GNU-XTask: Optimizing Fine-Grained Parallelism through Dynamic Load Balancing on Multi-Socket Many-Core Systems

THE UNIVERSITY OF CHICAGO

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

[1]The University of Chicago [2]Intel Corporation [3]Illinois Institute of Technology

globus labs

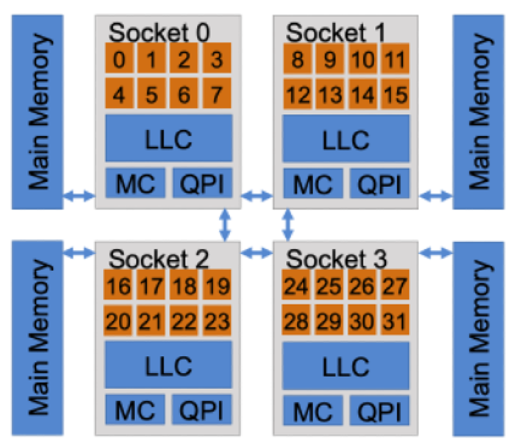
ILLINOIS TECH

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** load balancing 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.

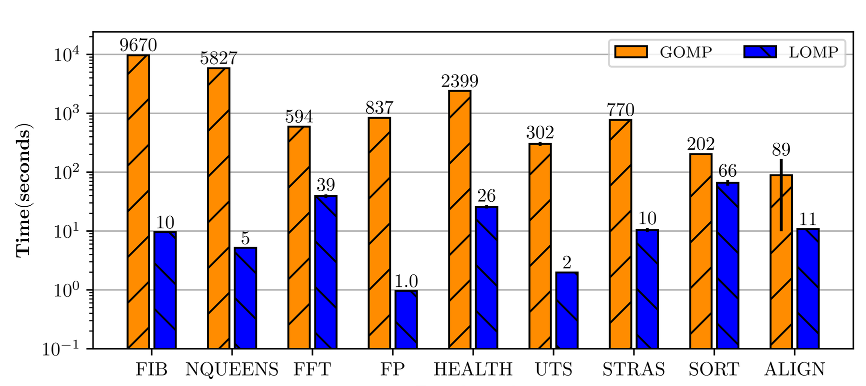
Background and Motivation

- **Higher Concurrency Level.** Concurrency level of modern computers are increasing from hundreds in CPUs to thousands in 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).



NUMA Architecture (Credit: Deters et al.)

libgomp

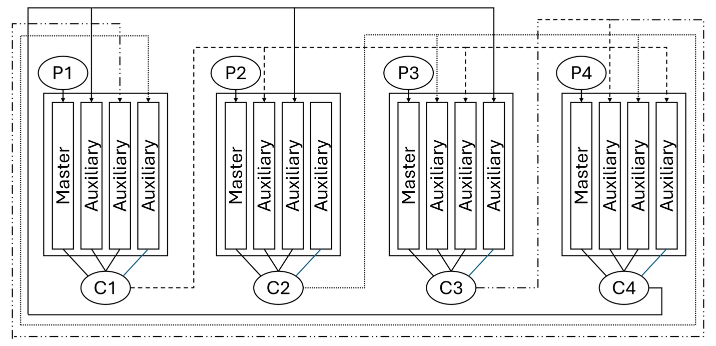


GOMP vs. LOMP Performance on BOTS

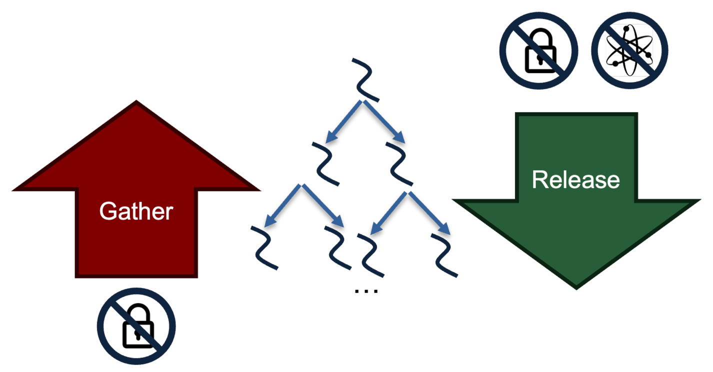
XGOMP/XGOMPTB: Enabling Fine-Grained Parallelism in GNU-OpenMP

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

1. **A globally shared lock:** All threads contending 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).**



