ACA Final Project Report

Raghav Donakanti 2021101024
Roja Sahoo 202111014

November 2024

1 Introduction

1.1 Project Overview

This project aims to analyze the power consumption of various memory hierarchy configurations (L1/L2 caches and main memory) in a RISC-V system using the gem5 simulator. The memory hierarchy significantly affects both performance and power consumption, with parameters such as cache size, associativity, and block size playing critical roles. By simulating different cache configurations and running diverse workloads, the project seeks to explore the trade-offs between power efficiency and performance in RISC-V memory hierarchies.

1. **Configure gem5 for RISC-V Memory Hierarchies**
   The first step involves configuring gem5 to simulate RISC-V cores with adjustable L1/L2 caches and DRAM. Various memory hierarchy models will be created by experimenting with different cache sizes, associativity levels, and block sizes.

2. **Select Benchmark Workloads**
   A combination of compute-bound and memory-bound workloads will be chosen for evaluation. Specifically, AES will serve as the compute-bound workload, while a tiled matrix multiplication (MatMul) will be used to represent memory-bound workloads. These benchmarks will provide insight into the effects of cache and memory configurations on performance and power consumption.

3. **Integrate Power Estimation Tools**
   McPAT will be integrated with gem5 to estimate power consumption based on activity statistics collected during simulations. This will allow for accurate power profiling of different memory hierarchy configurations.

4. **Run Simulations and Collect Data**
   Simulations will be run using the selected benchmark workloads across various cache configurations. Data such as cache hit/miss rates, simulation time, and power consumption will be collected for analysis.

5. **Analyze Power-Performance Trade-offs**
   The final analysis will compare power consumption across different cache configurations and workloads. The impact of parameters such as cache size and associativity on both performance and power usage will be thoroughly examined to identify the optimal configurations for power efficiency and performance balance.

1.2 About Gem5 and Its Usage

Gem5 is an open-source, event-driven simulator designed to model microarchitectural and system-level aspects of computer systems. It enables the analysis and testing of hardware configurations, architectures, and software environments without the need for actual hardware. Gem5 can simulate a complete system in full system mode (FS mode), which includes devices and an operating system, or run user-space programs in syscall emulation mode (SE mode), where system services are emulated by the simulator.

In this project, we explored various simulation environments such as FS and RISC-V configurations. We found that the `se.py` script met our requirements by providing the necessary parameters to modify, ensuring that all options were correctly configured for our analysis.

1.3 About McPAT and Its Usage

McPAT is an integrated framework for modeling power, area, and timing, focusing on power consumption and area estimation under a target clock rate constraint. It allows for simultaneous consideration of power, area, and timing, providing a complete power envelope for various systems. McPAT supports the modeling of manycore processors, with flexible configurations for cores, uncore, and system components, including I/O. It uses technology models from ITRS projections and modern processor models, and it offers a high-level, extensible framework for architectural research.

In this project, we leveraged McPAT's ability to calculate energy consumption for each event based on hardware parameters. We modified a template file to suit our needs and used it to process the simulation statistics from gem5, converting them into XML format for McPAT. Running McPAT then generated accurate power consumption statistics, which were broken down by various memory and cache components for the given workload.

2 Objective of Our Study

In this study, we aim to explore several key questions regarding the power consumption and performance of different memory and cache configurations. Specifically, we focus on how various factors such as cache size, memory type, and the characteristics of workloads like AES and Tiled MatMul influence system behavior. The study is structured around the following questions:

2.1 Q1: How Does Power Consumption and Cache Hits/Misses Vary with Tile Sizes in Tiled MatMul for Large and Small Matrix Sizes?

In this section, we explore the relationship between power consumption, cache behavior (hits and misses), and the tile sizes in Tiled MatMul. Key points to consider are:

- **Effect of Tile Size:** Varying the tile sizes affects the locality of data access, which in turn impacts the cache hit and miss rates. For small matrix sizes, the impact may be less pronounced, but as the matrix size increases, optimizing tile size could significantly improve cache utilization and reduce power consumption.
- **Matrix Size Dependency:** Larger matrices typically require more complex memory management and can benefit from larger tile sizes, while smaller matrices may not see as significant a benefit. This can affect both the execution time and power usage due to cache behavior.
- **Cache Utilization:** Larger tile sizes might lead to more cache hits, thus reducing memory access latency and improving performance, but could also increase the cache miss penalty if not properly optimized.

- **Trade-off Analysis:** A balance needs to be struck between minimizing cache misses and optimizing tile size, considering the power and performance trade-offs for different matrix sizes.

2.2 Q2: Which Program, AES or Tiled MatMul, Uses More Power and Why? Does Changing the DRAM Type Affect Their Cache Usage and Power Consumption?

This question investigates the power consumption patterns of AES and Tiled MatMul under different configurations, including the type of DRAM used. Key points to explore are:

- **Memory-bound vs. Compute-bound:**
  - MatMul is generally memory-bound, especially for naive implementations with small input sizes. For larger matrices, an optimized implementation becomes more compute-bound, relying heavily on floating-point units.
  - AES, on the other hand, is typically compute-bound due to its reliance on logical and shift operations. However, if accelerated, it can become memory-bound due to its low operational intensity and dependence on DRAM accesses.
- **Impact of DRAM Type:** For MatMul, which requires high bandwidth, using higher bandwidth RAM such as DDR5 or HBM could reduce time per byte and power per byte, improving both performance and power efficiency. For AES, the impact of high-bandwidth RAM is likely minimal, but it is worth investigating.
- **Execution Time Breakdown:** Execution time for both programs consists of two main components: (i) floating point/integer/logical operations and (ii) data movement. Data movement is particularly influenced by matrix size and cache architecture. For larger matrices, memory accesses become a significant contributor to power consumption.
- **Program Characterization:** The power consumption and performance depend on several factors, including the implementation, input size, and system architecture. Tools like Intel Advisor can be used to analyze the performance roofline and further characterize these benchmarks.

2.3 Q3: How Does Power Consumption Vary Between AES and MatMul with Different Cache Size and Associativity Configurations? How Are Cache Hits and Misses Affected?

