

Comparative Study of AtomicSimpleCPU and TimingSimpleCPU Models with RISC-V and x86 Architectures Using a Hello World Benchmark in gem5

1. **Question** Compare the performance of AtomicSimpleCPU and TimingSimpleCPU models using default configuration script given in config → example. You can use hello world binary inside tests → test-progs → hello → bin → riscv → hello. Discuss the major differences you observe in the m5out → stats.txt for both models.
2. **Question** Repeat task A, but this time compare riscv and x86 based on TimingSimpleCPU model. Please highlight the differences you observe.

Analysis:

To begin with, let's understand what is AtomicSimpleCPU and TimingSimpleCPU.
Types of CPUs which are compatible with gem5.

- AtomicSimpleCPU
- TimingSimpleCPU
- MinorCPU
- O3CPU
- KvmCPU

Before going to the AtomicSimpleCPU and TimingSimpleCPU, we have to know about BaseSimpleCPU.

1. BaseSimpleCPU:

The BaseSimpleCPU serves as a foundation by holding architected state and common stats for SimpleCPU models. It defines functions for interrupt checking, fetch request setup, pre-execute setup, post-execute actions, and advancing the PC. It implements the ExecContext interface. However, it cannot run independently and must be used through one of its subclasses—either AtomicSimpleCPU or TimingSimpleCPU.

There are three types of **Memory Access** are available in gem5 but we are going to discuss only two types, they are

- (a) **Timing** Timing accesses provide the most detailed representation, encompassing realistic timing with queuing delay and resource contention modeling. After a timing request is sent, the sending device will receive either a response or a NACK if the request couldn't be completed. It's important to note that Timing and Atomic accesses cannot coexist in the memory system.

- (b) **Atomic** Atomic accesses are quicker than detailed access, serving purposes like fast-forwarding and cache warming. They return an estimated completion time without considering resource contention or queuing delay. The response for an atomic access is provided when the function returns. Notably, Atomic and Timing accesses cannot occur simultaneously in the memory system.
2. **AtomicSimpleCPU:** The AtomicSimpleCPU is a type of SimpleCPU that relies on **atomic memory accesses**. It uses estimates of latency from these atomic accesses to gauge the overall time for cache access. Derived from BaseSimpleCPU, it includes functions for reading and writing memory, as well as a "tick" function that outlines activities in each CPU cycle. Additionally, it establishes the port for connecting to memory and links the CPU to the cache.
 3. **TimingSimpleCPU:** The TimingSimpleCPU is a variation of SimpleCPU that employs **timing memory accesses**, introducing stalls on cache accesses and waiting for the memory system's response before continuing. Similar to the AtomicSimpleCPU, the TimingSimpleCPU is derived from BaseSimpleCPU, inheriting the same set of functions. It establishes the port for connecting to memory and links the CPU to the cache. Additionally, it defines the essential functions for managing responses from memory to the sent-out accesses.

Note: By default AtomicSimpleCPU is set.

