

Performance Evaluation of RISC-V Micro-Benchmarks in gem5: A Comparative Analysis of Out-of-Order and In-Order Execution Modes

1. **Problem:** Generate the binaries of micro-benchmark suite for RISC-V ISA using riscv-gnu-toolchain.
2. **Problem:** Run the applications (any 10) in gem5 in SE mode using out-of-order and in-order execution. Analyze and compare the results and discuss it in your report.

Solution 1:

Installation Steps:

To install riscv-gnu tool we can follow the the below command

1. riscv-gnu-toolchain:

The below command

Listing 1: Command to install riscv-gnu-toolchain

```
1 $ sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev libmpfr-  
    dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc  
    zlib1g-dev libexpat-dev
```

is used to install a set of essential development tools and libraries on a Debian-based Linux system. Let's break down the components and provide a brief description for each:

- **autoconf:** A tool for generating configure scripts to simplify software compilation and installation.
- **automake:** A tool for automatically generating Makefile.in files from Makefile.am files.
- **autotools-dev:** A package that contains various scripts and configuration files used by the Autotools (autoconf, automake, and libtool).
- **curl:** A command-line tool for making HTTP requests, often used for downloading files and resources from the internet.
- **python3:** The Python programming language interpreter (version 3).
- **libmpc-dev:** Development files for the MPC (Multiple Precision Complex) library, which is used for mathematical computations.
- **libmpfr-dev:** Development files for the MPFR (Multiple Precision Floating-Point Reliable) library, used for high-precision arithmetic.
- **libgmp-dev:** Development files for the GMP (GNU Multiple Precision) library, which provides arithmetic operations on large numbers.

- **gawk:** The GNU implementation of the AWK programming language.
- **build-essential:** A meta-package that includes a collection of essential tools and packages required for building and compiling software on a Linux system.
- **bison:** A parser generator that is used to generate parsers for programming languages or data formats.
- **flex:** A tool for generating scanners, which are used for tokenizing input.
- **texinfo:** A typesetting system used for software documentation.
- **gperf:** A tool for generating perfect hash functions.
- **libtool:** A generic library support script that hides the complexity of using shared libraries behind a consistent interface.
- **patchutils:** A collection of utilities for manipulating patch files.
- **bc:** An arbitrary-precision numeric processing language.
- **zlib1g-dev:** Development files for the zlib library, which provides data compression.
- **libexpat-dev:** Development files for the expat XML parsing library.

These packages are commonly needed for building and compiling software from source on a Linux system, ensuring that the necessary dependencies and tools are available.

Listing 2: Command to Clone the repo

```
1 $ git clone https://github.com/riscv/riscv-gnu-toolchain
```

- (a) After cloning the repository, I selected an installation path for building the Newlib cross-compiler. In my configuration, I opted to create a directory named "riscv" within the cloned "riscv-gnu-toolchain" folder. The chosen path looks like this:

/home/kd/riscv-gnu-toolchain/riscv/

Within this directory, I proceeded with the compilation process for the Newlib cross-compiler. Afterward, I created an additional folder named "bin" inside the "riscv" directory, as per my setup. Following that, I appended the subsequent path to my .bashrc file:

/home/kd/riscv-gnu-toolchain/riscv/bin

To add the file to **.bashrc** run the following command

Listing 3: Command to open bash file

```
1 sudo gedit .bashrc
```

Once the text editor is open, append the specified path to the file. In my case, I added the following path:

export PATH="/home/kd/riscv-gnu toolchain/riscv/bin:\$PATH"

Save the file and close the text editor.

- (b) Now, if you've chosen an installation path for the Newlib cross-compiler, such as **"/home/kd/riscv-gnu-toolchain/riscv,"** and added **"/home/kd/riscv-gnu-toolchain/riscv/bin"** to your PATH using the .bashrc file, you can proceed with the configuration by executing the following command:

Listing 4: Command to Configure

```
1 ./configure --prefix=/home/kd/riscv-gnu-toolchain/riscv}
```

Remember to replace "--prefix=" with your chosen installation path. After this, you're ready to initiate the build process for the Newlib cross-compiler.

(c) **make**

After running the **make** command, there will be a significant duration during which the riscv-gnu-toolchain downloads files, totaling approximately 7GB. This process is time-consuming.

1. Micro-benchmarks:

Microbenchmarks are small benchmarks designed to test a component of a larger system.

