Accelerating Irregular Algorithms on GPGPUs Using Fine-Grain Hardware Worklists

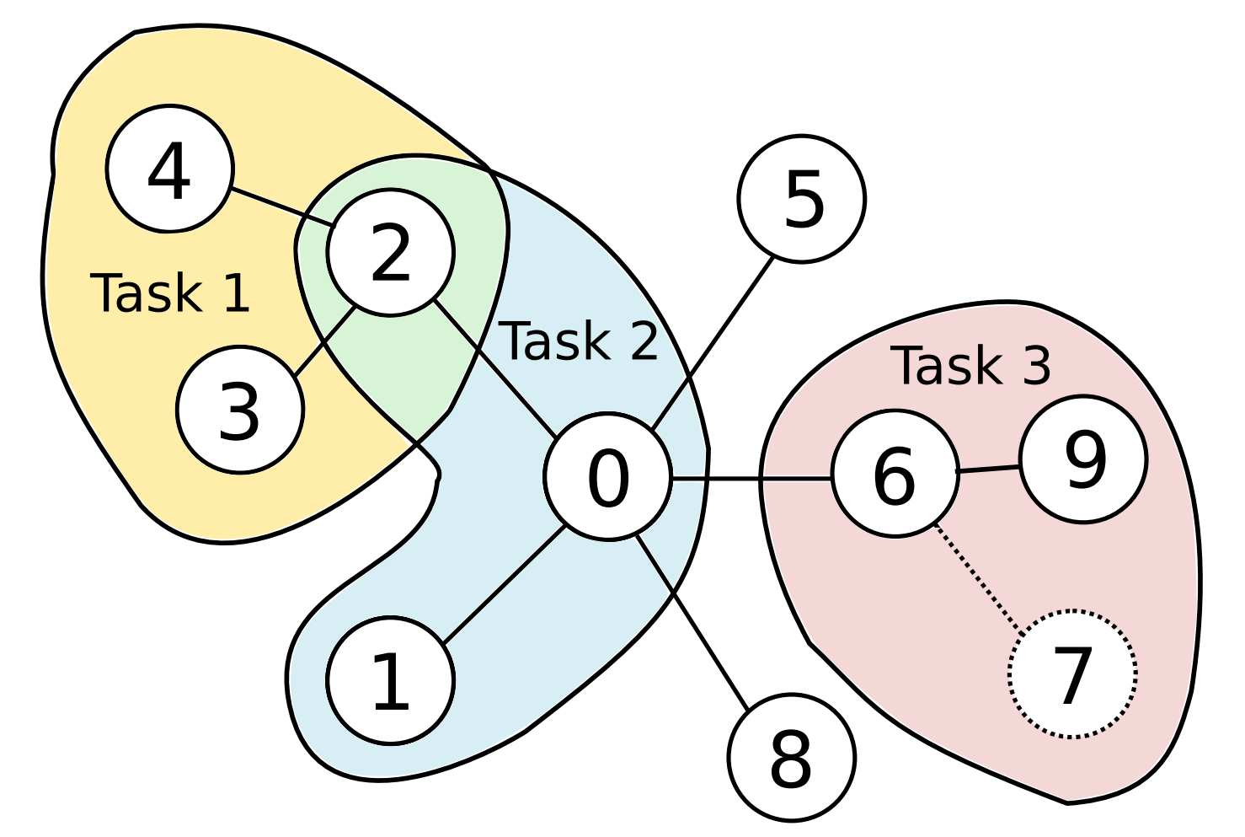
Ji Kim and Christopher Batten
Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University

1 Abstract

In this paper, we propose a novel fine-grain hardware worklist for GPGPUs that **addresses the classic weaknesses of data-driven implementations of irregular algorithms**. A set of distributed hardware worklist banks are tightly integrated with the GPGPU lanes are used to reduce memory contention and software overheads. We also detail multiple work redistribution schemes of varying complexity that can be employed to improve load balancing. Furthermore, a virtualization mechanism supports seamless work spilling and refilling. We evaluate challenging irregular algorithms from the LonestarGPU benchmark suite on a cycle-level simulator. We found that **using hardware worklists on a GPGPU yields speedups ranging from 1.2–2.4× over highly optimized software baselines on a nominal GPGPU.**

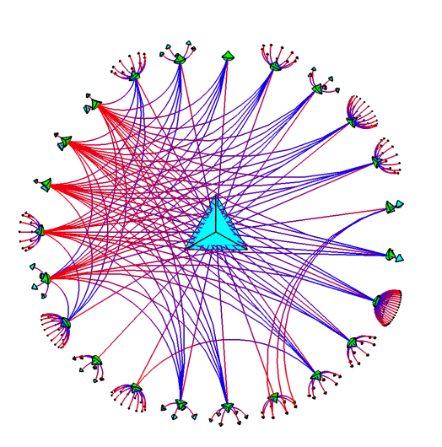
2 Motivation

GPGPUs excel at exploiting conventional data parallelism to achieve high performance and energy efficiency. However, it is much more challenging to map more irregular **amorphous data parallel** applications to GPGPUs which allows tasks to have conflicting accesses, to be generated dynamically, and to modify the underlying data structure. Even aggressive software optimizations do not fully mitigate issues with memory contention, suboptimal load balancing, and software overhead.

