ECE511 Mp1 Report

Gem5 Introduction

Leon Ku

leonku2

1/31/2025

3.3 Weak Processor

single-core, 1GHz, 64-byte cache line, 16KB 4-way L1 instruction cache, 32KB 8-way L1 data cache, no L2 cache

a. required simulation output statistics

statistic name

Benchmark execution time in seconds	simSeconds
Instructions per cycle (IPC)	system.cpu.ipc
Number of instructions (use simInsts)	simInsts
Number of µOps (use simOps)	simOps
Cache hit ratio for L1I	1 - system.cpu.icache.overallMissRate::total
Cache hit ratio for L1D	1 - system.cpu.dcache.overallMissRate::total
Total DRAM bandwidth	system.mem_ctrls.dram.bwTotal::total

X86 - fft

Benchmark execution time in seconds	0.130814
Instructions per cycle (IPC)	0.415735
Number of instructions (use simInsts)	54384178
Number of µOps (use simOps)	89348813
Cache hit ratio for L1I	0.999772
Cache hit ratio for L1D	0.983943
Total DRAM bandwidth	147459629 Byte/Second

X86 – Cholesky

Benchmark execution time in seconds	0.992516
Instructions per cycle (IPC)	0.428104
Number of instructions (use simInsts)	424899948
Number of µOps (use simOps)	635731236
Cache hit ratio for L1I	0.999944
Cache hit ratio for L1D	0.971852
Total DRAM bandwidth	416623901 Byte/Second

X86 – fmm

Benchmark execution time in seconds	0.454774
Instructions per cycle (IPC)	0.479979
Number of instructions (use simInsts)	218282059
Number of µOps (use simOps)	347494268
Cache hit ratio for L1I	0.999770
Cache hit ratio for L1D	0.985633
Total DRAM bandwidth	89439540 Byte/Second

Riscv – fft

Benchmark execution time in seconds	0.062289
Instructions per cycle (IPC)	0.711708
Number of instructions (use simInsts)	44331481
Number of µOps (use simOps)	44331504
Cache hit ratio for L1I	0.999877
Cache hit ratio for L1D	0.977675
Total DRAM bandwidth	309494813 Byte/Second

Riscv – Cholesky

Benchmark execution time in seconds	0.639823
Instructions per cycle (IPC)	0.669378
Number of instructions (use simInsts)	428283923
Number of µOps (use simOps)	428283954
Cache hit ratio for L1I	0.999956
Cache hit ratio for L1D	0.969736
Total DRAM bandwidth	646657492 Byte/Second

Riscv – fmm

Benchmark execution time in seconds	0.266382
Instructions per cycle (IPC)	0.670811
Number of instructions (use simInsts)	178692073
Number of µOps (use simOps)	178697653
Cache hit ratio for L1I	0.999904
Cache hit ratio for L1D	0.983898
Total DRAM bandwidth	149354582 Byte/Second

b. Experiment Commands

For this part, I used **se.py** to run my simulations. I modified the example command provided in the MP1 document to match the configurations needed for each benchmark. Below are the commands I ran for each benchmark:

X86 - fft:

```
build/X86/gem5.fast --outdir=m5out/mp1/part3.3/x86/fft/ configs/deprecated/example/se.py \
        --cpu-type=X86MinorCPU \
        --cpu-clock=1GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=16kB \
        --l1i_assoc=4 \
        --l1d_size=32kB \
        --l1d_assoc=8 \
        --num-cpus=1 \
        --cmd=mp1_provided/X86/FFT --options="-p1 -m16"
```

X86 - Cholesky:

```
build/X86/gem5.fast --outdir=m5out/mp1/part3.3/x86/cholesky/
configs/deprecated/example/se.py \
        --cpu-type=X86MinorCPU \
        --cpu-clock=1GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=16kB \
        --l1i_assoc=4 \
        --l1d_size=32kB \
        --l1d_assoc=8 \
        --num-cpus=1 \
        --cmd=mp1_provided/X86/CHOLESKY --options="-p1
mp1_provided/CHOLESKY_INPUT/tk15.O"
```

X86 - fmm:

```
build/X86/gem5.fast --outdir=m5out/mp1/part3.3/x86/fmm/ configs/deprecated/example/se.py \
        --cpu-type=X86MinorCPU \
        --cpu-clock=1GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=16kB \
        --l1i_assoc=4 \
        --l1d_size=32kB \
        --l1d_assoc=8 \
        --num-cpus=1 \
        --cmd=mp1_provided/X86/FMM --options="-p1"
```