<https://gem5art.readthedocs.io/en/latest/tutorials/microbench-tutorial.html>

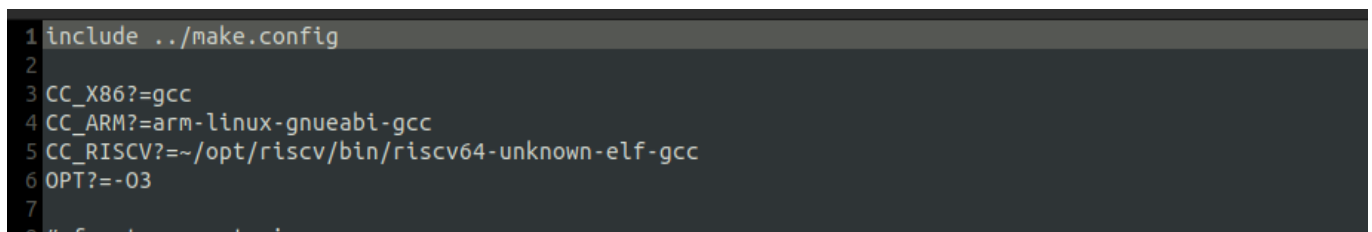
To download the microbenchmarks we can use the below link.

(a) **\$ git clone <https://github.com/darchr/microbench.git>**

(b) **\$ cd microbench**

Before proceeding to the next step, modifications need to be made to the **make.rule** and **rand_c_arr.py** files located within the cloned **microbenchmark** repository.

As, we see in the below picture line number 5,



```
1 include ../make.config
2
3 CC_X86?=gcc
4 CC_ARM?=arm-linux-gnueabi-gcc
5 CC_RISCV?=~/opt/riscv/bin/riscv64-unknown-elf-gcc
6 OPT?=-O3
7
8 # for tree visualization
```

Figure 1: make.rule file

we see the following command

Listing 5: make.rule file original

```
1 CC_RISCV?=~/opt/riscv/bin/riscv64-unknown-elf-gcc
```

Instead of using the previously mentioned path at line number 5, provide the path to the folder you created. Inside the "bin" directory of this folder, numerous files were generated during the "make" process. In my instance, I specified the following path:

Listing 6: make.rule file modified path

```
1 CC_RISCV? = /home/kd/riscv-gnu-toolchain/riscv/bin/riscv64-unknown-elf-gcc
```

Take note that we have omitted the tilde (~) symbol after the "=" sign when assigning our path. Instead of relying on the tilde ~ to denote the home directory, it is essential to furnish the complete absolute path.

The second correction is in the file **rand c arr.py**. In original file, it is given as

Listing 7: rand_c_arr.py

```
1  #!/usr/bin/python
```

It is necessary to modify it to "python3" (or the appropriate Python version for your system) based on your system's Python version. In my case, it is set to **python3**, as illustrated below:

Listing 8: rand_c_arr.py

```
1  #!/usr/bin/python3
```

Or if it doesn't work you can change as the following

Listing 9: rand_c_arr.py

```
1  #!/usr/bin/env python3
```

If encountering an error like **bad interpreter**, the aforementioned modification enables the system to locate the Python interpreter from the PATH environment variable, effectively resolving the "bad interpreter" issue.

(c) **\$ make RISCv**

After running the above command. It should generate all the binary files of the microbench's. In my case it worked. It generated all the binary RISCv files of all the microbench's.

Solution 2:

To run an application in SE mode we need **se.py** file. which is available at **configs/example/se.py**
All gem5 BaseCPU's take the naming format **{ISA}{Type}CPU**. Ergo, if we wanted a RISCv Minor CPU we'd use **RiscvMinorCPU**.

The Valid ISAs are:

- Riscv
- Arm
- X86
- Sparc
- Power
- Mips

The CPU types are:

- AtomicSimpleCPU
- O3CPU
- TimingSimpleCPu
- KvmCPU
- MinorCPU

To run the application in **out-of-order** we need to go with **cpu types as O3CPU** and for **in-order** it is **MinorCPU**.

Below we can see the complete command for running an application in gem5 in SE mode using out-of-order and in-order execution.

Listing 10: out-of-order

```
1 build/RISCV/gem5.opt configs/example/se.py --cmd=/home/kd/gem5/microbench/DPT/bench.RISCV --  
  cpu-type=RiscvO3CPU --caches
```

- In the above command **--cmd = it/is/the/path/where/the/microbench/binary/is/located.**(which was done in Lab 5 problem 1 at the end by \$ make RISCV)
- By changing the microbench marks in the above command run Any 10 application as per the assignment. Rest try running other benchmarks for your understanding and practice.
- If we are using the out-of-order CPU type then it will work only with caches. Therefore at the end of the command we enabled the caches by **--caches**.
- We can verify the same, that is whether the stats files is generated in out-of-order file or not. We can check the config.ini file. We can see the same in below Figure 2, at line number 51.

```

50
51 [system.cpu]
52 type=BaseO3CPU
53 children=branchPred dcache decoder dtb_walker_cache fuPool icache interrupts
54 LFSTSize=1024
55 LQEntries=32
56 LSQCheckLoads=true
57 LSQDepCheckShift=4
58 SQEntries=32
59 SSITSize=1024
60 activity=0
61 backComSize=5
62 branchPred=system.cpu.branchPred
63 cacheLoadPorts=200
64 cacheStorePorts=200
65 checker=NULL

```

Figure 2: Out-of-order config.ini output file

Similarly, for In-order execution, we can follow the below command.

