Performance Benefits of Multi-threading in Multi-program Environment

Agenda:

- Story
- Abstract
- Analyzing experimental results achieved so far
- Questions and counter-arguments
- Roadmap: next experiments, writing

Story:

Multi-program environment -> Progress imbalance of the instances

S.D. of runtime goes up as #instances goes up

The reason of the progress imbalance: imbalance in cache-sharing

Instances with a transient stall in the middle
-> lose share of the cache -> never gains it back
-> ends up having more S.D. of runtime

More parallelism breeds advantage in multi-program environment
Two programs run sequentially and concurrently: one has single-threaded, another is multi-threaded -> ratio of performance goes up

Abstract

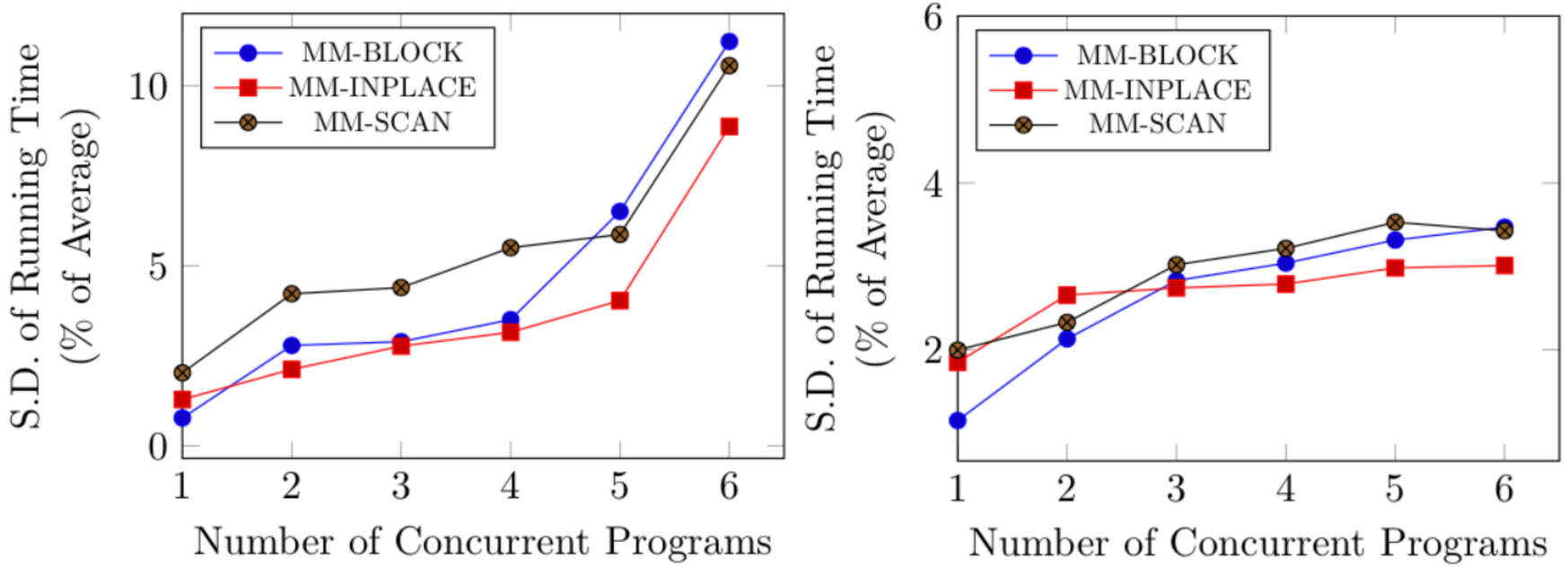
In most modern systems, programs run concurrently and share a memory. Hence, understanding the behavior of a cache shared among multiple processes is crucial for application designers. Dice et al.\ have observed that several concurrent homogeneous threads, threads running copies of the same program, may suffer from an imbalance in cache-residency when they share a memory.

