Reproducibility Report for ACM SIGMOD 2021 Paper: "MxTasks: How to Make Efficient Synchronization and Prefetching Easy"

TIANZHENG WANG, Simon Fraser University, Canada

The provided code and scripts are very complete and usable. Results were successfully reproduced using the scripts easily without additional changes.

1 INTRODUCTION

The paper being reproduced is titled "MxTasks: How to Make Efficient Synchronization and Prefetching Easy" which is published in SIGMOD 2021, by Jan Mühlig and Jens Teubner of TU Dortmund [1].

The reproducibility test included 10 experiments (generating 10 figures) that correspond to the experiments done by the original paper. Overall, the reproduced results showed similar trends compared to what was reported by the original paper. There were some exceptions where at larger scales the trends differ, but it largely hints at the need for additional tuning instead of poor reproducibility of the submission, because the nature of software prefetching mechanisms requires careful selection of a set of parameters different between CPU models.

2 SUBMISSION

The submission itself is a PDF that provides detailed instructions to reproduce the results. For code and scripts, the submission provided a link to a GitHub repo (https://github.com/jmuehlig/mxtasking-reproducibility) which is the only repo that the reviewer needs to clone.

The repo included the necessary scripts to run the experiments, including a "master" script that runs all the necessary steps (e.g., automatically cloning the actual code repos, compiling code, and issuing commands) to finish experiments with plotting (all figures) without manual/additional work needed. Additional scripts, such as those for installing the dependencies and checking environment setup are also provided.

Overall, the experience of reproducing the results is very smooth with zero issues. Simply running the dependency installation script followed by the master script worked out of the box. *This is the ideal reproducibility submission.*

3 HARDWARE AND SOFTWARE ENVIRONMENT

Experiments are conducted on a dual-socket server with configurations shown in Table 1 ("Repro Review"). The main difference is that the machine used has a better CPU with more cores and runs at a higher frequency.

4 REPRODUCIBILITY EVALUATION

4.1 Process

Everything was very straightforward, following the instructions in the submission and running scripts. No additional communications with the authors were needed. The experiments were repeated twice and we obtained consistent results.

4.2 Results

Here we describe the results obtained by the reproducibility process. We describe each experiment/figure included in the submission.

Table 1. Hardware & Software environment

	Paper	Repro Review
CPU	2 × Intel Xeon Gold 6226	2 ×Intel Xeon Gold 6224R
cores	12 × 2	20 × 2
GHz	2.7	3.1
RAM	Not available	375GB
Storage	Not applicable	Not applicable
OS	Ubuntu 20.04	Ubuntu 20.04.2 LTS
Compiler	Clang 10.0.0	Clang 10.0.0

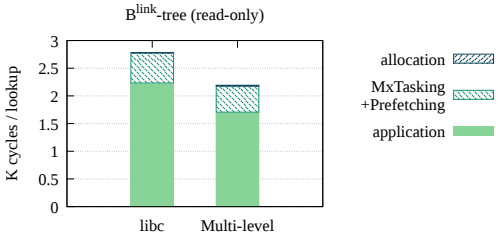

Fig. 1. Task allocation overhead (original paper's Figure 7).

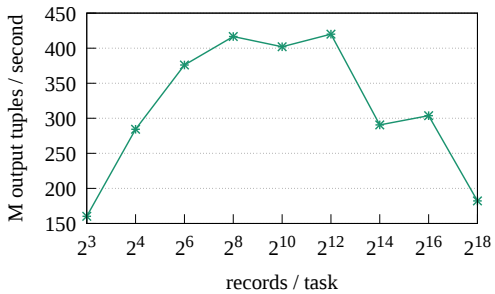

Fig. 2. Hash join under various task-granularities (original paper's Figure 9)

4.2.1 Task allocation overhead. The original paper measured CPU cycles breakdown under glibc and the proposed multi-level task allocation for a single Blink-tree lookup to show the number of cycles spent on task allocation, application, and MxTasking+Prefetching. Fig 1 shows the reproduced results. Compared to the original paper's result (Figure 7), the libc version here takes fewer cycles for allocation (still larger than the proposed multi-level approach). The proposed multi-level approach performed very well to show negligible allocation overhead. The overall amount of performance improvement is similar to the original paper's reported results.

4.2.2 Hash join under various task-granularities. This experiment joins the consumer and order tables in TPC-H using core-local hash tables, using different numbers of records per task. Fig 2 shows the reproduced results, which although showed more variations, exhibit the same trend as reported by the paper (Figure 9). The raw throughput is higher compared to the reported results, mainly because the machine we used here has a better CPU (higher frequency, larger cache, more cores).