Listing 11: In-order

```

1 build/RISCV/gem5.opt configs/example/se.py --cmd=/home/kd/gem5/microbench/DPT/bench.RISCV --
  cpu-type=MinorCPU --caches

```

We can verify the same, by checking the config.ini file. As we can see the below Figure 3, line number 51.

```

48 init_perf_level=0
49 voltage_domain=system.voltage_domain
50
51 [system.cpu]
52 type=BaseMinorCPU
53 children=branchPred dcache decoder dtb_walker_cache executeFuncUnits icache
54 branchPred=system.cpu.branchPred
55 checker=NULL
56 clk_domain=system.cpu_clk_domain
57 cpu_id=0
58 decodeCycleInput=true
59 decodeInputBufferSize=3
60 decodeInputWidth=2
61 decodeToExecuteForwardDelay=1
62 decoder=system.cpu.decoder
63 do_checkpoint_insts=true

```

Figure 3: In-order config.ini output file

Observation:

We will analyze the outcomes of 40 distinct microbenchmarks, with a focus on several key performance indicators: Instruction Per Cycle (IPC), the total number of cycles taken to complete tasks, memory utilization, power consumption, and the average power consumed by DRAMs. The findings for each of these performance metrics are illustrated in separate graphs.

- **IPC(Instruction Per Cycle):**

- **Higher IPC in O3CPU:** The O3CPU generally shows higher IPC values for all benchmarks when compared to the MinorCPU. This is indicative of the O3CPU's out-of-order execution capability, which allows it to execute multiple instructions simultaneously, filling in the cycles that would otherwise be idle due to instruction dependencies.
- **Lower IPC in MinorCPU:** The MinorCPU has lower IPC values, which is characteristic of in-order CPUs where instructions are executed sequentially. This can lead to cycles where the CPU is idle, waiting for the previous instructions to complete, hence a lower IPC.
- **Benchmark Differences:** The difference in IPC between the two CPU models varies across benchmarks. For instance, the 'Cce_O3CPU' benchmark has an IPC of 1.924649 compared to 'Cce_MinorCPU' with 0.732573, showing a significant performance advantage for the O3CPU. This suggests that the 'Cce' benchmark may have more instruction-level parallelism that the O3CPU can exploit.
- **Low IPC Benchmarks:** Some benchmarks like 'CS1_O3CPU' and 'CS1_MinorCPU' show low IPC values for both CPUs but are still higher for the O3CPU. This could indicate complex instructions that take more cycles to complete or poor parallelism within the benchmark itself.
- **Consistency Across Models:** While IPC values are consistently higher for the O3CPU, the pattern of IPC across benchmarks is similar for both CPU models. This suggests that the benchmarks have inherent characteristics that affect IPC regardless of the CPU's execution model.
- **Benchmarks with Minor Differences:** Some benchmarks, such as 'Cch_O3CPU' and 'Cch_MinorCPU', show a smaller gap in IPC between the two CPU models. This indicates that these particular benchmarks may not benefit as much from out-of-order execution, possibly due to fewer opportunities for parallelism or fewer instruction dependencies.
- **Performance Implications:** The IPC values provide insight into the performance you can expect from each CPU model. For tasks requiring high throughput and performance, the O3CPU appears to be a better fit. However, for simpler tasks or when power efficiency is a priority, the MinorCPU might be more suitable.

We can check the above observations of IPC in below Figure 4:

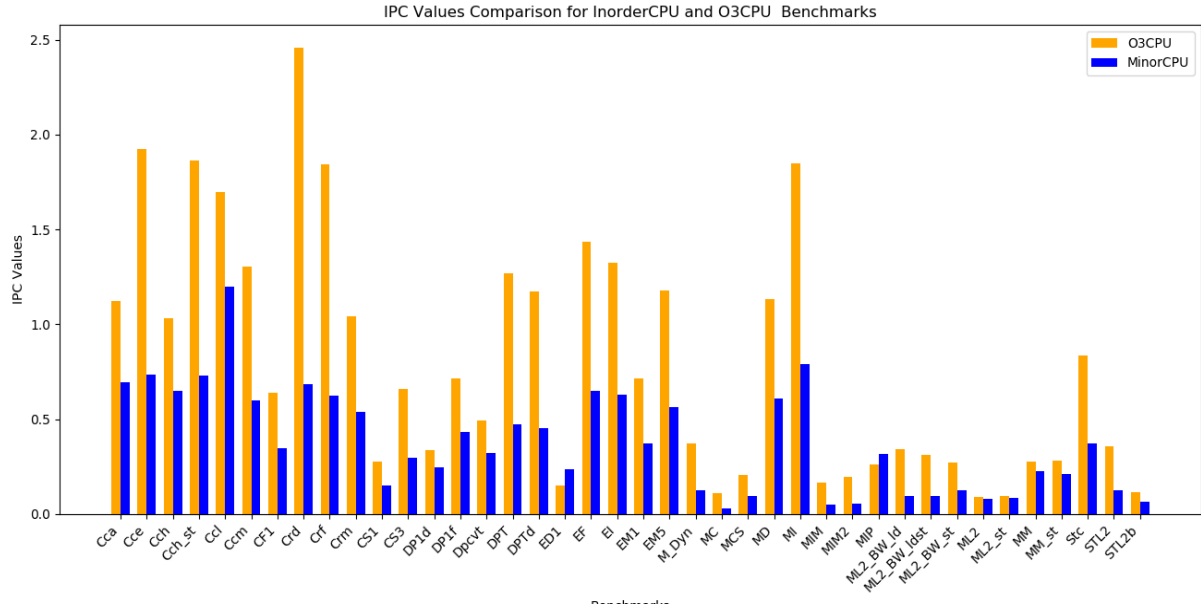


Figure 4: IPC comparison of Inorder & Out-of-order

- **Number of Cycles:**

The comparison of the number of cycles (numCycles) for both the O3CPU and MinorCPU across various benchmarks provides insightful observations into the performance characteristics of these CPU models under different computational workloads.

- **Overall Cycle Count:**

- * The O3CPU tends to have lower cycle counts in several benchmarks compared to the MinorCPU, indicating higher efficiency and possibly faster execution times for certain tasks.
- * The MinorCPU shows significantly higher cycle counts in benchmarks such as 'Crd_MinorCPU' and 'M_Dyn_MinorCPU', suggesting that certain operations are more cycle-intensive under in-order execution models.

- **Benchmark-Specific Performance:**

- * For compute-intensive benchmarks like 'Cch_O3CPU' and 'Cch_MinorCPU', the difference in cycle counts is particularly pronounced, highlighting the efficiency of out-of-order execution in handling complex computational tasks.
- * Memory-intensive benchmarks, indicated by 'ML2_BW_ld_O3CPU' and 'ML2_BW_ldst_MinorCPU', show that the O3CPU manages to execute operations with fewer cycles, suggesting better memory handling and data prefetching capabilities.
- * **Efficiency in Handling Complex Operations:**
- * The O3CPU's architecture allows it to perform well in benchmarks with both high computational and memory demands, as seen in 'Crd_O3CPU' and 'M_Dyn_O3CPU', by utilizing out-of-order execution to mitigate instruction pipeline stalls.

- **Efficiency in Handling Complex Operations:**

- * The O3CPU's architecture allows it to perform well in benchmarks with both high computational and memory demands, as seen in 'Crdr_O3CPU' and 'M_Dyn_O3CPU', by utilizing out-of-order execution to mitigate instruction pipeline stalls.
- * The MinorCPU exhibits a general increase in cycle counts across all benchmarks, which could reflect the limitations of in-order execution in optimizing the instruction execution flow.

– Variability in Performance Gap:

- * The gap in performance, as measured by cycle counts, varies across different benchmarks, suggesting that the impact of CPU architecture (out-of-order vs. in-order) on performance is highly dependent on the specific nature of the workload.
- * Benchmarks with less complexity or fewer dependencies between instructions ('Cca_O3CPU' vs. 'Cca_MinorCPU') show a smaller difference in cycle counts, indicating that the advantages of out-of-order execution diminish in simpler tasks.

– Implications for System Design:

- * The observed data underscores the importance of choosing the right CPU architecture based on the expected workload. Systems requiring high throughput and efficiency for complex operations would benefit from an out-of-order execution model.
- * The increased cycle counts for the MinorCPU in certain benchmarks might not necessarily translate to inferior performance in real-world applications, especially when considering power consumption and thermal output, which were not directly addressed in the provided data.

The above explained observation can be observed in the below Figure 5.

