# POSTER: Exploiting Multi-Level Task Dependencies to Prune Redundant Work in Relax-Ordered Task-Parallel Algorithms

**Masab Ahmad, Mohsin Shan, Akif Rehman, Omer Khan**
University of Connecticut, Storrs, CT, USA

***Abstract*—Work-efficient task-parallel algorithms enforce ordering between tasks using queuing primitives. Such algorithms offer limited parallelism due to queuing constraints that result in data movement and synchronization bottlenecks. Speculatively relaxing order of tasks across cores using the Galois framework shows promise as false dependencies generated by strict queuing constraints are mitigated to unlock task parallelism. However, relaxed ordering results in redundant work, for which Galois relies on static measures to improve work-efficiency. This paper proposes a dynamic multi-level parent-child task dependency checking mechanism in Galois to prune redundant work by exploiting monotonic properties of shared data values. Evaluation on a 40-core Intel Xeon multicore shows an average of 2× performance improvements over state-of-the-art ordered and relax ordered graph algorithms.**

## I. Introduction

Ordered algorithms achieve work-efficiency by executing tasks (or functions) in a particular order. For example, in graph algorithms, such order is enforced using queuing primitives [1], where graph vertices (parents) create edge tasks (children). Ordering is required on tasks due to read-write dependencies on preceding or future tasks with parent-child relationships. Due to the usage of queuing primitives to enforce order, exploiting parallelism requires thread synchronization to maintain global order among cores, which limits performance scalability on parallel machines. Therefore, prior literature has proposed unordered algorithms that break consistency guarantees using data structures that violate atomicity [1]. While this allows for more parallelism, it results in work inefficiency since the algorithm requires multiple iterations to converge on a solution. The Galois framework trades off parallelism with work efficiency, and enables relaxed ordering of tasks by only enforcing local order on tasks within a core using per-core priority queues [2], [3]. Since global ordering of tasks is not enforced, the Galois framework exposes plentiful task-level parallelism. However, relax ordering results in redundant work, which is pruned using static algorithmic mechanisms (e.g. $\Delta$-stepping algorithm for shortest path computations). Such static measures do not fully exploit the work-efficiency and parallelism tradeoffs, and thus a dynamic work pruning mechanism is proposed in this paper.

A multi-level task dependency checking mechanism is proposed for relax-ordered algorithms that efficiently prunes out redundant work at runtime. For example, in graphs, the edge-based task processing allows speculative shared data value updates [2]. As global order is not maintained, algorithmic convergence is iteratively managed over these shared data updates. Slow propagation of these shared data values results in redundant processing of tasks. This issue can be solved by either making races in shared memory faster, or by doing multi-level parent-child dependency checking to prune tasks whose parents have been overwritten by new data values. These checks exploit the *monotonic property of shared data values*, where an algorithm's converging shared data values either only increase or decrease (e.g. decreasing per-node distances in shortest path algorithm). Many relax-ordered algorithms exhibit this property, hence this paper proposes a novel multi-level task dependency check. When a monotonic shared data value is updated by a task, it may violate global order as its latest counterpart task may have been executed in another core. A check across multiple parent-child levels detects and prunes this task from being processed redundantly. This detection is done using shared memory reads of the latest parent values, and comparing them to the old values that were stored alongside the task at insertion into the priority queue. The multi-level task dependency check is then performed based on the monotonic property of a given algorithm. The proposed check is implemented on top of the Galois framework, and evaluated on a 40-core Intel Xeon multicore. Results for a range of relax-ordered algorithms show an average of 2× performance improvement over the Galois relax-ordered, as well as state-of-the-art ordered and unordered algorithms.

## II. Task Pruning in Relax-Ordered Algorithms

The Galois framework executes relax-ordered algorithms by implementing a per-core priority queue, where task ordering is only enforced locally within a core. To exploit load-balanced task parallel execution, Galois ordered by integer metric (obim) scheduler is used. The obim scheduler uses a dynamic work stealing heuristic to balance the task distribution among cores. After enqueuing tasks, the framework uses operators to dequeue tasks from cores. The monotonic property of shared data values is used to dynamically check multiple parent-child levels, which improves Galois's static redundant work pruning operators. Based on the number of levels being checked, the shared data values of the parents of this task (that is being enqueued) are pushed into the queue. These old values are then checked against the latest value when the task is dequeued.

Consider the single source shortest path (SSSP) algorithm that operates on the decreasing distances of each task from the source node. These distances are maintained using the shared distance array, $D[]$. Once a task ($v_{id}$) is pushed to a per-core priority queue, the current distance of its parent (or for 2-level check, the parent's parent) is also stored alongside the task entry. When this task is dequeued based on its priority, its state is checked since other core(s) may have pushed the same task, $v_{id}$, with a lower distance. Note, these distances are atomically updated. The proposed check extracts the enqueued distances of the task and its parents, and

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00062

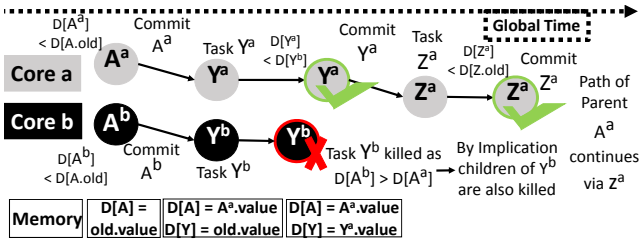


Fig. 1. Two-Level Task Pruning Check based on Monotonic Shared Data Values in the SSSP Algorithm.