In this section, we analyze the impact of cache size and associativity on the power consumption and cache efficiency for both AES and MatMul. Points to explore include:

- **Impact of Cache Size:** Increasing cache size generally improves matrix multiplication performance by enhancing data locality, leading to fewer cache misses. However, for AES, which is less sensitive to cache size, the improvements may be less pronounced.
- **Effect of Cache Associativity:** High associativity reduces conflict misses in MatMul, improving both performance and power efficiency. For AES, the effect of associativity on power consumption might be smaller, but it could still impact overall cache efficiency.
- **Loop Tiling Optimization:** Loop tiling can be applied to MatMul to optimize the use of L1/L2 caches, potentially reducing cache misses and improving both power consumption and performance.

- **Power-Performance Trade-offs:** By adjusting cache configurations and associativity, we can better understand the trade-offs between power usage and performance, particularly in terms of reducing cache misses and optimizing data access patterns.

3 Common Methodology

3.1 Experiment Paramters

Here are the common parameters we varied for all our three experiments -

Parameter	**Description**	**Options/Details**
`--l1i_size`	Sets the size of the Level 1 instruction cache (L1I).	-
`--l1d_size`	Sets the size of the Level 1 data cache (L1D).	-
`--l2_size`	Sets the size of the Level 2 cache (L2).	-
`--l1d_assoc`	Defines the associativity of the L1 data cache (how many locations in the cache a particular block of memory can be stored in).	-
`--l1i_assoc`	Defines the associativity of the L1 instruction cache.	-
`--l2_assoc`	Defines the associativity of the Level 2 cache.	-
`--cacheline_size (block size)`	Sets the size of cache lines in bytes (amount of contiguous memory data that is transferred between the main memory and the cache in a single operation).	-
`--mem-size`	Specifies the total memory size for the simulated system.	8192MB
`--num-cpus`	Defines the number of CPUs in the simulation.	-
`--cpu-clock`	Sets the clock frequency of each CPU core.	-
`--smt`	Enables or disables simultaneous multi-threading (SMT) for CPUs.	-
`--cpu-type`	Sets CPU types among:	`RiscvAtomicSimpleCPU`, `RiscvMinorCPU`, `RiscvNonCachingSimpleCPU`, `RiscvO3CPU`, `RiscvTimingSimpleCPU`
`--mem-type`	Defines the type of memory to use in the simulation.	`DDR3_2133_8x8`, `DDR4_2400_8x8`, `DDR5_8400_4x8`
`--caches`	Enable to use caching (L1 cache).	-
`--l2cache`	Enable to use L2 cache.	-

Table 1: Common Parameters Varied in All Experiments

We kept the following parameters constant across our experiments, varying the rest according to the specific needs of each experiment:

- `--cpu-type` = `RiscvO3CPU`
- `--smt` = `False`

- `--num-cpus` = 1
- `--mem-size` = 8192MB
- `--cacheline_size` = 64
- `--cpu-clock` = 2GHz

3.2 Gem5 - Program Simulation

We created a script to run a sweep, generating the required output files—`stats.txt` and `config.json`—in the `m5out` directory.

3.3 McPAT - Power Analysis

After obtaining the output from Gem5, we followed these steps for McPAT execution:

1. We ran a script (`Gem5McPATParser.py`) to generate the required XML files from the `stats.txt` and `config.json` files. The template used was a custom one based on McPAT templates. We mainly ensured that the stats name strings were correctly matched and turned off power-gating, as it was causing excessive delays.
2. We then executed McPAT on these XML files with a print level of 5 to obtain the output files containing relevant power consumption statistics.

3.4 Output Stats

The following output statistics were reported for all three experiments:

3.4.1 From `stats.txt`:

- `system.cpu.icache.overallHits::total`
- `system.cpu.icache.overallMisses::total`
- `system.cpu.dcache.overallHits::total`
- `system.cpu.dcache.overallMisses::total`
- `system.l2.overallHits::total`
- `system.l2.overallMisses::total`
- `system.mem_ctrls.dram.rank0.averagePower`

3.4.2 From McPAT Output (all in Watts):

- Instruction Cache Runtime Dynamic Power
- Data Cache Runtime Dynamic Power
- L2 Cache Runtime Dynamic Power
- Bus Cache Runtime Dynamic Power

4 Experiment 1

4.1 Aim

To investigate how power consumption and cache behavior (hits and misses) are influenced by varying tile sizes in Tiled MatMul, and how these effects differ for large and small matrix sizes.

4.2 Parameters

1. **MatMul Parameters:** The following parameters were varied for the MatMul experiment:

```
{
    "program_path": "matmul.cpp",
    "parameters": [
        {"name": "N", "values": [64, 128], "constraints": [{"name": "numerator", "key": 1}]},
        {"name": "TILE_SIZE", "values": [16, 32, 64],
        "constraints": [{"name": "denominator", "key": 1}]}
    ]
}
```

2. **Cache Parameters:**

```
{
    "name": "l1d_size", "values": ["32kB"],
    // made smaller so we can see tile fitting in cache
    "name": "l1i_size", "values": ["32kB"],
    "name": "l1d_assoc", "values": [2],
    "name": "l1i_assoc", "values": [2],
    "name": "l2_size", "values": ["2MB"],
    "name": "l2_assoc", "values": [8],
    "name": "cacheline_size", "values": [64],
    "name": "mem-size", "values": ["8192MB"],
    "name": "mem-type", "values": ["DDR5_8400_4x8"],
    "name": "cpu-type", "values": ["RiscvO3CPU"],
    "name": "num-cpus", "values": [1],
    "name": "cpu-clock", "values": ["2GHz"]
}
```

4.3 Expectations

Matrix multiplication is a fundamental operation in deep learning architectures, especially in transformers. It is primarily a memory-bound operation due to the large number of memory accesses required when handling large matrices, especially when the code is unoptimized.

An optimized implementation of matrix multiplication is Tiled Matrix Multiplication, which aims to minimize the data movement from DRAM, speed up memory access, and shift the program towards the compute-bound region. Without tiling, a significant number of cache misses occur because each row of the matrix is fetched independently, resulting in poor locality of reference.

Tiling addresses this issue by fetching smaller square matrices (tiles), which drastically reduces the number of cache misses. The tile size must be carefully adjusted to fit within the L1D or L2 cache sizes, ensuring that each tile fits perfectly within the cache.