Riscv – fft:

```
build/RISCV/gem5.fast --outdir=m5out/mp1/part3.3/riscv/fft/ configs/deprecated/example/se.py \
        --cpu-type=RiscvMinorCPU \
        --cpu-clock=1GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=16kB \
        --l1i_assoc=4 \
        --l1d_size=32kB \
        --l1d_assoc=8 \
        --num-cpus=1 \
        --cmd=mp1_provided/RISCV/FFT --options="-p1 -m16"
```

Riscv – Cholesky:

```
build/RISCV/gem5.fast --outdir=m5out/mp1/part3.3/riscv/cholesky/
configs/deprecated/example/se.py \
        --cpu-type=RiscvMinorCPU \
        --cpu-clock=1GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=16kB \
        --l1i_assoc=4 \
        --l1d_size=32kB \
        --l1d_assoc=8 \
        --num-cpus=1 \
        --cmd=mp1_provided/RISCV/CHOLESKY --options="-p1
mp1_provided/CHOLESKY_INPUT/tk15.O"
```

Riscv – fmm:

```
build/RISCV/gem5.fast --outdir=m5out/mp1/part3.3/riscv/fmm/
configs/deprecated/example/se.py \
        --cpu-type=RiscvMinorCPU \
        --cpu-clock=1GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=16kB \
        --l1i_assoc=4 \
        --l1d_size=32kB \
        --l1d_assoc=8 \
        --num-cpus=1 \
        --cmd=mp1_provided/RISCV/FMM --options="-p1"
```

---

c. Answer Questions

◊ **Which ISA has better performance for each benchmark program, in terms of execution time and IPC? Is the performance difference significant enough for us to conclude that one ISA outperforms the other?**

Benchmark	x86 (seconds)	RISC-V (seconds)	Speedup (RISC-V over x86)
FFT	0.1308	0.0623	2.1x faster
Cholesky	0.9925	0.6398	1.55x faster
FMM	0.4548	0.2664	1.71x faster

Benchmark	x86 IPC	RISC-V IPC	IPC Gain (RISC-V over x86)
FFT	0.4157	0.7117	71% higher
Cholesky	0.4281	0.6694	56% higher
FMM	0.48	0.6708	40% higher

RISC-V is clearly faster than x86 for all benchmarks. The performance difference is **significant** (1.5x-2.1x speedup). The **higher IPC of RISC-V** shows it utilizes available cycles more efficiently.

I think this is because RISC-V has a simpler instruction set which allows more consistency on instruction throughput. The complexity of X86 leads to more overhead due to instruction decoding therefore increases execution time.

Looking at each benchmark separately. FFT requires more computation, where RISC-V wins due to its reduced instruction latency. Cholesky and FMM require more memory access. X86 allows multiple embedded memory accesses within an instruction, making cache behavior less predictable. This may potentially lead to more L1D cache miss compared to RISC-V where all memory access is done with load/store instructions, which is more predictable.

◊ **Which ISA has a better L1I cache hit ratio? Does the choice of ISA have a significant impact on L1D, L2, and L3 cache hit ratio, respectively?**

Benchmark	x86 L1I Hit Ratio	RISC-V L1I Hit Ratio	Difference
FFT	0.999772	0.999877	0.0001
Cholesky	0.999944	0.999956	0.00001
FMM	0.99977	0.999904	0.0001

Benchmark	x86 L1D Hit Ratio	RISC-V L1D Hit Ratio	Difference
FFT	0.9839	0.9777	x86 is better
Cholesky	0.9719	0.9697	x86 is better
FMM	0.9856	0.9839	x86 is better

RISC-V has a slightly better L1I hit ratio, but the difference is **negligible**. **x86 has a slightly better L1D hit ratio**, meaning its data accesses are slightly more cache-friendly. RISC-V is the clear winner in performance, but x86 has **marginally better** L1D cache efficiency.

I think why x86 has a slightly better L1D hit ratio is because CISC instructions are more complex and are allowed to perform multiple operations. It allows x86 to reuse dcache data leading to more cache hits.

