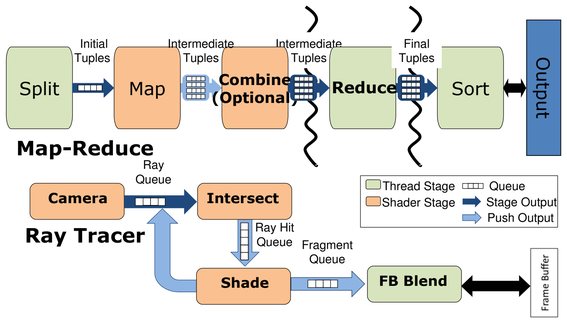
Extending GRAMPS Shaders

Jeremy Sugerman
PPL Retreat, 5 June 2009

GRAMPS

- Reminder: Run-time for producer-consumer parallelism based on stages and queues.

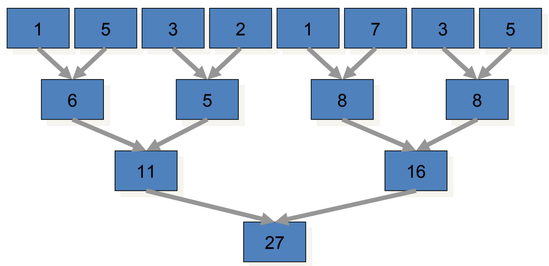


- "Shader" stage is the primary tool for data-parallelism. It is effectively "Map".
- Benefits: auto-instancing, auto queue mgmt, implicit parallelism model, 'shader cores'
- Constraints: 1 input queue, 1 input element per instance.

Reductions

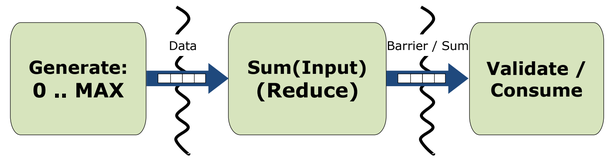
- Central to Map-Reduce (duh), data-parallel apps
- Strict form: sequentially and unlimited buffering
  - E.g., find median, depth order transparency
- Associativity, commutativity enable parallel, incremental reductions.
  - In practice, **many** of the reductions actually used (all Brook / GPGPU, most Map-Reduce)

Logarithic Parallel Reduction



Reductions in GRAMPS

(Currently) Requires Thread stages:



Strict Reduction:

```
sumThreadMain(GrEnv *env) {
   sum = 0;
   /* Block for entire input */
   GrReserve(inputQ, -1);
   for (i = 0 to numPackets) {
      sum += input[i];
   }
   GrCommit(inputQ, numPackets);

   /* Write sum to buffer or output */
}
```

Incremental / Partial Reduction:

```
sumThreadMain(GrEnv *env) {
   sum = 0;
   /* Consume one packet at a time */
   while (GrReserve(inputQ, 1) != NOMORE){
      sum += input[0];
      GrCommit(inputQ, 1);
   }

   /* Write sum to buffer or output */
}
```

Note: Even the incremental version is still single threaded! (Until / unless manually parallelized)

Enabling GRAMPS Shaders for Partial Reductions

- Appeal:
  - Stream and GPU languages allow reduce
  - Important to utilize shader cores
  - Remove / simplify programmer boiler plate
  - Enable auto parallelism and instancing
- Obstacles
  - Where to store partial / running result
  - Handling multiple inputs (spanning packets)
  - Detecting / signaling termination
  - Avoiding a proliferation of stage types

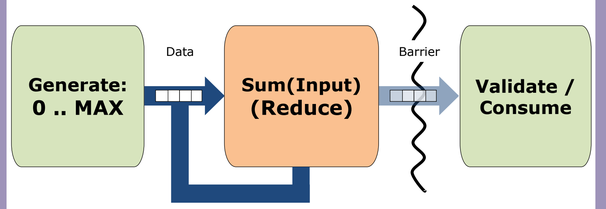
Shader Environment Enhancements:

- Stage / kernel takes N inputs at a time (Must handle < N available)
- Invocation reduces N : 1 (Stored as an output key?)
- GRAMPS can (will) merge input across packets
  - → No guarantees on per packet shared headers!

➢ **Not** a completely new type, not just GPGPU reduce

Reduction Using Extended Shaders

Sum() is now a Shader stage:



- An N:1 shader and a graph cycle reduce in place, in parallel.
- 'Barrier' queue only gets NOMORE (when Generate is done and Sum has reduced to a single item).

Scheduling Reduction Shaders

- Highly correlated with execution graph cycles
  - Scheduling heuristic for such cycles: pre-empt upstream to contain working set
- Free space in the loopback caps parallelism
  - Maximal parallelism requires 1/Nth free.
  - A single free space is sufficient to guarantee forward progress

➢ Note: Logarithmic versus linear reduction has become application transparent and entirely a GRAMPS run-time / scheduler decision.

Concluding, Other Thoughts

- Data-parallel reductions are an important case.
- Nice to include them by generalizing existing Shader versus adding a distinct Reduce stage.
- In addition to reductions, non-recursive filtering is also now possible. Remains to be seen how interesting that is.
- "Next" data-parallel operations after map and reduce? Are there more that run well on commodity many-core hardware?