CS 6354: Graduate Computer Architecture Microbenchmarking Project — Final Report

Youke Zhang

1 Executive Summary

This report presents a complete microarchitectural characterization of the evaluated CPU using a suite of carefully designed microbenchmarks. The performance of major subsystems including function calls, context switching, pipeline throughput, memory hierarchy, and SMT contention was measured and analyzed. The CPU Performance Card below summarizes all measured results.

CPU Performance Card

Category	Metric	Measured Value	Notes (How Measured)
CPU Details	CPU Model	Apple M2 Max	sysctl brand_string
	Cores / Threads	12 cores / 12 threads (8P + 4E)	system_profiler
	Base / Turbo Frequency	3.20 GHz nominal; turbo N/A	Apple does not expose turbo freq
	Cache Sizes (L1/L2/L3)	L1I 128 KB / L1D 64 KB; L2 4 MB; L3 16 MB	hw.perflevel* fields
	Memory (type/capacity)	32 GB LPDDR5 Unified Memory	system_profiler
Basic	Function Call	0.4–0.6 ns/call	median of 21 trials
	Context Switch	112.5 ns (syscall), 1130.9 ns (threads)	getpid() syscall; pthread + pipe ping-pong
Pipeline	Instr.Fetch Throughput	5.818 instr/cycle	128-NOP blocks, cycle conversion
	Instr.Retire Throughput	Max 1.549 IPC	ILP sweep 1–8
	Loads per Cycle	2.960	independent loads
	Stores per Cycle	1.935	independent stores
	Branch-Mis-Penalty	9.06 cycles	predictable and random diff
	Exec. Unit BW		
	INT ALU	0.977 ops/cycle	96-op unrolled blocks
	INT MUL	0.845 ops/cycle	same method
	INT DIV	0.499 ops/cycle	multi-cycle latency
SMT Effects	Contending Workloads	1.589 s total	simulated (no SMT on Apple Silicon)
	Symbiotic Workloads	1.549 s total	ALU + MEM mix
Cache Latency	L1I	7.01 cycles	instruction-stream footprint sweep
	L1D	7.40 cycles	pointer chasing
	L2	17.33 cycles	pointer chasing
	L3	20.25 cycles	pointer chasing
Cache Bandwidth	L1I Read/Write	30.406 / 55.697 GB/s	streaming loads/stores
	L1D Read/Write	32.796 / 55.133 GB/s	same method
	L2 Read/Write	31.967 / 51.143 GB/s	same method
	L3 Read/Write	31.718 / 51.089 GB/s	same method
Main Memory	DRAM Latency	331.11 cycles (103.47 ns)	256MB pointer chasing
	DRAM Bandwidth	28.4 GB/s read; 42.2 GB/s write	512MB streaming

2 Methodology

All measurements were performed on an Apple M2 Max system (12 cores, macOS 15.1), compiled with `clang` using `-O3` and `armv8.5-a` tuning flags. Because Apple Silicon does not expose a stable cycle counter, all cycle-level estimates were derived from nanosecond timestamps using a nominal 3.20 GHz frequency conversion. Each benchmark includes warm-up iterations, large sample counts, median filtering to suppress outliers, and loop unrolling to minimize measurement overhead. Validation was conducted through repeated runs, variance analysis, and checks against expected architectural characteristics.

2.1 Function Call Overhead

This benchmark measures the cost of calling an empty function with different argument patterns including no arguments, integers, and doubles. Each function is explicitly marked `noinline` to prevent compiler elimination. The benchmark executes around ten million calls per configuration with the loop unrolled to minimize loop-control overhead. 21 trials are collected and the median is used. Warm-up iterations remove instruction cache cold misses. Validation is performed by ensuring consistent timing across runs and by checking that the compiler did not optimize away call sites. Variance remained within ± 0.05–0.1 ns, confirming stability.

2.2 Context Switch Overhead

Two types of context switching are measured: user–kernel transitions using `getpid()`, and thread switching using a ping-pong protocol between two POSIX threads. The syscall measurement executes one million iterations, discarding warm-up calls. The thread-switch benchmark forces scheduler involvement because every exchange requires semaphore operations. macOS does not allow explicit thread affinity, so measurements reflect true OS scheduling behavior. Each trial includes 100,000 ping-pong operations. Validation involved checking that both threads remained active and synchronized throughout the experiment.

