GRAMPS: A Programming Model for Heterogenous, Commodity, Many-Core Systems

Jeremy Sugerman (with Kayvon Fatahalian, Solomon Boulos, David Lo, Daniel Sanchez, Richard Yoo, Kurt Akeley, Christos Kozyrakis, Pat Hanrahan)

Motivation

Hardware:

- Core counts are rising: scale-out is coming to rival scale-up.
- Heterogeneity is increasing: applications are adopting CPU *and* GPU / data-parallel regions.
- ➢ Programming parallel & heterogeneous is **hard**.

(Also, multi-platform/configuration is important)

Software:

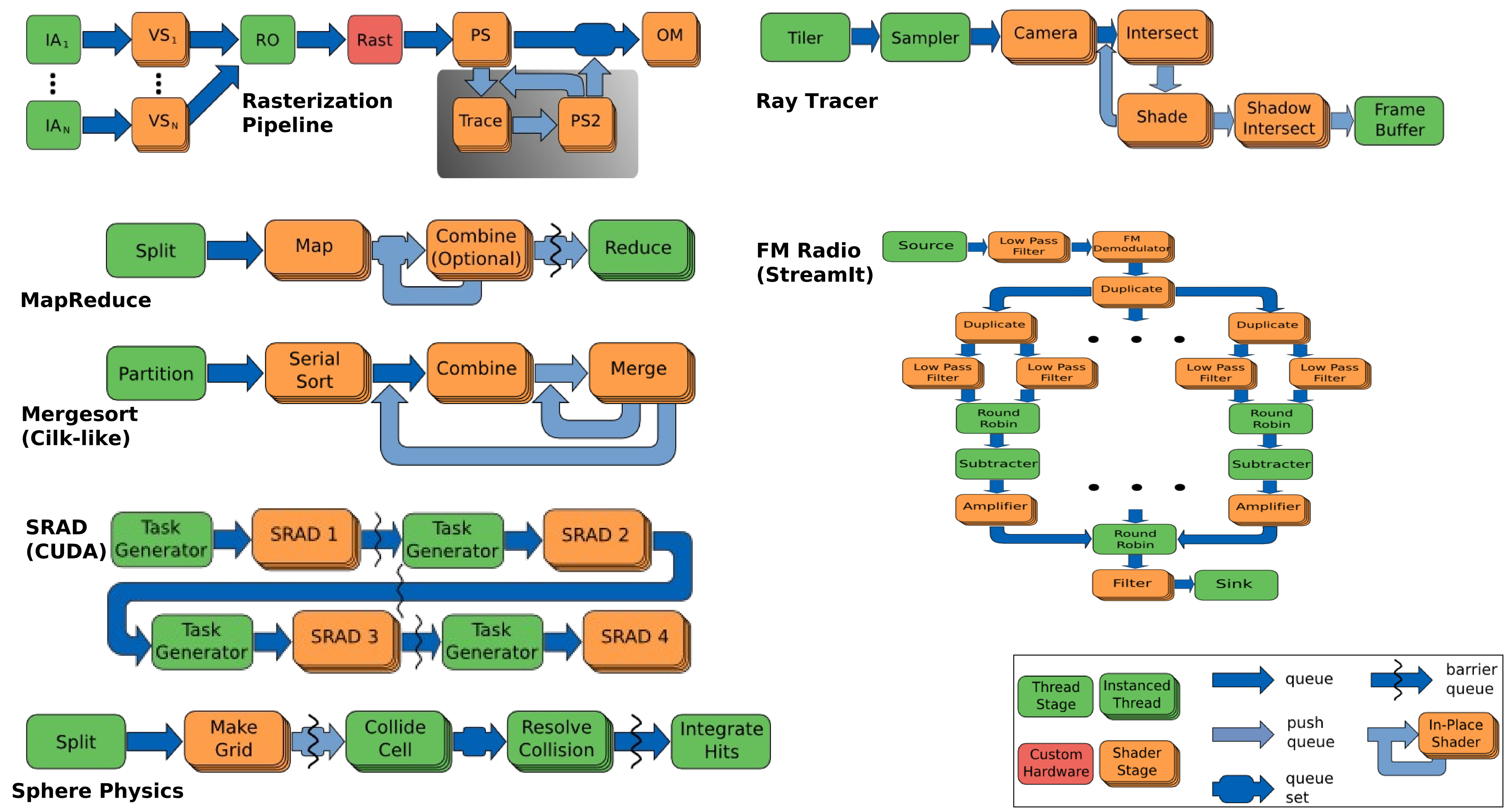
- Coherence matters: processing groupings of coherent 'work' is efficient.
- Irregularity matters: interesting applications are data-dependent and/or adaptive.
- Producer-consumer matters: interesting applications generate intermediate 'work'.
- ➢ Identify and exploit coherence at **run-time**.