As matrix sizes increase, the number of operations grows, which in turn affects cache performance. The effect of tile size on cache misses in L1D and L2 caches becomes more pronounced with larger matrices. We expect that increasing the tile size will influence the cache miss rate, which will, in turn, impact power usage. If the tile size is too large, it may lead to more cache misses and increased memory accesses, reducing the effectiveness of the cache and potentially leading to higher cache power consumption. On the other hand, well-optimized tile sizes that fit within the cache may reduce memory accesses and improve power efficiency.

Thus, the relationship between tile size, cache misses, and power consumption is crucial for optimizing matrix multiplication performance, especially for large matrix sizes.

4.4 Results and Observations

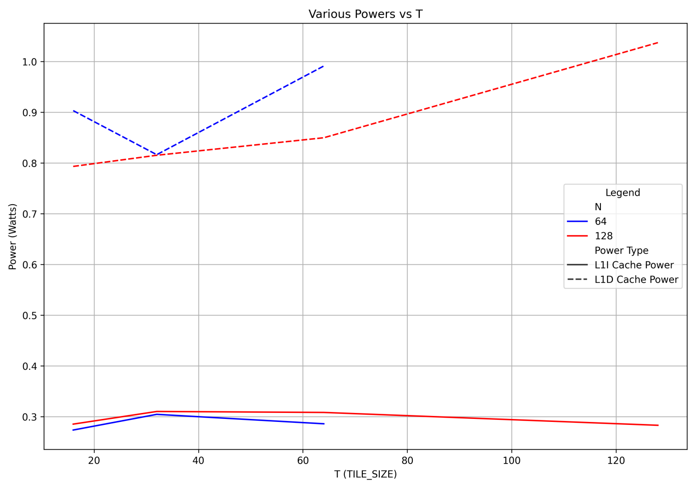

Figure 1: L1 Data Cache Power, L1 Instruction Cache Power vs Tile Size (for N=64, 128)

- The sequence of instructions, including loops and function calls, does not change significantly with varying tile sizes. However, larger tile sizes result in more data being accessed per tile.

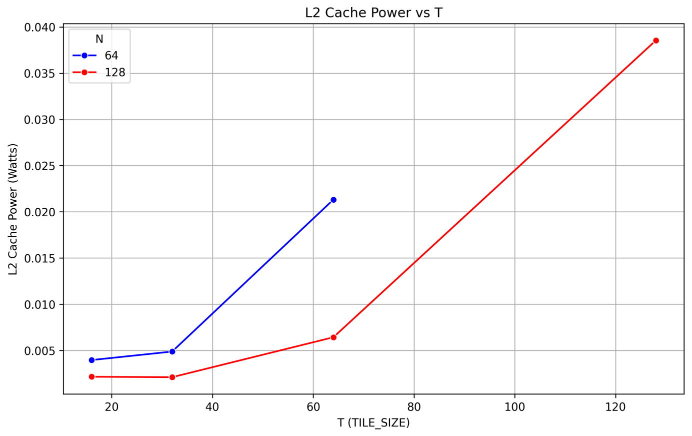

Figure 2: L2 Cache Power vs Tile Size (for N=64, 128)

- Higher power consumption is observed due to more frequent Level 2 (L2) cache accesses, as the Level 1 data (L1D) cache becomes full and cannot accommodate all the required data.

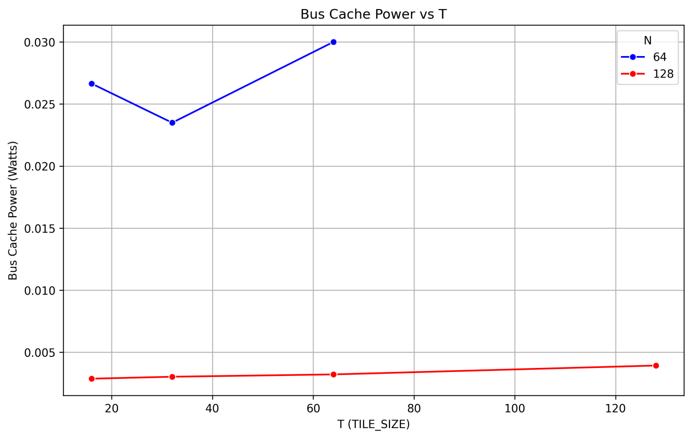

Figure 3: Bus Cache Power vs Tile Size (for N=64, 128)

- Larger working sets may not fit entirely within the L1 or L2 caches. This leads to fewer cache-to-cache transfers and increased DRAM accesses, resulting in higher bus cache power consumption.

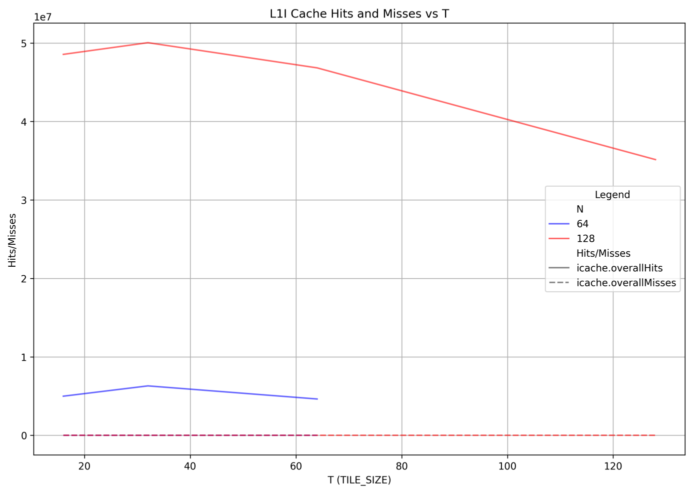

Figure 4: Instruction cache overall Hits, Instruction cache overall misses vs Tile Size (for N=64, 128)

- A standard trend is observed: as the number of instructions increases, so do the cache hits. Since instruction cache (I-cache) misses remain largely unchanged and overall hits increase, it indicates that the I-cache size is sufficient for the workload.