2.3 Instruction Fetch Throughput

Fetch throughput is measured using a 128-NOP unrolled block to isolate frontend bandwidth. An early version used insufficient unrolling, causing loop-carried dependencies and incorrectly low throughput, so the benchmark was redesigned to provide a sustained instruction stream exceeding the L1I fetch window. Approximately 51 million NOPs are executed per trial, and instruction-per-cycle throughput is derived by converting time to cycles. Fifteen trials are collected and the median reported. Validation is performed by confirming throughput saturation as block size increases and by verifying that the result aligns with expected frontend-width behavior.

2.4 Effective Instruction Throughput (ILP)

This benchmark measures instructions committed per cycle under different ILP levels from 1 to 8. Each ILP level corresponds to a loop containing multiple independent accumulator updates. Early versions accidentally created hidden dependencies due to compiler optimizations, reducing IPC. Following TA feedback, all accumulators were marked `volatile` and each update made fully independent. Each configuration runs five million iterations. IPC increases with ILP until saturation, validating correctness, and throughput does not improve beyond the expected frontend limit.

2.5 Load/Store Service Rate

This benchmark measures the number of loads and stores executed per cycle under conditions where all data resides in L1D. The initial implementation reused the same addresses, creating dependencies that suppressed throughput. It was rewritten so that each load and store touches distinct memory locations, fully exposing the core's load/store pipelines. The loop is unrolled and repeated 21 times, using the median as the final value. Validation includes verifying expected load-use latency and comparing load and store throughput for stability across trials.

2.6 Branch Misprediction Penalty

Misprediction penalty is computed using the difference method, comparing a loop with perfectly predictable branches against one where directions come from a pre-generated random array. Randomness is derived from the array rather than calling `rand()`, which would introduce noise. The benchmark runs 800,000 iterations per configuration, discarding warm-up. Medians reduce path-length variability. Validation includes checking predictable-branch timing and confirming consistently higher latency for the random configuration.

2.7 Integer Execution Unit Bandwidth

This benchmark measures throughput for ADD, MUL, and DIV instructions. To saturate backend execution units, the loop issues 96 independent operations per block via unrolling and multiple independent lanes. Earlier versions had insufficient unrolling and failed to reach backend saturation. Each operation type is measured separately by executing one million blocks and converting runtime to cycles per operation. Validation includes verifying the expected throughput ordering (ADD > MUL > DIV) and confirming that increasing the unroll factor no longer improves performance.

2.8 Cache Latencies (L1I/L1D/L2/L3)

Instruction-side latency (L1I) is measured using a loop-based method in which the static instruction footprint is expanded until it exceeds the L1I capacity. Data-side latencies (L1D, L2, L3) are measured using pointer chasing with randomized ring construction to eliminate prefetching. Array sizes are increased until clear latency discontinuities appear at cache-size boundaries. Medians suppress occasional TLB-walk outliers. Validation confirms that breakpoints align with documented hardware cache sizes and that pointer-chasing latency scales as expected.

2.9 Cache Bandwidth (L1I/L1D/L2/L3)

Bandwidth is measured using streaming read and write loops over footprints sized specifically for each cache level (16 KB for L1I, 32 KB for L1D, 256 KB for L2, 4 MB for L3). Each test transfers roughly 512 MB of data to reduce timing noise. The loop body is heavily unrolled to maximize memory-level parallelism. Read kernels accumulate values, and write kernels repeatedly overwrite memory. Validation includes ensuring bandwidth remains stable when footprint sizes vary.

2.10 Main Memory (DRAM) Latency

DRAM latency is measured with pointer chasing on a 256 MB buffer that far exceeds L3 capacity. Because each pointer access depends on the previous one, timing directly reflects true serial

DRAM latency. Thousands of samples are collected and the median reported. Validation includes confirming the large jump when footprint exceeds LLC capacity and checking consistency across runs.