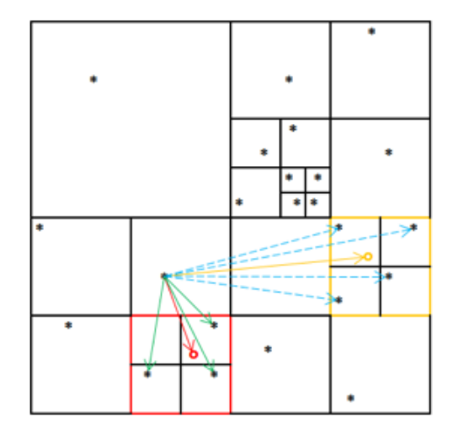


Example Amorphous Data Parallel Applications

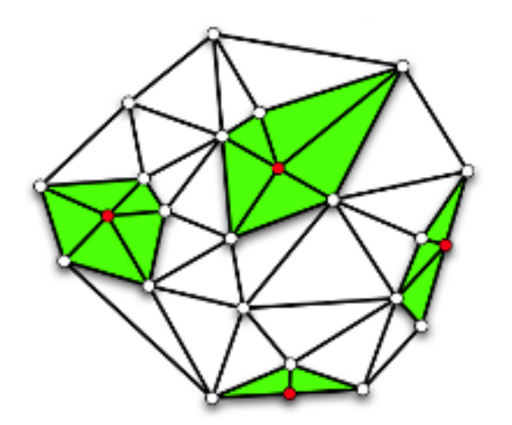
Breadth-First Search



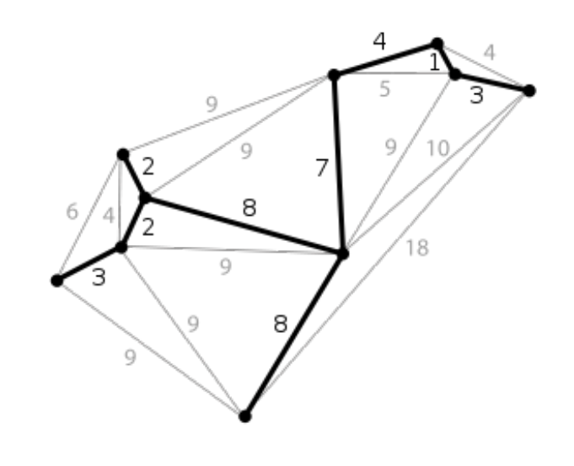
Barnes-Hut N-Body



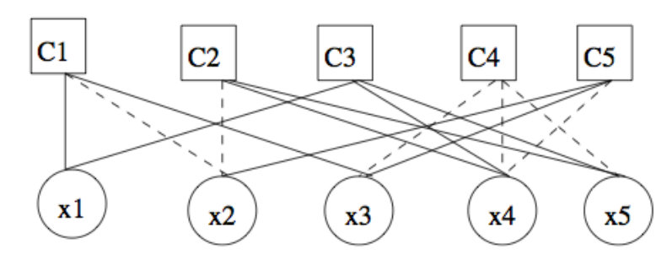
Delaunay Mesh Refinement



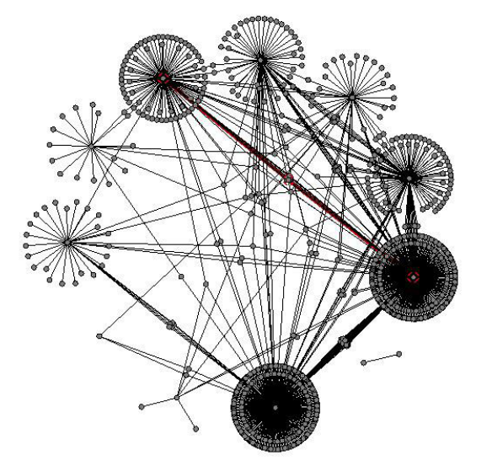
Minimum Spanning Tree



Survey Propagation



Single-Source Shortest-Path



- ▷ **Breadth-First Search –** Calculates number of hops from a source node to all other nodes in an unweighted graph.
- ▷ **Barnes-Hut N-Body –** Performs an N-body simulation using an octree.
- ▷ **Delaunay Mesh Refinement –** Fixes triangles in a mesh that violate geometric constraints.
- ▷ **Minimum Spanning Tree –** Computes a subset of nodes in a weighted graph that spans all nodes with a minimum cost.
- ▷ **Survey Propagation –** Heuristic SAT solver to determine the probability of a boolean statement being true.
- ▷ **Single-Source Shortest Path –** Calculates cost from a source node to all other nodes on a weighted graph.

3 Mapping Irregular Algorithms to GPGPUs

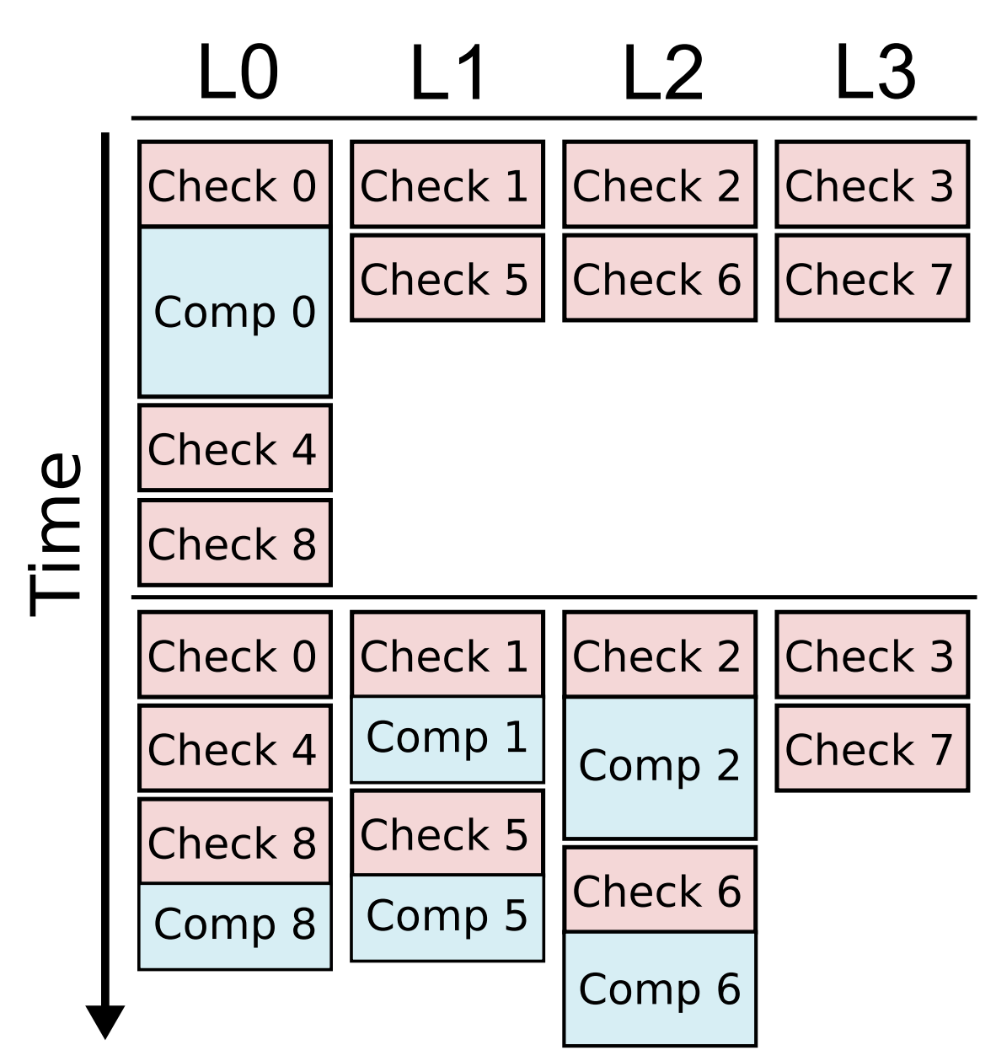
Irregular algorithms iteratively apply a set of operators on a subset of elements in the data structure which are referred to as **active nodes**. The check operator determines whether or not the element assigned to the thread is an active node or not. The compute operator performs the actual work required for the algorithm to progress and can generate more work by activating **inactive nodes**. There are two standard approaches to mapping irregular algorithms to GPGPUs.