4.2.3 Impact of software prefetching on Blink-tree. This experiment tests MxTasking's effectiveness on hiding memory stalls in accessing Blink-tree. Fig. 3–5 show the results. Compared to the original paper's result, overall the reproduced figures showed the same trend. The only exception is in the middle figures of Fig. 3 and 4: In Fig. 3 after 40–50 threads both prefetch and no-prefetch variants start to show lower throughput, whereas in the original paper, Figure 10a showed increasing performance as core count increases. In Fig. 4 after 40–50 threads "prefetch" started to show more stall cycles, and the original paper's Figure 10b showed that no-prefetch had consistently higher stalls for all core counts. We suspect the reason can be (1) hyperthreading's

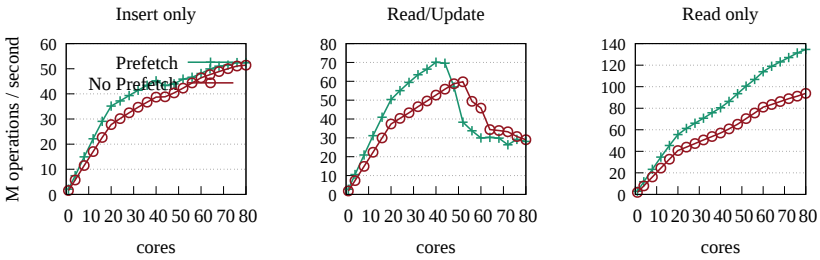

Fig. 3. Blink-tree throughput (original paper's Figure 10a)

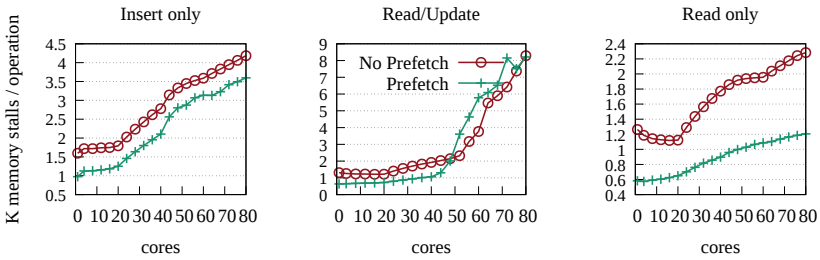

Fig. 4. Memory stalls per operation (original paper's Figure 10b)

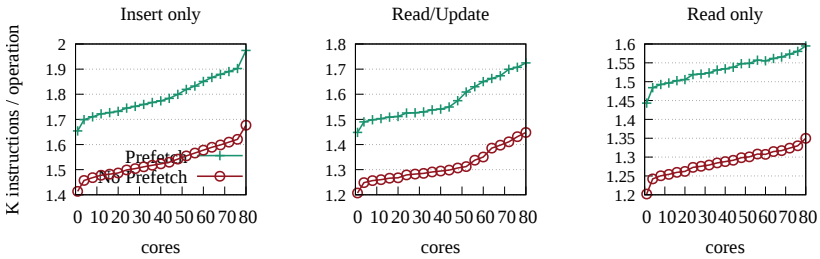

Fig. 5. Instructions per operation (original paper's Figure 10c)

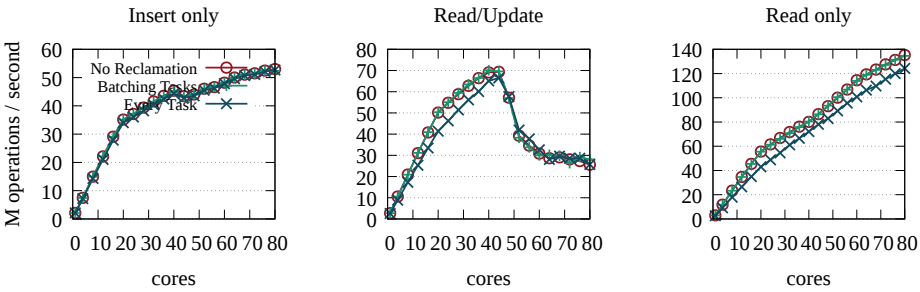

Fig. 6. Epoch-based reclamation scalability (original paper's Figure 11).

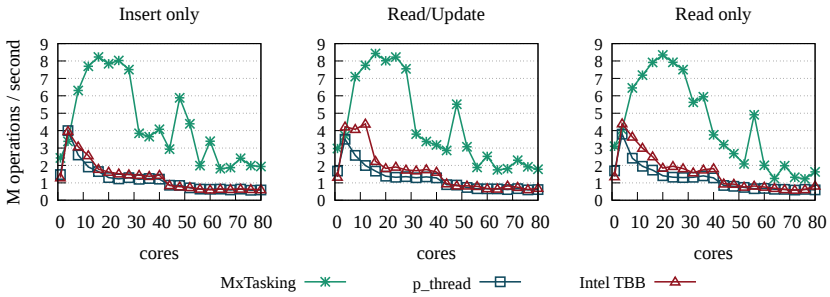

Fig. 7. Serialized synchronization (original paper's Figure 12a)