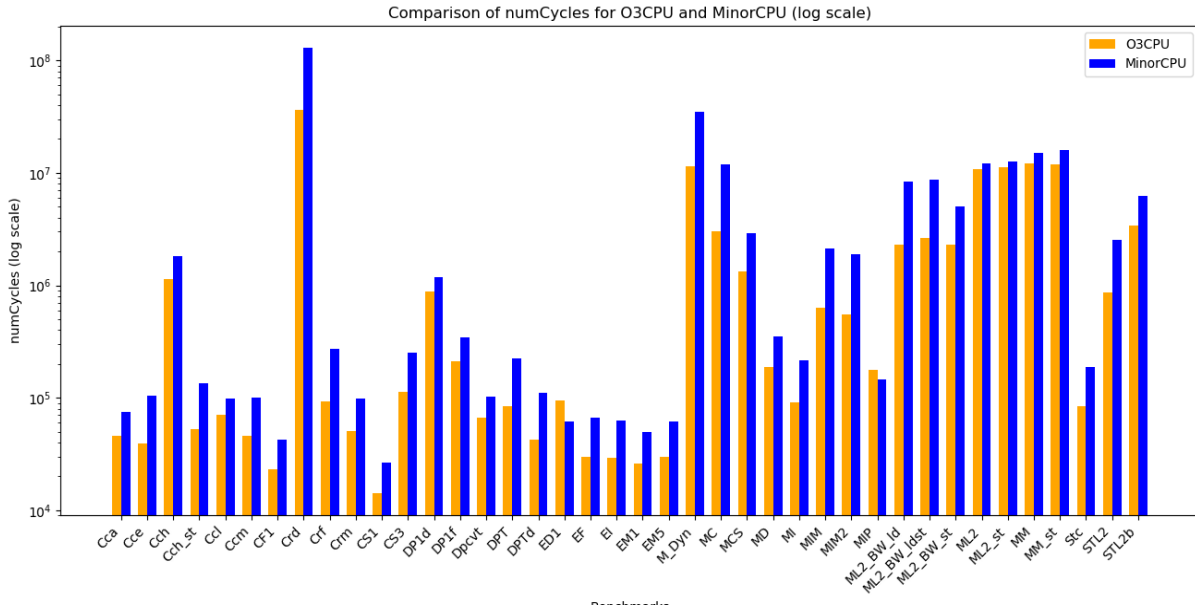


Figure 5: Comparison of No.of Cycles for Inorder & Out-of-order

- **Memory Usage Comparison Between MinorCpu & O3CPU:** To check the comparison of both the MinorCPU and O3CPU we can see the below Figures 6 & 7 respectively.
 - **Memory Read and Write Operations:**
 - * The memory read and write operations across different benchmarks. The **MemRead** and **MemWrite** values are consistent between the two datasets for corresponding benchmarks, indicating similar memory access patterns regardless of the CPU model.
 - * The benchmarks 'Crd' and 'M_Dyn' show exceptionally high memory read values, likely due to intensive memory operations within these benchmarks.
 - **Data Cache Hits and Accesses:**
 - * The 'dcache.overallHits' and 'dcache.overallAccesses' metrics indicate the efficiency of the data cache in handling memory operations. The out-of-order CPU data suggest a higher number of cache hits and accesses for certain benchmarks like 'Cch.O3CPU' and 'M_Dyn.O3CPU', which could imply better cache utilization or more memory-intensive operations compared to the assumed MinorCPU.
 - * Some benchmarks exhibit significant differences in data cache hits and accesses between the datasets, possibly reflecting the architectural differences in handling cache operations.
 - **Cache Miss Rate:**
 - * The 'dcache.overallMissRate' metric shows variability across benchmarks, with some having higher miss rates, which might indicate more challenging memory access patterns or less effective cache utilization.
 - * Notably, benchmarks with complex memory interactions or those that exceed the cache capacity tend to have higher miss rates, demonstrating the limits of cache effectiveness in certain scenarios.
 - **Performance Efficiency:** The O3CPU, with its out-of-order execution capability, likely offers better performance efficiency for benchmarks with high memory access and complex operations, as suggested by the overall higher cache hits and lower miss rates in some benchmarks compared to the initial dataset.
 - **Cache Utilization:** The comparison suggests that the O3CPU may have better cache utilization in certain benchmarks, as indicated by the difference in cache hits and accesses. This could be due to the out-of-order execution allowing for more flexible and efficient use of the cache.
 - **Impact of Architectural Differences:** The architectural differences between the CPU models (out-of-order vs. in-order execution) have a clear impact on memory operation efficiency and cache utilization. The O3CPU's ability to reorder instructions allows it to manage cache resources more effectively, reducing stall times and potentially improving overall performance.
 - **Benchmarks as Performance Indicators:** Specific benchmarks like 'Crd', 'M_Dyn', and 'MM_st' serve as critical indicators of performance differences between CPU models. High memory operations in these benchmarks test the limits of cache efficiency and memory management strategies.

- **Power Usage Comparison between MinorCPU & O3CPU :**

Given the power metrics for both the MinorCPU and the O3CPU across various benchmarks, we can derive some comparative insights into their power efficiency and consumption patterns:

- **Total Energy Consumption:**

The MinorCPU generally exhibits higher total energy consumption (**dram.rank0.totalEnergy** and **dram.rank1.totalEnergy**) for most benchmarks when compared to the O3CPU. This could be indicative of the MinorCPU's potentially less efficient power management or the nature of the in-order execution model, which might not utilize resources as effectively as the out-of-order execution model of the O3CPU.