XQueue design on a 4-core system

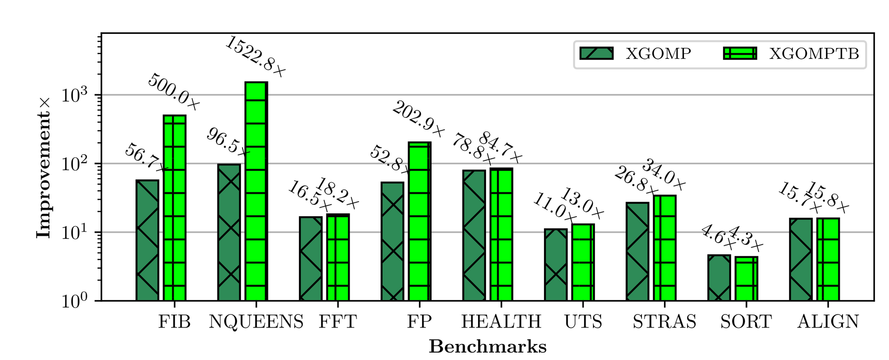


Distributed Tree Barrier Design

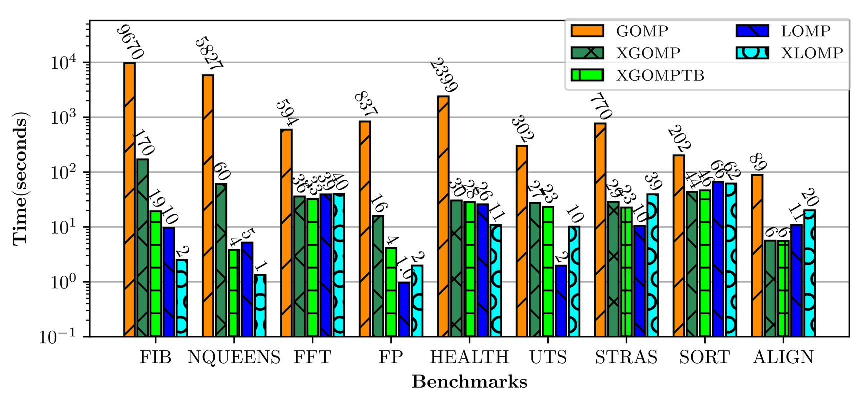
XGOMP/XGOMPTB Evaluation

We evaluate our XGOMP/XGOMPTB using all of nine applications from BOTS benchmark. We first compare our XGOMP/XGOMPTB with GNU OpenMP (GOMP). We also include LLVM OpenMP (LOMP) and LLVM OpenMP using XQueue (XLOMP) — in our results comparison.

The use of XQueue and the distributed tree barrier can improve performance by up to 1522.8× compared to GNU OpenMP.

