ECS201A - ASSIGNMENT3

Hezhi Xie, Mo Lyu
Email: hezxie@ucdavis.edu, molyu@ucdavis.edu
Github UserName: helenxie-bit, MOLl02
Github TeamName: architect

1 ANALYSIS AND SIMULATION

1. What will be the average speed-up of `HW3BigCore` over `HW3LittleCore`? Can you predict an upper bound using your pipeline parameters? **Hint**: You can predict an upper bound for the speed-up using Amdahl's law with optimistic values.

Golden Cove and Gracemont are microarchitectures developed by Intel. Both feature out-of-order execution, maximizing instruction-level parallelism and overall performance. Golden Cove is designed for high-performance computing with wider execution pipelines, larger cache sizes, enhanced branch prediction, and improved vector processing. Gracemont targets power-efficient and low-power computing, balancing performance and energy efficiency. Their related parameters are shown below:

Processor	Width	Reorder Buffer Size	Number of Physical Registers (Integer)	Number of Physical Registers (Floating Point)
Golden Cove	Decode: 6 Rename: 6	512	280	332
Gracemont	Decode: Dual 3 Rename: 5	256	214	207

Table 1: Parameters of Golden Cove and Gracemont

Referring to Intel Golden Cove and Gracemont architectures separately, we configured HW3BigCore and HW3LittleCore with the following parameters:

Processor	Width	Reorder Buffer Size	Number of Physical Registers (Integer)	Number of Physical Registers (Floating Point)
BigCore	6	512	280	332
LittleCore	3	128	100	100

Table 2: Parameters of Simulated Processors

According to Amdahl's law, the overall performance improvement gained by optimizing a single part of a system is limited by the proportion of the part that cannot be improved, which can be formulated in the following way:

$$S = \frac{1}{(1-p) + \frac{p}{s}}$$

Where:

- S is the overall speedup of the execution of the whole task;
- p is the proportion of the part benefiting from improved resources originally occupied;
- s is the speedup of the part of the task that benefits from improved system resources.

Therefore, to obtain the upper bound for the speed-up, we solely focus on enhancing the pipeline width while disregarding other constraints (such as reorder buffer size, number of physical registers). Additionally, we assume that all components of the system can be enhanced with increased pipeline width. Thus, $p = 1$ and $s = 6/3 = 2$. According to the formula of Amdahl's Law, the overall speed-up equals 2, indicating that the upper bound for the average speed-up of `HW3BigCore` over `HW3LittleCore` is **2**.

2. Do you think all the workloads will experience the same speed-up between `HW3BigCore` and `HW3LittleCore`?

We don't think all the workloads will experience the same speed-up between `HW3BigCore` and `HW3LittleCore`.

This is because different workloads have varying instruction arrangements and dependencies, which will lead to different pipeline results. For example, despite having higher pipeline width, reorder buffer size, and physical registers under `HW3BigCore`, certain part of the workloads may not be fully parallelized due to data dependencies. Consequently, the proportion of workloads benefiting from these enhancements differs. Therefore, the speed-up between `HW3BigCore` and `HW3LittleCore` varies across different workloads.

2 STEP I: PERFORMANCE COMPARISON

1. What is the speed up of `HW3BigCore` over `HW3LittleCore` for each workload?

The speedups of `HW3BigCore` over `HW3LittleCore` for each workload are shown below:

Workload	BigCore_simTicks	LittleCore_simTicks	Speedup
Matrix Multiplication	11,222,717,500	11,224,568,500	**1.0002**
Breadth First Search (BFS)	26,256,815,000	34,293,120,500	**1.3061**
Bubble Sort	4,623,909,500	4,652,532,000	**1.0062**
N-Queens	56,046,536,500	66,156,290,000	**1.1804**

Table 3: Speedups of Different Workloads

2. What is the average improvement in IPC of `HW3BigCore` compared to `HW3LittleCore` over all the workloads? **CAUTION**: Make sure to use the correct mean to report average IPC improvement.

We choose to use geometric mean to compute the average improvement in IPC, because IPC improvements are unitless, which makes arithmetic comparisons meaningless. With the IPC improvements for the four workloads listed in Table 4, the calculated average improvement in IPC is **1.1161** using the geometric mean formula.