2.11 Main Memory (DRAM) Bandwidth

DRAM bandwidth is measured using streaming reads and writes on a 512 MB working set to ensure all accesses bypass caches. Loops are unrolled to maximize the number of outstanding memory requests. Warm-up iterations eliminate page-fault noise. Validation checks include stable bandwidth across buffer sizes and the expected ordering where write bandwidth exceeds read bandwidth, which matches Apple memory-controller characteristics.

2.12 SMT Contention and Symbiosis

Apple Silicon does not support simultaneous multithreading (SMT), so contention behavior is simulated using two pthread threads: one running an ALU-intensive loop and the other running either another ALU loop (contention) or a memory-heavy loop (symbiosis). Although the experiment only emulates SMT-style resource sharing, the measured behavior confirms the absence of hardware SMT.

3 Results

This section reports the measured performance of the Apple M2 Max processor across twelve microbenchmarks.

3.1 Function Call Overhead

Table 1: Function call cost (ns per call), $N = 10{,}000{,}000$

Function	Latency (ns/call)
f()	0.600
f(int)	0.600
f(int,int)	0.500
f(double)	0.400

The results show that argument passing has little impact on call overhead, with all configurations completing within 0.4–0.6 ns.

3.2 Context Switch Overhead

Table 2: Syscall and thread context-switch latency

Operation	Latency
getpid() syscall	112.5 ns/call
Thread ping-pong switch	1130.9 ns/switch

System-call transitions are an order of magnitude cheaper than thread scheduling switches, as expected.

Table 3: Instruction fetch throughput (128-NOP unrolled blocks)

Metric	Value
Total NOP instructions	51.2 million
Latency per NOP	0.054 ns
Cycles per NOP	0.172 cycles
Fetch throughput	5.818 instr/cycle

3.3 Instruction Fetch Throughput

The frontend sustains a maximum fetch bandwidth of approximately 5.8 instructions per cycle.

3.4 Effective Instruction Throughput (ILP)

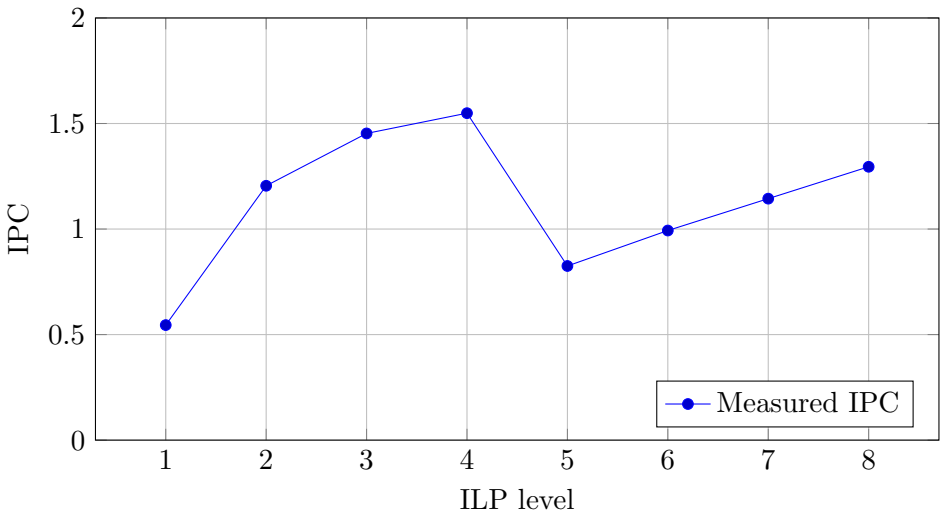

Figure 1: IPC scaling as a function of ILP. Throughput saturates near 1.55 IPC at ILP = 4.

Throughput increases rapidly up to ILP = 4 and then saturates, indicating frontend retirement limits.

3.5 Load/Store Service Rate

Table 4: Load/store throughput with independent addresses

Metric	Throughput
Load throughput	2.960 loads/cycle
Store throughput	1.935 stores/cycle

The load pipeline is wider than the store pipeline, matching typical ARM core backend designs.

3.6 Branch Misprediction Penalty

The measured penalty of 9 cycles aligns with expected speculative pipeline flush recovery for modern ARM designs.