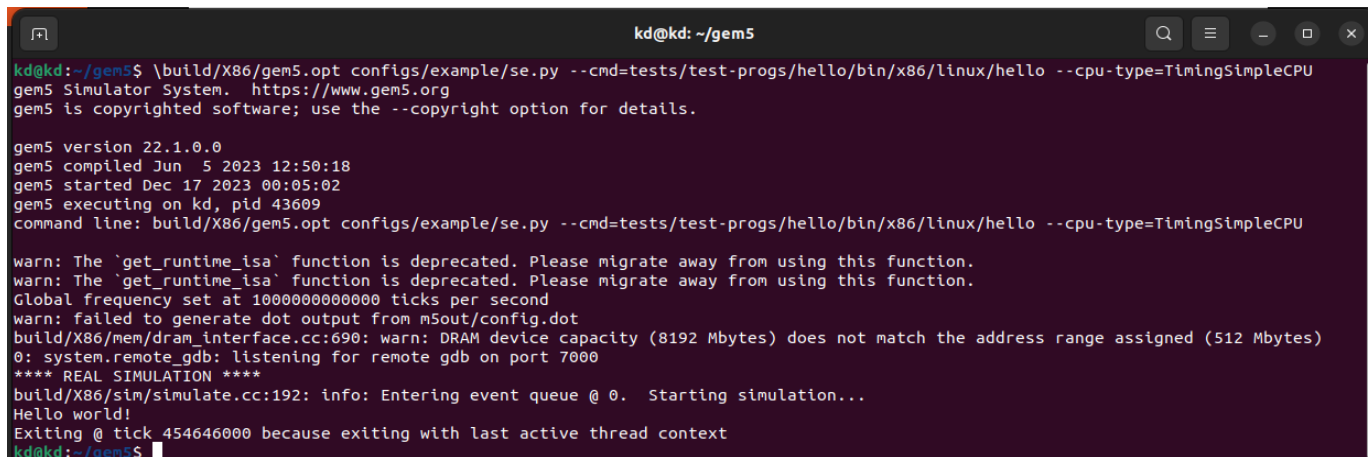
1 TimingSimpleCPU on X86 ISA

Building on our previous lab discussion about creating and analyzing stats files, a comparable procedure is employed for generating the stats file in the TimingSimpleCPU. This involves making specific modifications to the gem5 command line, as outlined below.

Listing 1: Command to Run Hello World in TimingSimpleCPU

```
1 \build/X86/gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/x86/linux/hello --cpu-type=TimingSimpleCPU
```

The below is the output of the Hello World Benchmark for the above command. And respective stats.txt and config.ini files are in the directory "/gem5/m5out/".



```
kd@kd: ~/gem5
kd@kd:~/gem5$ \build/X86/gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/x86/linux/hello --cpu-type=TimingSimpleCPU
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 22.1.0.0
gem5 compiled Jun  5 2023 12:50:18
gem5 started Dec 17 2023 00:05:02
gem5 executing on kd, pid 43609
command line: build/X86/gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/x86/linux/hello --cpu-type=TimingSimpleCPU

warn: The 'get_runtime_isa' function is deprecated. Please migrate away from using this function.
warn: The 'get_runtime_isa' function is deprecated. Please migrate away from using this function.
Global frequency set at 1000000000000 ticks per second
warn: failed to generate dot output from m5out/config.dot
build/X86/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
**** REAL SIMULATION ****
build/X86/sim/stimulate.cc:192: info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 454646000 because exiting with last active thread context
kd@kd:~/gem5$
```

Figure 1: gem5 TimingSimpleCPU output

Stats.txt provides insights into the configured system's performance. To verify the accurate implementation of our configuration, the config.ini file can be examined for the sanity check.

```
[system.cpu]
type=BaseTimingSimpleCPU
children=decoder interrupts isa mmu power_state tracer workload
branchPred=Null
checker=Null
clk_domain=system.cpu_clk_domain
cpu_id=0
decoder=system.cpu.decoder
do_checkpoint_insts=true
do_statistics_insts=true
eventq_index=0
function_trace=false
function_trace_start=0
interrupts=system.cpu.interrupts
isa=system.cpu.isa
max_insts_all_threads=0
max_insts_any_thread=0
mmu=system.cpu.mmu
numThreads=1
power_gating_on_idle=false
power_model=
power_state=system.cpu.power_state
```

Figure 2: gem5 TimingSimpleCPU Config.ini file

As we see in the above Figure 2, the **simobject system.cpu** has a **type=BaseTimingSimpleCPU**. So with this we can make sure that we have executed the command for the desired CPU.

Upon examination, we've generated simple code utilizing the TimingSimpleCPU to print the "Hello, World!" message. The simulation involves executing **630648** instructions solely for this printing task. The choice of TimingSimpleCPU, employing memory access with realistic timing, queuing delay, and resource contention modeling.

A few important statistics from stats.txt files to observe.