Workload	BigCore_IPC	LittleCore_IPC	Improvement
Matrix Multiplication	0.7536	0.7535	**1.0002**
Breadth First Search (BFS)	0.6047	0.4630	**1.3061**
Bubble Sort	1.3679	1.3595	**1.0062**
N-Queens	1.3716	1.1620	**1.1804**
Average Improvement in IPC			**1.1161**

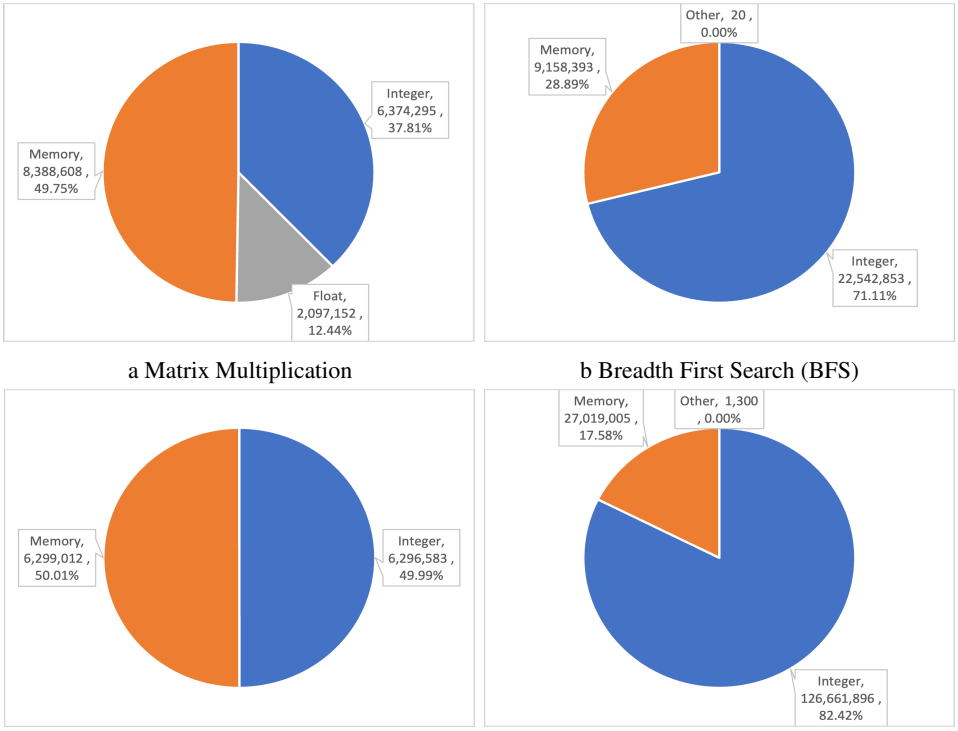
Table 4: Improvements in IPC of Different Workloads

3. Some workloads show more speedup than others. Which workloads show high speedup, and which show low speedup? Look at the benchmark code (both the `.c` and `.s` files may be useful) and speculate the algorithm characteristics that influence the IPC difference between `HW3BigCore` and `HW3LittleCore`. What characteristics do applications have that lead to low performance improvement and what characteristics lead to high performance improvement?

Based on the results presented in Table 3, the sequence of speedup for different workloads is as follows: **Breadth First Search (BFS)** $>$ **N-Queens** $>$ **Bubble Sort** $>$ **Matrix Multiplication**. This indicates that Breadth First Search (BFS) and N-Queens exhibit relatively high speedup, while Bubble Sort and Matrix Multiplication show relatively low speedup.

