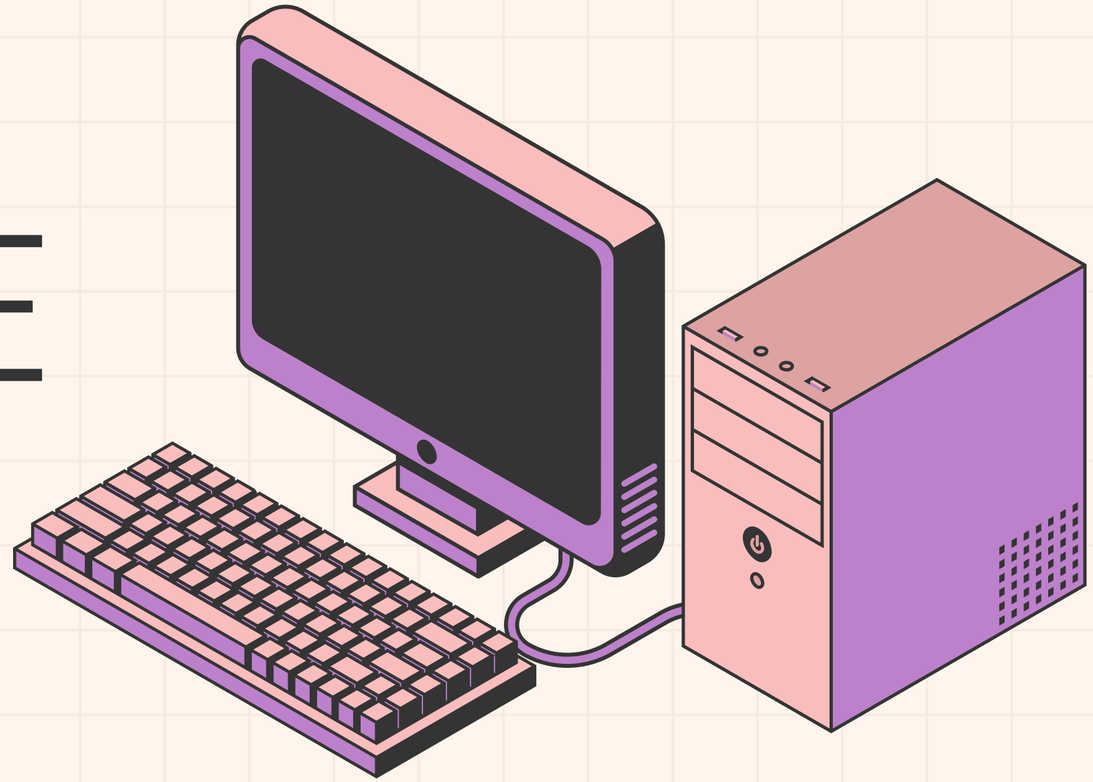
COMPUTER ARCHITECTURE

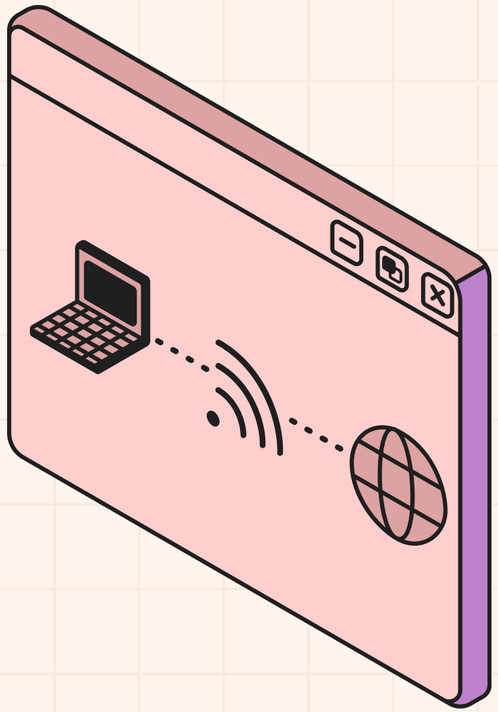


GROUP – 8

PRAKHAR PALOD	2301MC17
PRASHANT NIGAM	2301MC18
ABHINANDAN PORWAL	2301MC54
SHUBHAM PRAKASH	2302MC11

OVERVIEW

OUR PROJECT MEASURES MEMORY ACCESS LATENCY—THAT IS, THE DELAY BETWEEN REQUESTING AND RECEIVING DATA FROM MEMORY. THE IDEA IS TO SIMULATE VARIOUS MEMORY TRAVERSAL PATTERNS (USING POINTER CHASING AND INDEXED SCANNING) AND CAPTURE HOW LONG EACH ACCESS TAKES. THIS IS DONE BY:
INITIALIZING A LARGE MEMORY BUFFER.



PREPARING IT SO THAT ACCESSES JUMP AROUND BASED ON A GIVEN STRIDE.

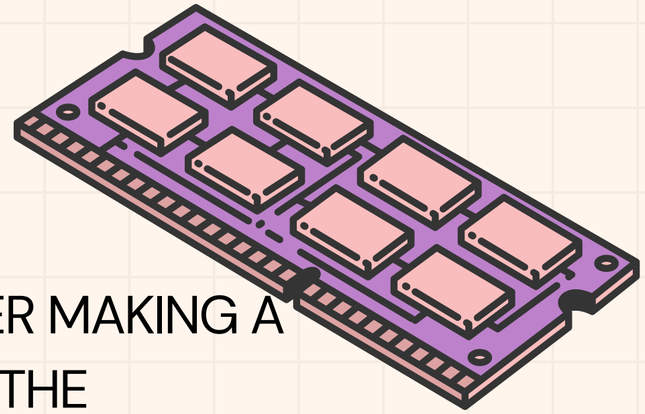
MEASURING THE AVERAGE ACCESS LATENCY OVER MANY ITERATIONS.

RUNNING TESTS UNDER DIFFERENT CONFIGURATIONS:

- FORWARD VS. BACKWARD POINTER SCANS: THESE DETERMINE HOW THE MEMORY IS NAVIGATED.
- INDEX-BASED SCANS VS. POINTER CHASING: TWO DIFFERENT METHODS TO ACCESS ELEMENTS.
- SINGLE-THREADED VS. CONCURRENT (TWO THREADS) EXECUTION.
- VARIOUS STRIDE VALUES.

FINALLY, RESULTS ARE RECORDED AND PLOTTED USING A LOGARITHMIC SCALE FOR THE X-AXIS (MEMORY SIZE) WITH VISUAL CUES FOR CACHE SIZES (L1, L2, L3).

MEMORY LATENCY AND MEMORY HEIRARCHY



MEMORY LATENCY IS THE TIME IT TAKES FOR THE CPU TO RETRIEVE DATA AFTER MAKING A REQUEST. IT SIGNIFICANTLY IMPACTS PERFORMANCE—NO MATTER HOW FAST THE PROCESSOR IS, SLOW MEMORY ACCESS CAN CAUSE DELAYS IN EXECUTION.