Simulation Results		
Parameter	Value	Description
simSeconds	0.000455	# Number of seconds simulated (Second)
simInsts	630648	# Number of instructions simulated (Count)
hostInstRate	55730	# Simulator instruction rate (inst/s) ((Count/Second))
system.cpu.numCycles	909292	# Number of cpu cycles simulated (Cycle)
system.cpu.exec_context.thread_0.numIntInsts	10195	# Number of integer instructions (Count)
system.cpu.exec_context.thread_0.numMemRefs	2027	# Number of memory refs (Count)
system.cpu.exec_context.thread_0.numLoadInsts	1084	# Number of load instructions (Count)
system.cpu.exec_context.thread_0.numStoreInsts	943	# Number of store instructions (Count)
system.cpu.exec_context.thread_0.notIdleFraction	1.000000	# Percentage of non-idle cycles (Ratio)
system.cpu.exec_context.thread_0.idleFraction	0.000000	# Percentage of idle cycles (Ratio)
system.cpu.exec_context.thread_0.numBranches	1306	# Number of branches fetched (Count)
system.cpu.mmu.dtb.rdAccesses	1084	# TLB accesses on read requests (Count)
system.cpu.mmu.dtb.wrAccesses	943	# TLB accesses on write requests (Count)
system.cpu.mmu.dtb.rdMisses	11	# TLB misses on read requests (Count)
system.cpu.mmu.dtb.wrMisses	9	# TLB misses on write requests (Count)
system.cpu.mmu.itb.rdAccesses	0	# TLB accesses on read requests (Count)
system.cpu.mmu.itb.wrAccesses	7286	# TLB accesses on write requests (Count)
system.cpu.mmu.itb.rdMisses	0	# TLB misses on read requests (Count)
system.cpu.mmu.itb.wrMisses	37	# TLB misses on write requests (Count)
system.mem_ctrls.dram.rank0.averagePower	613.606223	# Core power per rank (mW) (Watt)
system.mem_ctrls.dram.rank1.averagePower	583.602264	# Core power per rank (mW) (Watt)
system.mem_ctrls.dram.rank0.totalEnergy	278973615	# Total energy per rank (pJ) (Joule)
system.mem_ctrls.power_state.pwrStateResidencyTicks::UNDEFINED	421689000	# Cumulative time (in ticks) in various power states (Tick)

Table 1: Simulation Results of TimingSimpleCPU on X86

The simulation results provide insights into the performance of the CPU system under consideration. Here are key observations based on the presented statistics. We will do detailed comparison of the TimingSimpleCPU and AtomicSimpleCPU in Section 3 of this report.

CPU Cycle Simulation:

The simulation involved a total of 9,09,292 CPU cycles. This metric serves as a fundamental indicator of the overall computational activity during the simulated period.

2 AtomicSimpleCPU on X86 ISA

To get the stats file of the AtomicSimpleCPU, just we need to change the cpu-type to AtomicSimpleCPU as follows

Listing 2: Command to Run Hello World in AtomicSimpleCPU

```
1 \build\X86\gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/x86/linux/hello --
  cpu-type=AtomicSimpleCPU
```

The below is the output of the Hello World Benchmark for the above command. And respective stat files and config.ini files are in the path "/gem5/m5out/".

```
kd@kd: ~/gem5
kd@kd:~/gem5$ \build/X86/gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/x86/linux/hello --cpu-type=AtomicSimpleCPU
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 22.1.0.0
gem5 compiled Jun  5 2023 12:50:18
gem5 started Dec 17 2023 15:56:50
gem5 executing on kd, pid 53369
command line: build/X86/gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/x86/linux/hello --cpu-type=AtomicSimpleCPU

warn: The 'get_runtime_isa' function is deprecated. Please migrate away from using this function.
warn: The 'get_runtime_isa' function is deprecated. Please migrate away from using this function.
Global frequency set at 1000000000000 ticks per second
warn: failed to generate dot output from m5out/config.dot
build/X86/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
**** REAL SIMULATION ****
build/X86/sim/simulate.cc:192: info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 5943000 because exiting with last active thread context
kd@kd:~/gem5$
```

Figure 3: gem5 AtomicSimpleCPU output

Similar to TimingSimpleCPU, to verify the accurate implementation of our configuration for AtomicSimpleCPU, the config.ini file can be examined for the sanity check.

```
51 [system.cpu]
52 type=BaseAtomicSimpleCPU
53 children=decoder interrupts isa mmu power_state tracer worklo
54 branchPred=Null
55 checker=Null
56 clk_domain=system.cpu_clk_domain
57 cpu_id=0
58 decoder=system.cpu.decoder
59 do_checkpoint_insts=true
60 do_statistics_insts=true
61 eventq_index=0
62 function_trace=false
63 function_trace_start=0
64 interrupts=system.cpu.interrupts
65 isa=system.cpu.isa
66 max_insts_all_threads=0
67 max_insts_any_thread=0
68 mmu=system.cpu.mmu
69 numThreads=1
70 power_gating_on_idle=false
71 power_model=
72 power_state=system.cpu.power_state
```

Figure 4: gem5 AtomicSimpleCPU Config.ini file output

As we see in the above Figure 4. the **simobject system.cpu** has a **type=BaseAtomicSimpleCPU**. So with this we can make sure that we have executed the command for the desired CPU.