In this work, we first present a counter-intuitive phenomenon. We run several concurrent homogeneous instances, multiple copies of the same program, and each copy accesses private data of the same size and shares a memory. We observe a progress imbalance of the program instances; the program instances finish at different times. We compare the difference in running time of the program instances, and we observe that the relative standard deviation of the running time increases with the number of program instances. The more concurrent programs we run, the more progress imbalance among the programs we observe.

Secondly, we perceive experimentally that the potential reason for the progress imbalance is stemming from an imbalance in cache-sharing among the program instances. In a heterogeneous environment, the program instances with a uniform transient stall loose their share of the cache and fail to gain that share back throughout the execution, leading to a much higher running time.

Lastly, we empirically show that more parallelism in a program provides performance benefits in a multi-program environment. Program instances with multithreading require less running time and incur less I/O when run concurrently with program instances without parallel threads when these instances share a memory. This performance advantage increases with problem size.

There is a progress imbalance of the program instances in a multi-program environment

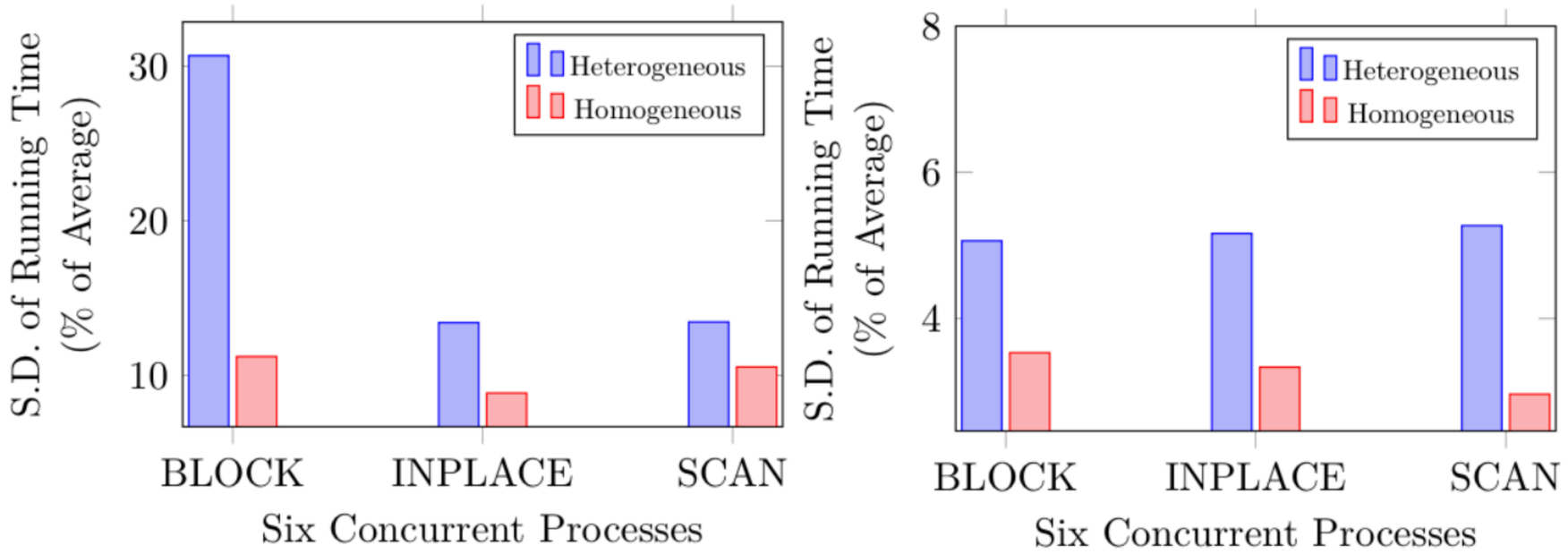


(a) Average running time of a program instance normalized to that when a single instance runs.

(b) Difference of extremes of running time of a program instance relative to the standard deviation.

Figure 1 Multi-program performance as up to six concurrent program instances of two cache-oblivious and a cache-aware matrix multiplication algorithms run and share a memory.