3.4 Powerful Processors

dual-core, 2.8GHz, 64-byte cache line, 32KB 8-way L1 instruction cache, 64KB 8-way L1 data cache, per-core 2MB 16-way L2 cache, unified 6MB 24-way L3 cache

a. required simulation output statistics

statistic name

Benchmark execution time in seconds	simSeconds
Instructions per cycle (IPC)	avg(system.cpu0.ipc, system.cpu1.ipc)
Cache hit ratio for L1I	1 - avg(system.cpu0.icache.overallMissRate::total, system.cpu1.icache.overallMissRate::total)
Cache hit ratio for L1D	1 - avg(system.cpu0.dcache.overallMissRate::total, system.cpu1.dcache.overallMissRate::total)
Cache hit ratio for L2	1 - avg(system.cpu0.l2cache.overallMissRate::total, system.cpu1.l2cache.overallMissRate::total)
Cache hit ratio for L3	1 - system.l3.overallMissRate::total
Total DRAM bandwidth	system.mem_ctrls.dram.bwTotal::total

X86 – fft

Benchmark execution time in seconds	0.009348
Instructions per cycle (IPC)	1.513652
Cache hit ratio for L1I	0.999726
Cache hit ratio for L1D	0.9509855
Cache hit ratio for L2	0.40108699999999997
Cache hit ratio for L3	0.251169
Total DRAM bandwidth	353145085.0 Byte/Second

X86 – Cholesky

Benchmark execution time in seconds	0.065507
Instructions per cycle (IPC)	1.7381885
Cache hit ratio for L1I	0.9998985
Cache hit ratio for L1D	0.9649105
Cache hit ratio for L2	0.837538
Cache hit ratio for L3	0.131646
Total DRAM bandwidth	676161899.0 Byte/Second

X86 – fmm

Benchmark execution time in seconds	0.018939
Instructions per cycle (IPC)	2.169289
Cache hit ratio for L1I	0.9998535
Cache hit ratio for L1D	0.9830985
Cache hit ratio for L2	0.781736
Cache hit ratio for L3	0.040487
Total DRAM bandwidth	53651611.0 Byte/Second

Riscv – fft

Benchmark execution time in seconds	0.007564
Instructions per cycle (IPC)	2.182705
Cache hit ratio for L1I	0.999576
Cache hit ratio for L1D	0.932718
Cache hit ratio for L2	0.4016325
Cache hit ratio for L3	0.263888
Total DRAM bandwidth	438327811.0 Byte/Second

Riscv – Cholesky

Benchmark execution time in seconds	0.063845
Instructions per cycle (IPC)	1.570187
Cache hit ratio for L1I	0.999922
Cache hit ratio for L1D	0.947829
Cache hit ratio for L2	0.84264
Cache hit ratio for L3	0.1334
Total DRAM bandwidth	685534231.0 Byte/Second

Riscv – fmm

Benchmark execution time in seconds	0.019326
Instructions per cycle (IPC)	1.7357875
Cache hit ratio for L1I	0.9998995
Cache hit ratio for L1D	0.9756835
Cache hit ratio for L2	0.789101
Cache hit ratio for L3	0.038828
Total DRAM bandwidth	50104794.0 Byte/Second

b. Experiment Commands

For this part, I used **se.py** to run my simulations. I modified the example command provided in the MP1 document to match the configurations needed for each benchmark. Below are the commands I ran for each benchmark:

X86 – fft

```
build/X86/gem5.fast --outdir=m5out/mp1/part3.4/x86/fft/ configs/deprecated/example/se.py \
        --cpu-type=X86O3CPU \
        --cpu-clock=2.8GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=32kB \
        --l1i_assoc=8 \
        --l1d_size=64kB \
        --l1d_assoc=8 \
        --l2cache \
        --l2_size=2MB \
        --l2_assoc=16 \
        --l3cache \
        --l3_size=6MB \
        --l3_assoc=24 \
        --num-cpus=2 \
        --cmd=mp1_provided/X86/FFT --options="-p2 -m16"
```

X86 – Cholesky

```
build/X86/gem5.fast --outdir=m5out/mp1/part3.4/x86/cholesky/
configs/deprecated/example/se.py \
        --cpu-type=X86O3CPU \
        --cpu-clock=2.8GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=32kB \
        --l1i_assoc=8 \
        --l1d_size=64kB \
        --l1d_assoc=8 \
        --l2cache \
        --l2_size=2MB \
        --l2_assoc=16 \
        --l3cache \
        --l3_size=6MB \
        --l3_assoc=24 \
        --num-cpus=2 \
        --cmd=mp1_provided/X86/CHOLESKY --options="-p2
mp1_provided/CHOLESKY_INPUT/tk15.O"
```