Let us now observe some key points from the simulation results for AtomicSimpleCPU.

Simulation Results		
Parameter	Value	Description
simSeconds	0.000006	# Number of seconds simulated (Second)
simInsts	5701	# Number of instructions simulated (Count)
hostInstRate	103287	# Simulator instruction rate (inst/s) ((Count/Second))
system.cpu.numCycles	11887	# Number of cpu cycles simulated (Cycle)
system.cpu.exec_context.thread_0.numMemRefs	2027	# Number of memory refs (Count)
system.cpu.exec_context.thread_0.numLoadInsts	1084	# Number of load instructions (Count)
system.cpu.exec_context.thread_0.numStoreInsts	943	# Number of store instructions (Count)
system.cpu.exec_context.thread_0.notIdleFraction	1.000000	# Percentage of non-idle cycles (Ratio)
system.cpu.exec_context.thread_0.idleFraction	0.000000	# Percentage of idle cycles (Ratio)
system.cpu.exec_context.thread_0.numBranches	1306	# Number of branches fetched (Count)
system.cpu.mmu.dtb.rdAccesses	1084	# TLB accesses on read requests (Count)
system.cpu.mmu.dtb.wrAccesses	943	# TLB accesses on write requests (Count)
system.cpu.mmu.dtb.rdMisses	11	# TLB misses on read requests (Count)
system.cpu.mmu.dtb.wrMisses	9	# TLB misses on write requests (Count)
system.cpu.mmu.itb.rdAccesses	0	# TLB accesses on read requests (Count)
system.cpu.mmu.itb.wrAccesses	7286	# TLB accesses on write requests (Count)
system.cpu.mmu.itb.rdMisses	0	# TLB misses on read requests (Count)
system.cpu.mmu.itb.wrMisses	37	# TLB misses on write requests (Count)
system.mem_ctrls.dram.rank0.averagePower	384.048460	# Core power per rank (mW) (Watt)
system.mem_ctrls.dram.rank1.averagePower	384.048460	# Core power per rank (mW) (Watt)
system.mem_ctrls.dram.rank0.totalEnergy	2282400	# Total energy per rank (pJ) (Joule)
system.mem_ctrls.power_state.pwrStateResidencyTicks::UNDEFINED	5943000	# Cumulative time (in ticks) in various power states (Tick)

Table 2: Simulation Results of AtomicSimpleCPU on X86

3 TimingSimpleCPU on RISC-V ISA

The below command is to run the "Hello World!" benchmark on RISC-V ISA using TimingSimpleCPU.

Listing 3: Command to Run Hello World in TimingSimpleCPU in RISC-V ISA

```
1 build/RISCV/gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/riscv/linux/hello
   --cpu-type=TimingSimpleCPU
```

The below is the respective output of the respective benchmark.

```
kd@kd: ~/gem5
kd@kd:~/gem5$ build/RISCV/gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/riscv/linux/hello --cpu-type=TimingSimpleCPU
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 22.1.0.0
gem5 compiled Jun  5 2023 16:54:57
gem5 started Dec 17 2023 20:44:34
gem5 executing on kd, pid 56969
command line: build/RISCV/gem5.opt configs/example/se.py --cmd=tests/test-progs/hello/bin/riscv/linux/hello --cpu-type=TimingSimpleCPU

warn: The 'get_runtime_isa' function is deprecated. Please migrate away from using this function.
warn: The 'get_runtime_isa' function is deprecated. Please migrate away from using this function.
Global frequency set at 1000000000000 ticks per second
warn: failed to generate dot output from m5out/config.dot
build/RISCV/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
**** REAL SIMULATION ****
build/RISCV/sim/simulate.cc:192: info: Entering event queue @ 0. Starting simulation...
build/RISCV/sim/mem_state.cc:443: info: Increasing stack size by one page.
Hello world!
Exiting @ tick 421689000 because exiting with last active thread context
kd@kd:~/gem5$
```