Analyzing the benchmark code, we identify several algorithm characteristics that may influence the IPC difference between `HW3BigCore` and `HW3LittleCore`:

- **Instruction-Level Parallelism (ILP)**: Algorithms with higher ILP benefit more from out-of-order execution capabilities. Thus, when utilizing `HW3BigCore`, workloads with higher ILP can execute more instructions simultaneously, leading to a higher IPC improvement. For instance, the ROI part of the Breadth First Search (BFS) workload includes many independent instructions that can be executed concurrently, resulting in greater ILP utilization and IPC improvement.
- **Compute-Intensive vs. Memory-Intensive Workloads**: Compute-intensive algorithms performing complex arithmetic operations may exhibit higher IPC improvements on `HW3BigCore`, which features wider execution units and more execution resources. Conversely, memory-intensive algorithms may see more modest IPC improvements on `HW3BigCore` due to memory access latencies and bottlenecks. Analysis of the committ instruction types reveals that the Breadth First Search (BFS) and N-Queens workloads consist of a higher proportion of arithmetic operations, leading to greater IPC improvement. In contrast, the other two workloads consist of a higher proportion of memory operations, resulting in a lower IPC improvement.



a Matrix Multiplication

b Breadth First Search (BFS)

c Bubble Sort

d N-Queens

Figure 1: Distribution of Commit Instruction Types under Different Workloads

- **Branch Predictability**: Algorithms with predictable branching behavior benefit from branch prediction mechanisms in out-of-order processors, resulting in fewer pipeline stalls and higher IPC. The ROI part of the Breadth First Search (BFS) workload contains more branches, which may benefit more from `HW3BigCore` usage.

Therefore, lower ILP, increased memory intensity, and reduced branch predictability result in more modest performance improvements, while higher ILP, greater compute intensity, and improved branch predictability contribute to high performance enhancements.

4. Which workload has the highest IPC for `HW3BigCore`? What is unique about this workload?

According to the data in Table 4, the **N-Queens** workload exhibits the highest IPC for `HW3BigCore`.

Based on Figure 1d, the **N-Queens** workload **predominantly consists of arithmetic operations**, with integer operations making up 82.42% of its instructions. Workloads characterized by a larger share of straightforward instructions, such as arithmetic operations, typically exhibit higher IPC due to the efficient execution of these instructions. This is particularly noticeable when utilizing `HW3BigCore`, where compute-intensive algorithms tend to demonstrate higher IPC due to wider execution units and more execution resources.

Moreover, the **BubbleSort** workload also exhibits a high IPC for `HW3BigCore`. We think this can be attributed to its **low computational complexity per iteration** and **high data locality**. Each iteration of the bubble sort's inner loop involves simple comparisons and swaps, which translate into basic assembly instructions such as lw (load word), sw (store word), and basic arithmetic operations. These instructions typically have low execution latencies and can often be executed in a single cycle. Moreover, despite BubbleSort's relatively high pro-

portion of memory operations, it accesses array elements in a predictable linear manner, resulting in high spatial and temporal locality. This access pattern effectively utilizes the CPU cache, leading to fewer cache misses and quicker data retrieval from main memory.

3 STEP II: MEDIUM CORE

1. If you were to use the speed-up under only one workload from the four workloads you used before, which workload would you choose? Why?

The choice of workload for speed-up would be **Breadth First Search (BFS)**, based on the data provided.

This decision is guided by the highest speedup factor of 1.3061 for BFS when comparing `HW3BigCore` to `HW3LittleCore`, as well as the considerable room for improvement signified by its lower IPC on the big core. The high improvement signals that the workload benefits greatly from the core's advanced features, clearly demonstrating the core's influence on enhancing computational efficiency. Meanwhile, a low IPC implies that the workload is not yet encountering a performance bottleneck on the new core, suggesting that there is still headroom for further improvements.

Additionally, the BFS workload is characterized by a high percentage of integer ALU operations (71.11%) and memory reads (27.04%), indicating that it is both compute and memory-bound. This duality suggests that optimizations such as improving cache utilization and enhancing integer ALU efficiency could lead to significant performance gains. The performance metrics and micro-operations distribution strongly advocate for prioritizing BFS for speed-up endeavors. The micro-operations distribution for the workloads is represented in the table below:

Workload	IntAlu	MemRead	MemWrite	FloatMultAcc
Matmul	37.81%	37.31%	12.44%	12.44%
BFS	71.11%	27.04%	1.85%	-
BubbleSort	49.99%	33.28%	16.73%	-
N-Queens	82.42%	17.12%	0.46%	-

Table 5: Distribution of Micro-Operations

2. Create a **pareto frontier** plot with cost on the y-axis and performance on the x-axis. This will be a scatter plot with 6 points: the two "big" and "LITTLE" cores as well as your 4 middle-ground designs. Then, "connect the dots" on the "best" designs.