Topology-Driven Approach

- ▷ Work is determined based on thread index
- ▷ All elements are visited whether or not they are active
- ▷ Number of threads spawned is equal to the number of elements

```
def topo_driven:
  idx = get_tid()
  my_node = nodes[idx]
  if check( my_node ):
    compute( my_node )
    *done_ptr = false

def main:
  done = false
  while not done:
    done = true
    topo_driven<<<N>>>( nodes )
```

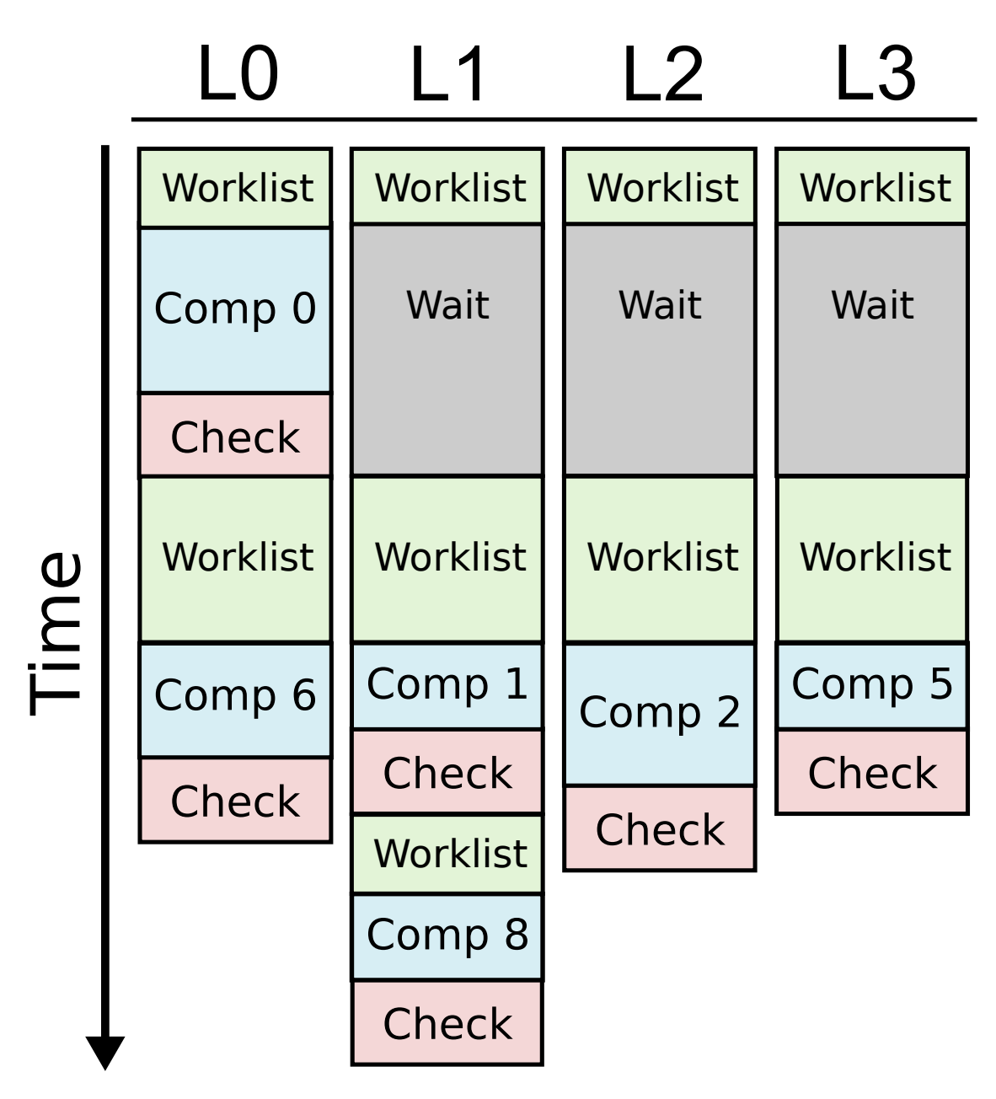


Data-Driven Approach

- ▷ Work is determined by accessing a shared software worklist
- ▷ Only active nodes are visited
- ▷ Number of threads spawned is equal to the number of hardware threads

```
def data_driven:
  while idx = wl.pull():
    my_node = nodes[idx]
    compute( my_node )
    for all neighbors of my_node:
      if check( neighbor ):
        wl.push( idx )

def main:
  init_wl<<<N>>>( nodes, wl )
  data_driven<<<M>>>( nodes, wl )
```



