Dynamic Fine-Grain Scheduling of Pipeline Parallelism

Daniel Sanchez, David Lo, Jeremy Sugerman, Richard Yoo, Christos Kozyrakis

Stanford University Pervasive Parallelism Laboratory

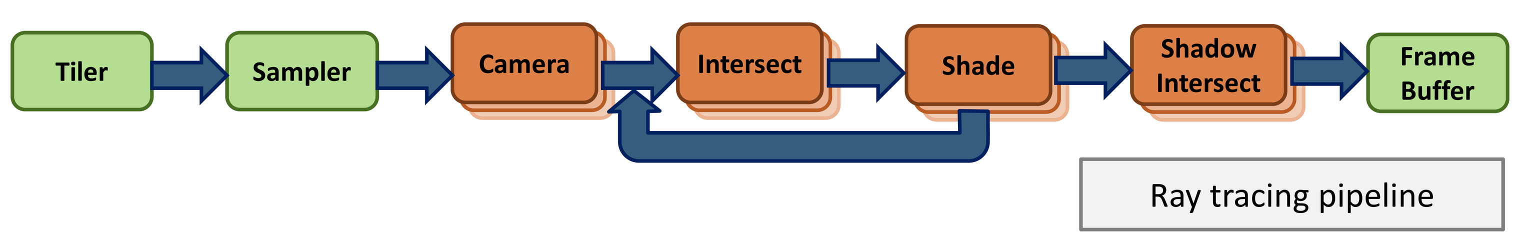
Overview

- Pipeline-parallel applications are hard to schedule
  - Existing techniques either ignore pipeline parallelism, cannot handle its dependences, or suffer from load imbalance
- Contributions:
  - Design a GRAMPS runtime that dynamically schedules pipeline-parallel applications efficiently
  - Show it outperforms typical scheduling techniques from multicore, GPGPU and Streaming programming models

PACT 2011. *"Dynamic Fine-Grain Scheduling of Pipeline Parallelism"*

Pipeline-Parallel Applications

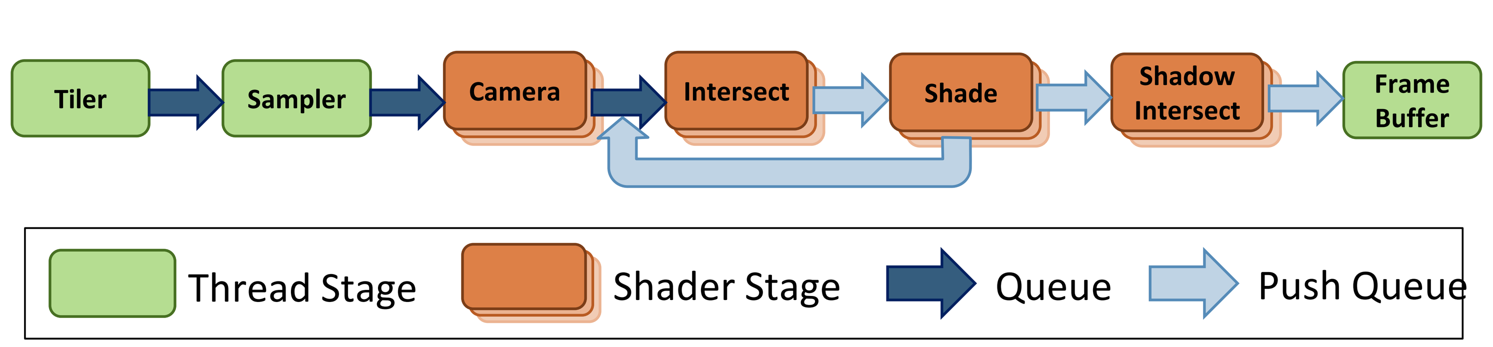
- Some models (e.g. streaming) define applications as a graph of stages that communicate explicitly through queues
  - Each stage can be sequential or data-parallel
  - Arbitrary graphs allowed (multiple inputs/outputs, loops)



- ✓ Well suited to many algorithms
- ✓ Producer-consumer communication is explicit → Easier to exploit to improve locality
- × Traditional scheduling techniques have issues dynamically scheduling pipeline-parallel applications

