Exploring Fine-Grained Parallelism in Data-flow Runtime Systems on Many-Core Systems

Wenyi Wang
The University of Chicago
Chicago, Illinois, USA
wenyiw@uchicago.edu

Maxime Gonthier
The University of Chicago
Chicago, Illinois, USA
mgonthier@uchicago.edu

Haibin Lai
Southern University of Science and Technology
Shenzhen, Guangdong, China
laihb2022@mail.sustech.edu.cn

Poornima Nookala
Intel Corporation
Chicago, Illinois, USA
nookala.poornima@gmail.com

Haochen Pan
The University of Chicago
Chicago, Illinois, USA
haochenpan@uchicago.edu

Ian Foster
The University of Chicago
Chicago, Illinois, USA
foster@uchicago.edu

Ioan Raicu
Illinois Institute of Technology
Chicago, Illinois, USA
iraicu@illinoistech.edu

Kyle Chard
The University of Chicago
Chicago, Illinois, USA
chard@uchicago.edu

Abstract

High synchronization overhead in frameworks like GNU OpenMP impedes fine-grained task parallelism on many-core architectures. We introduce three advances to GNU OpenMP: a lock-less concurrent queue (XQueue), a scalable distributed tree barrier, and two NUMA-aware, lock-less load balancing strategies.

Evaluated with Barcelona OpenMP Task Suite (BOTS) benchmarks, our XQueue and tree barrier improve performance by up to 1522.8×over the original GNU OpenMP. The load balancing strategies provide an additional performance improvement of up to 4×. We further apply these techniques to the TaskFlow runtime, demonstrating performance and scalability gains in selected applications while also analyzing the inherent limitations of the lock-less approach on x86 architectures.

Keywords

OpenMP, Fine-grained tasking, Load Balancing, Lock-less Programming, NUMA-Aware Scheduling, TaskFlow Runtime Systems

1 Introduction

The emergence of many-core Non-Uniform Memory Access (NUMA) systems complicates shared memory programming, primarily due to synchronization overhead and a lack of NUMA-awareness. We address these challenges within the task-parallel model, specifically OpenMP, whose implementations often scale poorly because of excessive lock-based synchronization [2, 3].

Our approach utilizes lock-less programming, which avoids both locks and atomic hardware primitives to manage shared data. We previously demonstrated the success of this method with a lock-less task queue in LLVM OpenMP, achieving up to 6× speedups [3].

In this work, we first target the widely used GNU OpenMP (GOMP) implementation. We redesign its fine-grained tasking system by integrating our novel lock-less concurrent queuing system – XQueue and inventing an efficient, hybrid lock-free and lock-less distributed tree barrier to overcome restrictive synchronization. These two optimizations combined improve the application performance by up to 1522.8×compared to GOMP. We then propose two novel lock-less, NUMA-aware dynamic load balancing (DLB) algorithms to mitigate the load imbalance issues introduced by XQueue. This optimization further improves performance by up to 4×. Finally, we extend the concepts to the TaskFlow runtime systems [1] and analyze the limitations of lock-less techniques on x86 architectures.

2 GNU-XTask: Fine-Grained Parallelism in GOMP with Lock-less DLB

We summarize our optimization strategies from our work [4] on GNU OpenMP with three implementation phases.

XGOMP: XQueue integration with GOMP: In this phase of implementation, we replace GNU's inefficient lock-based priority task queue with XQueue for massively parallel, fine-grained tasks.

XGOMPTB: Adding an Efficient Distributed Tree Barrier We replace XGOMP's lock-based centralized team barrier with a novel distributed tree barrier. Our design is the first hybrid distributed tree barrier for OpenMP tasking, featuring a lock-free gather phase and a lock-less release mechanism.

Lock-less NUMA-Aware Dynamic Load Balancing (DLB): XGOMPTB's static scheduler is NUMA-agnostic, which causes load imbalance. We address this by introducing two novel, lock-less, NUMA-aware DLB strategies for OpenMP: NUMA-aware Redirect Push (NA-RP) and NUMA-aware Work Stealing (NA-WS).

Our strategies rely on a lock-less messaging protocol where idle thieves send requests to busy victims via dedicated *round* and *request* memory cells.

3 Exploring Lock-less Scheduling in TaskFlow

To study the applicability of our lock-less techniques, we further extend XTask to the TaskFlow runtime systems [1], while maintaining TaskFlow's standard APIs. We explore two distinct lock-less scheduling strategies:

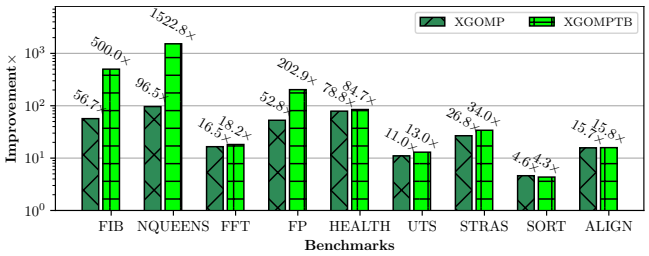

Figure 1: XGOMP/XGOMPTB performance improvement over GOMP

Local-First Work Sharing (LF X-TaskFlow). In this model, workers prioritize pushing to and consuming from their local queues. When idle, a worker sends out steal requests, prompting busy "victim" threads to actively share their work. This is designed to improve task and data locality observed with the static round-robin scheduler in our previous work.

Hybrid Scheduling (X-TaskFlow). We replace TaskFlow's Chase-Lev style deque with our XTask system, which combines a static round-robin scheduler with a lock-less dynamic work-stealing mechanism. A key improvement in this version was inspired by TaskFlow itself: we switched the Master Queue from First-In-First-Out (FIFO) to Last-In-First-Out (LIFO). This change significantly reduces stack consumption and improves cache utilization.

