ECE6960 Heterogeneous Programming Programming Assignment 1 Report

Cheng-Hsiang Chiu u1305418
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT-84112
{cheng-hsiang.chiu}@utah.edu

I. Introduction

In this assignment I implemented a thread pool to do matrix multiplications in parallel.

II. Implementations

There are several implementations in this assignment. They are:

1) Sequential solution. This is considered as the baseline. It is denoted as `Seq` in the experimental results.
2) Parallel with false sharing solution. This solution parallelizes the inner most loop of the matrix multiplication. The issue of this solution is false sharing, which arises from the fact that multiple threads trying to access the same cache line. It is denoted as `P_FS` in the experimental results.
3) Parallel without false sharing solution. This solution parallelizes the middle loop of the matrix multiplication and does not encounter the false sharing issue. It is denoted as `P_NFS` in the experimental results.
4) Parallel with block matrix size solution. This solution creates less numbers of tasks than the solution 3 using block matrix size. The block matrix size is set to 16. It is denoted as `P_B` in the experimental results.
5) Parallel with decentralized queues solution. This solution uses decentralized queues and does not use block matrix size. Each worker thread has its own local queue. The main thread inserts tasks to queues in a round-robin manner. It is denoted as `P_D` in the experimental results.
6) Parallel with decentralized queues and block matrix size solution. This solution uses decentralized queues and uses block matrix size. Each worker thread has its own local queue. The main thread inserts tasks to queues in a round-robin manner. The block matrix size is set to 16. It is denoted as `P_DB` in the experimental results.

III. Availability

The whole implementation is available in the Github repository, link.

IV. Build and Run Executable

There is a README in the uploaded compressed file and in the Github repository mentioned above. README provides very detailed information. In this section, I highlight the instructions to build and run the executable `main`. Please make sure CMake and g++ is installed.

To build the executable `main`, use the following commands,

```
mkdir build
cd build
cmake ../
make
```

To run the executable, use the following command in folder `build`,

```
./main
```

V. Results

The executable was built using g++ with -std=c++17 and -O3 on a Linux machine with Intel i7-9700K 8 Cores at 3.6 GHz and 32 GB RA. All results were obtained using Google benchmark [1].

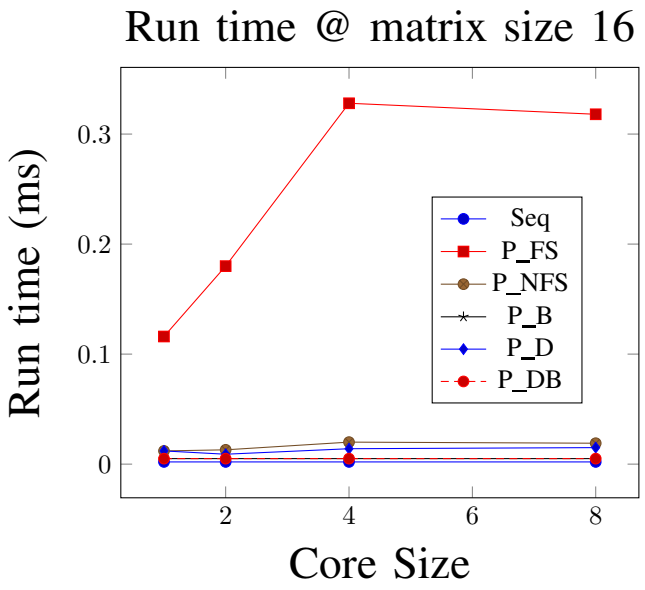

Fig. 1: Run time comparisons between all implementations of matrix size 16 at different core sizes.

We can see that `P_FS` is worse than `Seq` when matrix size is small (less than 1024) and is better at larger matrix size. At matrix size less than 256, `P_NFS`, `PB`, `PD`, and `PDB` have identical runtime performance. At 512 and 1024 matrix size, `P_NFS` and `PD` perform comparably; `PB` and `P_DB` perform

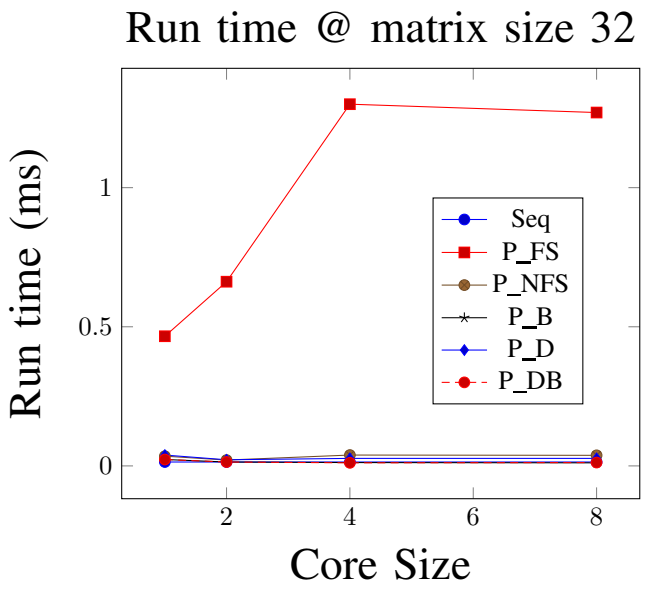

Fig. 2: Run time comparisons between all implementations of matrix size 32 at different core sizes.

