Lock-Free Priority-Aware Work Stealing

Akshay Joshi (akshayjo) and Amogha Hegde (amoghah)
Information Networking Institute, Carnegie Mellon University

Motivation

- Efficient load balancing is crucial for high performance parallel computing.
- Traditional work-stealing schedulers are oblivious to task priorities.
- This leads to priority inversion.
- Traditional schedulers favor throughput over priority enforcement.
- Goal: Respect task priority without centralized bottlenecks.

Existing Approaches and Challenges

- Single Deque: Fast but causes priority inversion.
- Global Priority Queue: Enforces order, but scales poorly.
- Multi-level queues [1] - decentralize priority but limited exploration in lock-free settings.
- How to combine lock-free scalability with priority-awareness?

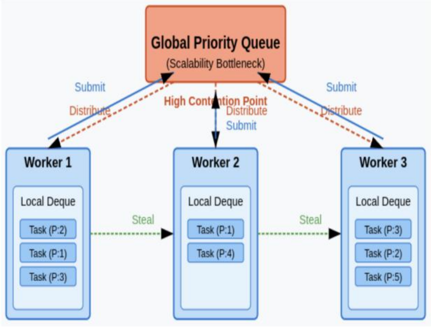

Fig. 1: Existing solution with Global coordinator

Methodology

- Multiple Lock-Free Deques per core - one per priority level.
- CAS-based operations for lock-free push/pop/steal.
- Memory consistency enforced via __sync_synchronize() barriers.
- Stealing prioritizes high-priority queues first; lower priorities considered only after exhausting higher ones.
- Local priority incremented only after checking all peer queues at the current level.
- Scalable across many-core systems.
- Randomized peer selection (~√n) to reduce overhead from exhaustive steal attempts and to analyze tradeoff.

Work Stealing Scenarios

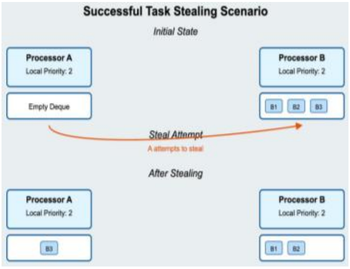

Fig. 2: Successful Steal

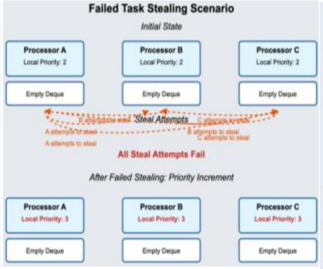

Fig. 3: Failed Steal

Fig. 4: Racing Steal

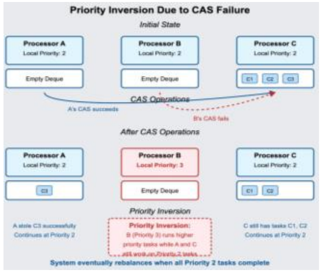

Fig. 5: Priority Inversion

Results and Discussions

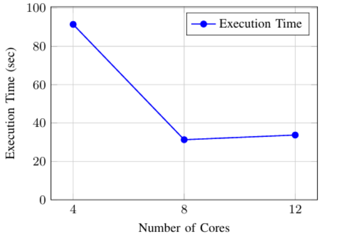

Fig. 6: Execution time vs No. of cores (tasks = 2880)

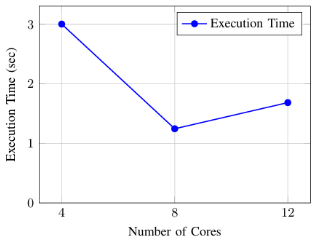

Fig. 7: Execution time vs No. of cores (tasks = 144)

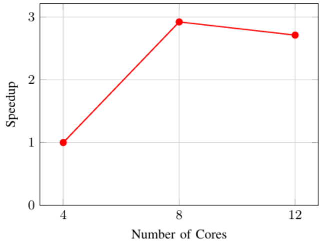

Fig. 8: Computational Speedup vs No. of Cores

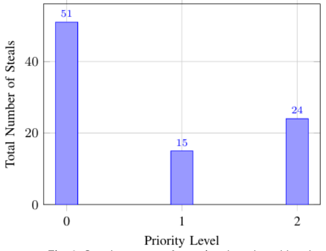

Fig. 9: Steal attempts for a simulated workload

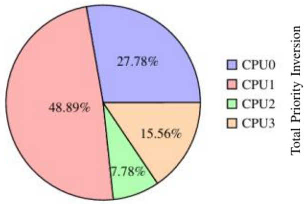

Fig. 10: Per core steals under simulated workload

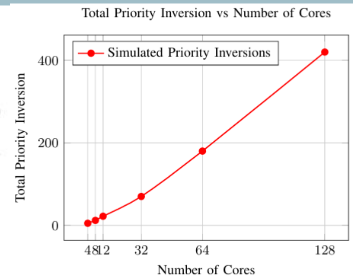

Fig. 11: Hypothetical increase in priority inversion as cores scale

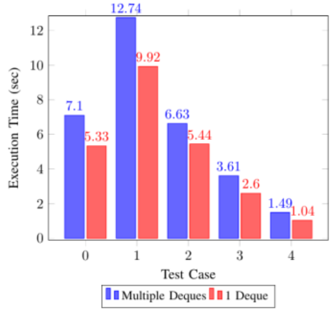

Fig. 12: Execution time comparison for 1 Deque vs Multiple Deques across test scenarios

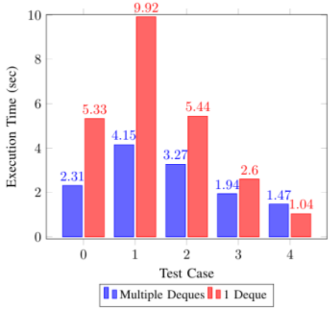

Fig. 13: Execution time comparison for 1 Deque vs Multiple Deques across test scenarios.

- Higher speedup when scaled from 4 to 8 cores but scaling further shows diminishing returns (Fig. 6 and 7).
- 1-deque (naive) is 15–30% faster (Fig. 12) but allows priority inversion. Our approach enforces priorities at cost of overhead.
- Inversion increases with core count due to limited peer checking (Fig. 11).
- In skewed workloads: High number of steal attempts at priority 0 shows that priority-based stealing helps recover balance (Fig. 9).
- Worst case: time to finish priority 0 tasks is approx. 3× faster (Fig. 13).
- Our system trades ~20% performance hit for strict priority enforcement.
- Ideal when task importance matters more than just throughput.

Acknowledgment

We would like to thank Prof. Todd C Mowry, and Prof. Brian P Railing to have provided us the opportunity and resources to work on this exciting project.

References

[1] Imam & Sarkar (2015) – Euro-Par Conference
[2] Chase & Lev (2005) – ACM SPAA
[3] Prokopec et al. (2015) – Euromicro PDP
[4] Lockless Work Stealing Deque in C by Paran Lee and Sho Nakatani