According to the analysis in question 1, let's first examine the Pareto frontier of the **Breadth First Search (BFS)** workload, which is shown in Figure 2.

Case	Width	ROB Size	Int-Regs	FP-Regs	Area Score	simTicks BFS	IPC
Little	3	128	100	100	4572	34,293,120,500	0.4630
Medium1	4	256	256	256	17424	26,958,729,500	0.5890
Medium2	5	256	128	128	19988	27,791,698,000	0.5713
Medium3	5	256	256	256	26644	26,681,495,000	0.5951
Medium4	5	512	256	256	39956	26,375,430,000	0.6020
BIG	6	512	280	332	60556	26,256,815,000	0.6047

Table 6: Performance and Configuration Metrics for Medium Cores

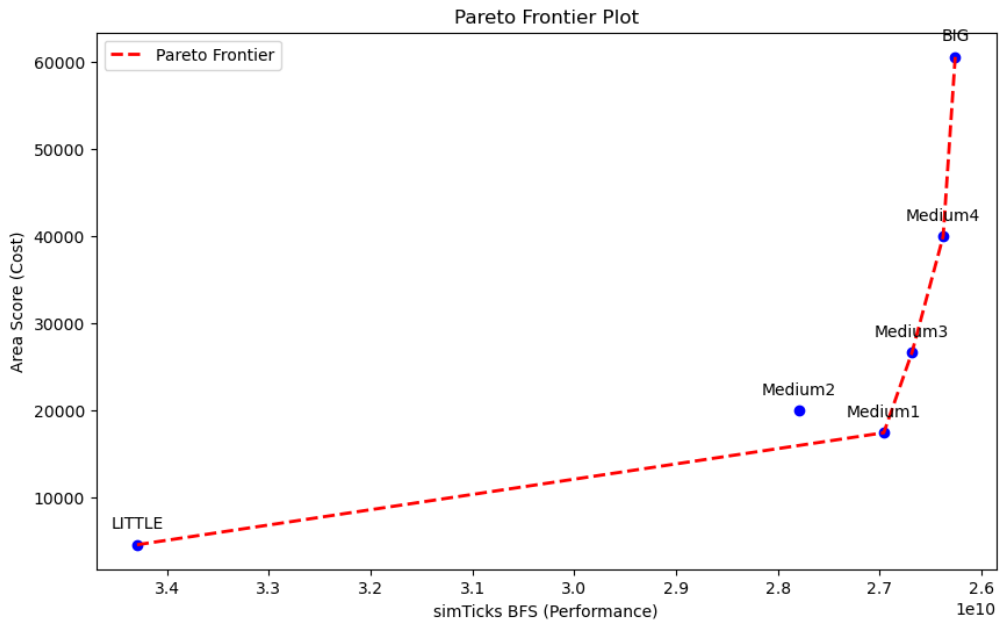


Figure 2: Pareto Frontier Plot Showing the Trade-off between Cost and Performance

Additionally, for **all workloads**, we have results shown in Table 7. We calculate the area score increasing ratio and the speed up over the LITTLE core, and calculate the geometric mean of four workload, the result is shown in Table 8. Accordingly, the Pareto frontier of all workloads is shown in Figure 3.

Case	Area Score	simTicks Matmul	simTicks BFS	simTicks Bubblesort	simTicks Queens
LITTLE	4,572	11,224,568,500	34,293,120,500	4,652,532,000	66,156,290,000
Medium1	17424	11,223,313,000	26,958,729,500	4,609,653,000	62,499,881,500
Medium2	19988	11,222,600,500	27,791,698,000	4,534,640,000	56,665,575,000
Medium3	26644	11,220,890,500	26,681,495,000	4,534,595,000	56,665,127,000
Medium4	39956	11,219,466,500	26,375,430,000	4,534,589,000	56,665,268,500
BIG	60556	11,222,717,500	26,256,815,000	4,623,909,500	56,046,536,500

