	16	8	8	4
5: Memports	1	2	2	1	1
6: L1D sets	512	512	512	512	1024
7: L1D ways	1	1	2	2	1
8: L1I sets	1024	128	512	512	1024
9: L1I ways	2	4	4	2	1
10: Unified L2 sets	2048	256	512	512	1024
11: Unified L2 block size	128	64	128	64	64
12: Unified L2 ways	2	8	2	4	4
13: TLB sets	16	8	16	64	32
14: L1D latency	1	2	2	2	1
15: L1I latency	4	4	4	2	1
16: Unified L2 latency	8	8	6	7	8
17: Branch predictor	perfect	Bimodal	2 level GAp	2 Level GAp	perfect

Conclusion

In this project, I explored a large space of processor configurations and designed heuristics for different optimization goals: EDP, $EDAP$, ED^2P, and ED^2AP. By exploiting architectural characteristics and tailoring the algorithm, I was able to find the “best” configuration across different optimizations. This project helped me better understand how microarchitectural choices affect performance, energy, and area in the real world systems.