Figure 5: RISC-V TimingSimpleCPU Output

To confirm the configuration, we can examine the config.ini file, as depicted in Figure 6 below. For a sanity check, we observe the simObject **system.cpu.isa** with **type=RiscvISA**. This ensures that our intended configuration has been executed successfully.

```
94 [system.cpu.isa]
95 type=RiscvISA
96 check_alignment=false
97 eventq_index=0
98
99 [system.cpu.mmu]
100 type=RiscvMMU
101 children=dtb itb pma_checker pmp
102 dtb=system.cpu.mmu.dtb
103 eventq_index=0
104 itb=system.cpu.mmu.itb
105 pma_checker=system.cpu.mmu.pma_checker
106 pmp=system.cpu.mmu.pmp
107
108 [system.cpu.mmu.dtb]
109 type=RiscvTLB
110 children=walker
111 entry_type=data
112 eventq_index=0
113 next_level=NULL
114 pma_checker=system.cpu.mmu.pma_checker
115 pmp=system.cpu.mmu.pmp
116 size=64
117 walker=system.cpu.mmu.dtb.walker
```

Figure 6: RISCv TimingSimpleCPU Config

The results presented in Table 3 represent the outcomes of the TimingSimpleCPU simulation on RISCv. Key findings are extracted from the stats files generated when executing the command provided in Listing 3 above.

Simulation Results		
Parameter	Value	Description
simSeconds	0.000422	# Number of seconds simulated (Second)
simInsts	5518	# Number of instructions simulated (Count)
hostInstRate	98338	# Simulator instruction rate (inst/s) ((Count/Second))
system.cpu.numCycles	843378	# Number of cpu cycles simulated (Cycle)
system.cpu.exec_context.thread_0.numIntInsts	5465	# Number of integer instructions (Count)
system.cpu.exec_context.thread_0.numMemRefs	2095	# Number of memory refs (Count)
system.cpu.exec_context.thread_0.numLoadInsts	1062	# Number of load instructions (Count)
system.cpu.exec_context.thread_0.numStoreInsts	1033	# Number of store instructions (Count)
system.cpu.exec_context.thread_0.notIdleFraction	1.000000	# Percentage of non-idle cycles (Ratio)
system.cpu.exec_context.thread_0.idleFraction	0.000000	# Percentage of idle cycles (Ratio)
system.cpu.exec_context.thread_0.numBranches	1224	# Number of branches fetched (Count)
system.cpu.mmu.dtb.misses	0	# Total TLB (read and write) misses (Count)
system.cpu.mmu.dtb.accesses	0	# Total TLB (read and write) accesses (Count)
system.cpu.mmu.itb.misses	0	# Total TLB (read and write) misses (Count)
system.cpu.mmu.itb.accesses	0	# Total TLB (read and write) accesses (Count)
system.mem_ctrls.dram.rank0.averagePower	666.304540	# Core power per rank (mW) (Watt)
system.mem_ctrls.dram.rank1.averagePower	503.898181	# Core power per rank (mW) (Watt)
system.mem_ctrls.power_state.pwrStateResidencyTicks::UNDEFINED	5943000	# Cumulative time (in ticks) in various power states (Tick)

Table 3: Simulation Results of TimingSimpleCPU on RISC-V

4 Comparision Between TimingSimpleCPU and AtomicSimpleCPU

1 Solution:

The observed simSeconds for TimingSimpleCPU is $455 \mu sec$, while for AtomicSimpleCPU, it is $6 \mu sec$. This discrepancy arises because TimingSimpleCPU employs Timing memory access, which accurately models realistic timing, incorporating queuing delay and resource contention.