TO REDUCE LATENCY, MODERN SYSTEMS USE A MEMORY HIERARCHY. THIS INCLUDES SMALL, FAST STORAGE AT THE TOP (REGISTERS), FOLLOWED BY MULTIPLE LEVELS OF CACHE (L1, L2, L3), AND FINALLY THE LARGER BUT SLOWER MAIN MEMORY (DRAM). EACH LEVEL BALANCES SPEED AND SIZE: HIGHER LEVELS ARE QUICKER BUT HOLD LESS DATA.

CACHES STORE DATA IN BLOCKS CALLED CACHE LINES (TYPICALLY 64 BYTES). WHEN DATA IS ACCESSED, AN ENTIRE LINE IS LOADED, IMPROVING PERFORMANCE FOR SEQUENTIAL READS. IF THE REQUIRED DATA IS ALREADY IN CACHE (CACHE HIT), ACCESS IS QUICK. IF IT'S NOT (CACHE MISS), THE SYSTEM MUST FETCH IT FROM A SLOWER LEVEL, INCREASING LATENCY.

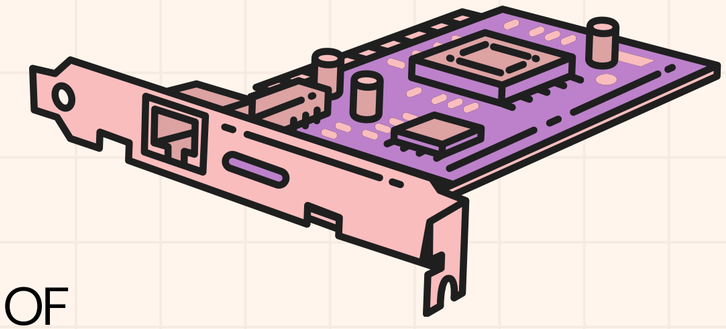
WHAT IS A STRIDE?

A STRIDE IS THE NUMBER OF BYTES BETWEEN SUCCESSIVE MEMORY ACCESSES DURING A SCAN. INSTEAD OF ACCESSING EVERY SINGLE BYTE OR EVERY ELEMENT ONE AFTER ANOTHER (WHICH IS CALLED STRIDE = 1), WE CAN ACCESS EVERY N-TH BYTE OR ELEMENT, WHERE N IS THE STRIDE SIZE.

WHY STRIDE MATTERS IN MEMORY ACCESS

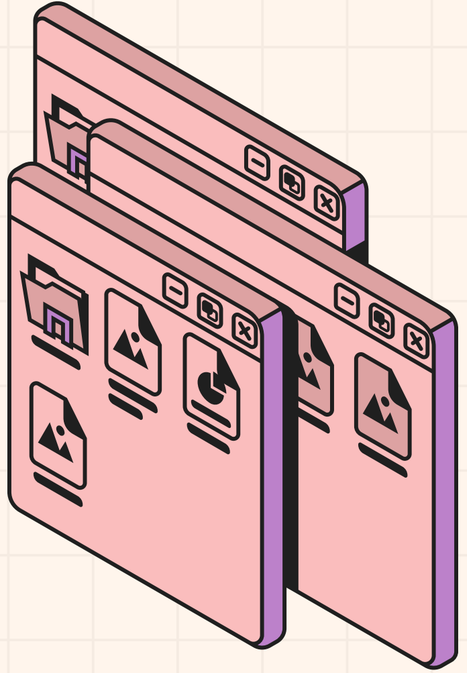
WHEN ACCESSING MEMORY, ESPECIALLY IN LARGE ARRAYS, THE STRIDE SIZE AFFECTS:

- CACHE UTILIZATION
- MEMORY BANDWIDTH USAGE
- OVERALL LATENCY



THIS IS BECAUSE MODERN CPUS LOAD MEMORY IN CACHE LINES—BLOCKS OF FIXED SIZE (USUALLY 64 BYTES). WHEN DATA IS ACCESSED, THE ENTIRE CACHE LINE IS LOADED INTO THE CACHE.