GRAMPS Programming Model

- Designed for dynamic scheduling of irregular pipeline-parallel workloads
- Two types of stages: Shader (data-parallel) and Thread (sequential)
- Thread stages are stateful, instanced by the programmer
  - Arbitrary number of input and output queues
  - Blocks on empty input/full output queue
  - Can be preempted by the scheduler
- Shader stages are stateless, automatically instanced
  - Single input queue, one or more outputs
  - Each instance processes an input packet
  - Does not block
- Stages send packets through fixed-size data queues
  - Queues can be ordered or unordered
  - Can enqueue full packets or push elements (coalesced by runtime)



Other Popular Programming Models

- Task-Stealing (Cilk, TBB / multicore)
  - Sea of "tasks"; task queues with work-stealing
  - ✓ Low overhead with fine granularity tasks
  - × No producer-consumer locality or aggregation
- Breadth-First (CUDA, OpenCL / GPGPU)
  - Pipeline / DAG of "kernels"; data-parallel shaders
  - ✓ Simple scheduler
  - × No producer-consumer, no pipeline parallelism
- Static (StreamIt / Streaming)
  - Graph of "stages" and "streams"; offline scheduling
  - ✓ No runtime scheduler overheads; complex schedules
  - × Cannot adapt to irregular workloads

GRAMPS Runtime Overview

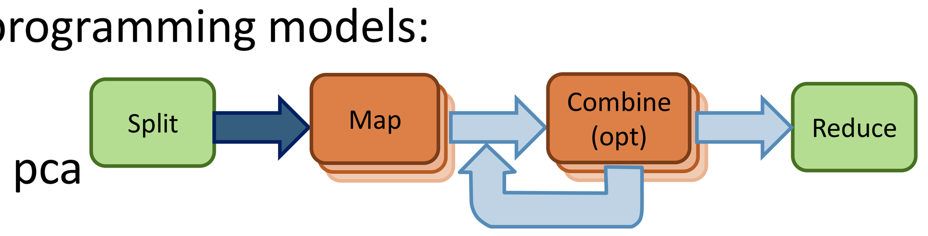
- Runtime = Scheduler + Buffer Manager
- Scheduler: Decide what tasks to run on where
  - Dynamic, low-overhead, keeps bounded footprint
  - Based on task-stealing with multiple task queues/thread
- Buffer Manager: Provide dynamic allocation of packets
  - Generic memory allocators are too slow for communication-intensive applications
  - Low-overhead solution, based on packet-stealing
- Backpressure: When a data queue fills up, disable dequeues and steals from queue producers
  - Producers remain stalled until packets are consumed, workers shift to other stages

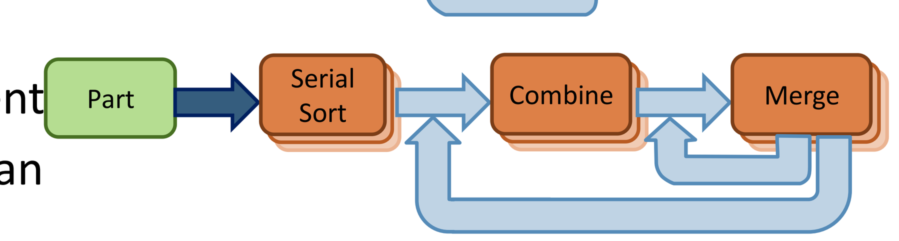
Alternative Schedulers

- GRAMPS scheduler can be substituted with other implementations to compare scheduling approaches
- Task-Stealing: Single LIFO task queue per thread, no backpressure
- Breadth-First: One stage at a time, may do multiple passes due to loops, no backpressure
- Static: Application is profiled first, then partitioned using METIS, and scheduled using a min-latency schedule, using per-thread data queues