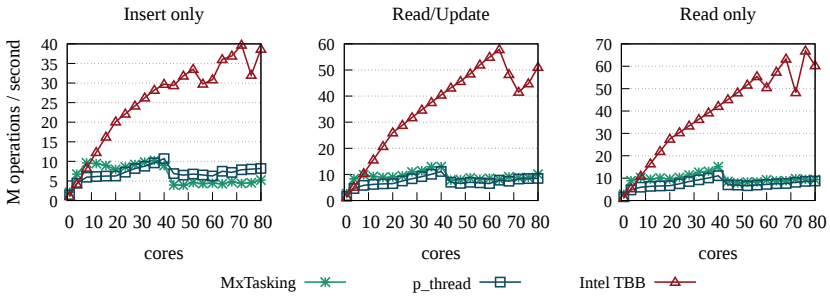

Fig. 8. Reader/writer locks (original paper's Figure 12b)

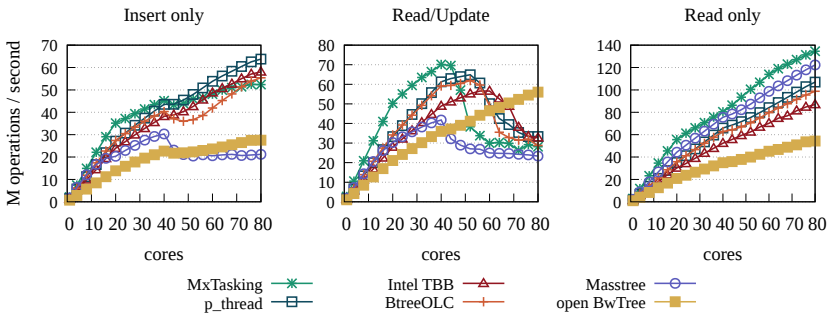

Fig. 9. Optimistic approaches (original paper's Figure 12c)

behavior may be different on the two CPUs, and (2) the access pattern somehow exhibits more locality under higher thread count so prefetching becomes a pure overhead.

In general, prefetching may behave quite differently across different CPU models, so this may mean more tuning is required to reproduce the exactly same results, but should not invalidate the reproducibility of the submission.

4.2.4 Scalability of epoch-based memory reclamation (EBMR). Fig. 6 reproduces the original paper's Figure 11 which shows the scalability of EBMR under varying core counts. Compared to the original paper's result, the only different trend shown is the performance drop after 40–50

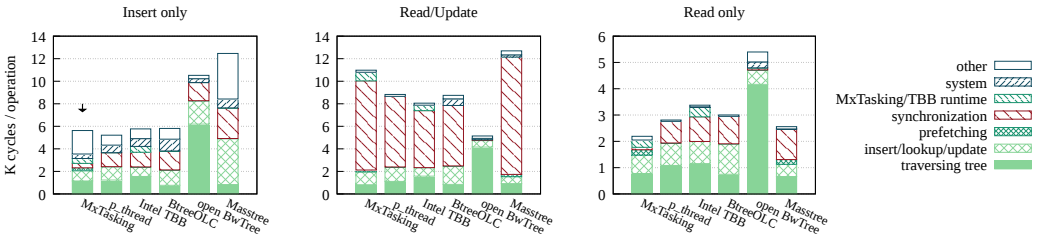

Fig. 10. Cycle breakdown for MxTasking and alternatives (original paper's Figure 13).

threads for read/update (as evidenced by the previous experiment as well). Again, we suspect more tuning is needed for this particular case.

4.2.5 Comparison of synchronization and programming models. Fig. 7–9 reproduce the original paper's Figures 12a–12c to compare different synchronization primitives and different tree structures. The trend shown here in Fig. 7–8 matches that reported by the original paper (Figures 12a–12b), although under high core count we see more fluctuations (the relative advantage of MxTasking is still maintained). Compared to the original paper, the testing machine here has more cores.

For the variants tested in Fig. 9, we observed the same trend except again for the read/update case in the middle figure (except for OpenBwTree), similar to the previous two experimental results.

4.2.6 Cycle breakdown for MxTasking and alternatives. The final experiment further shows the CPU cycle breakdown for MxTasking and other approaches for different index structures. Overall the results match those reported by the original paper. One observation is that due to the larger scale of the machine used here, synchronization cost appears to have increased compared to the original paper's results (obtained using a smaller-scale CPU).

5 SUMMARY

All the experimental results were correctly reproduced and showed overall trends similar to the original paper's. The process went very smoothly without any modification to the scripts and code provided.

REFERENCES

[1] Jan Mühlig and Jens Teubner. 2021. MxTasks: How to Make Efficient Synchronization and Prefetching Easy. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21)*. Association for Computing Machinery, New York, NY, USA, 1331–1344. https://doi.org/10.1145/3448016.3457268