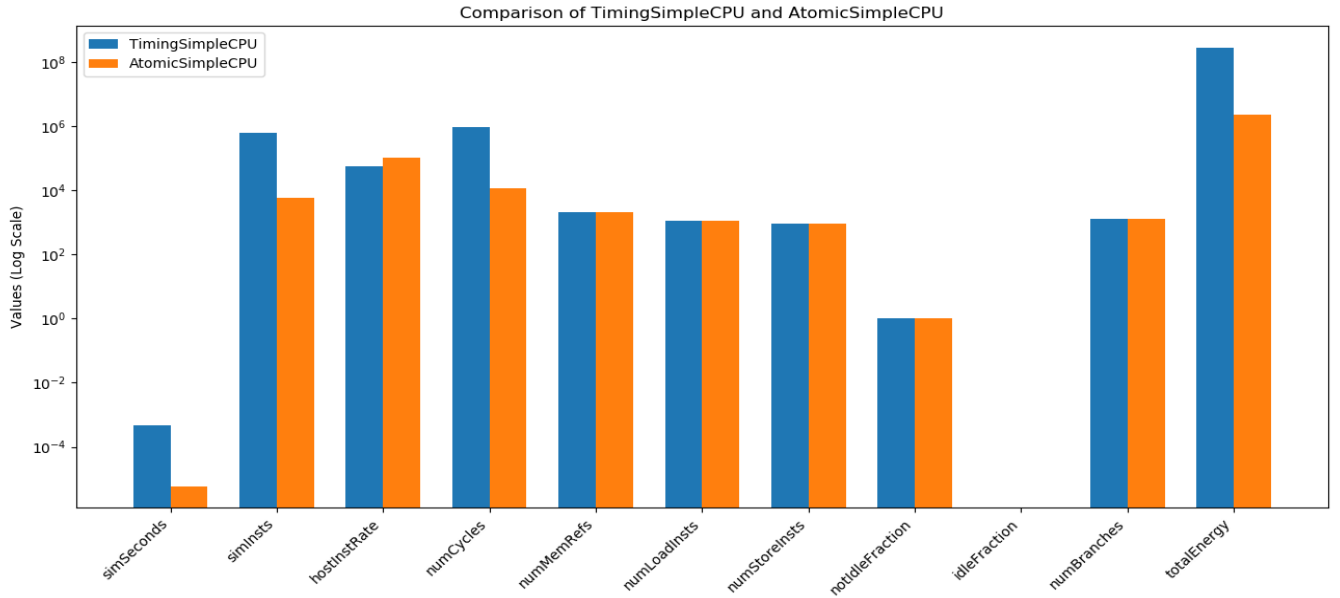


Figure 7: Comparision between TimingSimpleCPU and AtomicSimpleCPU

In contrast, AtomicSimpleCPU utilizes Atomic memory access, relying on services like fast-forwarding and providing an estimated completion time without accounting for resource contention or queuing delay.

In this comparison, there is a substantial increase in the number of CPU cycles, rising from 11,887 in AtomicSimpleCPU to 909,292 in TimingSimpleCPU. Similarly, the count of simulated instructions has seen a notable increase, climbing from 5,701 in AtomicSimpleCPU to 630,648 in TimingSimpleCPU. The simulator instruction rate has experienced a decrease from 103,287 in AtomicSimpleCPU to 55,730 in TimingSimpleCPU. Additionally, a significant rise is observed in the total energy per rank in pJ, increasing from 2,282,400 in AtomicSimpleCPU to 278,973,615 in TimingSimpleCPU. We can see the same in the Figure 5.

The average power per rank in TimingSimpleCPU is substantially higher compared to AtomicSimpleCPU. Each rank in TimingSimpleCPU consumes noticeably more power, indicating a higher demand for energy during simulation. TimingSimpleCPU incurs a significantly higher total energy consumption per rank when compared to AtomicSimpleCPU. The total energy in TimingSimpleCPU is considerably elevated, suggesting that the more accurate timing model comes at the cost of increased energy usage.

However, it's noteworthy that certain metrics, such as the number of memory references, the count of load and store instructions, the percentage of non-idle cycles, the percentage of idle cycles, and the number of branches fetched, have remained unchanged in both TimingSimpleCPU and AtomicSimpleCPU.

5 Comparison Between RISC-V and X86 based on TimingSimpleCPU

2 Solution:

The log-scale plot below illustrates the simulation results for both the RISC-V and X86 ISAs based on the TimingSimpleCPU. Upon closer examination, it is evident that the disparity in the number of simulated seconds between X86 and RISC-V is minimal. The X86 ISA incurs only an additional 33 microseconds for simulation compared to RISC-V.