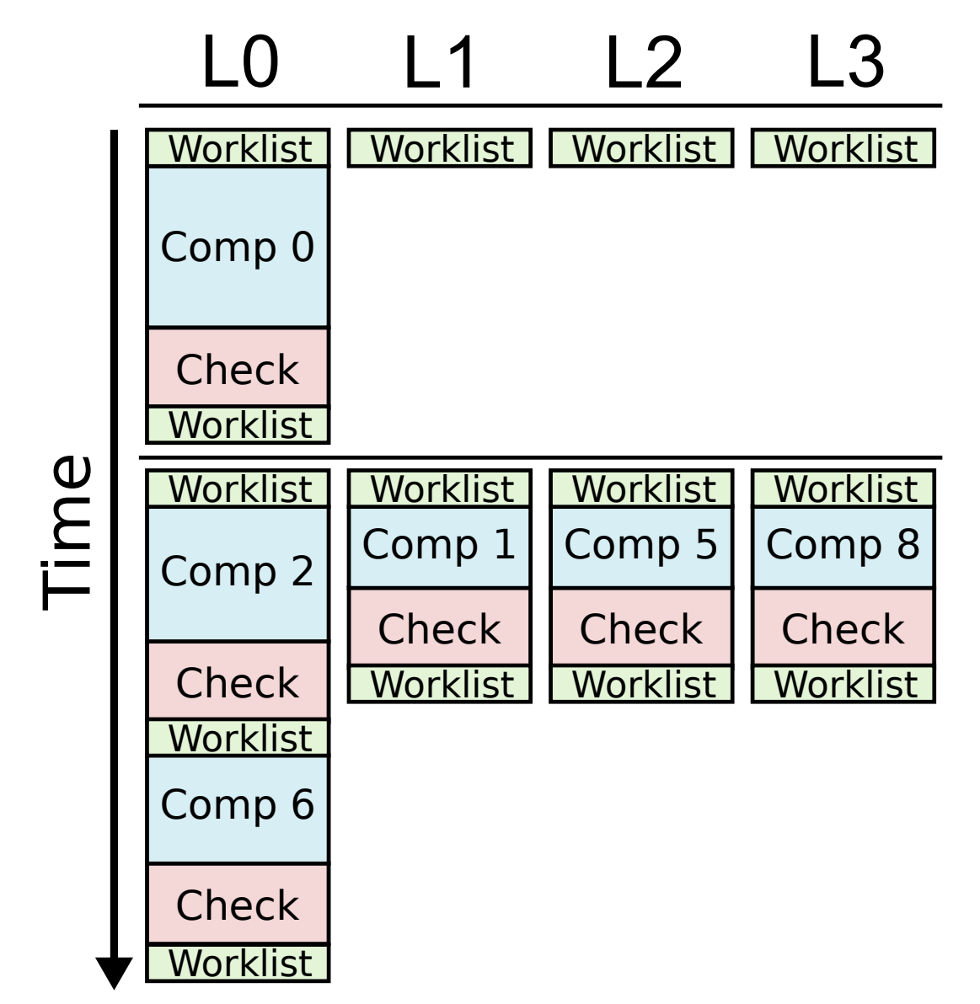
State of the Art Software Optimizations

We used benchmarks from multiple versions of the LonstarGPU benchmark suite and added our own implementations of software optimizations from previous work in order to select highly optimized topology- and data-driven implementations of each benchmark.

- ▷ **Double-buffering**
- ▷ Work chunking
- ▷ Atomic-reduced updates
- ▷ Work donating
- ▷ Hierarchical worklist
- ▷ Variable kernel configuration

```
def data_driven:
  for wid in range( start, end ):
    idx = inwl.pull( wid )
    my_node = nodes[idx]
    compute( my_node )
    for all neighbors of my_node:
      if check( neighbor ):
        outwl.push( neighbor.idx )

def main:
  init_wl<<<N>>>( nodes, inwl )
  while not inwl.empty():
    data_driven<<<M>>>( nodes, inwl, outwl )
    swap( inwl, outwl )
```