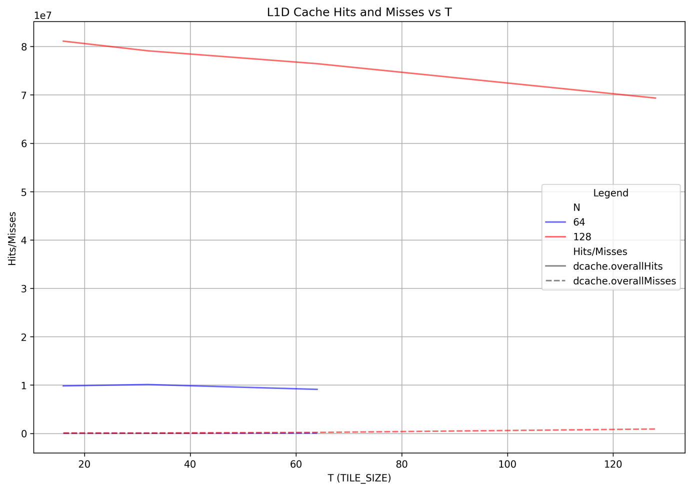

Figure 5: Data cache overall Hits, Data cache overall misses vs Tile Size (for N=64, 128)

- For a tile size of 64, the cache size is sufficient, resulting in mostly consistent hit rates. However, misses might reach their maximum due to frequent evictions. For a tile size of 128, the increased data accessed per tile potentially exceeds the cache's capacity, leading to more frequent evictions where new data replaces older cached data. This results in fewer hits and more misses.

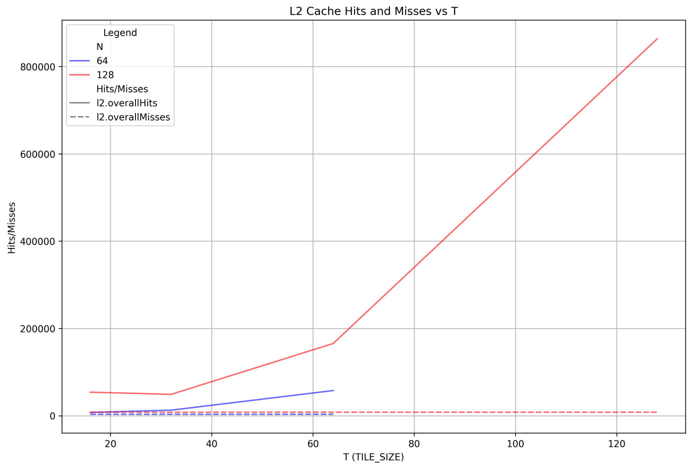

Figure 6: L2 cache overall Hits, L2 cache overall misses vs Tile Size (for N=64, 128)

- The L2 cache effectively handles data that cannot be accommodated by the L1 data cache. L2 cache misses remain relatively constant, indicating the absence of capacity misses, and most misses are compulsory. Hits in the L2 cache increase as it is sufficiently large to store the required data, improving overall performance.

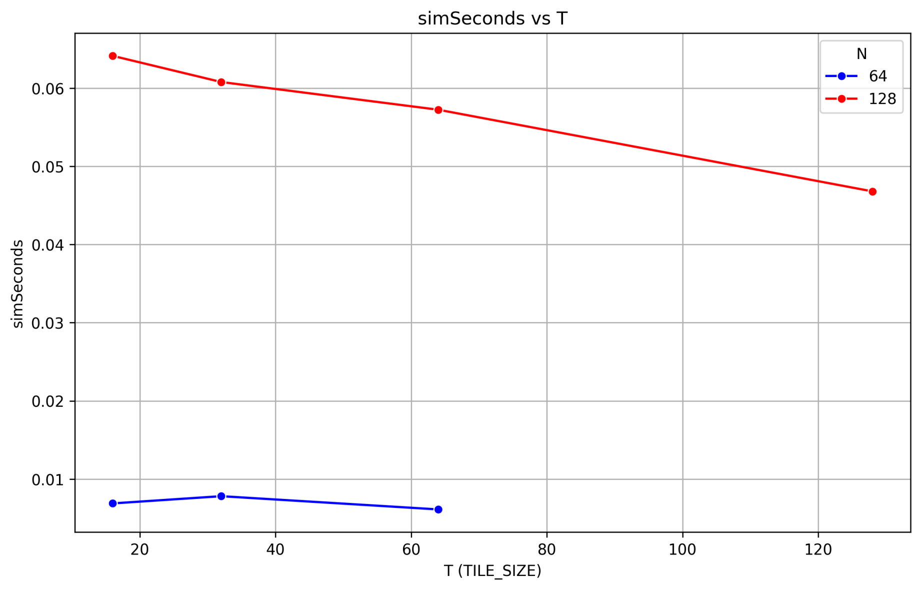

Figure 7: Simulation Time(seconds) vs Tile Size (for N=64, 128)

- A matrix size of 128 takes longer to execute due to higher memory and computational demands.

5 Experiment 2

5.1 Aim

To compare the power consumption of AES and Tiled MatMul, analyze the impact of DRAM type on cache usage, and investigate how these factors influence their power and performance characteristics.

5.2 Parameters

1. **MatMul Parameters:** The following parameters were varied for the MatMul experiment:

```
{
    "program_path": "matmul.cpp",
    "parameters": [
        {"name": "N", "values": [128], "constraints": [{"name": "numerator", "key": 1}]},
        {"name": "TILE_SIZE", "values": [32], "constraints": [{"name": "denominator",
        "key": 1}]}
    ]
}
```

2. **AES Parameters:** For AES, the size of the plaintext is 256 characters.

3. **Cache Parameters:**

```
{
    "name": "l1d_size", "values": ["32kB"],
    "name": "l1i_size", "values": ["32kB"],
    "name": "l1d_assoc", "values": [2],
    "name": "l1i_assoc", "values": [2],
    "name": "l2_size", "values": ["2MB"],
    "name": "l2_assoc", "values": [8],
    "name": "cacheline_size", "values": [64],
    "name": "mem-size", "values": ["8192MB"],
    "name": "mem-type", "values": ["DDR3_2133_8x8",
    "DDR4_2400_8x8", "DDR5_8400_4x8"],  // varied
    "name": "cpu-type", "values": ["RiscvO3CPU"],
    "name": "num-cpus", "values": [1],
    "name": "cpu-clock", "values": ["2GHz"]
}
```

5.3 Expectations

AES and MatMul exhibit distinct program characteristics in terms of memory dependence, CPU dependence, and cache behavior. To gain a high-level understanding of how these programs differ, we explored their performance under various conditions, including changes in DRAM types (DDR versions). This is particularly relevant for workloads with high memory access demands, as different DDR types can impact performance and power consumption.