TYPES OF STRIDE ACCESS PATTERNS

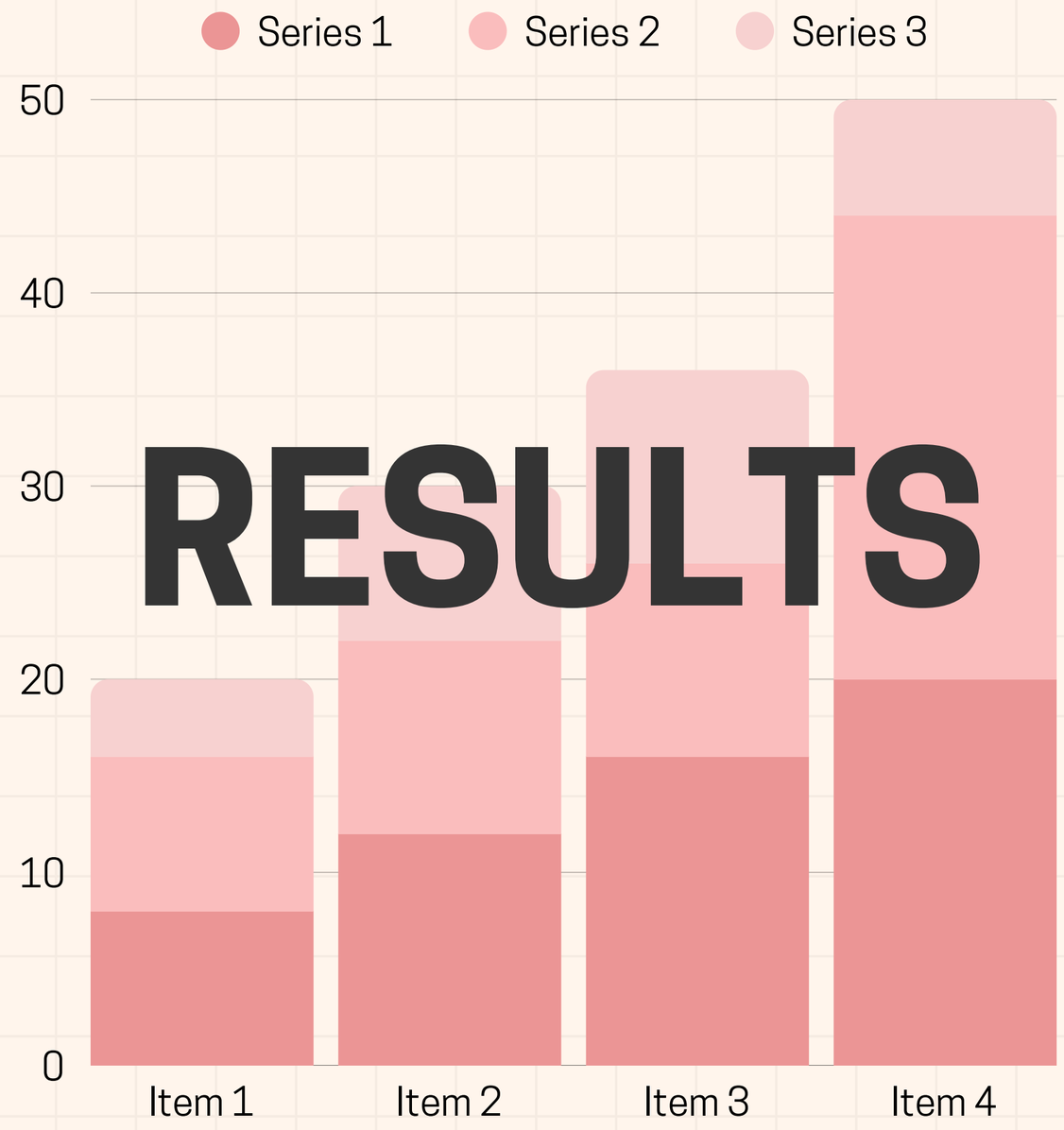


- SMALL STRIDE (E.G., 64 BYTES OR LESS):
  - ACCESSES HAPPEN WITHIN THE SAME OR ADJACENT CACHE LINES.
  - BETTER CACHE UTILIZATION.
  - HIGHER CHANCE OF CACHE HITS.
  - LOWER LATENCY.
- LARGE STRIDE (E.G., 128, 256, 512 BYTES OR MORE):
  - SKIPS MANY BYTES BETWEEN ACCESSES.
  - MORE LIKELY TO MISS CACHE LINES AND FETCH FROM HIGHER LEVELS OF MEMORY OR RAM.
  - HIGHER LATENCY.
  - CAN STRESS THE MEMORY SYSTEM AND REVEAL PERFORMANCE BOTTLENECKS.

WHAT WE DID IN THE PROJECT

IN OUR PROJECT, WE TESTED MEMORY ACCESS LATENCY USING DIFFERENT STRIDE SIZES (LIKE 128 B, 256 B, AND 512 B):

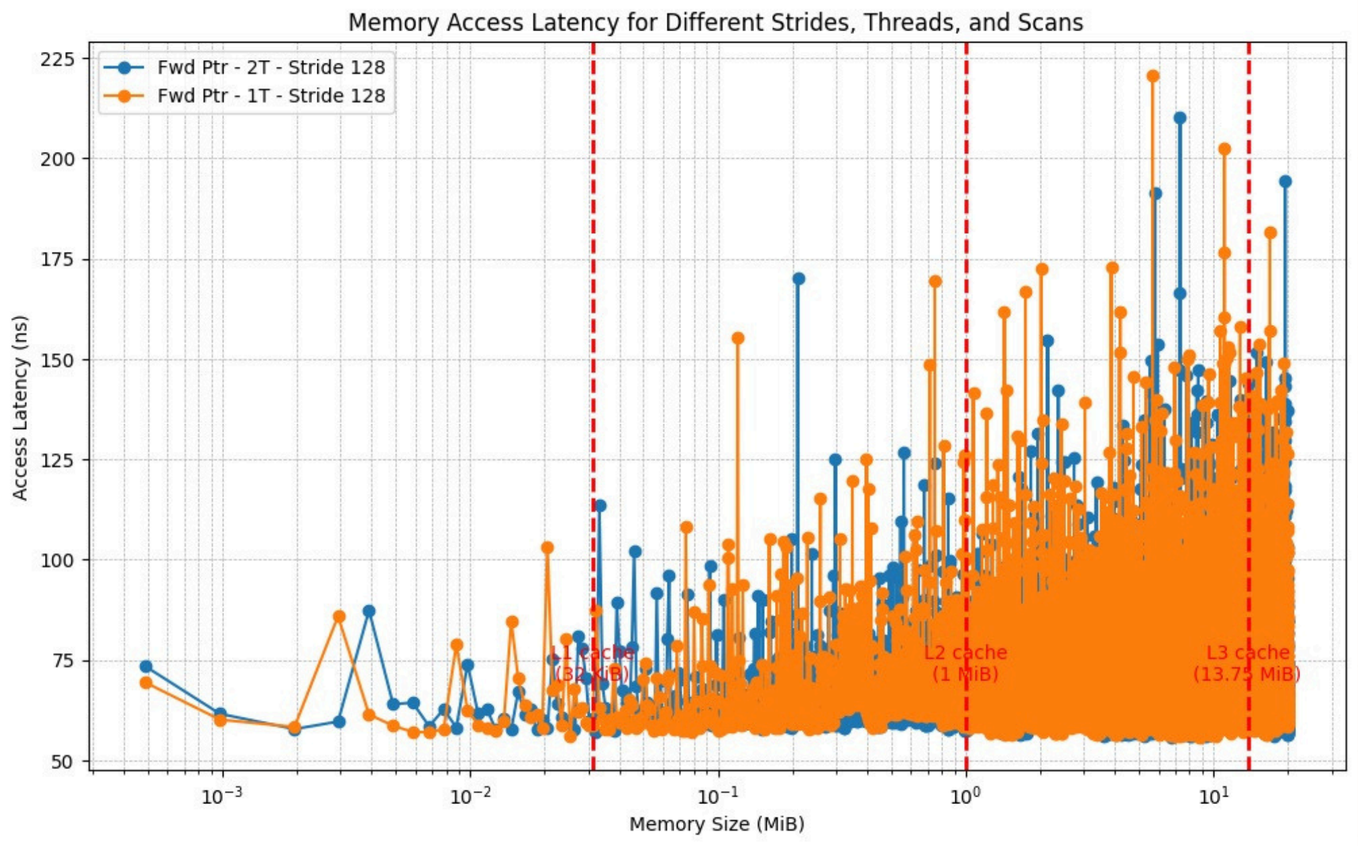
- WITH SMALL STRIDES, MEMORY ACCESSES STAY WITHIN THE SAME CACHE LINE OR NEARBY ONES.
- WITH LARGER STRIDES, WE INTENTIONALLY SKIPPED OVER MORE DATA TO SIMULATE POOR SPATIAL LOCALITY —FORCING MORE CACHE MISSES AND INCREASING ACCESS TIME.
- THIS HELPED US OBSERVE HOW STRIDE SIZE IMPACTS LATENCY AND HOW MEMORY HIERARCHY (L1, L2, L3, RAM) BEHAVES UNDER VARIOUS ACCESS PATTERNS.