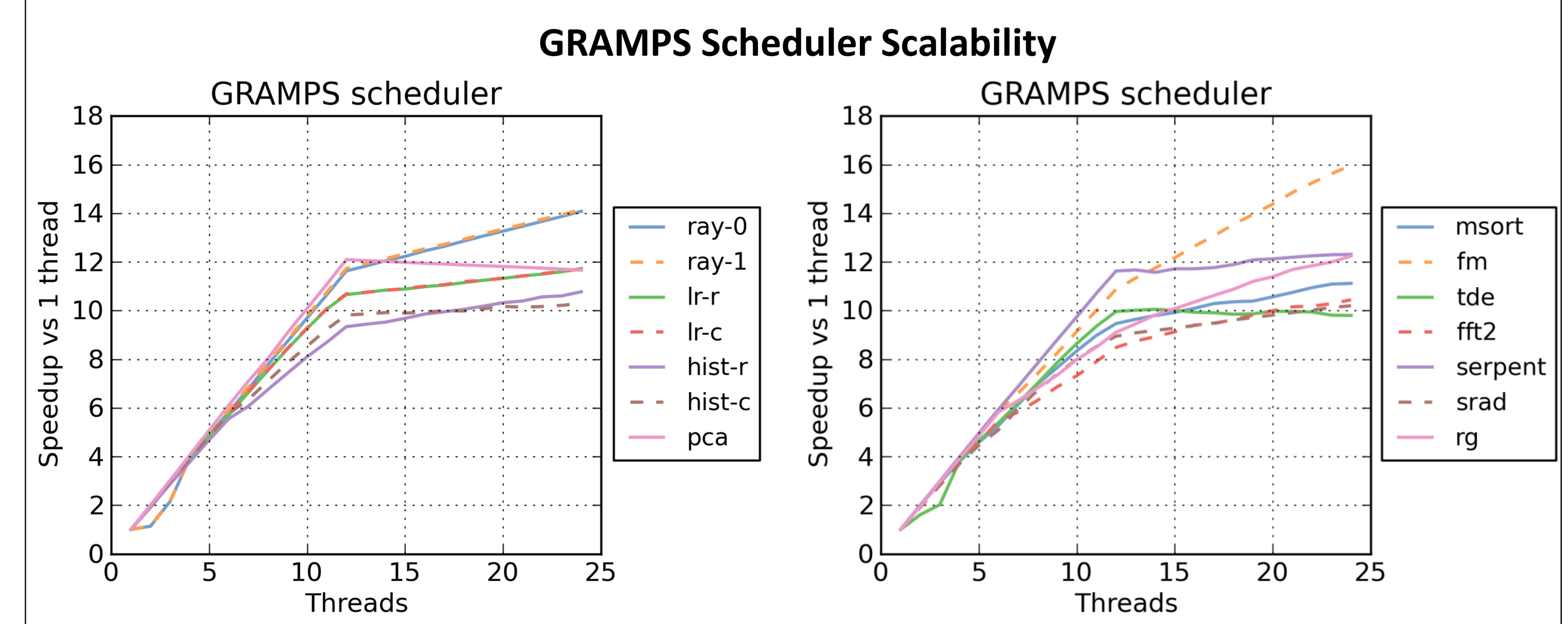
Methodology

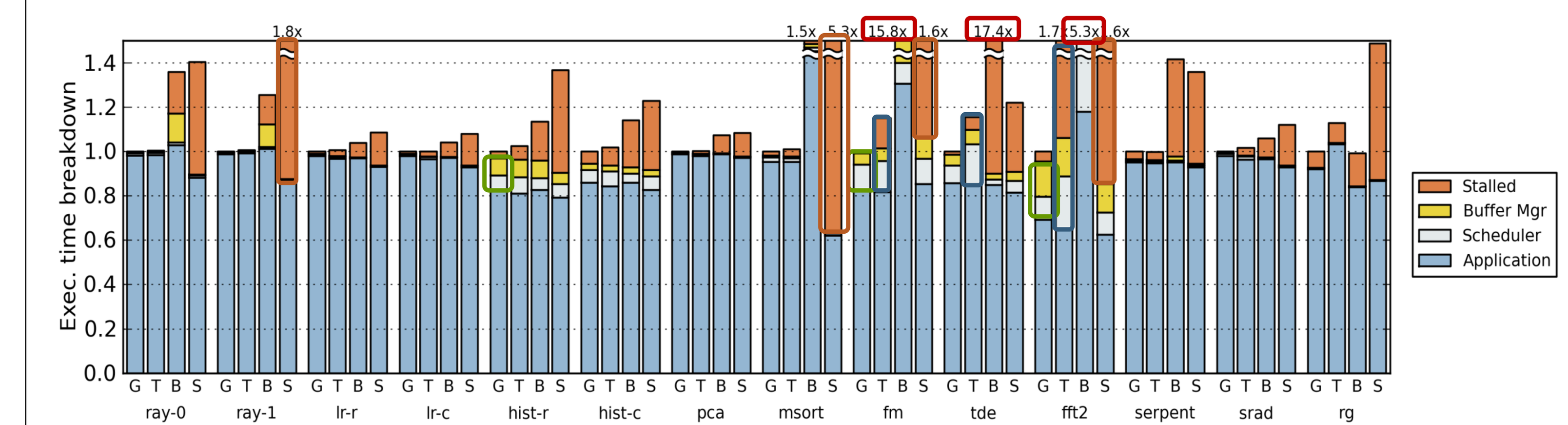
- Test system: 2-socket, 12-core, 24-thread Westmere
  - CPU: 32KB L1I+D, 256KB private L2, 12MB per-socket L3
  - Memory: 48GB 1333MHz DDR3 memory, 21GB/s peak BW
- Benchmarks from different programming models:
  - GRAMPS: raytracer
  - MapReduce: histogram, lr, pca
  - Cilk: mergesort
  - StreamIt: fm, tde, fft2, serpent
  - CUDA: srad, recursiveGaussian