MatMul can be memory-bound for naive implementations or small input sizes but becomes compute-bound when optimized for larger matrices. Its data movement largely depends on the matrix size and the cache architecture. For larger matrices, MatMul requires high bandwidth from DRAM, making higher-bandwidth RAM types, such as DDR5 or HBM, beneficial for reducing time per byte and power per byte.

AES, in contrast, is less likely to benefit significantly from high-bandwidth RAM due to its lower operational intensity and smaller data requirements. AES tends to be heavily cache-dependent, leveraging higher data locality and reduced data movement compared to MatMul.

Furthermore, MatMul is expected to generate more main memory accesses as its data often exceeds cache capacity, whereas AES operates more efficiently within the cache due to its smaller working set. Modern DRAM types like DDR5 offer lower latency and reduced power consumption per byte, which could enhance the efficiency of memory-dependent workloads like MatMul but may have minimal impact on AES.

5.4 Results and Observations

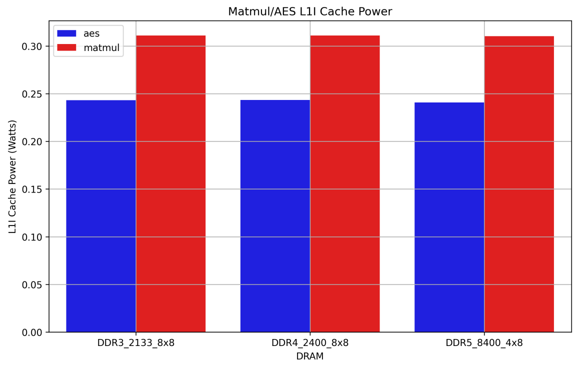

Figure 8: L1 Instruction Cache Power vs DRAM Memory Types (for Prog=AES, Matmul)

- For L1I cache power, memory-bound programs exhibit higher power consumption than compute-bound programs. AES shows relatively lower L1I power consumption since it involves fewer memory-related instructions being fetched.

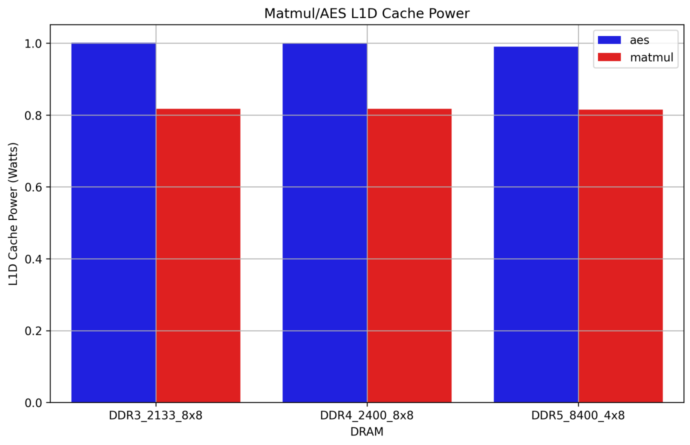

Figure 9: L1 Data Cache Power vs DRAM Memory Types (for Prog=AES, Matmul)

- For L1D cache power, compute-bound programs consume more power than memory-bound programs. AES has higher L1D power due to frequent data manipulation and computation, with data typically residing in the cache. This results in more direct cache hits compared to MatMul, which queries the main memory more frequently.

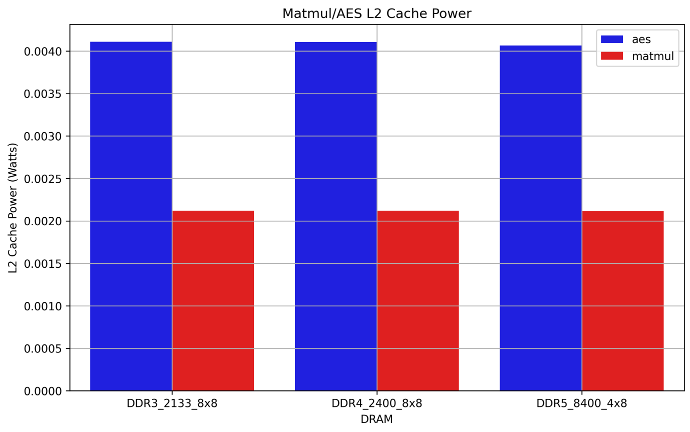

Figure 10: L2Cache Power vs DRAM Memory Types (for Prog=AES, Matmul)

- For L2 cache power, compute-bound programs again exhibit higher power consumption than memory-bound programs. This is due to frequent large memory accesses required for fetching matrix tiles that do not fit in the L1 data cache.

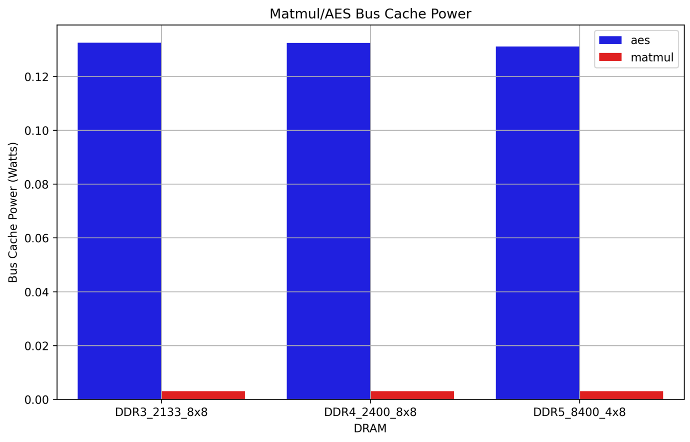

Figure 11: Bus Cache Power vs DRAM Memory Types (for Prog=AES, Matmul)

- Compute-bound programs have higher bus cache power consumption than memory-bound programs. However, memory-bound programs like MatMul make more frequent accesses to the main memory compared to compute-bound programs like AES.