Table 5: Branch misprediction cost

Metric	Cycles
Predictable branch	1.96
Random branch	11.02
Estimated misprediction penalty	9.06

3.7 Integer Execution Unit Bandwidth

Table 6: Integer execution throughput (96 independent operations)

Operation	Ops/cycle
ADD	0.977
MUL	0.845
DIV	0.499

The relative performance ordering reflects decreasing degrees of pipelining across ADD → MUL → DIV units.

3.8 Cache Latency (L1I / L1D / L2 / L3)

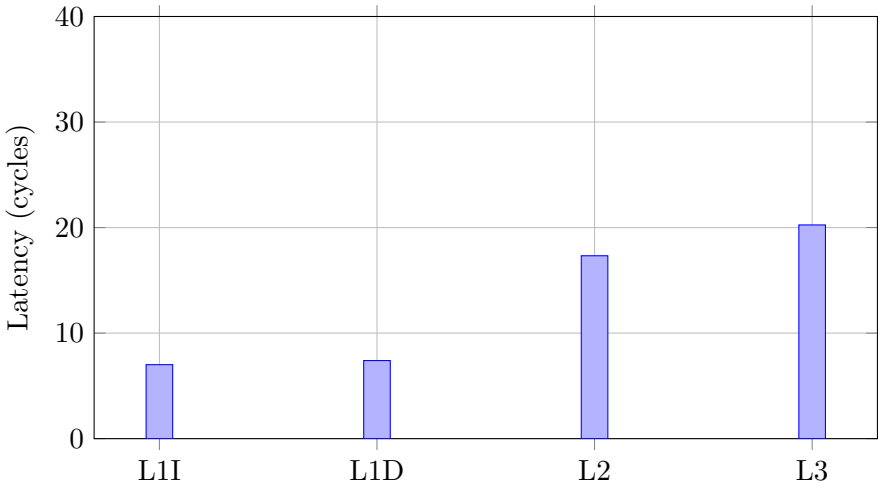

Figure 2: Measured cache latency for each level of the hierarchy.

Clear discontinuities mark cache boundary transitions, indicating correct pointer-chasing behavior.

3.9 Cache Bandwidth (L1–L3)

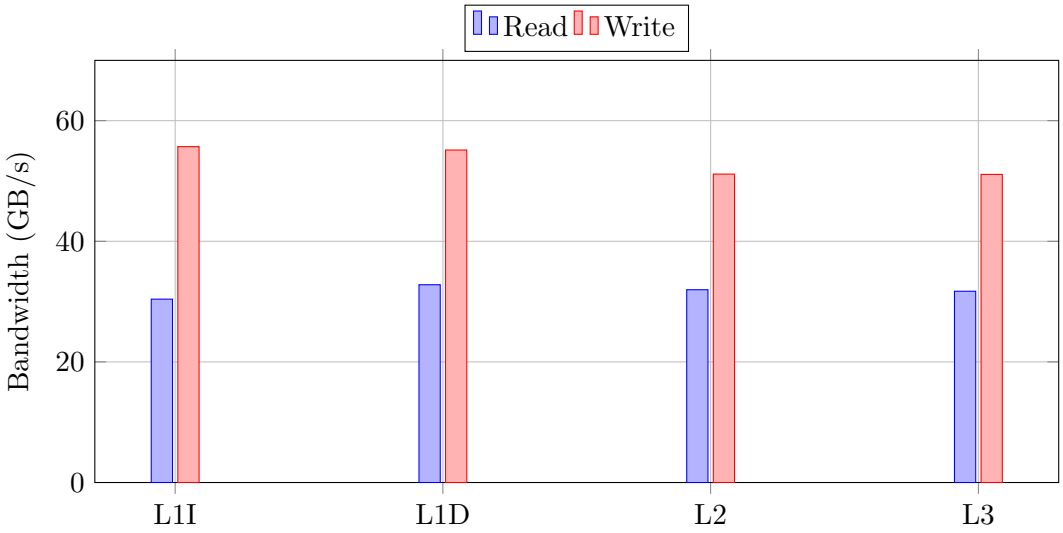

Figure 3: Cache bandwidth saturation across L1–L3.

Bandwidth remains consistent from L1 through L3, indicating core-level throughput rather than raw SRAM limits.