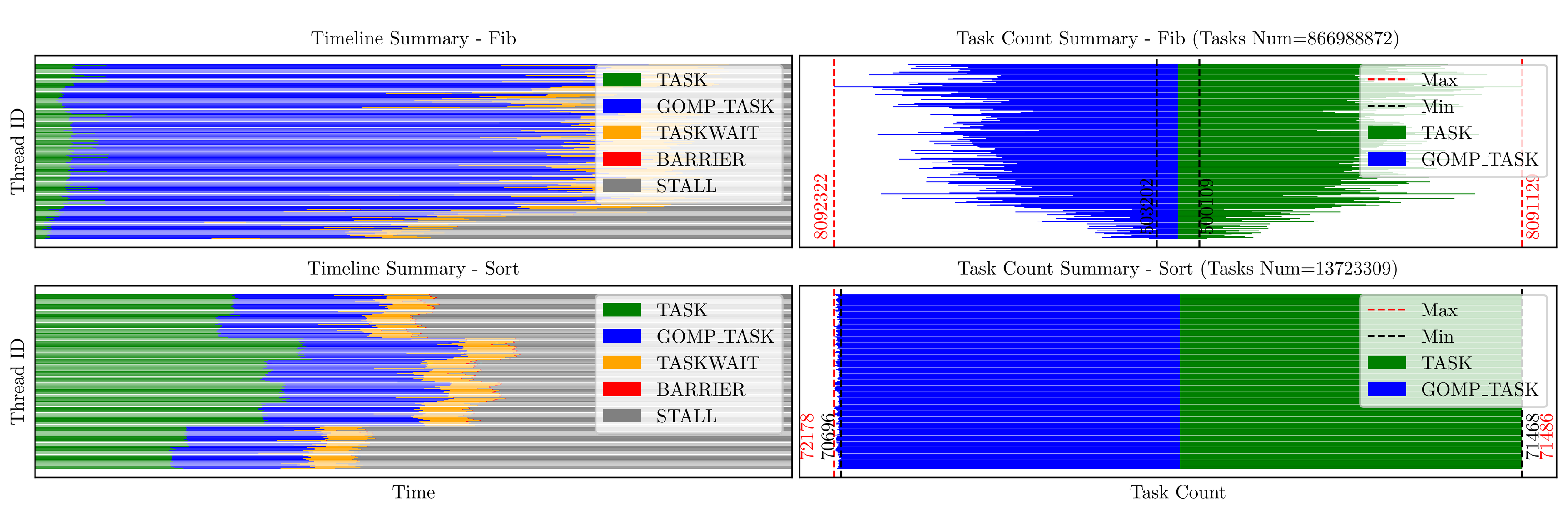


XGOMP/XGOMPTB performance improvement over GOMP



Absolute Execution Time of BOTS

However, XGOMPTB has load imbalance issues.

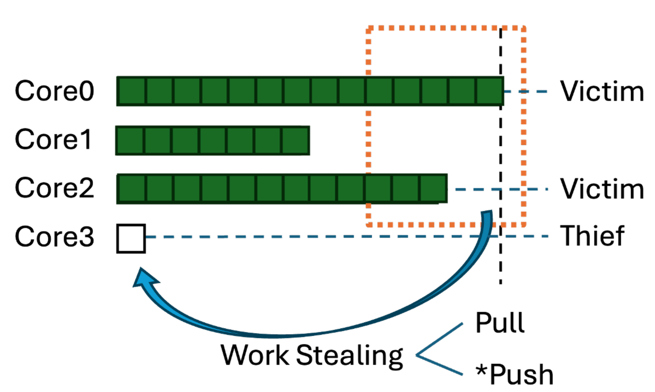


Load imbalance of **Fib** (above) and **Sort** (below); Timeline Summary (Left), Task Count Summary (Right)

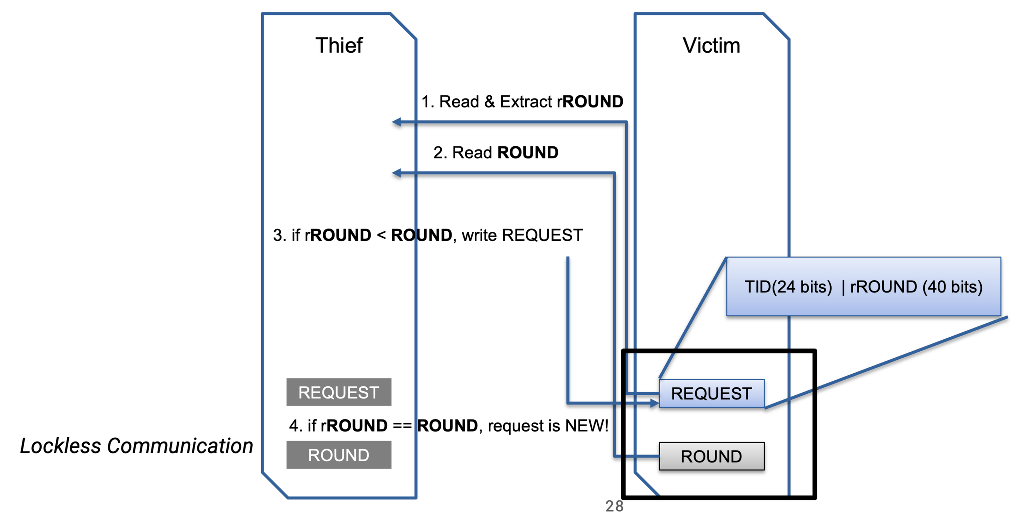
GNU-XTask: Achieving Lock-less NUMA-Aware Dynamic Load Balancing

We propose a novel lock-less communication protocol and develop two NUMA-Aware lock-less load balancing strategies based on it.

- **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.