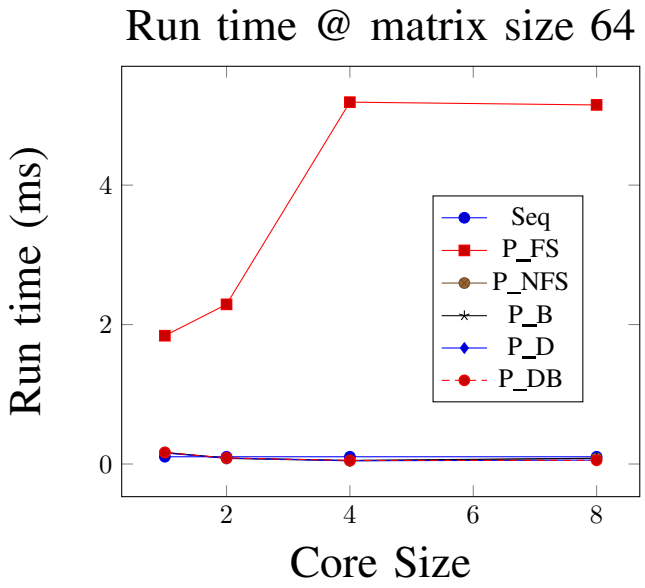

Fig. 3: Run time comparisons between all implementations of matrix size 64 at different core sizes.

comparably. Among all of the implementations, `P_DB` has the best performance regardless of the matrix sizes and core sizes.

References

[1] "Google benchmark," GitHub. [Online]. Available: https://github.com/google/benchmark

[2] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Vlsi module placement based on rectangle-packing by the sequence-pair," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1996.

[3] Z. L. Zhipeng Huang and J. Chen, "An improved simulated annealing algorithm with excessive length penalty for fixed-outline floorplanning," *IEEE Access*, 2020.

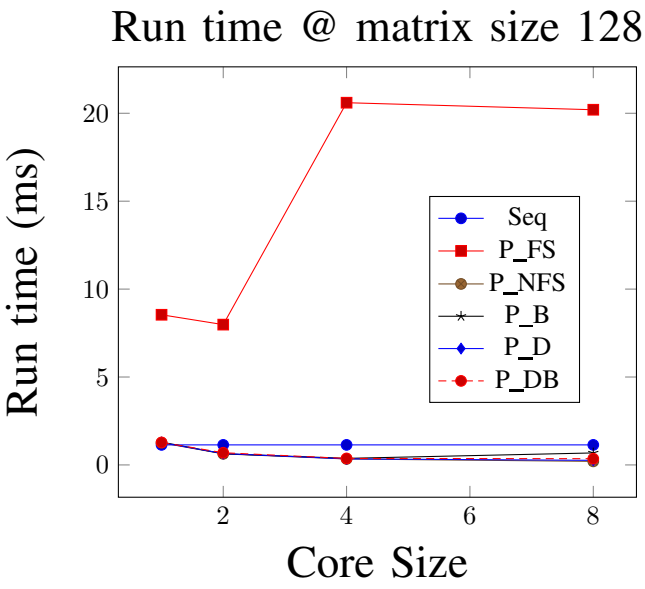

Fig. 4: Run time comparisons between all implementations of matrix size 128 at different core sizes.

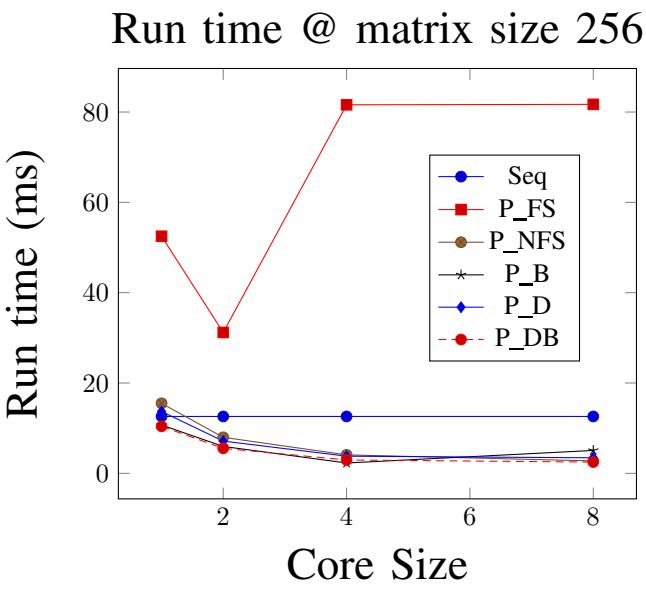

Fig. 5: Run time comparisons between all implementations of matrix size 256 at different core sizes.