Series 1
Series 2
Series 3
50
40
30
20
10
0
RESULTS
Item 1
Item 2
Item 3
Item 4

FORWARD POINTER WITH STANDARD STRIDE

KEY OBSERVATIONS

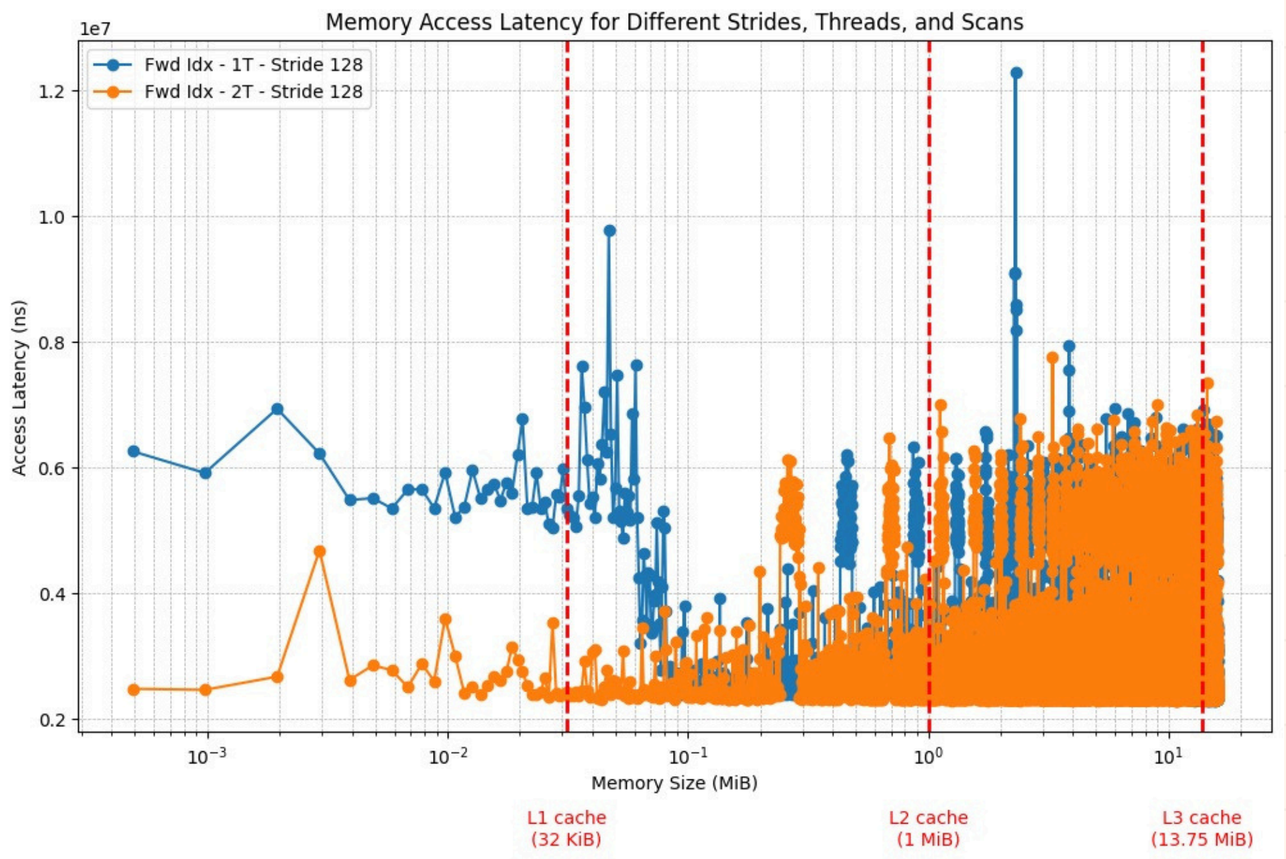
- BOTH THREAD CONFIGURATIONS SHOW SIMILAR BASELINE LATENCY IN SMALL MEMORY SIZES
- LATENCY GRADUALLY INCREASES AS MEMORY SIZE GROWS BEYOND L1 CACHE
- SINGLE-THREADED EXECUTION (ORANGE) SHOWS HIGHER PEAK LATENCIES, REACHING ABOUT 220 NS
- MORE LATENCY SPIKES OCCUR ONCE THE MEMORY SIZE EXCEEDS THE L2 CACHE (1 MIB)
- THE OVERALL LATENCY PATTERN IS MORE CHAOTIC THAN WITH INDEXING, SUGGESTING POINTER CHASING IS LESS PREDICTABLE
- LATENCY IS SIGNIFICANTLY HIGHER THAN IN INDEXING OPERATIONS (MEASURED IN HUNDREDS VS. MILLIONS)



FORWARD INDEXING WITH STANDARD STRIDE

KEY OBSERVATIONS

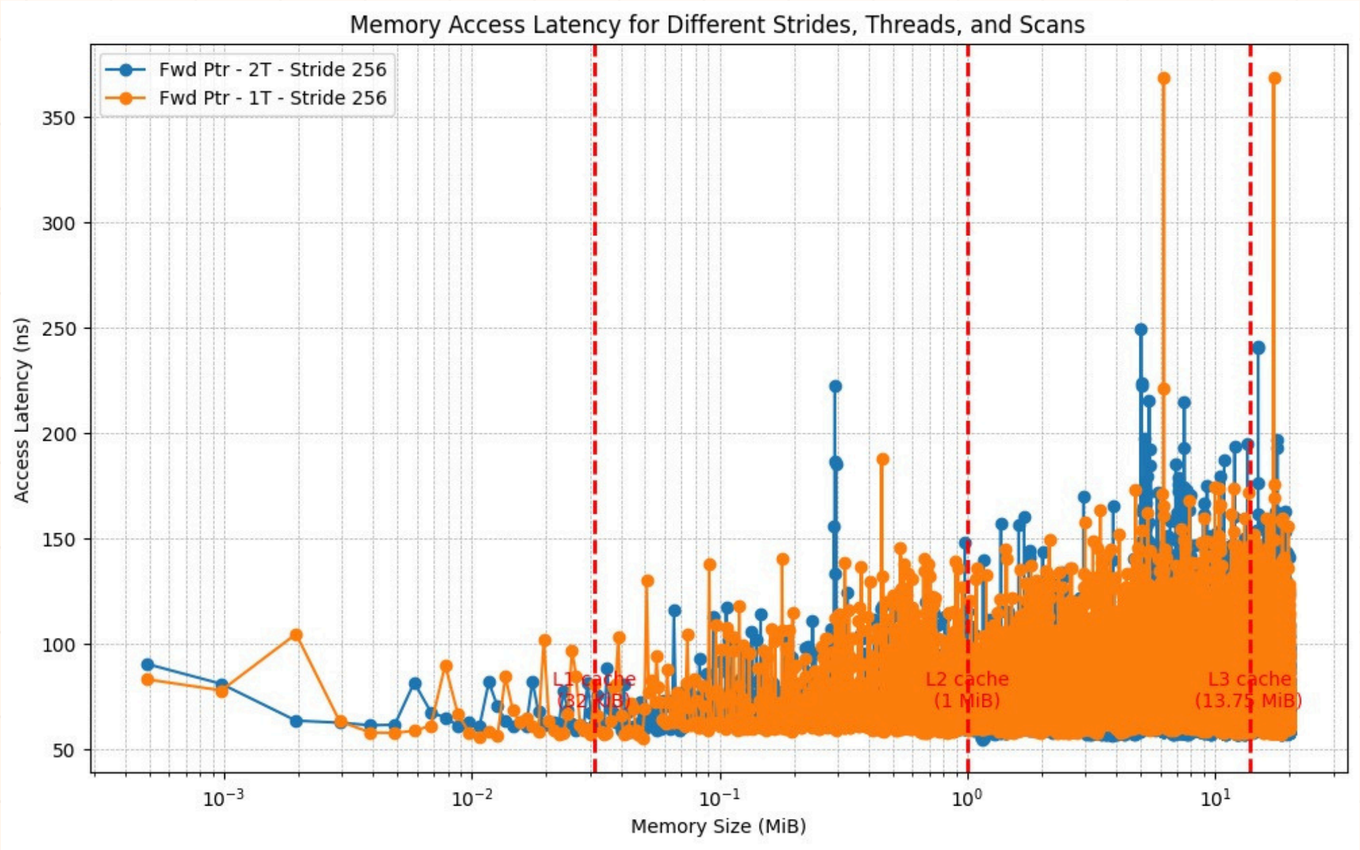
- SINGLE-THREADED ACCESS (BLUE) CONSISTENTLY SHOWS HIGHER LATENCY THAN TWO-THREADED ACCESS (ORANGE)
- SIGNIFICANT LATENCY SPIKES OCCUR AT VARIOUS MEMORY SIZES, PARTICULARLY AROUND 2-3 MIB
- THE LATENCY DIFFERENCE BETWEEN 1T AND 2T IS MOST PRONOUNCED IN SMALLER MEMORY SIZES
- MAXIMUM LATENCY REACHES APPROXIMATELY 12×10^7 NS FOR SINGLE-THREADED ACCESS
- TWO-THREADED EXECUTION SHOWS MORE STABLE PERFORMANCE WITH FEWER EXTREME SPIKES



