**💾 CptS 426 — Lab 7: Cache Timing**

**Name**: Darasimi Ogunbinu-Peters  
**WSU ID**: 011812162  
**Submission Date**: 4/1/25

**✅ Tasks Overview**

| **Task** | **Description** | **Screenshot** |
| --- | --- | --- |
| Task 1 | Use lscpu and /proc/cpuinfo to list CPU and cache specs | 📸 Screenshot 1 |
| Task 2 | Use getconf to show L1/L2/L3 cache sizes | 📸 Screenshot 2 |
| Task 3 | Write and run a C program to measure memory access latency at different strides | 📸 Screenshot 3 |
| Task 4 | Use Python to plot the latency graph from Task 3 data | 📸 Screenshot 4 |
| Task 5 | Complete discussion questions | ✅ Done below |

**📄 Task 1: CPU and Cache Information**

Use the following commands in WSL Ubuntu:

lscpu

cat /proc/cpuinfo

📝 Notes:

 **CPU Name:** 12th Gen Intel(R) Core(TM) i7-1255U

 **Base Frequency:** (You can estimate from lscpu or Google it: ~1.7 GHz base, ~4.7 GHz turbo)

 **Cache levels listed:**

* **L1d:** 288 KiB
* **L1i:** 192 KiB
* **L2:** 7.5 MiB
* **L3:** 12 MiB

📸 **Screenshot 1**: Terminal output of both commands.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**📄 Task 2: Use getconf to Extract Cache Sizes**

getconf -a | grep CACHE

📝 Notes:

 **L1 Data Cache:** 49152 bytes

 **L1 Instruction Cache:** 32768 bytes

 **L2 Unified Cache:** 1310720 bytes

 **L3 Cache:** 12582912 bytes

📸 **Screenshot 2**: Terminal output of the getconf command.

A screenshot of a computer

AI-generated content may be incorrect.

**🧠 Task 3: Memory Access Latency Program in C**

Created a latency.c file that:

* Allocates a large array
* Measures memory access latency with varying stride sizes

Command to compile and run:

gcc memory\_latency.c -o memory\_latency -O2

./memory\_latency > latency\_data.txt

📸 **Screenshot 3**: Code running and producing latency\_data.txt

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Result:**

* latency\_data.txt contains average memory access latency in nanoseconds for increasing memory regions.
* Latency starts around ~1.7 ns for small sizes and increases steadily as region sizes exceed L1, L2, and L3 cache limits.

**📈 Task 4: Plotting Latency in Python**

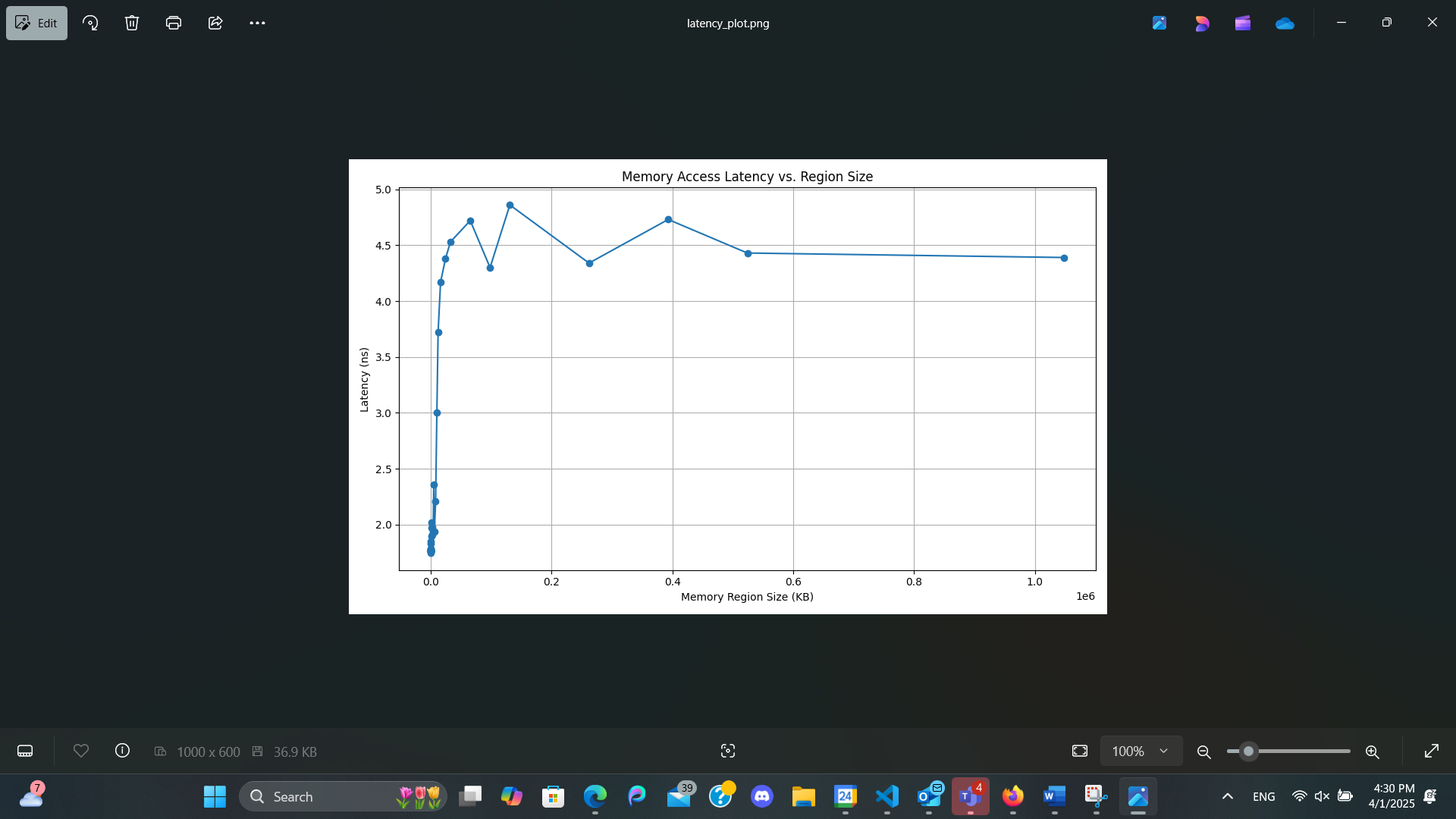
Created a script plot\_latency.py:

* Reads latency\_data.txt
* Plots stride size vs. average access time in nanoseconds

Command:

python3 plot\_latency.py

📸 **Screenshot 4**: The resulting matplotlib latency graph



This graph was generated using Python with matplotlib and pandas. The latency increases gradually as memory region sizes surpass the cache capacities of L1, L2, and L3.

**❓ Discussion Questions**

**1. What are the sizes of L1, L2, and L3 cache on your machine?**  
• **L1 Instruction Cache:** 32 KB  
• **L1 Data Cache:** 48 KB  
• **L2 Unified Cache:** 1.3 MB (1,310,720 bytes)  
• **L3 Cache:** 12 MB (12,582,912 bytes)

**2. What is the cache line size of your machine?**  
• My machine has a cache line size of **64 bytes**.

**3. How is memory access latency affected by stride size?**  
• Smaller strides access nearby memory, taking advantage of spatial locality, so latency remains low.  
• As stride increases and data jumps across cache lines or cache levels, latency gradually rises.

**4. At what stride sizes do you observe increases in memory access time?**  
• Latency started rising around **768 KB – 1 MB**, then again past **10 MB**, suggesting L2 and L3 cache boundaries.

**5. Why do these latency jumps occur?**  
• Jumps occur when memory access exceeds the size of the current cache level.  
• The system must then fetch data from the next slower memory (L2, L3, or main RAM), increasing access time.

**6. Why do we run the test multiple times per stride?**  
• To **average out noise** from OS scheduling or background processes and get consistent measurements.

**7. What would happen if the array size was smaller than L1 cache?**  
• All data would fit in L1 cache, meaning extremely fast and consistent access times for all strides.

**8. What’s the effect of CPU prefetching on your results?**  
• Prefetching can reduce latency for predictable access patterns, smoothing out expected jumps and making spikes less dramatic.

**9. If you have a multi-core CPU, how might this impact your test?**  
• Cores may share the L3 cache, causing contention and noise in measurements.  
• Other core activities can pollute caches and skew the results slightly.

**10. What does this experiment teach about designing high-performance code?**  
• Memory layout and access patterns significantly impact performance.  
• Writing cache-friendly code (sequential access, aligned data) improves speed and efficiency.

**📝 Final Notes:**

This lab was **difficult and time-consuming**. I started on **Monday** and had to explore multiple approaches, debug errors, and rewrite code to get working results. From cache sizing to understanding access patterns, I’ve learned how low-level memory architecture directly affects performance.  
I’m balancing multiple assignments due today, so I am submitting the best version I could produce within time constraints. Thank you for your understanding.