Table 7: More Performance and Area Score for Different Cores

Case	Area Score	Matmul	BFS	Bubblesort	Queens	Geometric Mean
LITTLE	1.0	1.0	1.0	1.0	1.0	1.0
Medium1	3.811	1.0001	1.2721	1.0093	1.0585	1.0797
Medium2	4.372	1.0002	1.2339	1.0260	1.1675	1.1027
Medium3	5.828	1.0003	1.2853	1.0260	1.1675	1.1140
Medium4	8.739	1.0005	1.3002	1.0260	1.1675	1.1173
BIG	13.245	1.0002	1.3061	1.0062	1.1804	1.1161

Table 8: Area Score Ratio and Speed-up Compared to LITTLE Core

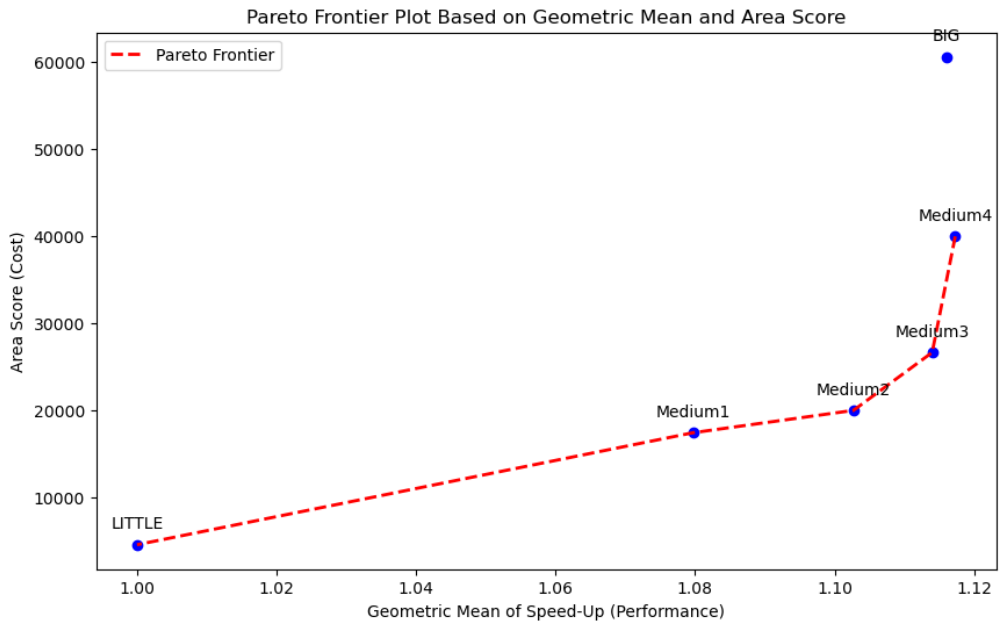


Figure 3: Pareto Frontier Plot Showing the Trade-off between Cost and Geometric Mean

3. Assume you are an engineer working to design this middle-ground core. Given this early analysis, which designs, if any, would you recommend your team to pursue developing? Explain why. (Note: you may want to annotate the above plot.)

The plot (Figure 2) reveals that `Medium1`, `Medium3`, and `Medium4` are positioned on the frontier, thus representing the most favorable trade-offs between cost and performance.

Notably, `Medium1` is located where the frontier begins its steep rise, indicating that it marks the point of diminishing returns in performance improvement relative to cost. This suggests that while `Medium1` offers a substantial level of performance, further enhancements in speed-up would entail a significant increase in cost, making `Medium1` an ideal choice for scenarios demanding high efficiency in terms of performance gain per unit of cost.

Also, from the plot (Figure 3), `Medium2` is a better choice here based on previous analysis. But our main focus is BFS workload and Medium1 is on the pareto frontier and has a flat slope. So we still choose `Medium1`.

For a core design D with a speed-up G_D and an area score ratio A_D (representing the cost), we define the trade-off ratio by:

$$\text{Trade-off Ratio}_D = \frac{G_D - 1}{A_D - 1} \times 100 \tag{1}$$

Here, $G_D - 1$ represents the performance improvement over the baseline and $A_D - 1$ represents the additional cost compared to the baseline. This ratio captures the efficiency of each design in providing performance gains relative to the extra cost incurred.