FORWARD POINTER WITH VARIABLE STRIDE

KEY OBSERVATIONS

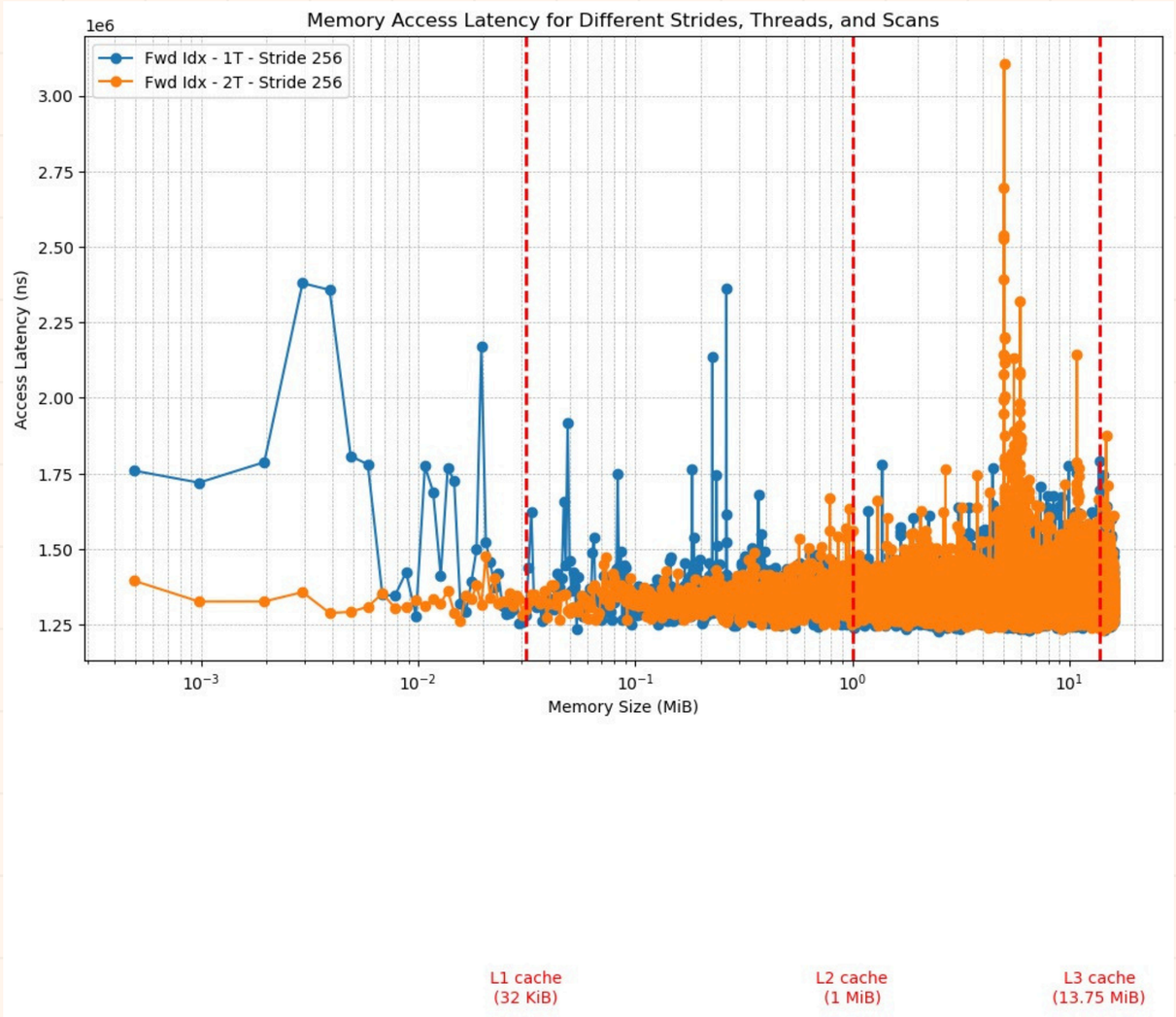
- BASELINE LATENCY STARTS AROUND 80-90 NS FOR BOTH THREAD CONFIGURATIONS
- LATENCY GRADUALLY INCREASES AS MEMORY SIZE GROWS
- EXTREME SPIKES OCCUR FOR SINGLE-THREADED EXECUTION (ORANGE), REACHING APPROXIMATELY 370 NS
- TWO-THREADED EXECUTION SHOWS FEWER EXTREME SPIKES BUT MORE CONSISTENT MODERATE SPIKES
- OVERALL PATTERN SHOWS INCREASING VOLATILITY AS MEMORY SIZE EXCEEDS CACHE BOUNDARIES
- LATENCY IS GENERALLY HIGHER THAN WITH STRIDE 128 POINTER TRAVERSAL