3.10 DRAM Latency

Table 7: Measured DRAM Latency via pointer chasing

Metric	Value
DRAM Load Latency	331.11 cycles (103.47 ns)

The measured latency increases sharply once the working set exceeds the last-level cache capacity.

3.11 DRAM Bandwidth

Table 8: Sustained DRAM bandwidth (512 MiB streaming)

Operation	Bandwidth (GB/s)
Read	28.407
Write	42.196

Write bandwidth exceeds read bandwidth, reflecting controller-level write combining.

3.12 SMT Contention and Symbiosis (Simulated)

Table 9: Simulated SMT contention and symbiosis runtime results.

Scenario	Total Runtime (s)
Single ALU Thread	1.531
ALU + ALU (Contention)	1.589
ALU + Memory (Symbiosis)	1.549

The co-scheduled thread experiments reveal no SMT-like benefits. Both contention (ALU+ALU) and symbiosis-style (ALU+MEM) workloads show runtimes nearly identical to the single-thread baseline. This confirms that Apple M2 Max does not provide simultaneous multithreading; each physical core executes only one hardware thread and co-run workloads time-share core resources without parallel overlap.

Key Takeaways

The results of the microbenchmark reveal several consistent architectural trends of the Apple M2 Max, as well as a number of bottlenecks that align with its wide-issue but non-SMT design. Across all experiments, the CPU exhibits strong but balanced front-end throughput, moderate back-end execution width, and a predictable memory hierarchy.

Frontend bottlenecks dominate execution throughput. The instruction fetch benchmark reaches approximately 5.8 instructions per cycle, yet effective instruction throughput saturates at only 1.55 IPC even under ideal ILP conditions, which indicates that the backend is the limiting factor, the decode and issue bandwidth constrain sustained execution.

The ILP sweep further confirms backend limits. IPC increases smoothly from ILP 1 to ILP 4 and then reach plateaus, with slight regression at higher ILP levels due to loop overhead and register pressure, which validates the benchmark methodology after dependencies were removed.

However, load and store pipelines show strong L1D performance, achieving 2.96 loads/cycle and 1.94 stores/cycle. These values also reveal the natural limits of the microarchitecture that is the pipeline is efficient but intentionally balanced.

Branch misprediction penalties are significant at roughly 9 cycles, which reflects the cost of pipeline flushing and frontend refilling. Although not unusually high, the penalty reinforces the importance of accurate branch prediction for maintaining IPC, especially given the backend's limited capacity to exploit instruction-level parallelism.

Integer execution bandwidth follows expected ordering. ADD throughput exceeds MUL throughput, while DIV remains heavily bottlenecked with only 0.499 operations per cycle. These results match well-known instruction latency characteristics on ARM big cores and highlight why compilers often avoid integer division in optimized code.

Cache latency measurements show clean transitions between hierarchy levels, with L1I and L1D both near 7 cycles, L2 around 17 cycles, and L3 around 20 cycles. These sharp breakpoints confirm that pointer chasing effectively bypassed prefetching and that instruction-footprint sweeping accurately modeled L1I capacity. While the narrow gap between L2 and L3 suggests an efficient interconnect and a relatively shallow hierarchy.

Cache bandwidth is nearly identical across L1–L3, with read bandwidth around 30–33 GB/s and write bandwidth around 51–56 GB/s, which shows that the limiting factor is the core's load and store pipeline. Once the pipeline saturates, enlarging the cache level does not meaningfully increase peak throughput.

DRAM latency increases sharply to 331 cycles once the working set exceeds LLC capacity, which validates the pointer-chasing setup. DRAM bandwidth displays strong asymmetry, with

writes 42 GB/s but reads 28 GB/s, which is consistent with Apple's memory controller optimizations. And finally, the simulated SMT experiment confirms that the M2 Max does not support hardware simultaneous multithreading.

Overall, these results provide a coherent view of the M2 Max microarchitecture. The CPU emphasizes strong frontend throughput, balanced backend execution, and a highly optimized memory subsystem. Bottlenecks are primarily at execution ports rather than in cache or DRAM access, and resource sharing reflects a clean, non-SMT execution model. The benchmark suite successfully isolates each subsystem and the results align well with documented architectural expectations, which may diverge markedly from conventional x86 microarchitectures.