- **DRAM Energy:**

Energy consumption in DRAM ranks is a significant component of overall power metrics. The O3CPU tends to be more energy-efficient in DRAM usage across most benchmarks. This efficiency could result from better memory access patterns, more effective prefetching, or other architectural advantages that reduce the need for energy-intensive memory operations.

- **Power State Residency:**

The `power_state.pwrStateResidencyTicks` metric, which likely measures the time spent in various power states, shows that the O3CPU spends less time in higher power states across many benchmarks compared to the MinorCPU. This suggests that the O3CPU may have more effective power-saving features or that its architecture allows it to complete tasks more quickly, thus entering low-power states sooner.

- **Efficiency:**

The O3CPU's architecture provides it with an efficiency advantage in both computational tasks and power management, leading to lower total energy consumption in many cases.

- **Architectural Impact on Power:** The data underscores the impact of CPU architecture on power consumption. The out-of-order execution model of the O3CPU not only enhances performance but also contributes to better overall energy efficiency.

- **Benchmark Sensitivity:** The difference in power consumption patterns between the two CPU models varies across benchmarks, indicating that the workload type (CPU-bound, memory-bound, etc.) significantly affects power efficiency.

The above discussed Power Usage Comparison between MinorCPU & O3CPU can be seen in the below Figures 8 and Figure 9.