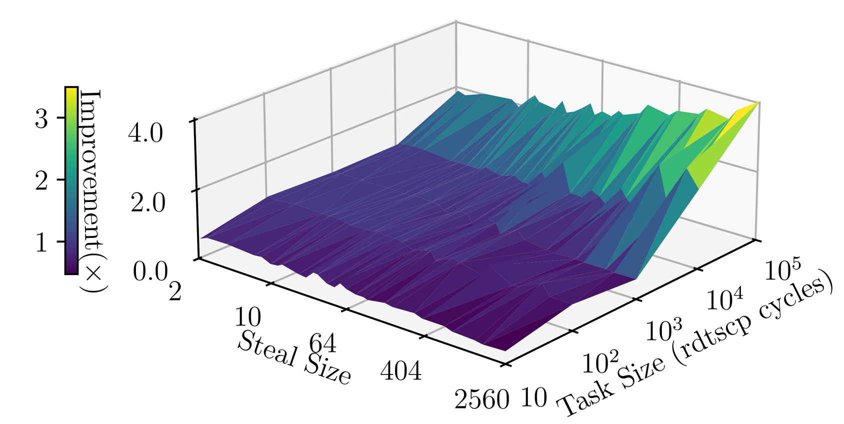
Work Stealing



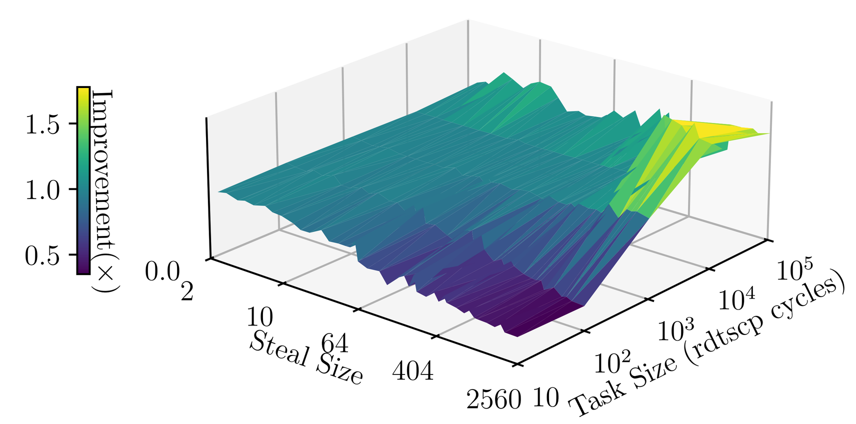
Lock-less Communication Protocol

GNU-XTask Evaluation with Parameter Sweeping

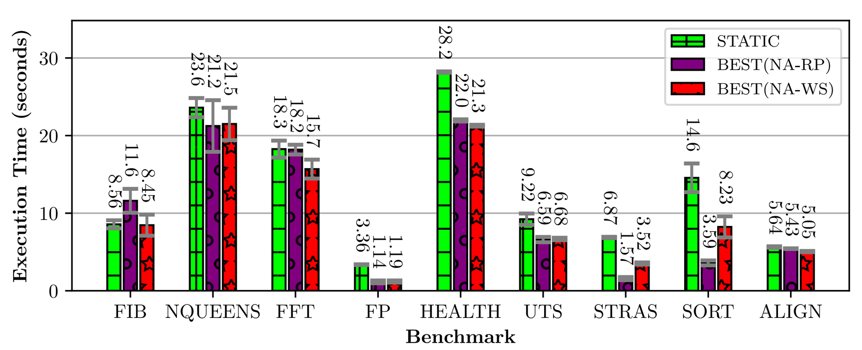
We went through all the combinations of DLB settings: 1) NA-RP is better at load balancing coarse-grained tasks with aggressive LB settings, up to 4× improvement is achieved 2) NA-WS is well-round method, both coarse-grained and fine-grained tasks can find an optimal setting.



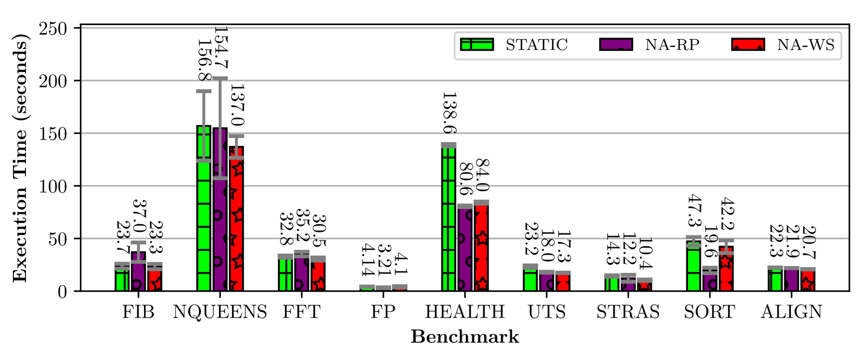
NA-RP Performance Speedup over XGOMPTB



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

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 fully NUMA-local