X86 - fmm

```
build/X86/gem5.fast --outdir=m5out/mp1/part3.4/x86/fmm/ configs/deprecated/example/se.py \
        --cpu-type=X86O3CPU \
        --cpu-clock=2.8GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=32kB \
        --l1i_assoc=8 \
        --l1d_size=64kB \
        --l1d_assoc=8 \
        --l2cache \
        --l2_size=2MB \
        --l2_assoc=16 \
        --l3cache \
        --l3_size=6MB \
        --l3_assoc=24 \
        --num-cpus=2 \
        --cmd=mp1_provided/X86/FMM --options="-p2"
```

Riscv – fft

```
build/RISCV/gem5.fast --outdir=m5out/mp1/part3.4/riscv/fft/ configs/deprecated/example/se.py \
        --cpu-type=RiscvO3CPU \
        --cpu-clock=2.8GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=32kB \
        --l1i_assoc=8 \
        --l1d_size=64kB \
        --l1d_assoc=8 \
        --l2cache \
        --l2_size=2MB \
        --l2_assoc=16 \
        --l3cache \
        --l3_size=6MB \
        --l3_assoc=24 \
        --num-cpus=2 \
        --cmd=mp1_provided/RISCV/FFT --options="-p2 -m16"
```

Riscv – Cholesky

```
build/RISCV/gem5.fast --outdir=m5out/mp1/part3.4/riscv/cholesky/
configs/deprecated/example/se.py \
        --cpu-type=RiscvO3CPU \
        --cpu-clock=2.8GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=32kB \
        --l1i_assoc=8 \
        --l1d_size=64kB \
        --l1d_assoc=8 \
        --l2cache \
        --l2_size=2MB \
        --l2_assoc=16 \
        --l3cache \
        --l3_size=6MB \
        --l3_assoc=24 \
        --num-cpus=2 \
        --cmd=mp1_provided/RISCV/CHOLESKY --options="-p2
mp1_provided/CHOLESKY_INPUT/tk15.O"
```

Riscv – fmm

```
build/RISCV/gem5.fast --outdir=m5out/mp1/part3.4/riscv/fmm/
configs/deprecated/example/se.py \
        --cpu-type=RiscvO3CPU \
        --cpu-clock=2.8GHz \
        --cacheline_size=64 \
        --caches \
        --l1i_size=32kB \
        --l1i_assoc=8 \
        --l1d_size=64kB \
        --l1d_assoc=8 \
        --l2cache \
        --l2_size=2MB \
        --l2_assoc=16 \
        --l3cache \
        --l3_size=6MB \
        --l3_assoc=24 \
        --num-cpus=2 \
        --cmd=mp1_provided/RISCV/FMM --options="-p2"
```

---

c. Answer Questions

◊ **Which ISA has better performance for each benchmark program, in terms of execution time and IPC? Is the performance difference significant enough for us to conclude that one ISA outperforms the other?**

Benchmark	x86 (seconds)	RISC-V (seconds)	Speedup (RISC-V over x86)
FFT	0.009348	0.007564	1.24x faster
Cholesky	0.065507	0.063845	1.03x faster
FMM	0.018939	0.019326	x86 is 1.02x faster

Benchmark	x86 IPC	RISC-V IPC	IPC Gain (RISC-V over x86)
FFT	1.5137	2.1827	44% higher
Cholesky	1.7382	1.5702	x86 is 11% higher
FMM	2.1693	1.7358	x86 is 20% higher

RISC-V is slightly **faster** for **FFT and Cholesky**, but **x86** is **marginally faster** for **FMM**. The differences are much smaller than in the powerful processor case. RISC-V has a significantly higher IPC for FFT, but x86 has a better IPC for Cholesky and FMM. Unlike the weak processor results, the performance gap is much smaller. **The difference is not large enough to conclude that one ISA is consistently better.**

Comparing the performance difference to the single-core weak processor scenario, we see that the gap is significantly smaller. This is because with a dual-core, high-performance setup, each system benefits from more functional units, increased parallelism, and better workload distribution. As a result, the impact of ISA efficiency is reduced, making the performance difference between x86 and RISC-V much smaller.