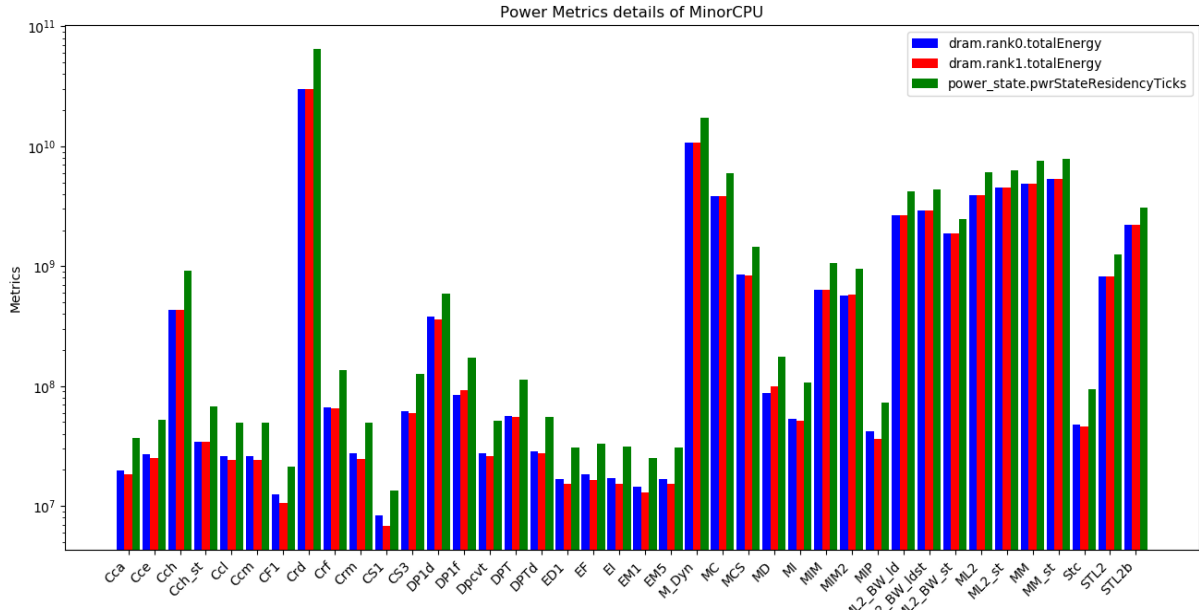


Figure 8: Power Metrics for Microbenchmarks of Inorder

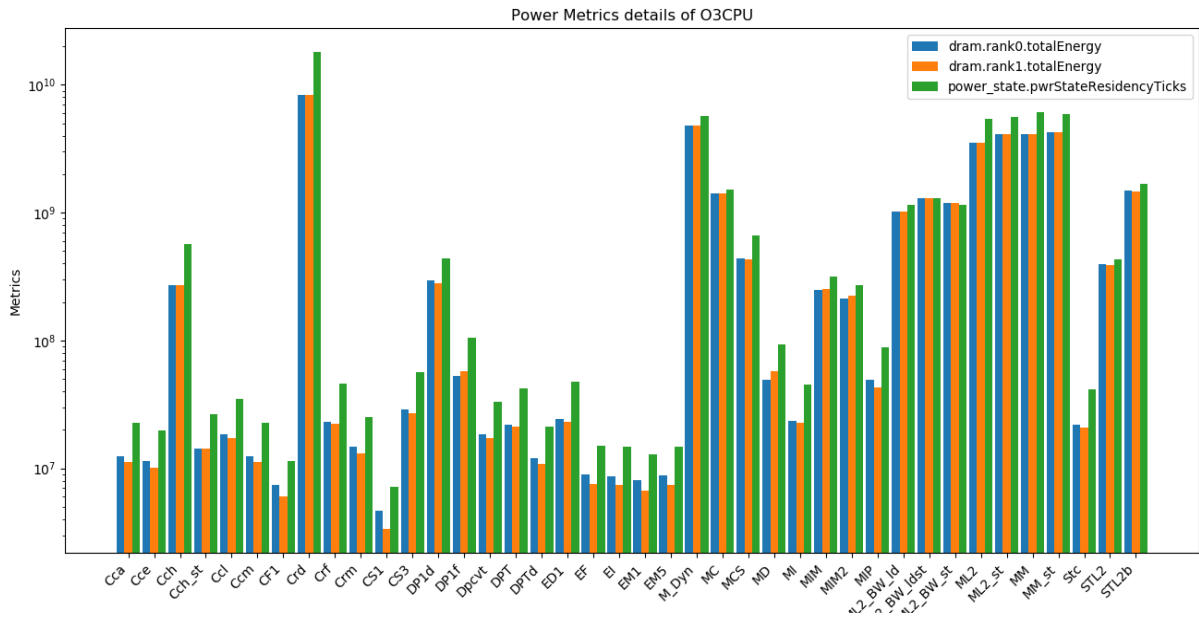


Figure 9: Power Metrics for Microbenchmarks of Out-of-order

- **Average DRAM Power Metrics Comparison in MinorCPU & O3CPU:**
 - **DRAM Power Consumption:**
 - * **MinorCPU vs. O3CPU:**

The data indicates that, on average, the O3CPU has slightly higher DRAM power consumption in several benchmarks compared to the MinorCPU. This could be due to the O3CPU's higher performance and potentially more aggressive memory access patterns facilitated by its out-of-order execution model.
 - **Efficiency and Performance:**
 - * **Higher Power, Higher Performance:**

The O3CPU's increased DRAM power consumption correlates with its higher performance, as evidenced by other provided metrics like IPC values and number of cycles. The O3CPU's ability to execute more instructions per cycle likely leads to increased memory access and, consequently, higher DRAM power consumption.
 - **Memory Access Patterns:**
 - * **Intensive Benchmarks:**

Benchmarks with particularly high DRAM power consumption, such as 'ML2_BW_ldst.O3CPU' and 'M_Dyn_O3CPU' for the O3CPU, and their counterparts in the MinorCPU, suggest that these benchmarks are more memory-intensive. The difference in power consumption between the CPUs in these benchmarks could reflect their respective efficiencies in handling memory-intensive tasks.
 - **Power Management:**
 - * **Power Efficiency:**

While the O3CPU tends to consume more power for DRAM, it's important to balance this with its overall performance gains. Effective power management strategies and architectural optimizations in the O3CPU may offset the higher power consumption by delivering significantly higher performance per watt than the MinorCPU.

We check the above explained observation about Average DRAM power metric is in the below Figures 10 and Figure 11.