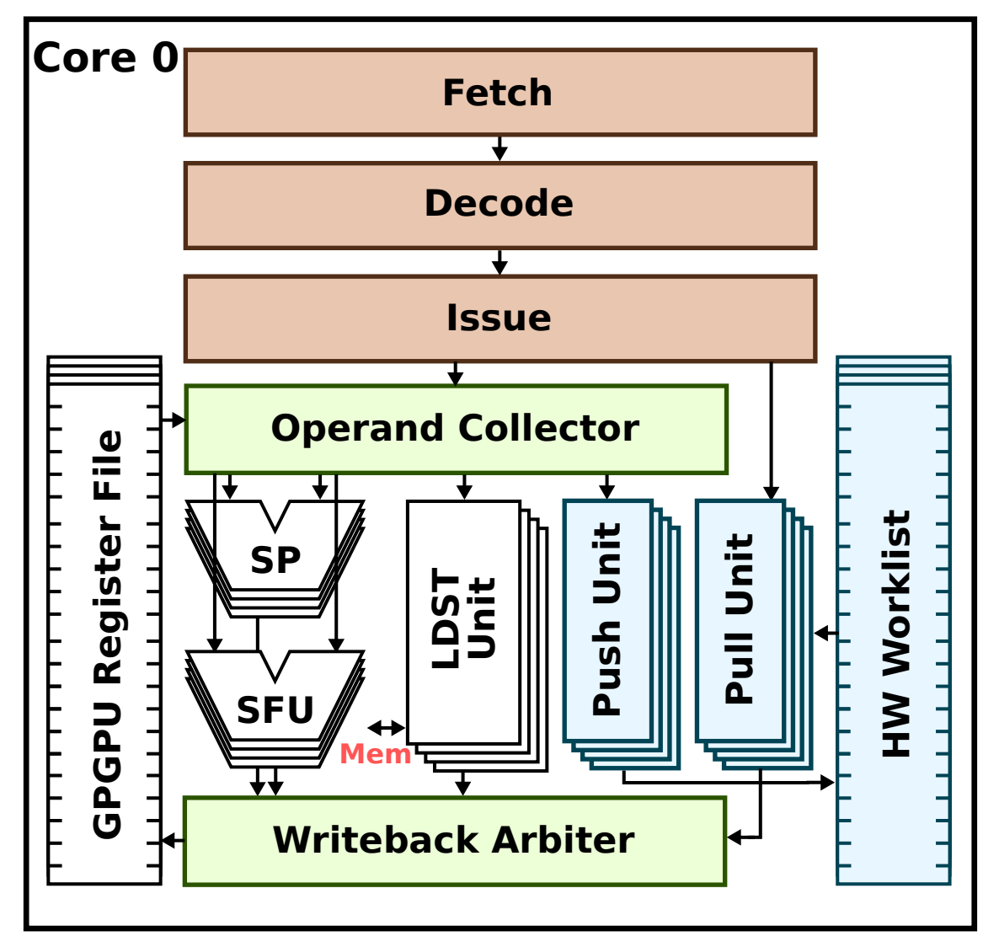
4 Fine-Grain Hardware Worklists

Fine-grain hardware worklists (HWWL) are implemented as **distributed banks** tightly integrated with the GPGPU lanes in order to reduce memory operations when interacting with the worklist. A **work redistribution** unit facilitates dynamic load balancing between banks within a core as well as across cores via a special redistribution network. A **virtualization** unit allows work that does not fit in the banks to seamless spill to an overflow buffer in memory and refill empty banks as necessary.

ISA Modifications

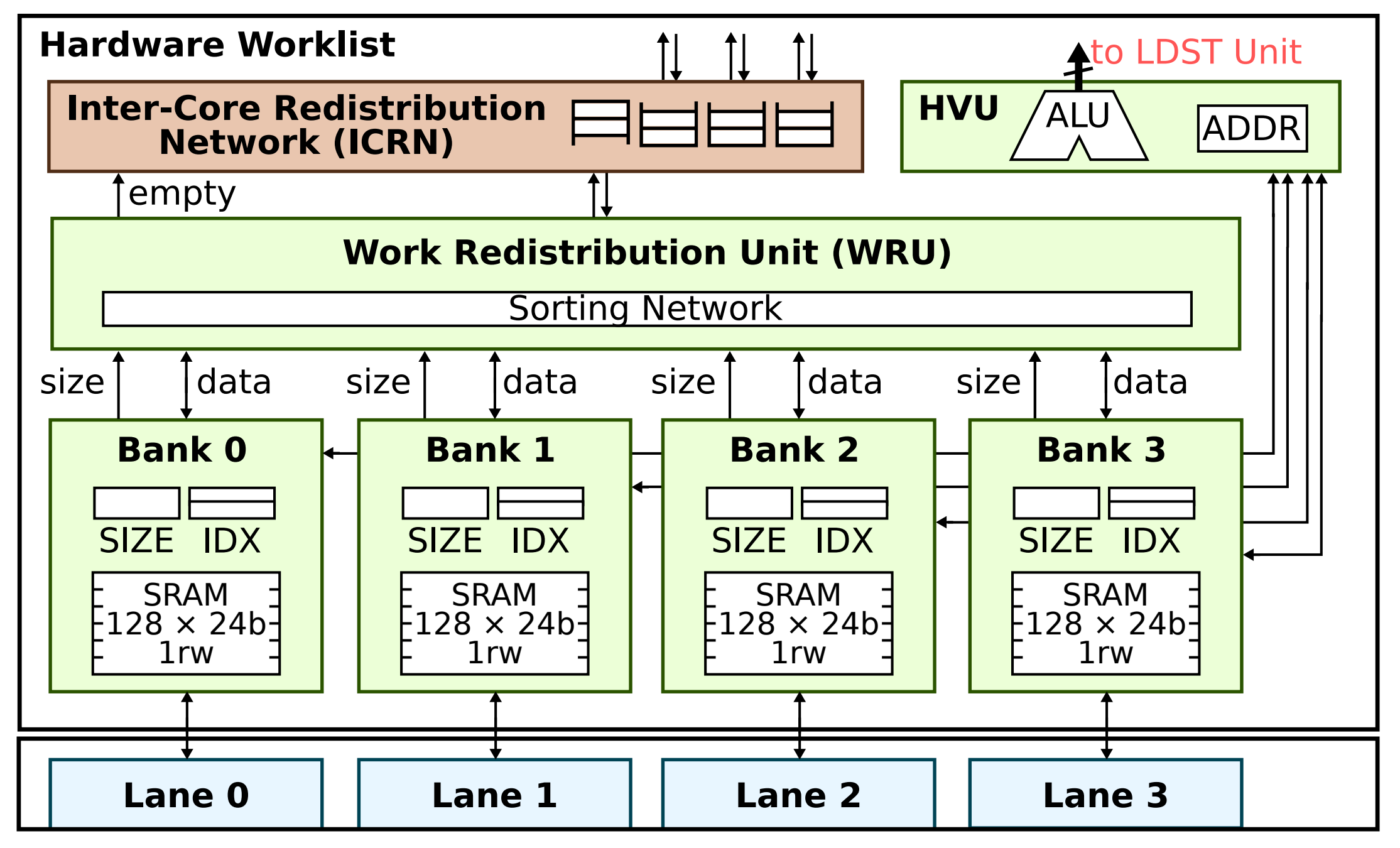
Instruction	Description
`wlinit r_d, r_s`	Initializes overflow buffer for virtualization.
`wlcfg r_s`	Configure partition mode (0=single,1=double).
`wlpull r_d, r_s`	Pulls work ID from HWWL, if local bank is empty: return WAIT if there is more work in system, or DONE otherwise.
`wlpush r_s, r_t`	Pushes work ID to HWWL, throws exception if overflow buffer is full.