Analyzing each benchmark separately: FFT is compute-heavy and relies less on memory access, allowing RISC-V's simpler ISA to reduce instruction decoding overhead, leading to lower execution time and higher IPC. However, in FMM and Cholesky, x86 performs better because the L2 and L3 cache layers significantly reduce the penalty for L1 cache misses. In this scenario, x86's instruction complexity is no longer a disadvantage but an advantage, as its higher code density and ability to perform multiple memory operations within a single instruction improve execution efficiency.

◊ **Which ISA has a better L1I cache hit ratio? Does the choice of ISA have a significant impact on L1D, L2, and L3 cache hit ratio, respectively?**

Benchmark	x86 L1I Hit Ratio	RISC-V L1I Hit Ratio	Difference
FFT	0.999726	0.999576	x86 is better
Cholesky	0.9998985	0.999922	RISC-V is better
FMM	0.9998535	0.9998995	RISC-V is better

Benchmark	x86 L1D Hit Ratio	RISC-V L1D Hit Ratio	Difference
FFT	0.9509	0.9327	x86 is better
Cholesky	0.9649	0.9478	x86 is better
FMM	0.9831	0.9757	x86 is better

Benchmark	x86 L2	RISC-V L2	x86 L3	RISC-V L3
FFT	0.4011	0.4016	0.2512	0.2639
Cholesky	0.8375	0.8426	0.1316	0.1334
FMM	0.7817	0.7891	0.0405	0.0388

x86 consistently has better L1D hit ratios across all benchmarks. **L2 and L3 differences are negligible, meaning ISA choice does not significantly impact multi-level caching**. **Neither ISA dominates.** Performance depends on the specific benchmark, and cache behaviors are similar except for **x86's slightly better L1D efficiency**.

x86 achieves a higher L1D cache hit ratio because its CISC architecture allows a single instruction to perform multiple memory operations while reusing the same cached data, reducing the likelihood of cache misses. In contrast, RISC-V follows a strict load/store architecture, where each memory operation is handled separately. Since RISC-V requires more individual load/store instructions to complete the same workload as x86, it increases the chances of cache evictions and misses, resulting in a lower L1D cache hit ratio.

4. Answer Questions

◊ **It may be expected that since CISC ISAs have higher code density (i.e., less assembly instructions for the same program) than RISC ISAs, CISC may have better performance by saving memory bandwidth. Is this really the case, based on your experiments? Why or why not?**

Weak processor, Single layer cache:

Benchmark	x86 **DRAM Bandwidth** (Bytes/sec)	RISC-V **DRAM Bandwidth** (Bytes/sec)	RISC-V vs. x86
FFT	147,459,629	309,494,813	2.1x higher
Cholesky	416,623,901	646,657,492	1.55x higher
FMM	89,439,540	149,354,582	1.67x higher

Benchmark	x86 (**execution seconds**)	RISC-V (**execution seconds**)	Speedup (RISC-V over x86)
FFT	0.1308	0.0623	2.1x faster
Cholesky	0.9925	0.6398	1.55x faster
FMM	0.4548	0.2664	1.71x faster

Looking at the DRAM bandwidth comparison, we can see that RISC-V consumes significantly more bandwidth than x86. This is because RISC-V does not support multiple memory operations per instruction and follows a simpler ISA that requires more frequent memory access. However, despite using more DRAM bandwidth, RISC-V still outperforms x86 by 1.5x ~ 2x in execution time, proving that higher memory bandwidth usage does not necessarily mean lower performance. Several other factors contribute to this outcome:

- x86 instructions are more complex and require multi-stage decoding, leading to pipeline stalls and lower IPC compared to RISC-V.
- The absence of L2 and L3 caches increases cache misses, especially for x86, which tends to execute multiple memory operations in a single instruction. This can cause more frequent L1D cache misses, leading to stalls.
- CISC's complex instruction structure introduces more dependency stalls, as more operands are accessed within a single instruction. This is particularly problematic in

a weak processor with limited instruction-level parallelism (ILP), where stalling has a greater impact on performance.