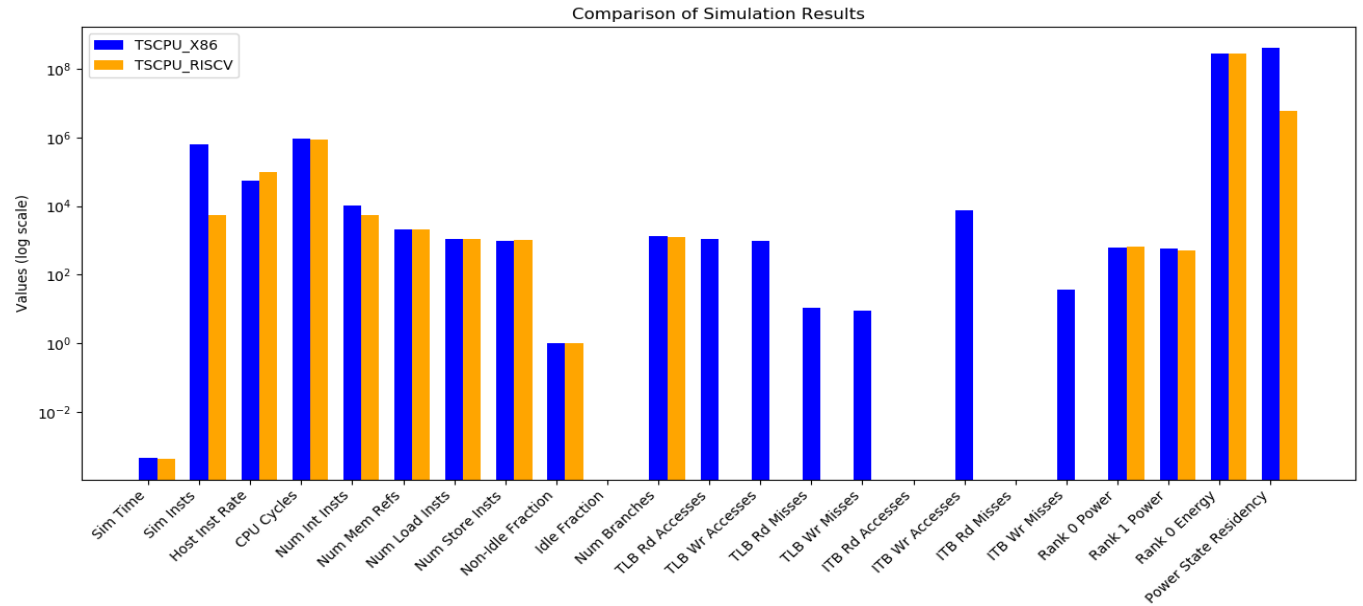


Figure 8: Comparison between X86 and RISC-V ISA using TimingSimpleCPU

The simulation results reveal notable differences, particularly in the count of integer instructions. The RISC-V architecture shows a significant variance, with only 5465 integer instructions compared to the X86 ISA's 10195, representing nearly a 50% reduction.

Additionally, substantial distinctions are observed in TLB accesses and misses on read and write requests. In the X86 ISA, there are 1084 TLB read accesses, 943 TLB write accesses, 11 TLB read misses, and 9 TLB write misses.

Contrastingly, the RISC-V ISA registers zero TLB accesses or misses, indicating a distinct absence of TLB activity in RISC-V simulations.

6 Conclusion:

In summary, the comparison between TimingSimpleCPU and AtomicSimpleCPU for X86 simulations reveals a trade-off between accuracy and efficiency. TimingSimpleCPU offers realistic timing modeling but results in longer simulation times, while AtomicSimpleCPU sacrifices some accuracy for faster simulations with lower power and energy consumption.

When comparing X86 and RISC-V based on TimingSimpleCPU, subtle differences exist in simulation times, with X86 slightly longer. The major distinction is in the count of integer instructions, where RISC-V shows a significant reduction. TLB activity in X86 is notable, whereas RISC-V has zero TLB accesses or misses. Choosing between ISAs should consider specific simulation goals, balancing accuracy and efficiency based on the desired trade-offs.