GPGPU Integration



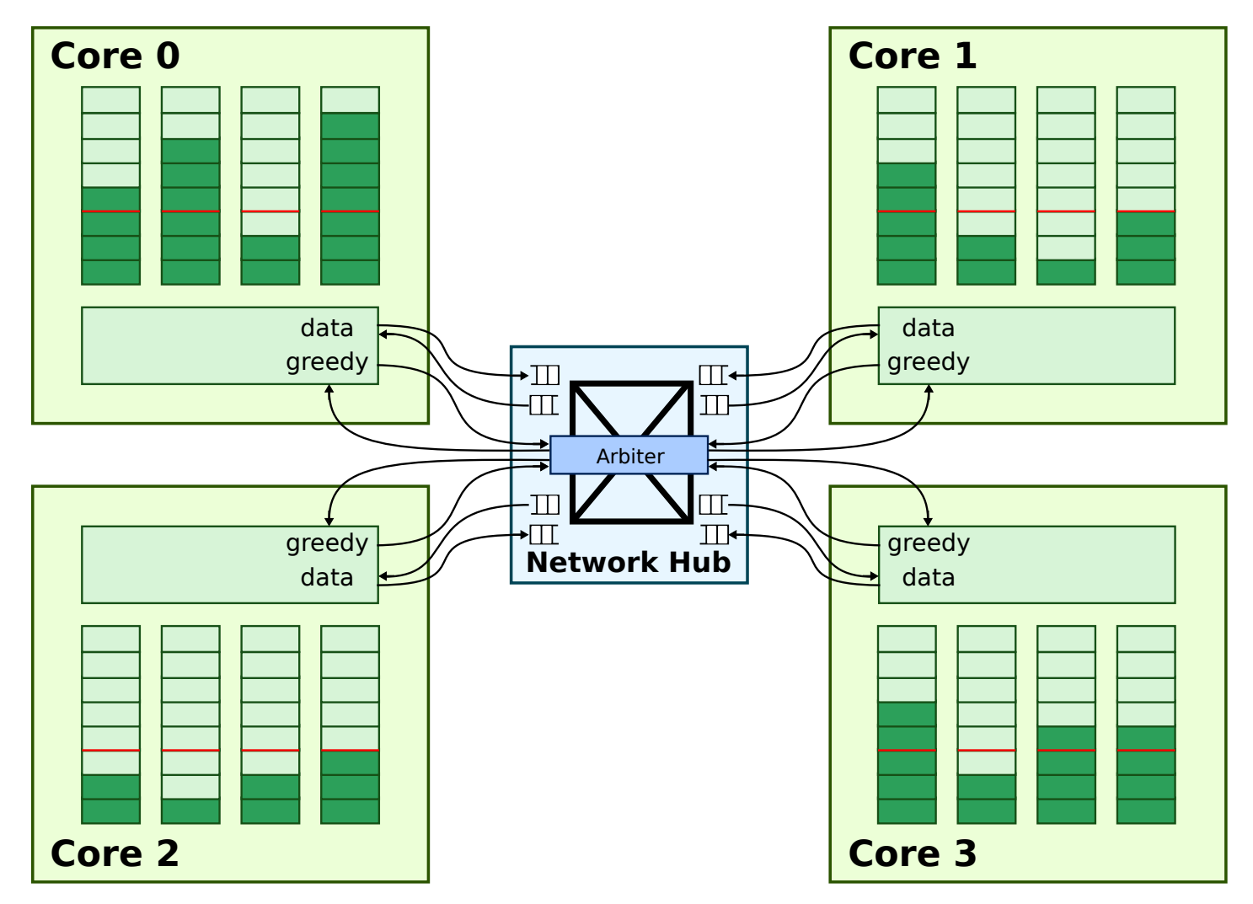
- ▷ **Distributed Banks –** Push units move work from the GPGPU register file to the HWWL banks and pull units move work in the opposite direction.
- ▷ **Work Redistribution –** The **threshold-based** and **sorting-based** schemes tradeoff hardware complexity for load balancing capabilities.
- ▷ **Virtualization –** The **on-demand** and **interval-based** schemes tradeoff energy efficiency for virtualization performance.

