Jose Rodriguez 9-15-2021

Matrix Multiply - Part 2

CS 4175

During this assignment, I extended the previous first part of the matrix multiply assignment by adding a parallel function that was able to multiply two matrices across different threads. This was accomplished through the use of the python pymp module which is a wrapper surrounding OpenMP.

The way in which I split off the work as I was parallelizing is by splitting it by the rows of the output. This way accomplished in the following way:

```
def matrixMultiplyParallelRow(matrix_a, matrix_b):
    assert len(matrix_a[0]) == len(matrix_b), 'Both matrices can be
multiplied'

    out = pymp.shared.array((len(matrix_a), len(matrix_b[0])),
dtype="uint32")

    with pymp.Parallel() as p:
        print(f'Number of thread: {p.thread_num} of {p.num_threads}')

        for row in p.range(0, len(out)):
            for col in range(len(out[0])):
                out[row][col] = sum(matrix_a[row][i] * matrix_b[i][col]
                                    for i in range(len(matrix_b)))

    return list(list(l) for l in out)
```

Difficulties Encountered

I encountered several issues as I was writing the assignment. For instance, I first used the pymp.shared.list instead of the pymp.shared.array. This led to a dramatic increase in the time that was used. Specifically, using the threaded version was now up to 20 times slower which is ridiculously slow. I think this makes sense since nowadays, memory tends to be a lot slower than processing speed relatively, so if the memory layout is slower, then it outweighs the benefits attained from parallelism.

Furthermore, another difficulty that I encountered is the memory type representation I was using. At first I used uint8. This worked fine for the smaller instances, but once I did 500 a

matrix of ones of size 100, I started getting 144 which I found odd. My immediate suspicion was that I might have introduced a race condition in my code, but upon further review, each cell would only be modified by only one thread due to the row constraint which I found odd. Then, when I decreased it to 260, I found that the answer was now a 260x260 matrix with only 4. Since 260-4 is 256 and since I was using uint8, I realized that the values were being added properly, only that they were overflowing due to the memory limitation that I was using.

Problems I was not able to overcome

I was able to overcome the problems I faced, and I think that I have fulfilled all of the requirements mentioned inside of the matrix multiply assignment's instructions.

Assignment duration

It took me around 3 hours to finish the assignment + around an hour to collect the request test data and write the report.

Performance Measurements

This section contains evidence of the runs at different numbers of threads.

One thread:

```
A x B (cropped):
Time taken: 8.825389699999505 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
A x B through blocked matrix multiply (cropped):
Time taken: 10.96976689999974 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
Number of thread: 0 of 1
A x B through row parallel matrix multiply (cropped):
Time taken: 8.86720349999996 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
enoumy@Athena:~/documents/matrix-multiply-Enoumy|master
```

Two threads:

```
A x B (cropped):
Time taken: 8.861879599999156 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
A x B through blocked matrix multiply (cropped):
Time taken: 11.024184500000047 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
Number of thread: 0 of 2
Number of thread: 1 of 2
A x B through row parallel matrix multiply (cropped):
Time taken: 4.524729700000535 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
```

Four threads:

```
A x B (cropped):
Time taken: 8.855907499999375 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
A x B through blocked matrix multiply (cropped):
Time taken: 10.918456499999593 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
Number of thread: 1 of 4
Number of thread: 2 of 4
Number of thread: 0 of 4
Number of thread: 3 of 4
A x B through row parallel matrix multiply (cropped):
Time taken: 2.346091499999602 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
```

Eight Threads:

```
A x B (cropped):
Time taken: 8.776834000000235 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
A x B through blocked matrix multiply (cropped):
Time taken: 10.857363999999507 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
Number of thread: 1 of 8
Number of thread: 2 of 8
Number of thread: 3 of 8
Number of thread: 4 of 8
Number of thread: 5 of 8
Number of thread: 6 of 8
Number of thread: 0 of 8
Number of thread: 7 of 8
A x B through row parallel matrix multiply (cropped):
Time taken: 1.3799946999988606 seconds
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
```

Furthermore, the results are tabulated into this table:

Test	Time taken	Improvement
No parallelism	8.77 seconds	
Single thread	8.86 seconds	0.98
Two threads	4.52 seconds	1.96
Four threads	2.34 seconds	1.93
Eight threads	1.37 seconds	1.7

Analysis

When the thread count moves from no parallelism to a single thread, it is roughly doing the same thing without any parallelism. However, this is still slower due to the overhead needed in order to run pymp.

Furthermore, upon every duplication of threads, the program gets faster. At first it gets around twice as fast, but as more threads are being duplicated, the multiplier starts decreasing from the initial 1.96. This makes sense since there is additional overhead of maintaining more and more threads that need to coordinate. Furthermore a thing that might be a blocker is that perhaps in the way that pymp is implemented, only one thread can write to a shared pool at once, which might result in increased wait times for all threads.

Observations and comments

I liked the assignment! It was really satisfying seeing the speed increase as I started using more threads in the program.

Output of cpuinfo

This section contains the output of running the CPU info string. The machine I ran it on had 8 cores and 16 threads.