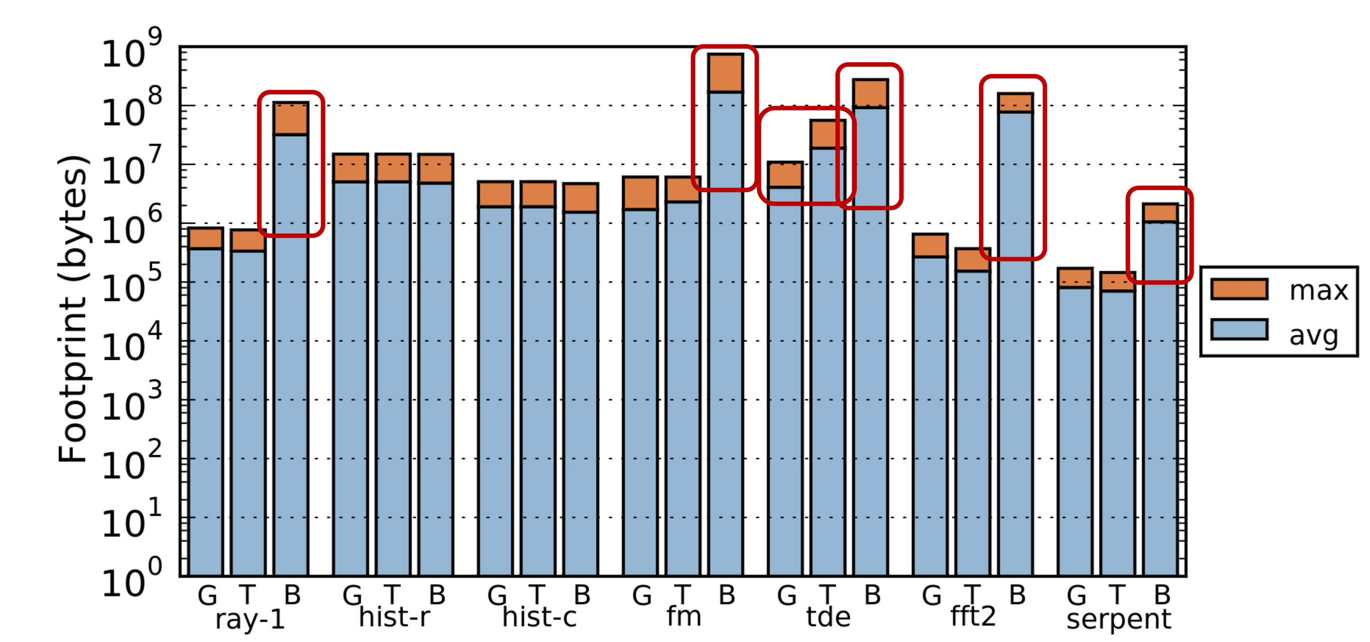


Evaluation





- Dynamic runtime overheads are small in GRAMPS
- Task-Stealing performs worse on complex graphs (fm, tde, fft2)
- Breadth-First does poorly when parallelism comes from pipelining
- Static has no overheads and better locality, but higher stalled time due to load imbalance



- Task-Stealing fails to keep footprint bounded (tde)
- Breadth-First has worst-case footprints → much higher footprint, memory bandwidth requirements

Conclusions

- Traditional scheduling techniques have problems with pipeline-parallel applications
  - Task-Stealing: fails on complex graphs , ordered queues
  - Breadth-First: no pipeline overlap, terrible footprints
  - Static: load imbalance with any irregularity
- GRAMPS runtime performs dynamic fine-grain scheduling of pipeline-parallel applications efficiently
  - Low scheduler and buffer manager overheads
  - Good locality