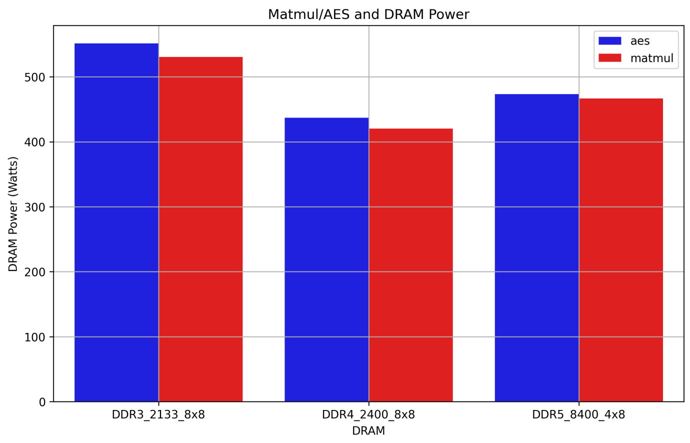

Figure 12: DRAM Average Power vs DRAM Memory Types (for Prog=AES, Matmul))

- The overhead of DDR5 appears to be significant for smaller matrix sizes, increasing the average power consumption. Despite this, DDR5 is generally more efficient in terms of both power and time for larger workloads.

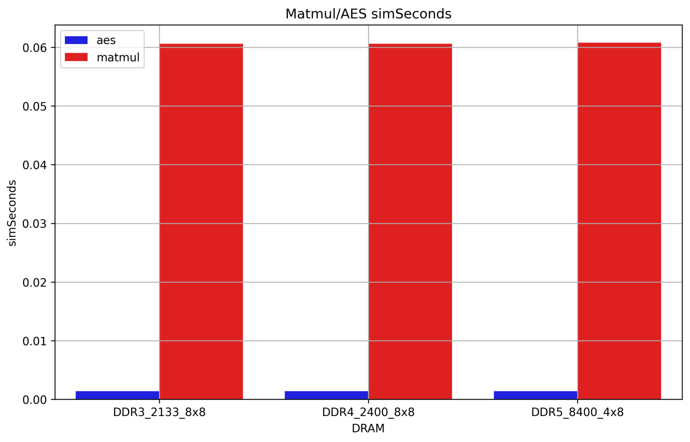

Figure 13: Simulation Time(seconds) vs DRAM Memory Types (for Prog=AES, Matmul)

- Memory-bound programs, such as Tiled MatMul, have longer simulation times due to their higher memory access requirements and heavy computational workload.

6 Experiment 3

6.1 Aim

To analyze how cache size and associativity configurations affect power consumption and cache efficiency in AES and MatMul, focusing on cache hits, misses, and the resulting performance-power trade-offs.

6.2 Parameters

1. **MatMul Parameters:** The following parameters were varied for the MatMul experiment:

```
{
    "program_path": "matmul.cpp",
    "parameters": [
        {"name": "N", "values": [64], "constraints": [{"name": "numerator", "key": 1}]},
        {"name": "TILE_SIZE", "values": [16], "constraints": [{"name": "denominator",
        "key": 1}]}
    ]
}
```

2. **AES Parameters:** For AES, the size of the plaintext is 256 characters.

3. **Cache Parameters:**

```
{
    "name": "l1d_size", "values": ["2kB", "16kB", "32kB"],
    "name": "l1i_size", "values": ["2kB", "16kB", "32kB"],
    "name": "l1d_assoc", "values": [2, 8],
    "name": "l1i_assoc", "values": [2, 8],
    "name": "l2_size", "values": ["1MB", "2MB"],
    "name": "l2_assoc", "values": [8, 16],
    "name": "cacheline_size", "values": [64],
    "name": "mem-size", "values": ["8192MB"],
    "name": "mem-type", "values": ["DDR3_2133_8x8",
    "DDR4_2400_8x8", "DDR5_8400_4x8"],  // varied
    "name": "cpu-type", "values": ["RiscvO3CPU"],
    "name": "num-cpus", "values": [1],
    "name": "cpu-clock", "values": ["2GHz"]
}
```

6.3 Expectations

6.4 Results and Observations

6.4.1 For AES

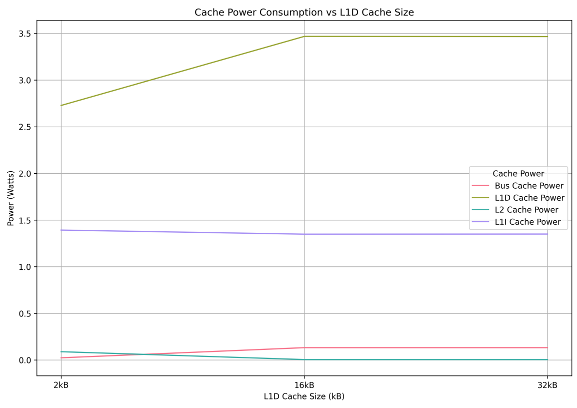

Figure 14: Caches Power vs L1D Cache Size

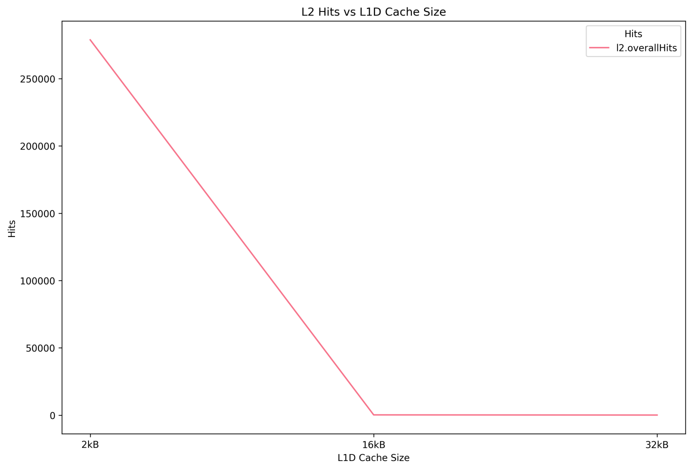

Figure 15: L2 Cache Hits vs L1D Cache Size

- In the graphs for AES, L1D cache power increases due to higher hardware complexity and more frequent L1D cache accesses. Conversely, L2 cache power decreases, as most data accesses are handled by L1, reducing the load on L2. Bus cache power increases as a result of more data movement between caches, driven by higher cache hit rates. Surprisingly, Dcache overall hits decrease, which is counterintuitive, and L2 overall hits also decrease due to fewer queries reaching L2 since L1 is managing most of the requests. Additionally, AES generally does not rely heavily on L2 cache. All trends eventually saturate at 16KB, which appears to be more than sufficient for this program.