FORWARD INDEXING WITH VARIABLE STRIDE

KEY OBSERVATIONS

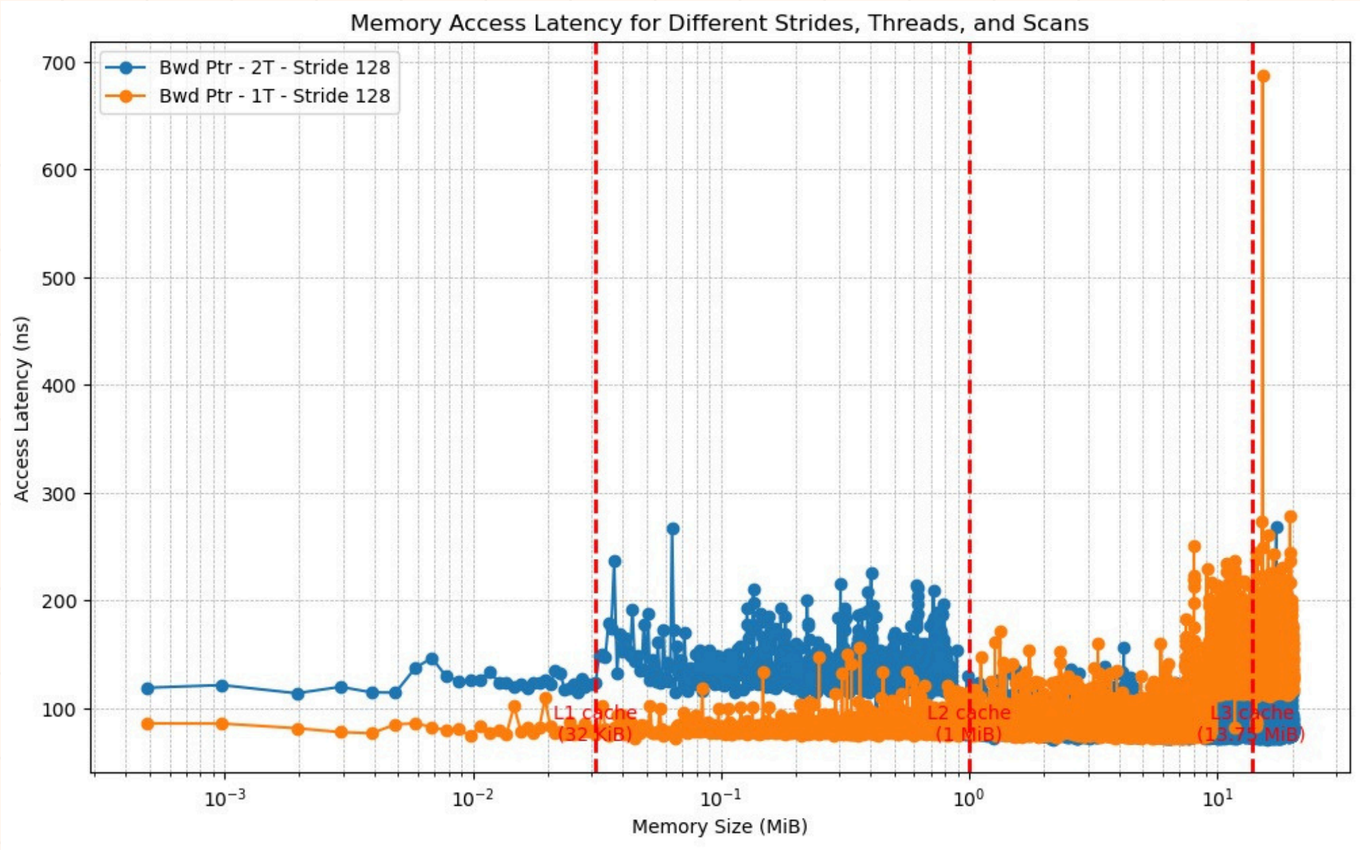
- LATENCY IS MEASURED IN 10^6 NS, SHOWING HIGHER OVERALL VALUES THAN WITH STRIDE 128
- TWO-THREADED EXECUTION (ORANGE) GENERALLY SHOWS LOWER LATENCY THAN SINGLE-THREADED (BLUE)
- EXTREME LATENCY SPIKE FOR 2T EXECUTION AROUND 3-4 MIB, REACHING OVER 3×10^6 NS
- SINGLE-THREADED EXECUTION SHOWS MORE FREQUENT BUT LESS EXTREME SPIKES
- THE PATTERN BECOMES MORE VOLATILE ONCE MEMORY SIZE EXCEEDS L2 CACHE



BACKWARD POINTER WITH STANDARD STRIDE

KEY OBSERVATIONS

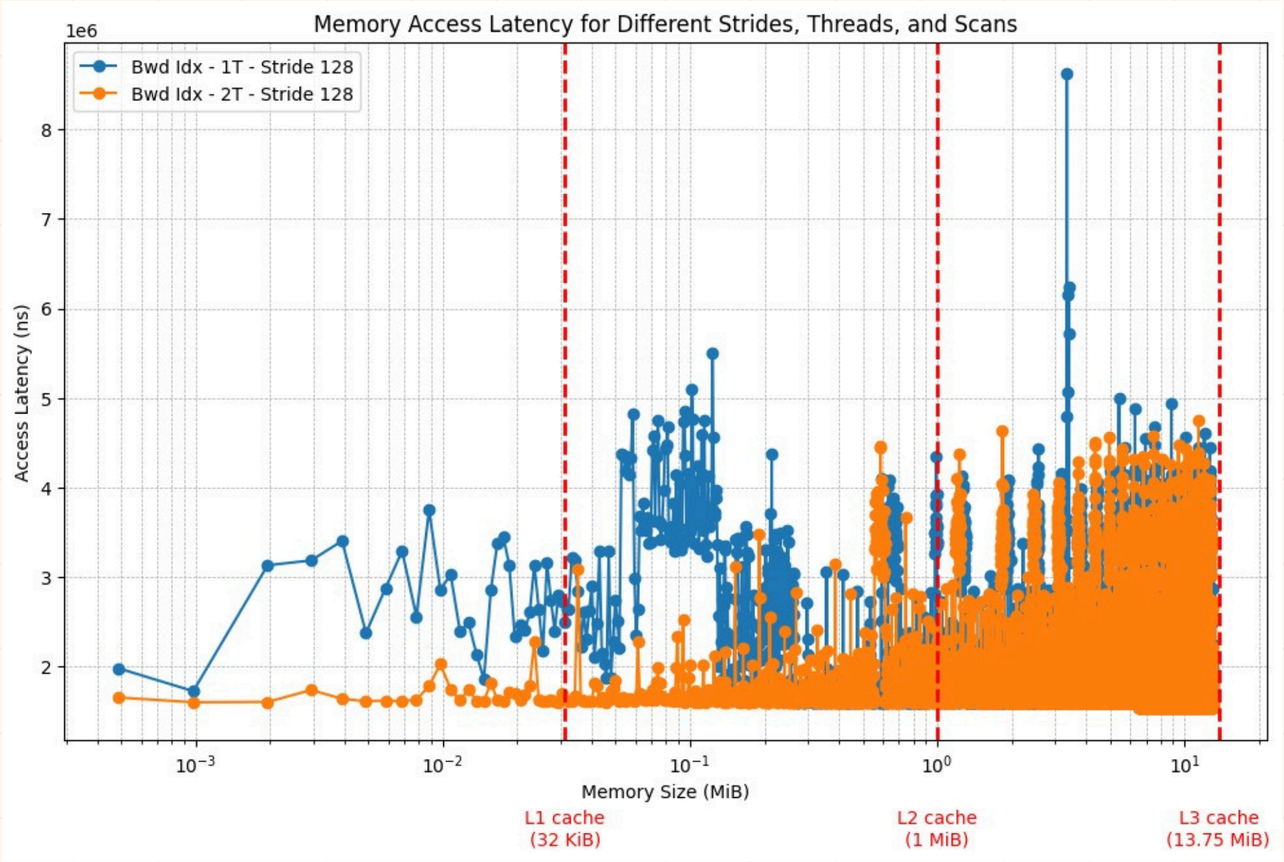
- SINGLE-THREADED EXECUTION (ORANGE) SHOWS CONSISTENTLY LOWER BASELINE LATENCY (~80 NS) THAN 2T (~120 NS) FOR SMALL MEMORY SIZES
- 2T EXECUTION (BLUE) SHOWS MORE FREQUENT AND HIGHER LATENCY SPIKES IN THE L1-L2 RANGE
- AN EXTREME SPIKE OCCURS FOR 1T EXECUTION AT THE LARGEST MEMORY SIZE, REACHING ~690 NS
- THE OVERALL LATENCY PATTERN IS LESS VOLATILE THAN WITH STRIDE 256
- PERFORMANCE DEGRADATION IS MORE GRADUAL ACROSS CACHE BOUNDARIES
- LATENCY IS GENERALLY LOWER THAN WITH STRIDE 256 UNTIL THE LARGEST MEMORY SIZES