(Also, codify and offload best practices)

The GRAMPS Programming Model

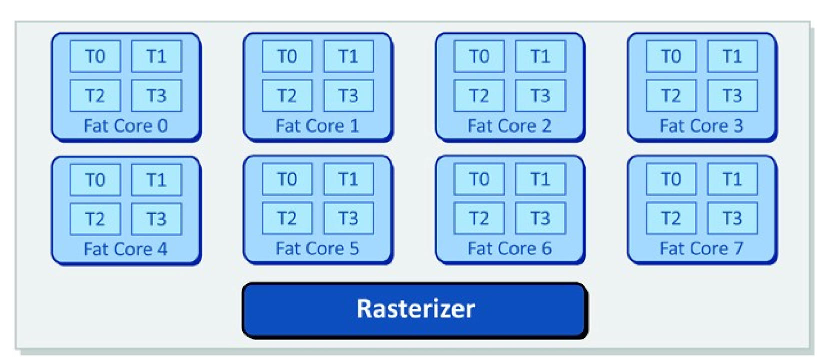
- Applications are graphs (or pipelines):
  - Independent stages connected via queues
- **Thread** stages:
  - Task-parallel, potentially stateful
  - Singleton or automatically instanced
  - Explicit `GrReserve`/`GrCommit` on queues
  - Potentially implemented in custom hardware
- **Shader** stages:
  - Data-parallel, independent stateless instances
  - Automatically instanced
  - Automatic pre-reserve/post-commit of input and fixed outputs
  - Run-time coalesced `GrPush` for variable / conditional output.
- **Queue Sets**:
  - A single logical queue with independently indexed subqueues
  - Parallelism with mutual exclusion: sequential per-subqueue, but many subqueues at once
  - Examples: Screen-space subdivision, per-key reductions in MapReduce

Example GRAMPS Applications

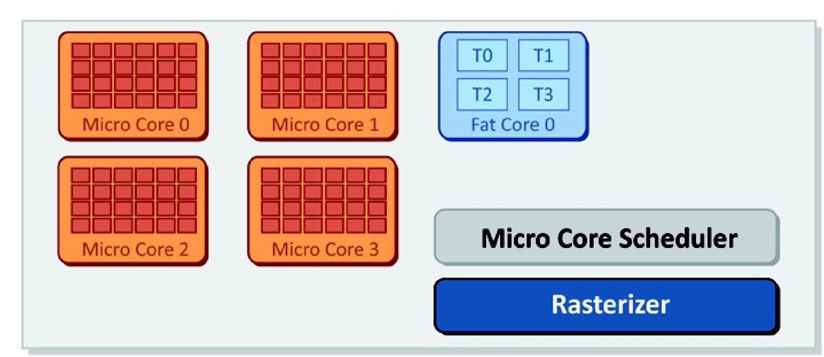


Example GRAMPS Run-Times / Hardware Configurations

Two simulated future rendering platforms:

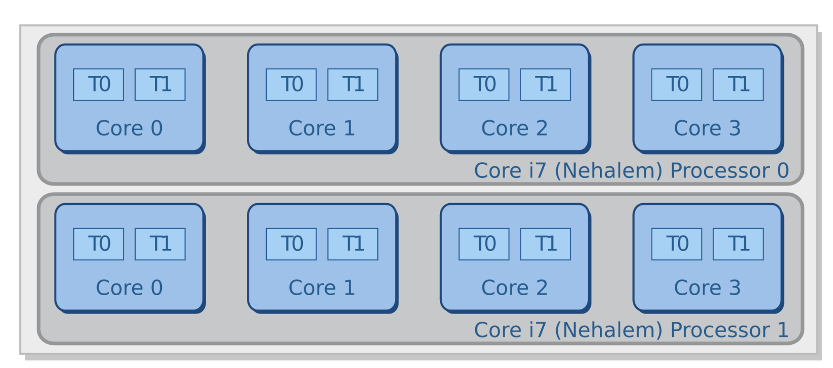


CPU-Like: 8 Fat Cores, Rast



GPU-Like: 1 Fat Core, 4 Micro Cores, Rast, Sched

One current (x86) general purpose platform:



