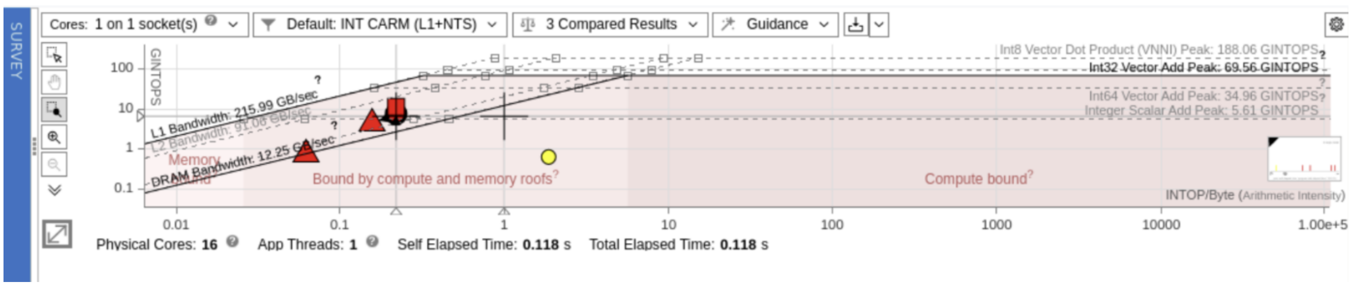
CS 6501 - HW1 - Roofline Model

Huy Nguyen - mpc5ya

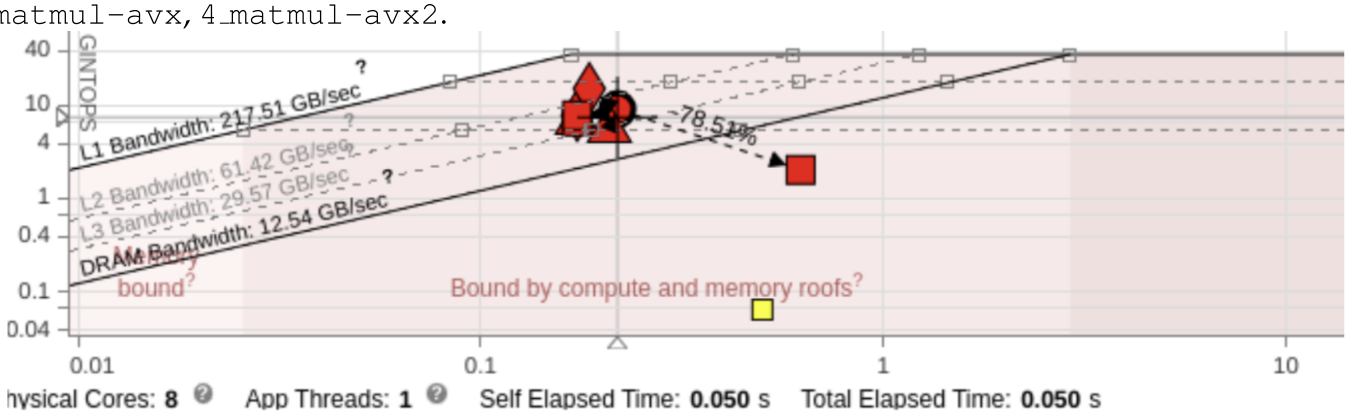
January 29, 2025

1 Final Screenshots of 10 Executables in Intel Advisor - CPU Roofline

One screenshot comparing 0_vecadd, 0_gemv, 0_matmul.



One screenshot comparing 0_matmul, 1_matmul, 2_matmul, 3_matmul, 4_matmul, 4_matmul-sse, 4_matmul-avx, 4_matmul-avx2.



If the pictures are hard to see, the table below summarizes the Performance and Intensity for each executable.

