# LITERATURE REVIEW: Latency-Hiding Work Stealing: Scheduling Interacting Parallel Computations with Work Stealing

Vladimir Milicevic
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
*vladimir.milicevic@carleton.ca*

November 3, 2021

## 1 Introduction

Modern computing systems employ hundreds of processor cores which support simultaneous multithreading. Many language extensions for parallelism exist, such as Cilk and OpenMP. When running programs written in these parallel languages, the runtime system instantiates user-level threads which are mapped to the processors using a scheduler without a prior knowledge of the threads, running as an online scheduler within the system.

Much prior work has focused on scheduling threads which do not incur latency penalties while operating, such as sending transactions over a network. While this simplifies previous parallel scheduling algorithms such as work stealing, this hinders performance as the scheduler runs threads to completion without any switching. This class of non-preemptive scheduler appears adequate when workloads are heavily computational such as traditional high-performance computing applications, where application threads rarely incur any latency penalties.

## 2 Literature Review

Recent works in parallel scheduling have focused on multi-dimensional job scheduling on parallel machines using game theory frameworks [5], scheduling jobs to fairly allocate IO bandwidth in network applications [6], and have gone finer-grain to study effective schedulers for parallel computation on hierarchical caches [2]. Other prior works which capture system IO latencies within a chosen online scheduling algorithm include minimizing maximum flow time (makespan) [4] which effectively minimizes job latency, efficient algorithms for scheduling uni-processor threads when incorporating inter-processor IO delays [7], scheduling unit-length tasks on asynchronous multi-processor systems [3]. Work has been conducted in parallel thread-level schedulers which batch threads during critical sections that access a common structure [1] to avoid hazards at the expense of blocking concurrent access by other threads. In general past works focus less on fine-grain scheduling of tasks with non-unit length threads.

Many workloads which exploit the inherent parallelism of modern hardware such as datacenter and database applications, are not solely computational but also display high communication usage patterns. Accessing remote storage or even invoke remote procedure calls on devices within a network. Access to these remote devices is often shown to incur various latency penalties while a thread waits for a response. Previous work in work-stealing scheduling algorithms do not account for the latency incurred by a particular thread. As such, this work presents a work-stealing algorithm that hides latency by allowing threads to suspend and be swapped out by the preemptive scheduler to improve the performance of the underlying scheduling algorithm. Such techniques which allow the schedule to prompt the latency-bound thread and run another useful thread improve performance by hiding latency.

The use of a preemptive scheduler to schedule user-level threads differs from operating system schedulers, as the preemptive parallel schedulers studied here must control the order in which millions of fine-grain threads execute, while an OS scheduler seeks to hide the latency of course-grain jobs in the order of thousands. The preemptive latency hiding work stealing model for schedulers explored in this work shows mathematically the behaviour of parallel threads that incur latency penalties, and develops a latency-hiding scheduling algorithm for such computation. The benefit of such a scheduler is the ability to allow threads to suspend, and be overtaken by other threads on the hardware.

## 3 Selected Work

The work by Muller et el.[8] extends the standard model of parallel computations as Directed Acyclic Graphs (DAGs, or dags) by allowing edges to have non-unit weights that represent the latency incurred by an instruction. General instructions such as integer or floating-point arithmetic operations incur no latency penalty, but IO operations or remote procedure calls represent non-unit weight edges. The span of the DAG accounts for the latency incurred on the critical path. The scheduling algorithm used in this work uses deques (double-ended queues) to model threads ready for execution. Different from prior work, this work uses multiple deques per-worker (almost, virtual-deques) which are switched out when appropriate. The introduced algorithm is online and requires no prior knowledge of the DAG or edge weights, but requires knowledge of which edges are non-unit weight. The notion of *Suspension Width* is introduced to indicate the number of non-unit edges on the critical path of a DAG.

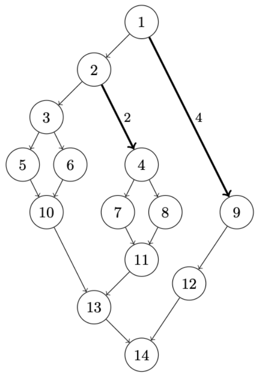

Figure 1: Instruction DAG including non-unit edges with suspension widths 2 and 4.

The offered runtime bound for a DAG with W work, S span, and $U >= 1$ suspension width running on a P-processor system is: $O(W/P + SU(1 + logU))$. When the suspension width is 0 (i.e., all unit-weight edges) the algorithm achieves the bound $O(W/P + S)$ which is identical to standard work stealing.

## References

[1] Kunal Agrawal, Jeremy T. Fineman, Kefu Lu, Brendan Sheridan, Jim Sukha, and Robert Utterback. Provably good scheduling for parallel programs that use data structures through implicit batching. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, page 84–95, New York, NY, USA, 2014. Association for Computing Machinery.

[2] Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallelizable jobs online to minimize the maximum flow time. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, page 195–205, New York, NY, USA, 2016. Association for Computing Machinery.

[3] Michael A. Bender and Cynthia A. Phillips. Scheduling dags on asynchronous processors. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, page 35–45, New York, NY, USA, 2007. Association for Computing Machinery.

[4] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Harsha Vardhan Simhadri. Scheduling irregular parallel computations on hierarchical caches. In *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, page 355–366, New York, NY, USA, 2011. Association for Computing Machinery.

[5] Leah Epstein and Elena Kleiman. Scheduling selfish jobs on multidimensional parallel machines. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, page 108–117, New York, NY, USA, 2014. Association for Computing Machinery.

[6] Ajay Gulati and Peter Varman. Lexicographic qos scheduling for parallel i/o. In *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '05, page 29–38, New York, NY, USA, 2005. Association for Computing Machinery.

[7] H. Jung, L. Kirousis, and P. Spirakis. Lower bounds and efficient algorithms for multiprocessor scheduling of dags with communication delays. In *Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '89, page 254–264, New York, NY, USA, 1989. Association for Computing Machinery.

[8] Stefan K. Muller and Umut A. Acar. Latency-hiding work stealing: Scheduling interacting parallel computations with work stealing. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, page 71–82, New York, NY, USA, 2016. Association for Computing Machinery.