Native: 2 Quad-Core Core i7's

Results and Analysis

Scheduling Mantra: "Maintain **high machine utilization** while keeping **working sets small**":

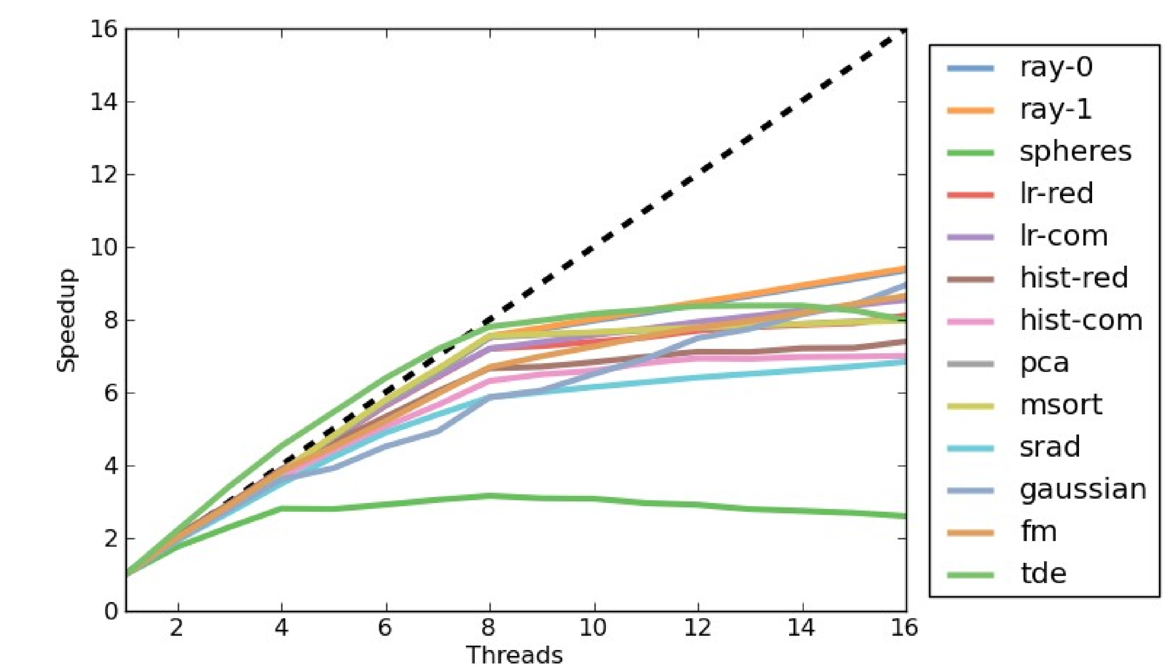
Simple proves effective:

- App-specified queue capacities
- Static stage priorities
- Limited preemption points

Study 1: Rendering (**CPU-Like**, **GPU-Like**):

- 3 scenes x { Rasterization, Ray Tracer, Hybrid }
- 95+% Utilization for all but fairy-rast (~80%).
- Small queues (working sets):
  - < 600KB **CPU-like**, < 1.5MB **GPU-like**

Study 2: General Purpose (**Native**):



- Plenty of parallelism, good scalability
- Working sets are no worse (often better) than task-stealing
- Minimal scheduling overheads

References

1. Sugerman J., Fatahalian K., Boulos S., Akeley K., and Hanrahan P. "GRAMPS: A Programming Model for Graphics Pipelines", ACM TOG, January 2009
2. Kozyrakis C., Lo D., Sanchez D., Sugerman J., Yoo R., "Comparing Parallel Programming Models using GRAMPS", submitted for publication, 2010