For both implementations, DLB occurs between the enqueue and dequeue operations, consistent with our earlier OpenMP framework.

4 Evaluation

4.1 GNU-XTask

Evaluation of XGOMP/XGOMPTB. We evaluate our implementations using all applications from BOTS benchmark and compare them with GOMP (see Figure 1) in [4]. The use of XQueue and the distributed tree barrier (XGOMPTB) can improve performance by up to **1522.8**× compared to GNU OpenMP.

Evaluation of GNU-XTask with Lock-less NUMA-Aware DLB. We evaluated all combinations of DLB settings, we found that: 1) NA-RP is better at load balancing coarse-grained tasks with aggressive LB settings, up to 4× improvement is achieved 2) NA-WS is a well-rounded method, both coarse-grained and fine-grained tasks can find an optimal setting.

Performance Tuning Guide. We provide guide in [4] for user to tune their application performance.

4.2 TaskFlow

The performance and scalability of our integrated TaskFlow variants were evaluated against the default TaskFlow, using optimal work-stealing settings for each. The results (Figure 2) highlight a nuanced performance landscape where the superior approach is highly dependent on application characteristics.

Our X-TaskFlow demonstrates superior performance and scalability in applications exhibiting map-reduce patterns or naturally balanced workloads. In these scenarios, the combination of our lock-less round-robin scheduler and DLB proves more efficient than the baseline's work-stealing mechanism, which incurs higher overhead due to a greater frequency of atomic operations.

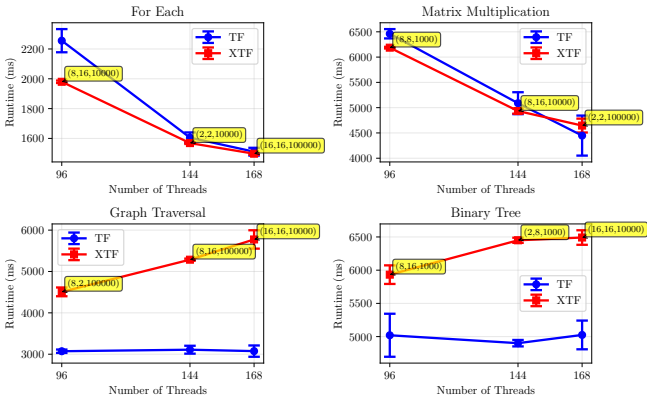

Figure 2: Representative performance results. X-axis is the Number of Threads, the Y-axis is Execution Time. Lower is better. Red line is ours, blue line is TaskFlow.

The preliminary results for LF X-TaskFlow show that it works only with applications that have extremely fine-grained tasks where their DAGs are unfolded in a generic tree path. Their performance results are worse than those of TaskFlow.

Limitations of Lock-less Programming

(1) We cannot achieve atomic *Read-Modify-Write (RMW)* operation as lock-less does not guarantee program order between threads.
(2) We leverage Total Store Order (TSO) on x86 platforms, this limits us to only use SPSC/work-sharing pattern as it guarantees store order, but not Store-Read order.
(3) *Work-sharing* requires prior knowledge of system load. Dynamically balancing load with work-sharing has the overhead of communication, plus the delay for the victim to be ready to share-work.

5 Summary

GNU OpenMP, although part of the mainstream compiler infrastructure GCC, is unable to harness modern high-performance CPUs with many cores, due to its use of locks and atomic operations. Our work improves GNU OpenMP performance by integrating a novel lock-less concurrent data structure, XQueue, and an efficient distributed tree barrier to achieve up to 1522.8× improvement compared to the original GNU OpenMP. We further improve performance through the use of dynamic load balancing strategies, demonstrating that with optimal settings, lock-less dynamic load balancing can achieve 4× more performance improvement compared to using static load balancing.

Our experiments show that good parameter choices are dependent on application characteristics. We provide guidance to help practitioners select parameter values based on their applications.

Lastly, we extend the OpenMP work in the data-flow system TaskFlow. Our evaluation, which yields mixed performance outcomes, highlights and discusses the inherent limitations of lock-less programming on modern x86 architectures.

References

[1] Tsung-Wei Huang, Dian-Lun Lin, Chun-Xun Lin, and Yibo Lin. 2022. Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System. *IEEE Trans. Parallel Distrib. Syst.* 33, 6 (June 2022), 1303–1320. doi:10.1109/TPDS.2021.3104255

[2] Agustín Navarro-Torres, Jesús Alastruey-Benedé, Pablo Ibáñez-Marín, and Maria Carpen-Amarie. 2021. Synchronization Strategies on Many-Core SMT Systems. In *IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. 54–63. doi:10.1109/SBAC-PAD53543.2021.00017 ISSN: 2643-3001.

[3] Poornima Nookala, Peter Dinda, Kyle C. Hale, Kyle Chard, and Ioan Raicu. 2021. Enabling Extremely Fine-grained Parallelism via Scalable Concurrent Queues on Modern Many-core Architectures. In *29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE, Houston, TX, USA, 1–8. doi:10.1109/MASCOTS53633.2021.9614292

[4] Wenyi Wang, Maxime Gonthier, Poornima Nookala, Haochen Pan, Ian Foster, Ioan Raicu, and Kyle Chard. 2025. Optimizing Fine-Grained Parallelism Through Dynamic Load Balancing on Multi-Socket Many-Core Systems. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 81–93. doi:10.1109/IPDPS64566.2025.00016