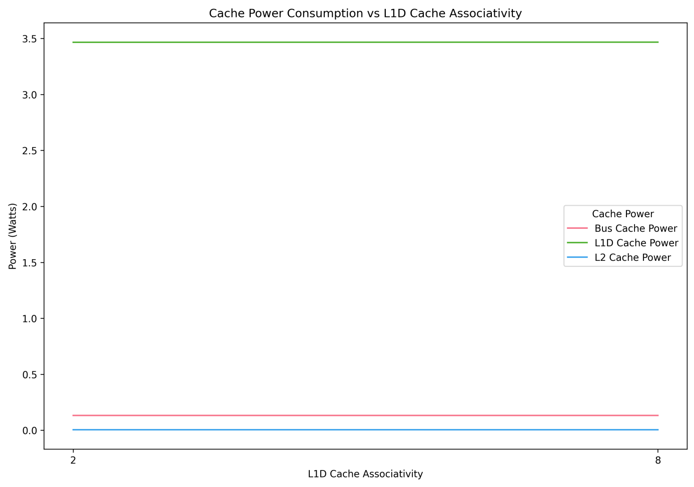

Figure 16: Caches Power vs L1D Associativity Size

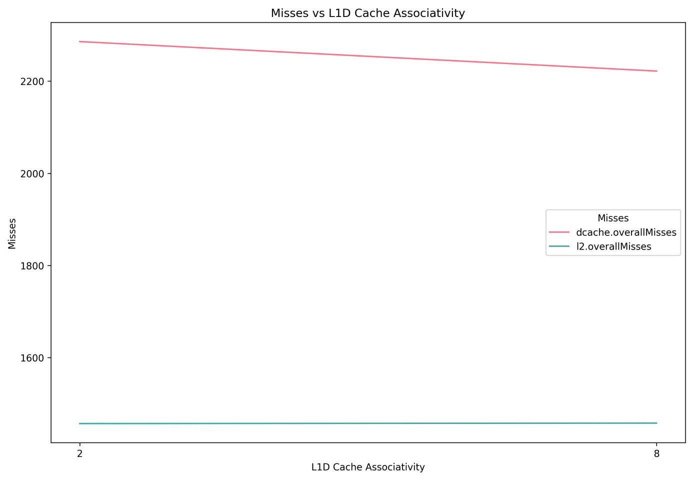

Figure 17: L2 Cache Hits vs L1D Associativity Size

- In the second set of graphs, there is little change across the variables, with all values remaining static. This is counterintuitive, as we might expect that higher hardware complexity (due to increased associativity) would result in higher power consumption, especially for L1D cache power. However, it seems that associativity does not significantly affect AES, likely because the program is compute-bound and not heavily dependent on cache performance.

6.4.2 For Tiled Matmul

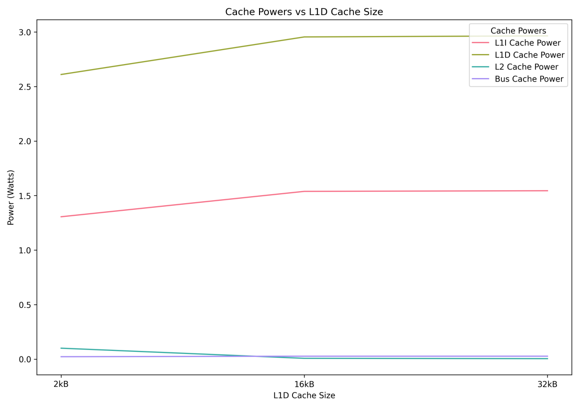

Figure 18: Caches Power vs L1D Cache Size

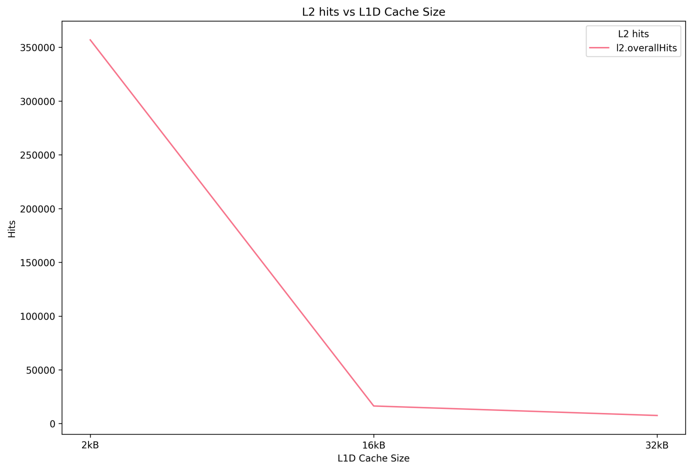

Figure 19: L2 Cache Hits vs L1D Cache Size

- In the case of tiled MatMul, L1D cache power increases due to higher hardware complexity and more frequent L1D cache accesses. L2 cache power decreases because most of the data accesses are handled by L1, shielding L2 from further requests. Bus cache power increases as more data is moved between caches, driven by higher cache hit rates. Dcache overall hits increase as more data is accessed, while L2 cache hits decrease due to fewer queries reaching L2, as L1 is handling most of the requests. With a 16KB configuration, L1 cache appears to be more than sufficient for this program.