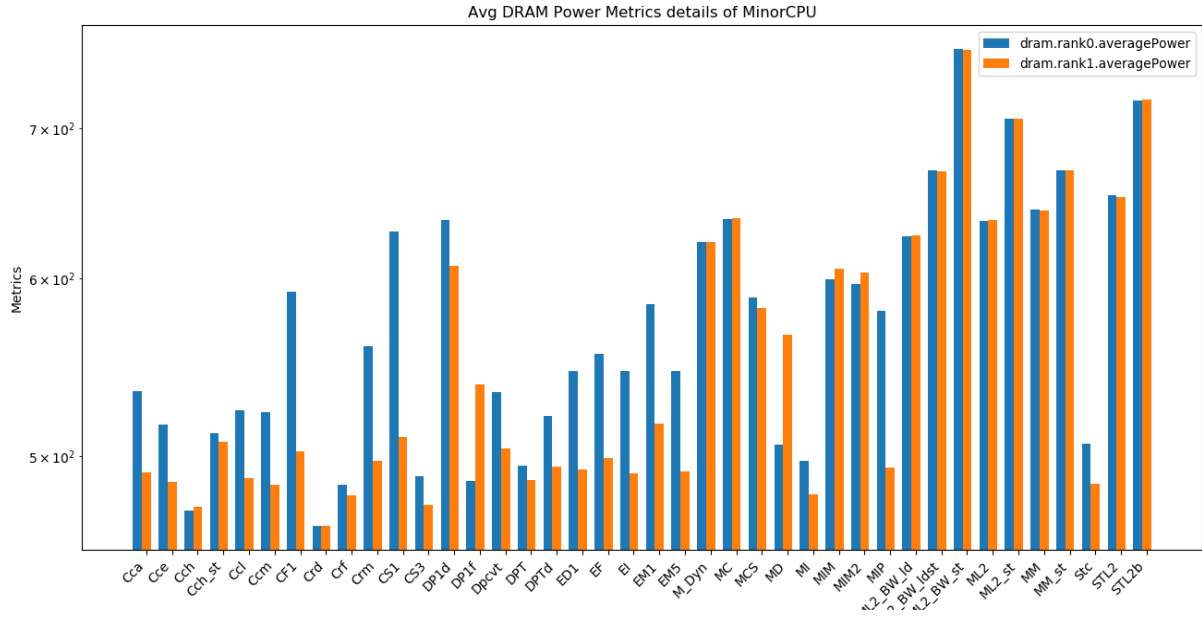


Figure 10: AvgPower Metrics for Microbenchmarks of Inorder

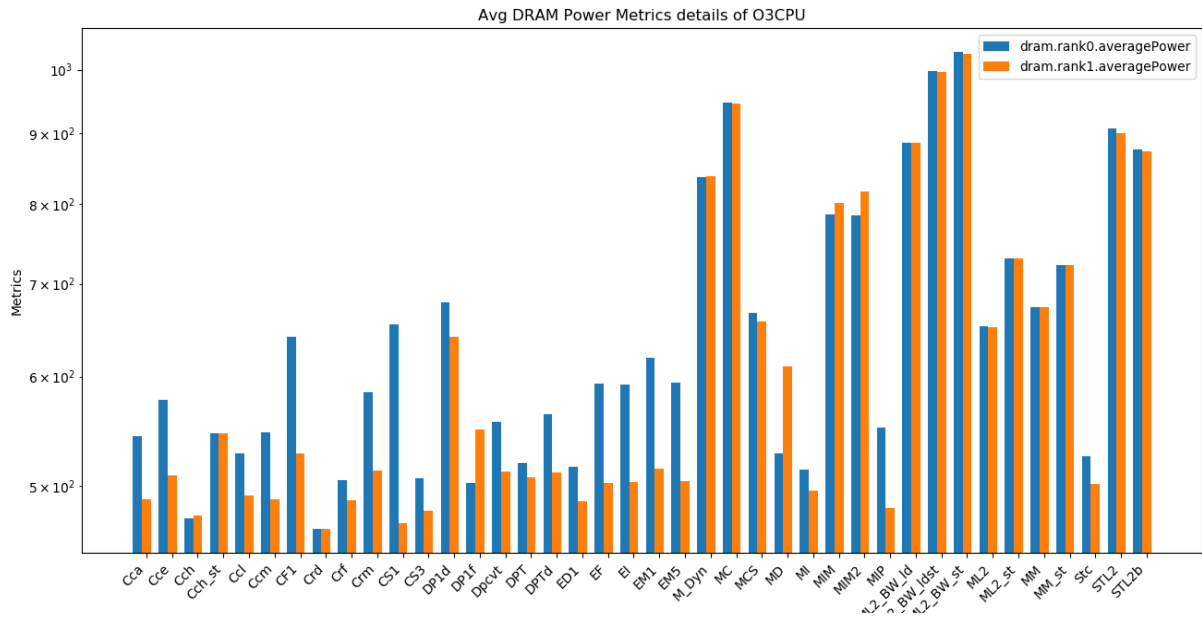


Figure 11: AvgPower Metrics for Microbenchmarks of Out-of-order

Note:

The comparison of O3CPU's higher average DRAM power and its lower total energy consumption can be explained simply:

- **Average DRAM Power:**

O3CPU shows slightly higher average power usage in DRAM, indicating more power used at any given moment, likely due to better performance and aggressive memory access.

- **Total Energy Consumption:**

Despite this, the O3CPU has lower total energy consumption across most benchmarks. This efficiency results from completing tasks quicker, which means it spends less time consuming power.

- **Efficiency Over Time:**

The O3CPU's architecture allows it to be active for shorter periods at high power, thus reducing overall energy usage despite higher instantaneous power levels.

In short, the O3CPU's higher performance leads to higher average power usage but lower overall energy consumption due to faster task completion and effective power management.

Conclusion:

The comprehensive analysis of O3CPU and MinorCPU across IPC, memory, power, and average DRAM power metrics highlights the O3CPU's enhanced performance and energy efficiency. Despite the O3CPU's marginally higher average DRAM power, its advanced architecture facilitates superior performance with reduced overall energy consumption. This efficiency, coupled with higher IPC values, positions the O3CPU as the optimal choice for demanding applications that prioritize both performance and energy efficiency, emphasizing the importance of selecting the appropriate CPU architecture tailored to specific performance needs and energy considerations.