Powerful processor, 3-layer cache:

Benchmark	x86 **DRAM Bandwidth** (Bytes/sec)	RISC-V **DRAM Bandwidth** (Bytes/sec)	RISC-V vs. x86
FFT	353,145,085	438,327,811	1.24x higher
Cholesky	676,161,899	685,534,231	1.01x higher
FMM	53,651,611	50,104,794	x86 uses slightly more

Benchmark	x86 (**execution seconds**)	RISC-V (**execution seconds**)	Speedup (RISC-V over x86)
FFT	0.009348	0.007564	1.24x faster
Cholesky	0.065507	0.063845	1.03x faster
FMM	0.018939	0.019326	x86 is 1.02x faster

Looking at these two tables, two interesting observations stand out:

1. The DRAM bandwidth difference between x86 and RISC-V is significantly smaller in the multi-layer cache system (1x~1.2x) than in the single-layer cache system (1.5x~2x).
2. The results suggest a counterintuitive trend where higher DRAM bandwidth usage correlates with better performance, contradicting the expected assumption that more DRAM access would lead to worse performance. For example, in FFT and Cholesky, RISC-V uses slightly more DRAM bandwidth and remains slightly faster than x86, while in FMM, x86 uses slightly more DRAM bandwidth and outperforms RISC-V.

The reasoning behind the first observation is that L2 and L3 caches absorb a significant portion of memory traffic, reducing the need for frequent DRAM accesses. Since both ISAs benefit from multi-level caching, the DRAM bandwidth gap between them shrinks. Additionally, with a dual-core system and more powerful processors, factors like prefetching, improved memory scheduling, and increased instruction-level parallelism help optimize DRAM access, **making DRAM bandwidth no longer the primary bottleneck for performance.**

Because DRAM bandwidth is no longer a bottleneck, this also means that higher DRAM bandwidth usage does **not** directly cause the observed performance improvement. Instead, the increased DRAM bandwidth usage is a byproduct of an optimized system that efficiently utilizes available empty cache spaces (for example prefetching). Therefore, our second observation is **misleading**. The actual boost in performance is likely driven by other factors, such as better cache efficiency, reduced memory stalls, and improved execution parallelism, which allow the system to make better use of DRAM without being hindered by memory latency.

Conclusion:

Our results show that higher code density in x86 does **not** directly lead to better performance by reducing memory bandwidth usage. In the single-layer cache system, RISC-V uses significantly more DRAM bandwidth than x86 but still outperforms it due to **simpler instruction decoding** and **higher IPC**. In the multi-layer cache system, the DRAM bandwidth difference shrinks because **L2 and L3 caches** absorb most memory accesses, making DRAM bandwidth less of a bottleneck for performance. The key takeaway is that memory bandwidth usage alone does not determine performance; instead, **caching efficiency, instruction scheduling, and execution parallelism play a larger role in overall system performance.**

◊ **Compare and contrast your findings in Part I and Part II. Based on your experiments, can you conclude which one has more impact on processor performance: the choice of ISA, or the microarchitectural design choices**

Benchmark	x86 (**execution seconds**)	RISC-V (**execution seconds**)	Speedup (RISC-V over x86)
FFT	0.1308	0.0623	2.1x faster
Cholesky	0.9925	0.6398	1.55x faster
FMM	0.4548	0.2664	1.71x faster

Weak processor, Single layer cache

Benchmark	x86 (**execution seconds**)	RISC-V (**execution seconds**)	Speedup (RISC-V over x86)
FFT	0.009348	0.007564	1.24x faster
Cholesky	0.065507	0.063845	1.03x faster
FMM	0.018939	0.019326	x86 is 1.02x faster

Powerful processor, dual core, 3-layer cache

Conclusion:

Based on our experimental results, **microarchitecture has a far greater impact on processor performance than ISA**. While ISA affects instruction decoding efficiency and memory access patterns, the shift from a single-core, single-layer cache system to a powerful dual-core, multi-layer cache system had a much greater influence on overall performance. As hardware becomes more advanced, the performance gap between ISAs shrinks, proving that microarchitecture optimizations benefit both ISAs and minimize ISA-specific differences. This is evident in the two tables: while the weak processor setup shows a **1.5x-2x** performance gap, the powerful processor setup reduces the gap to just **1x-1.2x**. This demonstrates that as hardware complexity increases, ISA-specific advantages become less relevant. This conclusion is also supported by the findings in Question 1, where multi-level caching, prefetching, and increased instruction-level parallelism significantly reduced the differences in performance, cache hit rates, and DRAM bandwidth between x86 and RISC-V, further proving that hardware advancements diminish ISA-specific advantages.