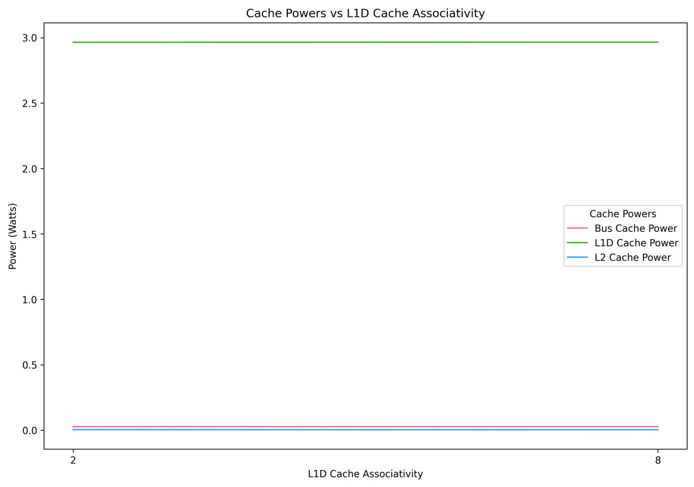

Figure 20: Caches Power vs L1D Associativity Size

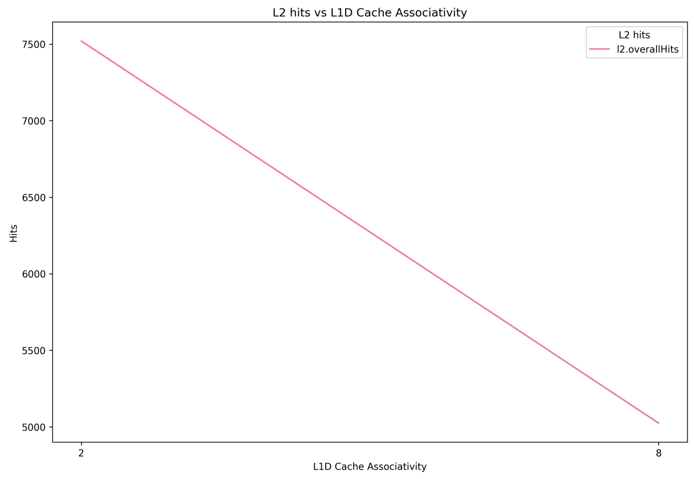

Figure 21: L2 Cache Hits vs L1D Associativity Size

- L1D cache hits increase and misses decrease, which is a standard behavior. L1D cache power also increases as it is being utilized more. On the other hand, L2 cache hits decrease and L2 power also decreases due to less frequent use. Bus cache power increases, while overall conflict misses are reduced. Associativity plays a significant role here, as MatMul benefits from efficient cache utilization.

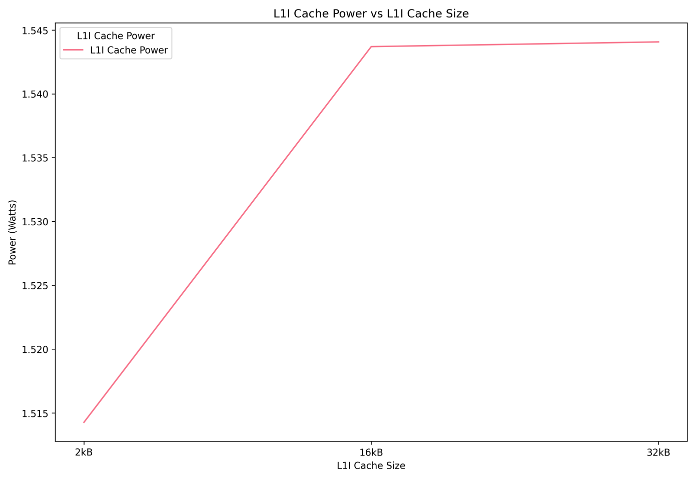

Figure 22: L1I Cache Power vs L1I Cache Size

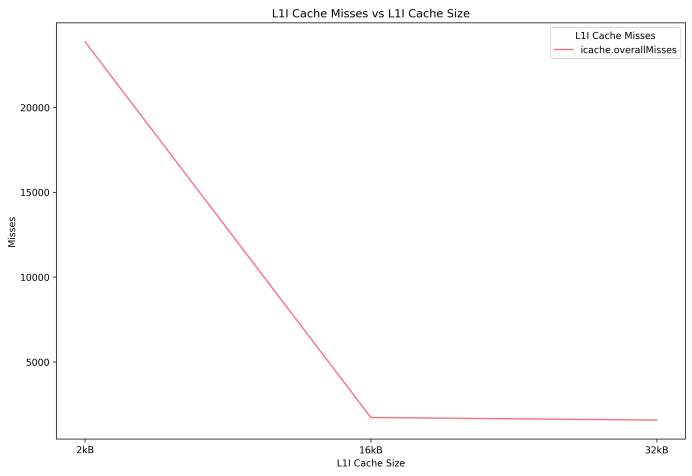

Figure 23: L1I Cache Misses vs L1I Cache Size

- Icache misses decrease and Icache power increases, as expected. Meanwhile, L2 cache power decreases as there is more space available for memory-heavy instructions in the Icache, especially with the increased Icache size.

7 Key Conclusions

- The memory-bound program, Tiled MatMul, requires more cache than our current configuration offers. As a result, it frequently relies on main memory, which is reflected in lower cache power consumption, fewer cache hits, and more cache misses.
- The compute-bound program, AES, fits comfortably within our maximum cache configuration and does not experience similar issues with cache misses.
- Tiled MatMul took more time to run than AES due to its higher memory requirements, which contributed to greater data movement and memory access latency.

- The more a cache is used, the more power it consumes. This relationship highlights the importance of balancing cache usage with power efficiency.
- Properly increasing associativity and cache size results in more cache hits and fewer misses, thereby improving both performance and power efficiency.
- Increasing the L1 data cache size reduces the reliance on L2 cache, resulting in more efficient data handling.
- Increasing L1 cache size is particularly beneficial for memory-bound instructions, as it helps accommodate more data within the cache.
- **Main Anomalies:**
  - For L1D cache hits, AES shows a decrease, while MatMul shows an increase as the L1D cache size increases.
  - DDR5 showed higher time and power consumption, possibly due to the overhead introduced by optimization processes.

8 Limitations & Unexplored Areas

Several limitations and unexplored areas remain in this study:

- MatMul fails to run for matrix sizes greater than 256 due to CPU usage constraints in the current setup.
- The bounds for AES and MatMul could not be pushed further due to the constraints of the Gem5 simulation environment.
- The circumstances under which the CPU-bound AES program might consume more power than the memory-bound MatMul (ridge point) remain unclear and need further investigation.
- Identifying the optimal configuration for compute and memory-bound programs, and determining the settings that enable the best performance with the least power consumption, remains an open challenge.

9 Resources

- Extensive Results
- Presentation
- Code and Stats files