Detailed Microarchitecture



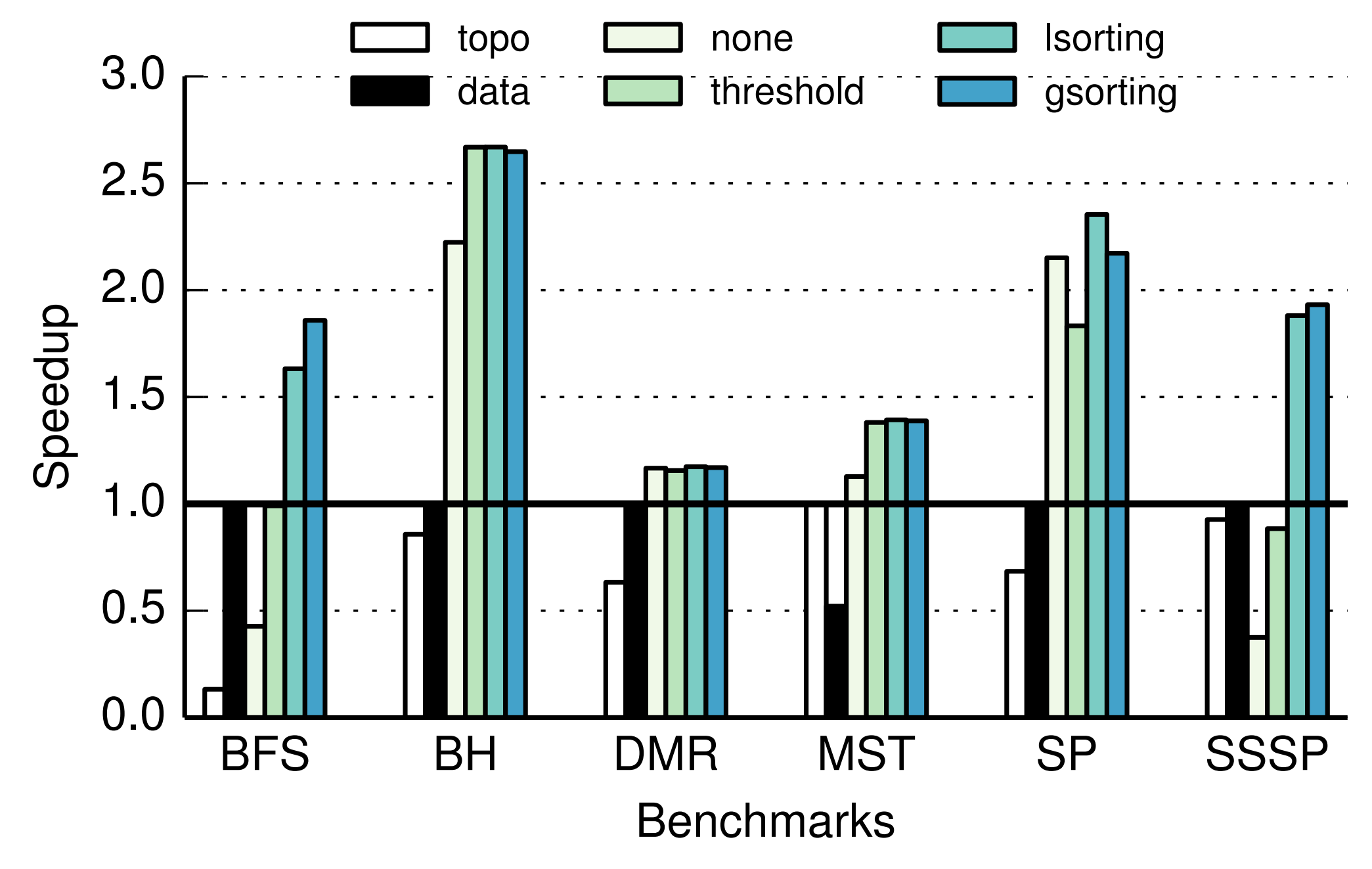
Inter-Core Redistribution Network

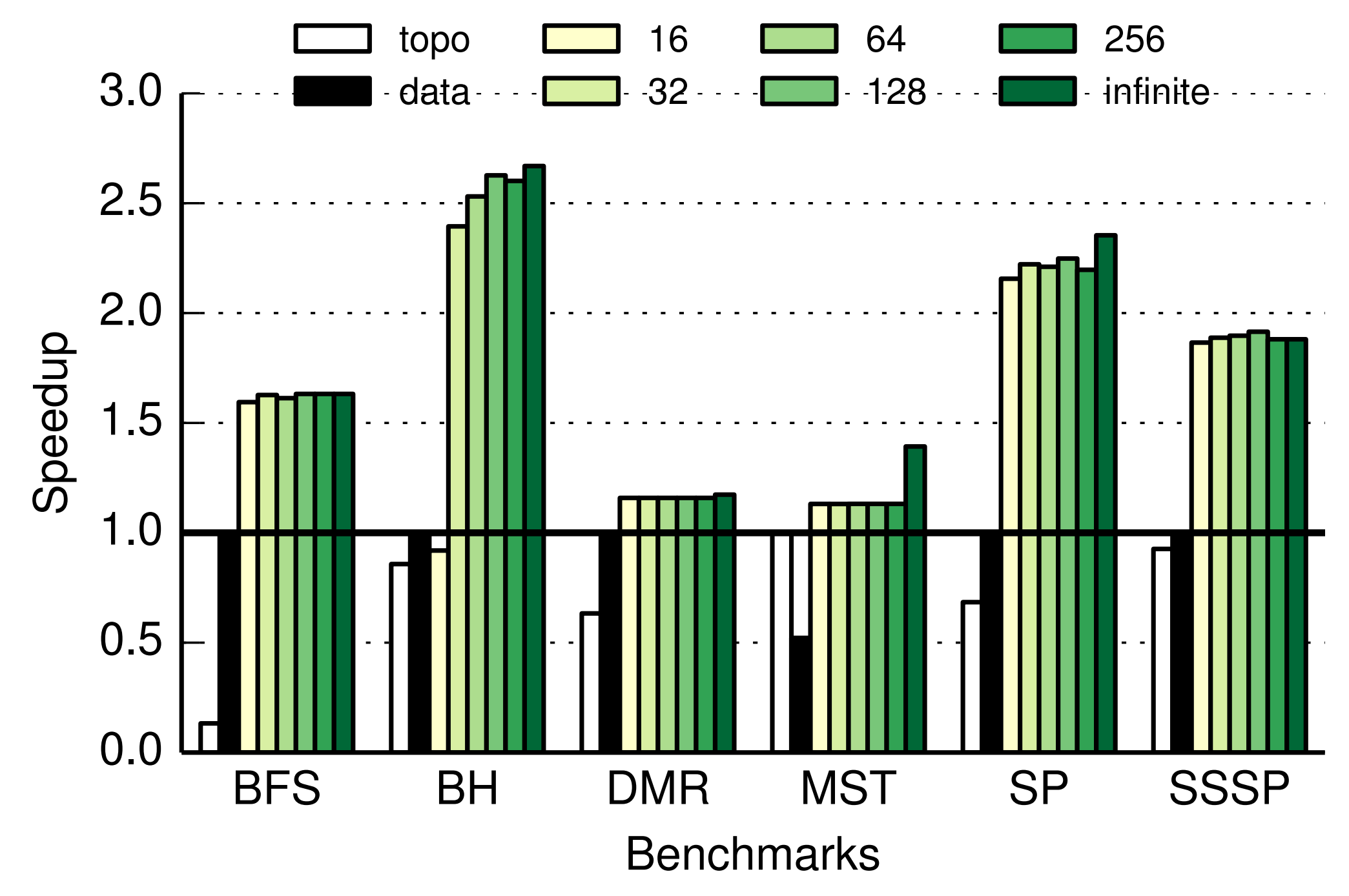
- ▷ Specialized network with tree topology takes two hops from source to destination.
- ▷ Hub arbiter determines which cores donate and receive based per-core metadata.
- ▷ Metadata describes the ratio of banks with excess work to banks with lacking work.