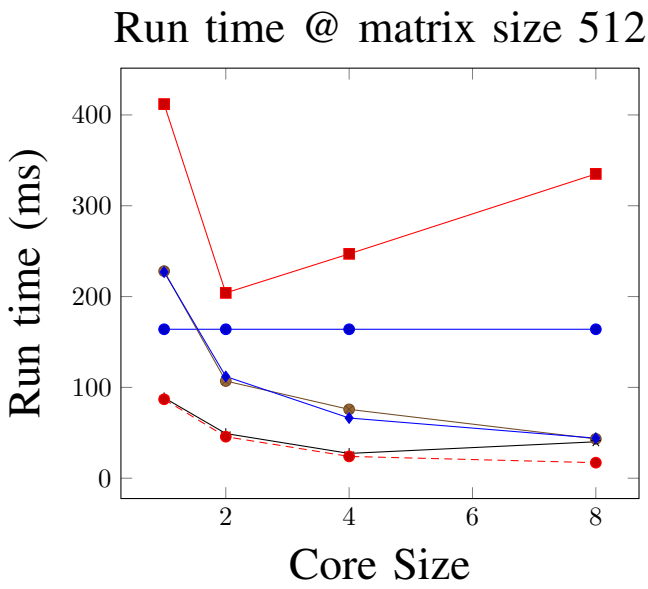

Fig. 6: Run time comparisons between all implementations of matrix size 512 at different core sizes. The legend is missing here. Please refer to the other figures for legend informations.

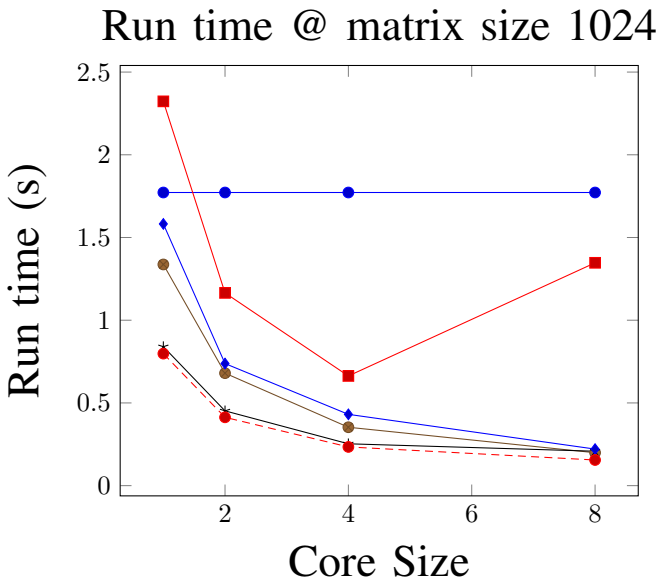

Fig. 7: Run time comparisons between all implementations of matrix size 1024 at different core sizes. The legend is missing here. Please refer to the other figures for legend informations.

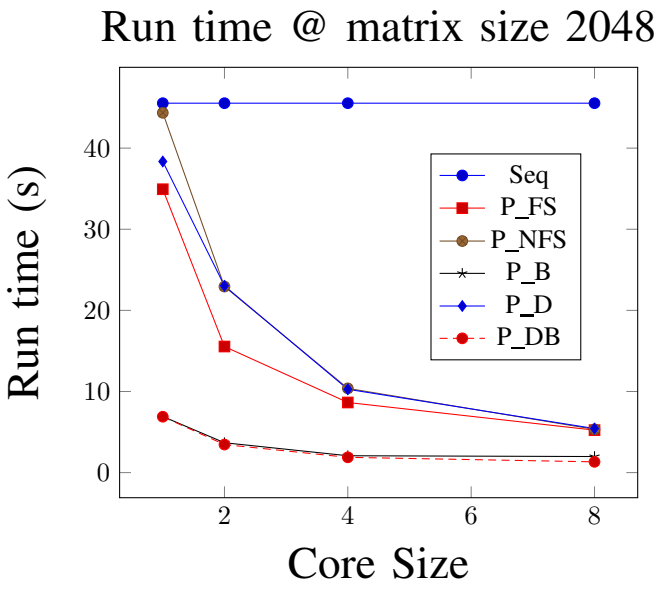

Fig. 8: Run time comparisons between all implementations of matrix size 2048 at different core sizes.