Exploring Fine-Grained Parallelism in Dataflow Runtime Systems on Multi-Socket Many-Core Systems

Wenyi Wang [1] Maxime Gonthier [1] Haibin Lai [2] Poornima Nookala [3] Haochen Pan [1] Ian Foster [1] Ioan Raicu [4] Kyle Chard [1]

[1]The University of Chicago [2]Southern University of Science and Technology [3]Intel Corporation [4]Illinois Institute of Technology

Performance Optimization for GNU-OpenMP Tasking and More

- We introduce XQueue, a lock-less concurrent queue implementation to replace GNU's priority queue and remove the global task lock.
- We develop a **scalable, efficient, and hybrid lock-free/lock-less** distributed tree barrier to address the high hardware synchronization overhead from GNU's centralized barrier.
- We develop two **lock-less and NUMA-aware** dynamic load balancing (DLB) strategies.
- We show that the use of XQueue and the distributed tree barrier can improve performance by up to 1522.8× compared to the original GNU OpenMP.
- We further show that lock-less load balancing can improve performance by up to 4× compared to GNU OpenMP using XQueue.
- We then show the limitation of XQueue in TaskFlow runtime system on modern x86 platforms.

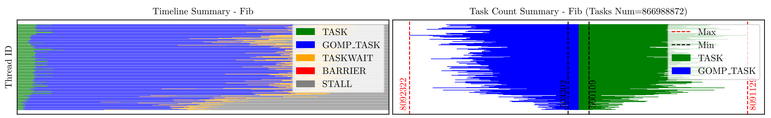
Background and Motivation

- **Higher Concurrency Level.** Concurrency level of modern computers are increasing from hundreds of CPUs to thousands of GPUs. For example:
  Mystic Eight-Socket System (Illinois Tech): 192 cores, 384 HW threads.
- **Non-Uniform Memory Access (NUMA)** architecture offers asymmetric access to cache and memory banks. Challenging to program using shared memory programming model.
- In **Task Parallel Programming Model,** computation is broken down into inter-dependent tasks that can be executed concurrently on various cores while respecting dependencies. GNU OpenMP has been a popular parallel library implementation of OpenMP that supports task parallel programming model.
- **GNU OpenMP (GOMP) is performing bad.** We use Barcelona OpenMP Task Suite (BOTS) to evaluate performance. GOMP is significantly (1000×) slower than LLVM OpenMP (LOMP).

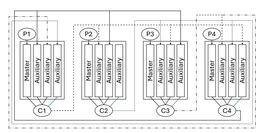
GNU-XTask: Fine-Grained Parallelism in GOMP with Lock-less DLB

Why is GNU-OpenMP (GOMP) much worse than LLVM OpenMP and how to optimize it?

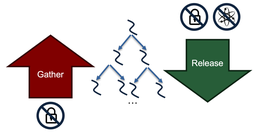
1. **A globally shared lock:** All threads contend for it, with large critical region. — **Remove it.**
2. **A redundant task queue structure:** One global task queue, one task queue per task. – **Replace it with XQueue (XGOMP).**
3. **A centralized team barrier:** All threads use it, with the global task lock. — **Replace it with Distributed Tree Barrier (XGOMPTB).**
4. XQueue **has load imbalance** due to its static round-robin approach. We propose a novel lock-less communication protocol and develop two NUMA-Aware lock-less DLB.
   - NUMA-Aware Redirect Push: (NA-RP) – Dynamically redirect new tasks to under-loaded workers.
   - NUMA-Aware Work Stealing: (NA-WS) – Steal, migrate work to under-loaded workers.



