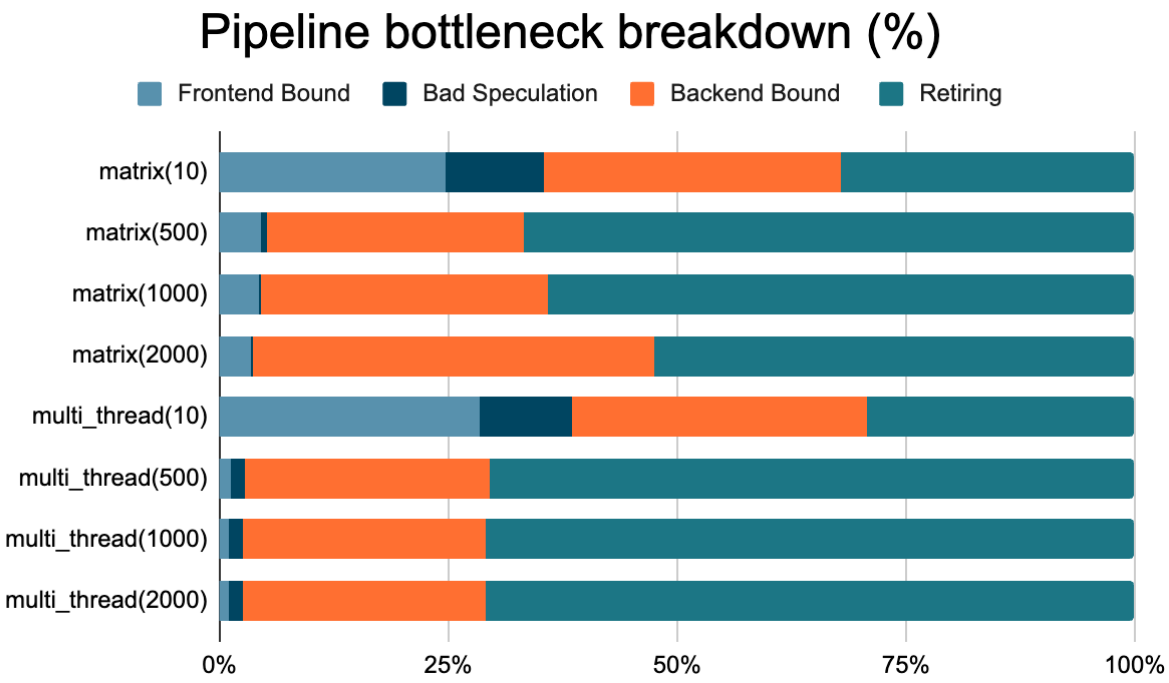
Report - Lab4

Siyu Meng sm2659



Multi_thread (8) by splitting rows to do matrix multiplication. For matrix multiplication in each thread, I used best performance blocking size and tiling in lab2 to speed up the runtime and optimize the pipeline breakdown.

Profiling process:

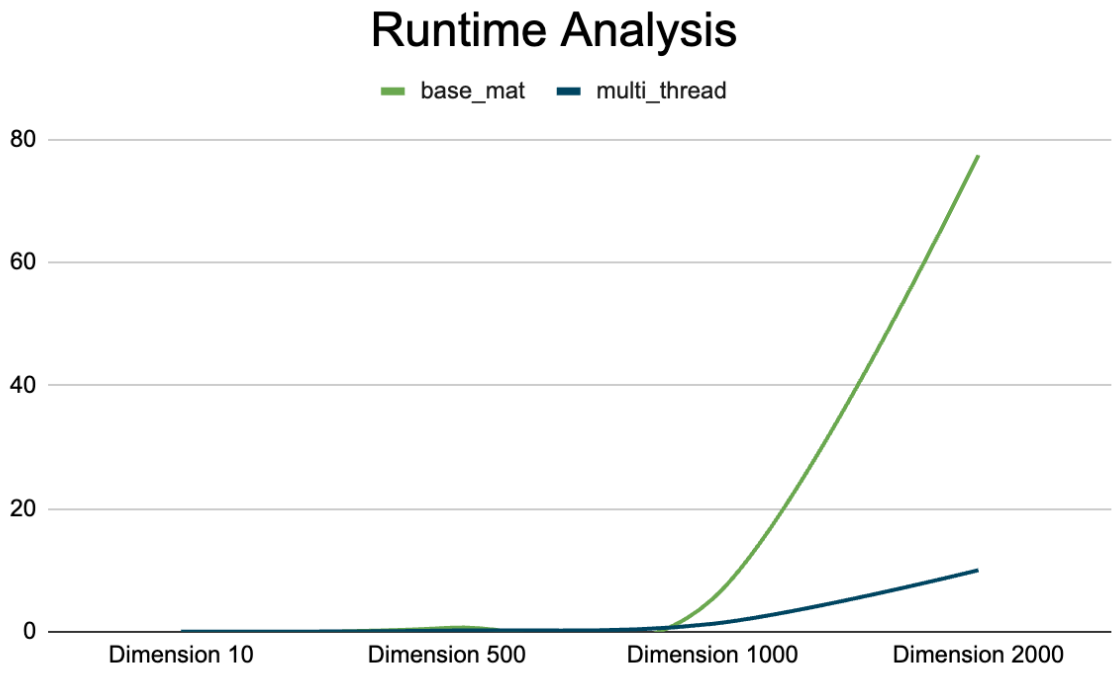
- Follow the instructions on piazza to run slurm job with request for 8 cores
- Profiling all_tests file with all 0-7 cores
- Using time utility to check if the real time is faster than user time to see if the multi threading does actually work. Also use the real time to draw the runtime analysis graph compared with naive matrix multiplication runtime.

Top-down analysis:

- For smaller input size, the pipeline breakdown does not have a big difference between naive matmul and mutithread matmul. Because the overhead in the multithread will over performance than its advantage for small input.
- For larger input sizes, the pipeline breakdown is stable.
    - Multi-threaded applications often exhibit better performance scalability with larger datasets. As the dataset size increases, the application can effectively utilize multiple threads, leading to a more balanced workload distribution and improved parallel scalability.
    - Large datasets may result in more regular and predictable memory access patterns, reducing contention for memory resources and optimizing cache

utilization. Improved memory access patterns can contribute to stability in the backend bound.

- With larger datasets, there may be increased opportunities for instruction-level parallelism. The frontend and backend metrics can benefit from a higher degree of parallel execution, leading to stable performance characteristics
- The retiring metrics stability can be attributed to efficient execution and completion of instructions in a well-optimized multithreaded matrix multiplication algorithm (blocking/tiling in lab2). As the dataset size increases, the retirement of instructions becomes more predictable and stable.



Runtime analysis:

- For small input sizes, the runtime does not have a big difference.
- As the input sizes become larger, the difference also increases significantly.
- Because we run the multithread matmul on 8 cores, the runtime of dimension 2000 for multithread is around 8 times faster than naive matmul.

Discussion:

- After finishing writing the multithread code and testing its correctness, I tried to measure the runtime. However, when the input size increases, the runtime of multithread is even worse than the naive one. I looked through the taskset and toplev.py documentation and tried to find why it is slow and how to profile the correct cores. After stucking for a long time, I went to office hours and found that I do not run the program on multi cores. That's why the runtime does not have any improvement.
- After solving the problem of running on multiple cores, I thought about combining the method in lab 2 about blocking matrix multiplication to well optimize the matmul. After change the matrix multiplication in each thread to blocking matrix multiplication, the

pipeline breakdown become more stable and the runtime for dimension 2000 improve around 2 seconds.