BACKWARD INDEXING WITH STANDARD STRIDE

KEY OBSERVATIONS

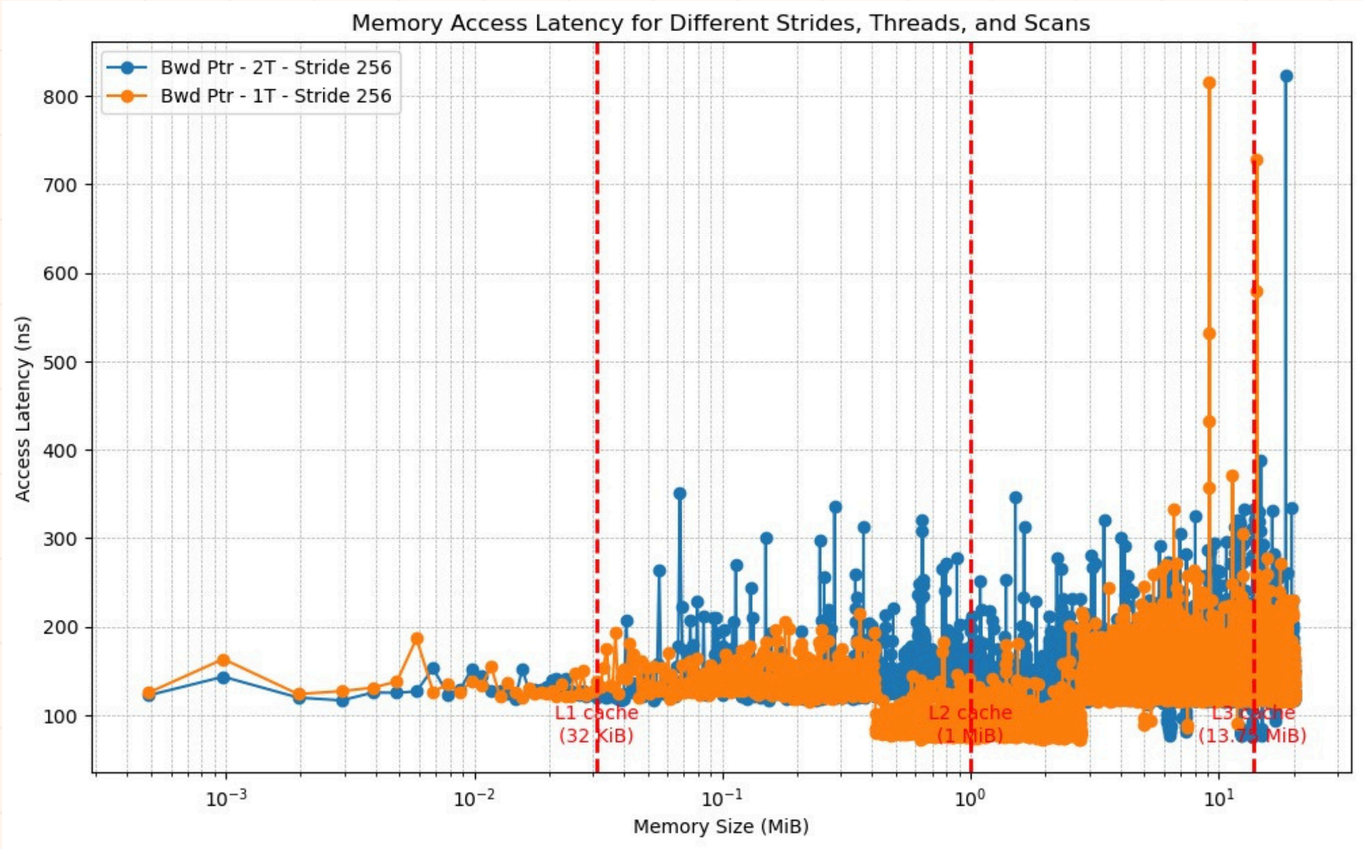
- SINGLE-THREADED EXECUTION SHOWS HIGHER BASELINE LATENCY AND MORE EXTREME SPIKES
- A MASSIVE SPIKE FOR 1T EXECUTION REACHES ~8.5×10^6 NS NEAR THE L2 CACHE BOUNDARY
- TWO-THREADED EXECUTION SHOWS MORE CONSISTENT PERFORMANCE WITH FEWER EXTREME SPIKES
- BOTH CONFIGURATIONS SHOW INCREASING LATENCY AS MEMORY SIZE GROWS
- THE OVERALL PATTERN IS MORE CHAOTIC THAN WITH STRIDE 256
- LATENCY IS MEASURED IN MILLIONS OF NANOSECONDS, ORDERS OF MAGNITUDE HIGHER THAN POINTER TRAVERSAL



