CS5312 High Performance Computer Architecture

Take Home Lab 2

# Step 4

## Task 5.1

Please note that these times are in milliseconds.

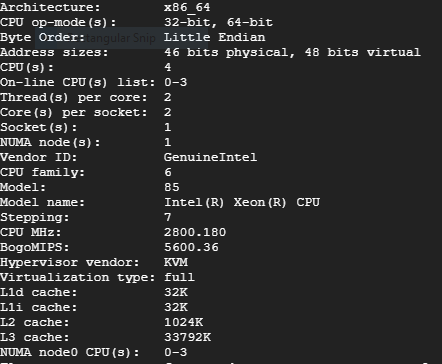
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Dimensions** | | **Execution Time** | | | | | | |
| **Listing 5** | **Listing 6** | | **Speed Up** | | **Listing 6 SSE** | **Speed Up** |
| 100\*100 | 100\*1 | 0.0302 | 0.0187 | 1.61 | | 0.0169 | | 1.79 |
| 200\*200 | 200\*1 | 0.1176 | 0.0744 | 1.58 | | 0.0624 | | 1.88 |
| 400\*400 | 400\*1 | 0.4665 | 0.2945 | 1.58 | | 0.2549 | | 1.83 |
| 800\*800 | 800\*1 | 1.8452 | 1.1682 | 1.58 | | 1.0421 | | 1.77 |
| 1600\*1600 | 1600\*1 | 7.3952 | 4.718 | 1.57 | | 4.1406 | | 1.79 |

## Task 6.1 and 7.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Dimensions** | | **Execution Time** | | | | |
| **Listing 7** | **Listing 7 SSE** | **Speed Up** | **Listing 7 Auto Vectorization** | **Speed Up** |
| 100\*100 | 100\*100 | 3.9738 | 2.4122 | 1.65 | 0.95 | 4.18 |
| 200\*200 | 200\*200 | 31.4334 | 19.1806 | 1.64 | 8.497 | 3.70 |
| 400\*400 | 400\*400 | 286.8759 | 171.6214 | 1.67 | 72.952 | 3.93 |
| 800\*800 | 800\*800 | 2596.103 | 1747.9119 | 1.49 | 649.634 | 4.00 |
| 1600\*1600 | 1600\*1600 | 23502.365 | 19013.4422 | 1.24 | 6057.310 | 3.88 |

# Step 5

## Specifications of the machine used



## Matrix-Vector multiplication

In Matrix-Vector multiplication first we have taken time using the naïve method. Then we tested the same method with loop unrolling with unrolling 4 iterations. Later we tested with 128 bit vectors which accommodates 4 elements at once. Which is parallel to the manual loop unrolling we have done.

As we can see from the data we acquired we have achieved considerable amount of speedup (nearly 1.6 times) by loop unrolling. This is mainly due to the fact that unrolling loops reduce the overhead occurs when running each iteration. By unrolling the loop 4 times we have reduced a significant amount of overhead per iteration. Hence resulting in the speedup.

In the next test, SSE is used for the same matrix vector multiplication. We have compiled the files to match the native SSE/AVX of the machine. 128 bit SSE variables were used which can hold 4, 32bit floats.

As we can see from the data this has given a significant boost to the speed up compared to the naïve version (nearly 1.8 speedup). Also when compared to the manual loop unrolled version this has provided nearly 1.12X speedup thanks to the streaming operations done through the special hardware.

I have provided some charts below. This problem corresponds to O(n2) problem. Therefore better comparison can be done using the approximate number of multiplications happen.

## Matrix-Matrix multiplication

This graph shows the time taken for each test. As we can see SSE version performs significantly better compared to the naïve version (nearly 1.5 times better). But as you can see the speedup drops with the size of the matrix. This can be attributed to limitations of the memory system. As the number of elements increase, the memory becomes saturated and cache misses increase. Which results in higher delays.

Auto vectorized method is much better compared to the SSE version. It provided nearly 4X improvement over the naïve method and nearly 2.6X improvement over the SSE version. In the compile time we could observe the message that

**matmat\_auto.c:74:13: note: loop vectorized**

**matmat\_auto.c:74:13: note: loop with 2 iterations completely unrolled (header execution count 72506056)**

Which indicates that a main loop is completely unrolled with vectorization. Which explains the huge gain in speedup.

This problem is O(n3). Hence x axis of the graph provided corresponds to the cube of the dimension of the matrix.