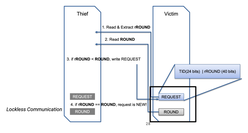
Load imbalance of **Fib**: Timeline Summary (Left), Task Count Summary (Right)



XQueue design on a 4-core system



Distributed Tree Barrier Design

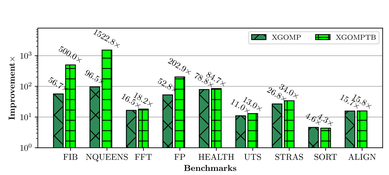


Lock-less Communication Protocol

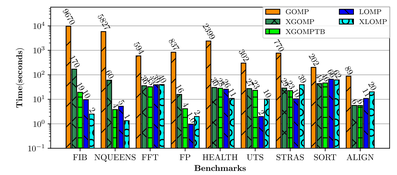
GNU-XTask Evaluation

Evaluation of XGOMP/XGOMPTB:

We evaluate our implementations using all 9 applications from the BOTS benchmark. We also evaluate LLVM OpenMP (LOMP) and LLVM OpenMP using XQueue (XLOMP). XQueue with the distributed tree barrier **(XGOMPTB)** allow to improve performance by up to 1522.8× compared to GNU OpenMP.



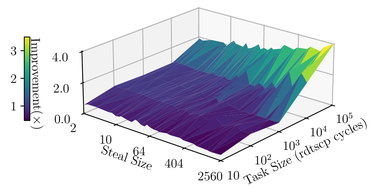
XGOMP/XGOMPTB performance improvement over GOMP



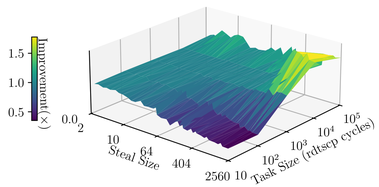
Execution Time of BOTS

Evaluation of GNU-XTask with Lock-less NUMA-Aware DLB:

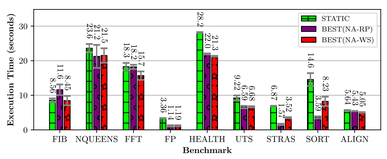
We evaluated all combinations of DLB settings: 1) NA-RP is better at load balancing coarse-grained tasks with aggressive LB settings, up to 4× improvement is achieved 2) NA-WS is a well-rounded method, both coarse-grained and fine-grained tasks can find an optimal setting.



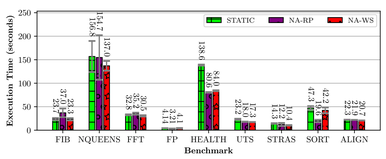
NA-RP Performance Speedup over XGOMPTB



NA-WS Performance Speedup over XGOMPTB



Execution Time Comparison with Optimal Settings
Small Problem Size



Execution Time Comparison with Settings from Small
Large Problem Size

GNU-XTask Performance Tuning Guide

We provide the following guide for user to tune their app performance.

- Fine-grained tasks (task size (S_{task})=10-100 cycles): Pick NA-WS, small steal size (S_{steal}), fully NUMA-local.
- When S_{task} increases, S_{steal} should increase, fully NUMA-local; NA-WS is still preferred.
- When $S_{task} > 10,000$ cycles, NA-RP is preferred; large S_{steal}, NUMA-local can be tuned down.
- When application characteristics are unclear, use NA-WS with small S_{steal} and NUMA-local.

Lock-less Exploration on Data-flow Programming

We explored the use of XTask in the data-flow runtime system TaskFlow.

We integrated XTask using TaskFlow's native API (X-TaskFlow). We show the performance and scaling of TaskFlow and X-TaskFlow with optimal work-stealing settings.

X-axis is the **Number of Threads**, the Y-axis is **Execution Time**. Lower is better. Red line is ours, blue line is TaskFlow.

X-TaskFlow outperforms TaskFlow and scales well in the first row, because these applications:

- Follow map-reduce pattern and have naturally balanced load in general.
- TaskFlow's approach with excessive atomic operations is suboptimal compared to ours.

TaskFlow outperforms X-TaskFlow and scale well in the other use-cases, because these applications:

- Have heavier load-imbalance issues in general.
- Our round-robin work-sharing approach breaks locality and cache benefits.

Limitations of lock-less approach:

1. Cannot achieve atomic *Read-Modify-Write (RMW)* operation by only using lock-less techniques.
2. Total Store Order (TSO) on x86 platforms limits us to only use SPSC/work-sharing pattern as it guarantees store order, but not Store-Read order.
3. Unavoidable communication overhead and delay for the victim to be ready to share work.