BACKWARD POINTER WITH VARIABLE STRIDE

KEY OBSERVATIONS

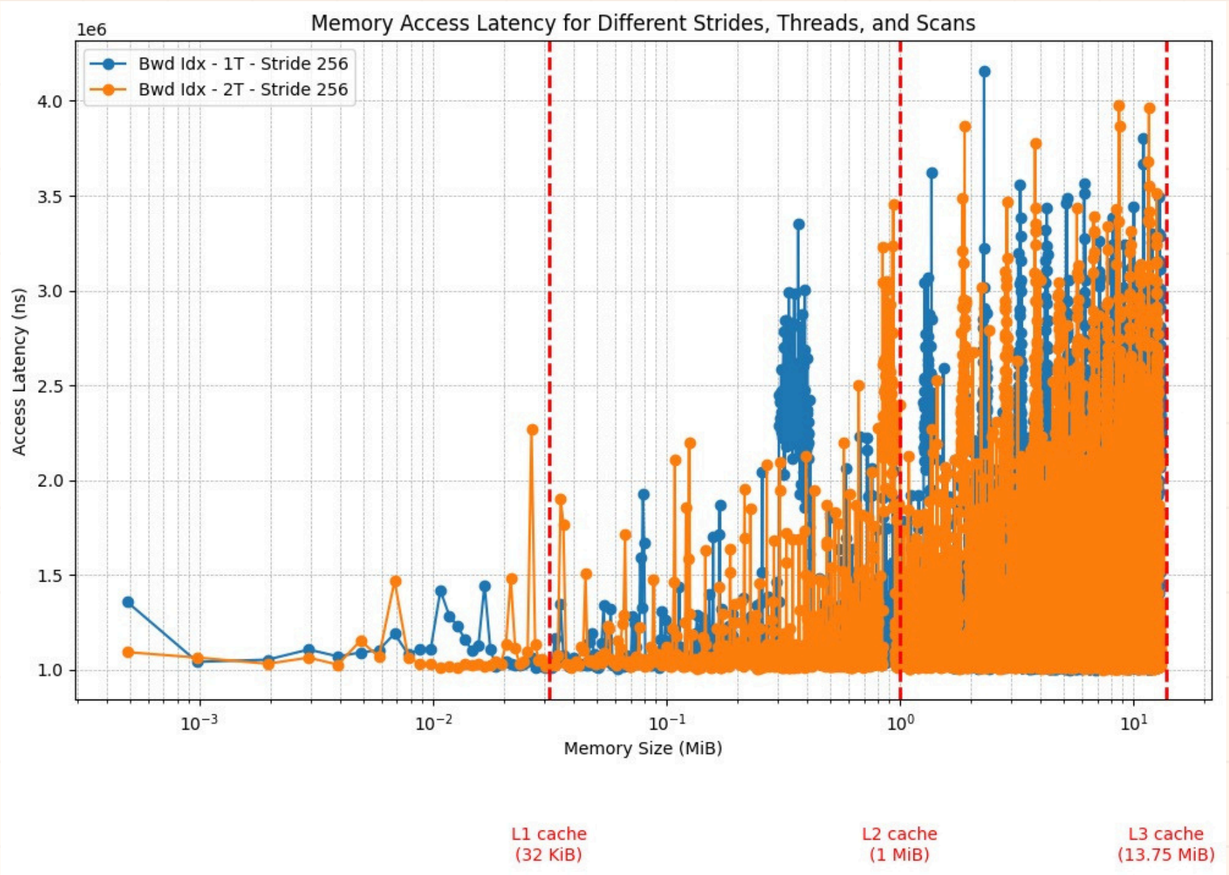
- BOTH THREAD CONFIGURATIONS SHOW SIMILAR BASELINE LATENCY (~120-150 NS) FOR SMALL MEMORY SIZES
- SIGNIFICANT LATENCY SPIKES OCCUR BEYOND L1 CACHE, WITH EXTREME SPIKES REACHING ~820 NS AT LARGE MEMORY SIZES
- 2T EXECUTION (BLUE) SHOWS MORE FREQUENT HIGH-LATENCY SPIKES IN THE L1-L2 RANGE
- BOTH CONFIGURATIONS EXPERIENCE DRAMATIC LATENCY INCREASES BEYOND L3 CACHE
- THE OVERALL PATTERN BECOMES MORE VOLATILE AS MEMORY SIZE INCREASES
- MAXIMUM LATENCY IS SIMILAR FOR BOTH THREAD COUNTS AT THE LARGEST MEMORY SIZES



BACKWARD INDEXING WITH VARIABLE STRIDE

KEY OBSERVATIONS

- BASELINE LATENCY STARTS AROUND 1×10^6 NS FOR BOTH THREAD CONFIGURATIONS
- LATENCY GRADUALLY INCREASES AS MEMORY SIZE GROWS, WITH DRAMATIC SPIKES BEYOND L1 CACHE
- SINGLE-THREADED EXECUTION SHOWS EXTREME SPIKES REACHING OVER 4×10^6 NS
- TWO-THREADED EXECUTION GENERALLY SHOWS MORE CONSISTENT BUT STILL ELEVATED LATENCY
- BOTH CONFIGURATIONS SHOW INCREASING VOLATILITY AS MEMORY SIZE EXCEEDS CACHE BOUNDARIES
- THE LATENCY SCALE (10^6 NS) IS DRAMATICALLY HIGHER THAN FOR POINTER TRAVERSAL (HUNDREDS OF NS)



GENERAL OBSERVATIONS ACROSS ALL GRAPHS:

- **ACCESS METHOD IMPACT:** BACKWARD INDEXING CONSISTENTLY SHOWS DRAMATICALLY HIGHER LATENCY (10^6 NS) THAN BACKWARD POINTER TRAVERSAL (HUNDREDS OF NS), SUGGESTING FUNDAMENTAL DIFFERENCES IN HOW THESE MEMORY ACCESS PATTERNS ARE HANDLED.

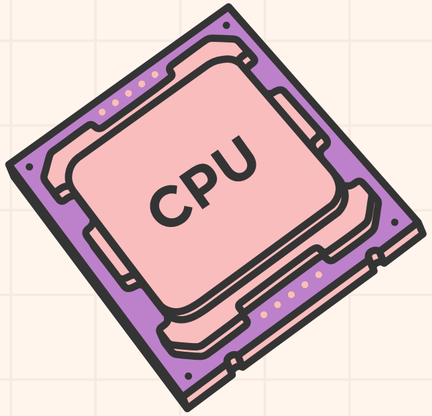
- **THREAD COUNT EFFECTS:** FOR POINTER TRAVERSAL, 2T EXECUTION GENERALLY SHOWS HIGHER LATENCY FOR SMALL MEMORY SIZES BUT SOMETIMES BETTER STABILITY FOR LARGE SIZES. FOR INDEXING, 2T EXECUTION OFTEN SHOWS MORE CONSISTENT PERFORMANCE WITH FEWER EXTREME SPIKES.

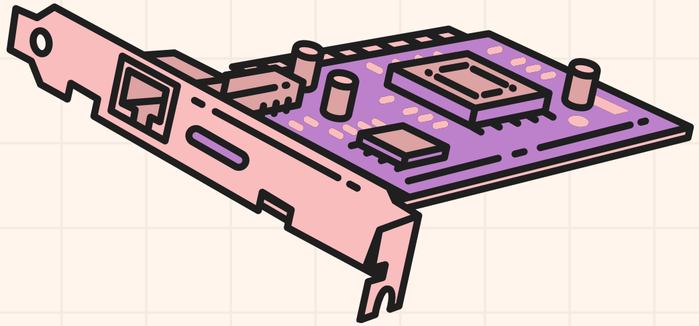
- **STRIDE SIZE INFLUENCE:** SMALLER STRIDE (128) GENERALLY RESULTS IN LOWER BASELINE LATENCY THAN LARGER STRIDE (256), PARTICULARLY FOR POINTER TRAVERSAL, DEMONSTRATING BETTER SPATIAL LOCALITY WITH SMALLER STRIDES.

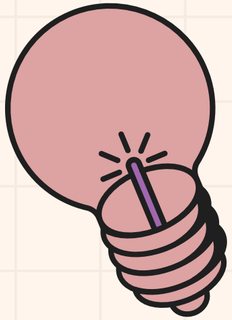
- **CACHE BOUNDARY IMPACT:** ALL GRAPHS SHOW CHANGING BEHAVIOR AT CACHE BOUNDARIES, WITH PERFORMANCE DEGRADING AS MEMORY ACCESSES MOVE BEYOND EACH CACHE LEVEL.

- **VOLATILITY PATTERNS:** LATENCY BECOMES INCREASINGLY VOLATILE AS MEMORY SIZE INCREASES, WITH THE MOST EXTREME SPIKES OCCURRING BEYOND L3 CACHE OR NEAR CACHE BOUNDARIES.

- **BACKWARD VS. FORWARD ACCESS:** COMPARED TO FORWARD ACCESS PATTERNS, BACKWARD TRAVERSAL SHOWS DIFFERENT LATENCY CHARACTERISTICS, SUGGESTING THE CPU'S PREFETCHER MAY BE LESS EFFECTIVE FOR BACKWARD MEMORY ACCESS.







THANK YOU