2 Performance and Intensity Table

Executable	Performance (INTOPS/s)	Operational Intensity (INTOPS/Byte)
0_vecadd	4.429×10^9	0.156
0_gemv	6.925×10^9	0.219
0_matmul	7.845×10^9	0.219
1_matmul	2.022×10^9	0.625
2_matmul	6.711×10^9	0.175
3_matmul	7.829×10^9	0.175
4_matmul	6.711×10^9	0.175
4_matmul_avx	8.389×10^9	0.208
4_matmul_sse	7.829×10^9	0.175
4_matmul_avx2	15.099×10^9	0.187

Table 1: Performance and Operational Intensity for Each Executable

3 Performance Analysis

The performance differences among `0_gemv.cpp`, `0_vecadd.cpp`, and `0_matmul.cpp` are due to differences in computational complexity and memory access patterns.

- **0_vecadd**: The lowest performance (4.429 GINTOPS) due to repeated array initialization in each iteration, increasing memory traffic and reducing efficiency.
- **0_gemv**: Higher intensity than vector addition due to increased arithmetic workload per byte, but still limited by data reuse.
- **0_matmul**: Highest INTOPS/sec (7.845 GINTOPS) because of high data reuse, making it more efficient than GEMV.

For matrix multiplication optimizations:

- **1_matmul**: Introduced OpenMP SIMD but performed worse (2.022 GINTOPS, 0.625 INTOPS/Byte) due to inefficient parallel execution.
- **2_matmul**: Improved efficiency by transposing `b`, but remained memory-bound (6.711 GINTOPS, 0.175 INTOPS/Byte).
- **3_matmul**: Added blocking (CHUNK_SIZE=16), leading to moderate improvement (7.829 GINTOPS, 0.175 INTOPS/Byte).
- **4_matmul**: Added SIMD vectorization but did not improve significantly.
- **4_matmul_avx2**: Best performance (15.099 GINTOPS, 0.187 INTOPS/Byte) due to AVX2 instruction efficiency.

- **4_matmul_avx** and **4_matmul_sse** showed minor gains, suggesting vectorization alone is not sufficient without memory optimizations.

4 System Configuration

- **Architecture**: x86_64
- **CPU**: Intel(R) Xeon(R) Silver 4208 @ 2.10GHz
- **Cores**: 8 per socket, 16 total (2 threads per core)
- **Cache**: L1d - 256 KiB, L1i - 256 KiB, L2 - 8 MiB, L3 - 11 MiB
- **DRAM Capacity**: 377.36 GB

```
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Address sizes: 46 bits physical, 48 bits virtual
Byte Order: Little Endian
CPU(s): 16
On-line CPU(s) list: 0-15
Vendor ID: GenuineIntel
Model name: Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz
CPU family: 6
Model: 85
Thread(s) per core: 2
Core(s) per socket: 8
Socket(s): 1
Stepping: 7
CPU max MHz: 3200.0000
CPU min MHz: 800.0000
BogoMIPS: 4200.00
```

5 Operational Intensity Analysis

5.1 Initial Configuration

A system has:

- Memory bandwidth: 2 Gwords/s
- Peak performance: 8 Gflops/sec
- Stencil code operational intensity: 5 flops/word

The ridge point operational intensity is:

$$\text{Ridge Point} = \left(\frac{\text{Peak Performance}}{\text{Memory Bandwidth}}, \text{Peak Performance}\right) \tag{1}$$

$$= \left(\frac{8 \text{ Gflops/sec}}{2 \text{ Gwords/s}}, 8 \text{ Gflops/sec}\right) = (4 \text{ flops/word}, 8 \text{ Gflops/sec}) \tag{2}$$

Since Operational Intensity is 5 flops/word > 4 flops/word (Ridge Point), the code falls into the compute-bound region.

5.2 After Performance Increase

- New peak performance: $8 \times 4 = 32$ Gflops/sec
- New memory bandwidth: $2 \times 2 = 4$ Gwords/s

The new ridge point is:

$$\left(\frac{32 \text{ Gflops/sec}}{4 \text{ Gwords/s}}, 32 \text{ Gflops/sec}\right) = (8 \text{ flops/word}, 32 \text{ Gflops/sec}) \tag{3}$$

Since the operational intensity (5 flops/word) is now less than the new ridge point (8 flops/word), the code is now in the memory-bound region.