2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)

Optimizing Fine-Grained Parallelism Through Dynamic Load Balancing on Multi-Socket Many-Core Systems

Wenyi Wang[1], Maxime Gonthier[1], Poornima Nookala[2], Haochen Pan[1], Ian Foster[1], Ioan Raicu[3], Kyle Chard[1]
[1]*Department of Computer Science, The University of Chicago, Chicago, Illinois, USA*
[2]*Intel, Chicago, Illinois, USA*
[3]*Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois, USA*
wenyiw@uchicago.edu, mgonthier@uchicago.edu, nookala.poornima@gmail.com,
haochenpan@uchicago.edu, foster@uchicago.edu, iraicu@iit.edu, chard@uchicago.edu

***Abstract*—Achieving efficient task parallelism on many-core architectures is an important challenge. The widely used GNU OpenMP implementation of the popular OpenMP parallel programming model incurs high overhead for fine-grained, short-running tasks due to time spent on runtime synchronization. In this work, we introduce and analyze three key advances that collectively achieve significant performance gains. First, we introduce XQueue, a lock-less concurrent queue implementation to replace GNU's priority task queue and remove the global task lock. Second, 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. Third, we develop two lock-less and NUMA-aware load balancing strategies. We evaluate our implementation using Barcelona OpenMP Task Suite (BOTS) benchmarks. 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.**

***Index Terms*—OpenMP, Fine-grained tasking, Load Balancing, Lock-less Programming, NUMA-Aware Scheduling**

I. Introduction

The emergence of many-core computing systems with con-

C (UPC) [3]. We focus on OpenMP, a task-centric model in which higher-level parallel constructs such as loops are translated into fine-granularity tasks with dependencies, which the runtime must dynamically schedule to available resources. When a task is enabled by some thread, it is conceptually queued for execution by a future available thread. Unfortunately, OpenMP implementations and their tasking data structures often scale poorly due to the excessive use of expensive synchronization operations, such as locks, to resolve dependencies [4]–[7].

Two approaches to avoiding the performance degradation associated with locks are *lock-free programming*, which typically relies on atomic hardware operations such as compare-and-swap to synchronize without locks, and *lock-less programming*, which relies on methods to safely manipulate shared data without using locks. We focus here on lock-less programming in which no atomic primitives are used. In prior work we proposed the use of a lock-less, concurrent, multi-producer multi-consumer (MPMC) task queue, called XQueue, in the LLVM-based OpenMP (LOMP) [4]. We showed significant performance improvements, often delivering 4–6× speedup, compared to native LLVM OpenMP by reducing the time spent