Culprit of the progress imbalance: imbalance in cache-sharing

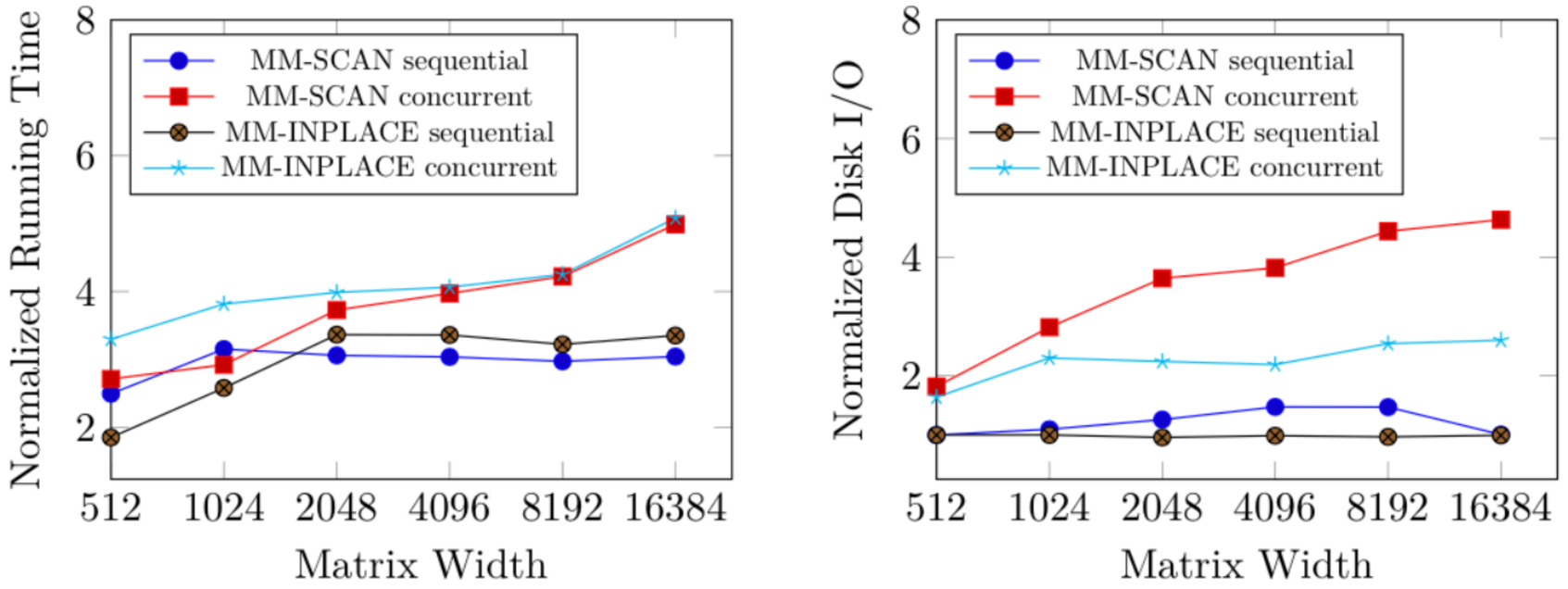


(a) Standard deviation (in % of average) of running time of the program instances in homogeneous and heterogeneous environment.

(b) Difference of extremes of running time of the program instances, normalized to the standard deviation.

Figure 2 Deviation of running time of the program instances in homogeneous and heterogeneous environment. Homogeneous environment: All instances pause at the end; Heterogeneous: Half of the instances pause in the middle, the other half pause at the end.

Multi-threading breeds performance advantage in a multi-program environment



(a) Normalized running time of the program instance with a single thread to that of the program instance with multi-threading.

(b) Normalized disk I/O incurred by the program instance with a single thread to that by the program instance with multi-threading.

Figure 3 Performance of a program instance with a single thread normalized to an instance with multiple threads when they are run either sequentially or concurrently (and share a memory). Multi-threading is increasingly advantageous for larger problem sizes.

Questions:

1. Why matrix multiplication? What are the other problems we should work on?
2. And, why are they good choices? Is there a theoretical argument?
3. Only one reference :(What are the other references, or topic of papers we should look into?

Resource needed:

1. Machine
2. Senior member, maybe to do theory?

Roadmap:

Next Experiments

Goal: ESA, Track B, 2021?