Case	Geometric Mean Ratio	BFS Ratio
Medium1	2.835	9.680
Medium2	3.046	6.937
Medium3	2.361	5.909
Medium4	1.516	3.879
BIG	0.948	2.500

Table 9: Performance-Cost Ratios for Geometric Mean and BFS

The performance-cost ratio of BFS, is a crucial metric in our assessment. A higher ratio signifies a design that delivers more performance gains for each additional unit of cost. Considering this metric, `Medium1` stands out with the highest ratio of 9.68, implying a more favorable balance between cost and performance.

Given these insights and the geometric means indicating overall performance, `Medium1` presents itself as an efficient choice for scenarios where budget constraints are significant. Given the analysis, the `Medium1` core design is the most promising candidate for development. `Medium1` provides a substantial performance increase for its cost. This balance of cost and performance is highly favorable for scenarios where economical design choices are as valued as computational improvements.

While the `HW3BigCore` offers the highest overall performance, its cost-performance ratio is lower than that of `Medium1`. Therefore, unless the absolute highest performance is required regardless of cost, `Medium1` stands out as the more judicious choice. It is especially suited for markets where cost-effectiveness is crucial without significantly compromising performance.

When evaluating the geometric mean, which represents an overall balance of performance across various workloads, we observe that `Medium2` has a marginally higher value than `Medium1`. However, this slight advantage in overall performance needs to be weighed against the cost implications. `Medium1` offers a more favorable cost-efficiency ratio, as evidenced by its higher performance-cost ratio in the BFS workload, which is a key consideration in our assessment.

In summary, our recommendation is to pursue further development of the **Medium1** design ($width = 4, rob\_size = 256, int\_regs = 256, fp\_regs = 256$) for a cost-effective middle-ground core, with considerations for **Medium4** ($width = 5, rob\_size = 512, int\_regs = 256, fp\_regs = 256$) as a secondary option for scenarios where additional performance is paramount.

4 STEP III: GENERAL QUESTIONS

1. Many phone chips (e.g. Arm Cortex-A series processors) and Intel's Alder Lake chips employ architectures that contain both big cores and little cores in a single system. The operating system (or Intel's "thread directory") can choose which core to use to run each thread. Assume that there is no context switching overhead, do you think that this system will be more or less efficient than have an equal number of medium cores? You may want to read the paper **Amdahl's Law in the Multicore Era by Hill and Marty** for inspiration. Note that you do not need to run any further experiments to answer this question. Back up your answer with logic.

The efficiency of a system employing a mix of big and little cores compared to one with an equal number of medium cores depends significantly on the nature of the workloads it is intended to handle.

Heterogeneous systems, characterized by their combination of big and little cores, offer the advantage of specialization. They can allocate compute-intensive tasks to big cores while assigning less demanding processes to little cores. This dynamic distribution of tasks, assuming negligible context switching overhead, facilitates more energy-efficient operations

and can potentially enhance overall performance for mixed workloads. The system's adaptability to the computational intensity of varying tasks is a key strength of this architecture.

Conversely, homogeneous systems equipped with only medium cores may lack this degree of specialization. While potentially offering balanced performance across various tasks and simpler scheduling mechanisms, these systems might not achieve the same level of efficiency in managing diverse workloads. Medium cores may neither provide the energy savings of little cores for lightweight tasks nor reach the computational prowess of big cores for demanding applications.

Drawing from Hill and Marty's insights on Amdahl's Law in the multicore era, the efficiency of a multicore system largely hinges on the balance between parallelizable and sequential components of the workload. Heterogeneous systems are likely more efficient for workloads that encompass a mix of tasks with diverse computational demands, as they can assign tasks to the most appropriate cores. In contrast, homogeneous systems with medium cores might be more efficient in scenarios where workloads are uniformly distributed or predominantly consist of tasks that do not necessitate the high computational capabilities of big cores or the energy efficiency of little cores.

In conclusion, the efficiency of heterogeneous systems with big and little cores relative to homogeneous systems with medium cores **is contingent upon the nature of the workload**. For diverse workloads with significant variability in computational intensity, heterogeneous systems are likely to be more efficient. However, for more uniform workloads, or in situations where the advanced computational power of big cores and the energy efficiency of little cores are not critical, homogeneous systems with medium cores could be equally or more efficient.