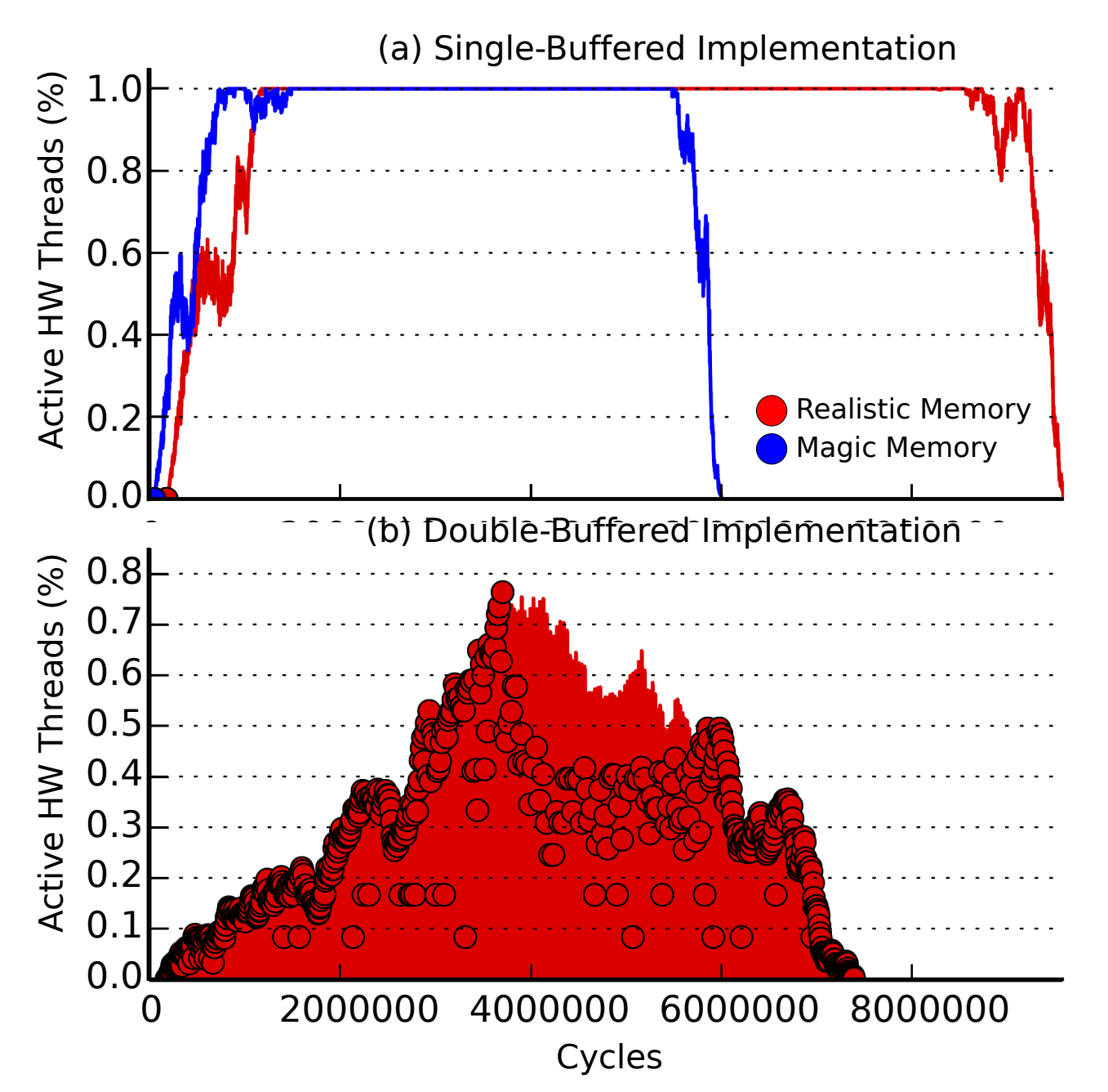
5 Evaluation

We used GPGPU-Sim 3.0 with four cores (16 lanes each) and a FIFO-based DRAM model. We compared the performance of highly optimized topology- and data-driven implementations of irregular algorithms from the LonestarGPU benchmark suite running on a nominal GPGPU to double-buffered data-driven implementations using fine-grain hardware worklists. All results are normalized to the best of the two software baselines.





Single-Buffer vs. Double-Buffer with Hardware Worklists



- ▷ Experiment on Breadth-First Search.
- ▷ Single-buffered data-driven implementation achieves higher resource utilization than double-buffered variant by overlapping super-steps together.
- ▷ Performance degradation from increased memory-access irregularity and larger cache footprint outweighs the benefits.
- ▷ Using magic memory shows that without these limitations, single-buffer can be viable when using hardware worklists.

6 Acknowledgments

This work was supported in part by NSF CAREER Award #1149464, an NDSEG Fellowship, and donations from Intel Corporation and NVIDIA Corporation. The authors acknowledge and thank the Lonestar Benchmark Suite team for their insightful advice, particularly Martin Burtscher, Rupesh Nasre, and Sreepathi Pai for generously providing additional code and their extensive help in understanding the various im- plementations of LSG benchmarks.

Publication: Appears in the Proceedings of the 47th Int'l Symp. on Microarchitecture (MICRO-47), Dec. 2014. **URL:** `http://www.csl.cornell.edu/~cbatten/pdfs/kim-gpgpu-hwwl-micro2014.pdf` **Contact Author:** Ji Kim, 364 Upson Hall, Ithaca, NY 14853, `jyk46@cornell.edu`