performs data accesses to lookup the latest distance values for all corresponding $D[]$ locations. If the distances of the current task and/or its parent(s) are lowest in memory, then the task, $v_{id}$ is executed. Otherwise, $v_{id}$ is considered redundant and pruned out. Figure 1 shows how 2-level checking occurs in the proposed algorithm in a monotonically decreasing case for SSSP, with task $Y^b$ being deemed redundant as another core updates its parent's $D[]$ with a lower distance than $D[A^b]$. In this example, two cores, $a$ and $b$, go down the same path from task $A$. Task $A^b$ first enqueues task $Y^b$, while parent $A^a$ then enqueues child task $Y^a$. By the time Core $b$ extracts task $Y^b$ from its priority queue, the lower distance value of $A^a$ has made it to memory and is visible to all cores. Core $b$ checks $D[Y^b]$, and finds that it is the lowest distance ($D[Y^a]$ has not propagated to memory yet). Before executing task $Y^b$, Core $b$ also checks if $A^b$'s distance has changed since it was enqueued. Task $Y^b$ was enqueued with a parent distance of $D[A^b]$, but it has been updated with a lower distance in $D[A]$, hence task $Y^b$ is pruned out and not executed.

## III. Evaluation

Figure 2 shows the completion times normalized to the Galois baseline. Various relax-level versions of the proposed multi-level checks are evaluated to quantify the tradeoffs between task pruning efficiency and the associated checking overheads. Unordered benchmarks are acquired by picking the best performing implementation from a variety of open source suites, such as Julienne-Ligra [4], PBBS [5], and CRONO [6]. The Kinetic Dependence Graphs (KDG) [7] version of Galois that parallelizes ordered algorithms is also compared. KDG globally orders tasks using thread synchronization primitives, allowing task execution and commits to occur as they do in the sequential case. Each benchmark is evaluated for a set of diverse inputs, i.e., California road network, CAGE-14, Orkut, Friendster, Twitter, and Connectomic graphs [8] [9].

The proposed multi-level checking version outperforms all other variants, primarily because of the reduction in redundant work. This reduction in total work is seen in Figure 3, which shows the total work (number of tasks executed after dequeues) normalized to work in the Galois baseline. A geometric work-efficiency improvement of 1.6–3× is observed for various multi-level checks over Galois. SSSP, A*, and MST benchmarks add significantly more tasks for the unordered implementations since their sequential algorithms are highly work efficient. KDG does not execute redundant tasks, hence its total work is minimal. However, KDG limits parallelism due to consistent checking for global ordering. While the

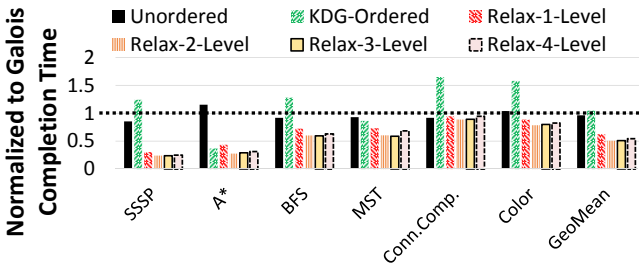


Fig. 2. Normalized Completion Time for Unordered, KDG-Ordered, and Proposed Algorithms Relative to the Galois Relax-Ordered Versions.

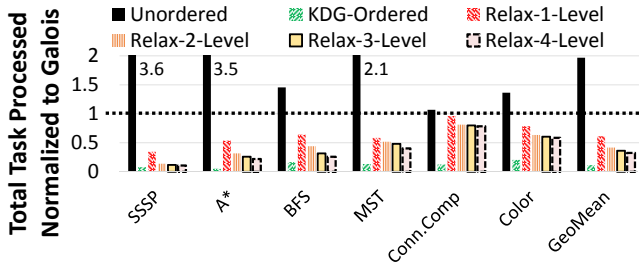


Fig. 3. Normalized Total Tasks Executed for Unordered, KDG-Ordered, and Proposed Algorithms Relative to the Galois Relax-Ordered Versions.

proposed deeper level checks, such as Relax-3-level maximize work reduction, the additional checking adds computations that worsen performance. Therefore, Relax-2-Level is picked since it enables an average 2× reduction in completion time, while optimizing task pruning over the Galois framework.

## Acknowledgments

This work was funded by the U.S. Government under a grant by the Naval Research Laboratory. This work was also supported by the National Science Foundation (NSF) under Grant No. CNS-1718481.

## References

[1] K. Pingali and et. al., "Ordered vs. unordered: A comparison of parallelism and work-efficiency in irregular algorithms," in *Proc. of the 16th ACM Symp. on Principles and Prac. of Parallel Prog.*, ser. PPoPP, 2011.

[2] D. Nguyen, A. Lenharth, and K. Pingali, "A lightweight infrastructure for graph analytics," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13, NY, USA.

[3] A. Lenharth, D. Nguyen, and K. Pingali, "Priority queues are not good concurrent priority schedulers," in *Euro-Par 2015: Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 209–221.

[4] J. Shun and et. al., "Julienne: A framework for parallel graph algorithms using work-efficient bucketing," in *Proc. of the 29th ACM SPAA*, 2017.

[5] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, A. Kyrola, H. V. Simhadri, and K. Tangwongsan, "Brief announcement: the problem based benchmark suite," in *SPAA*, 2012.

[6] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *2015 IEEE International Symposium on Workload Characterization*.

[7] M. A. Hassaan, D. D. Nguyen, and K. K. Pingali, "Kinetic dependence graphs," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: ACM, 2015.

[8] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on AI*, 2015.

[9] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, Eds., *The Shortest Path Problem, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, 2006*. DIMACS/AMS, 2009.