Doing More With GRAMPS

Jeremy Sugerman

10 December 2009

GCafe

Introduction

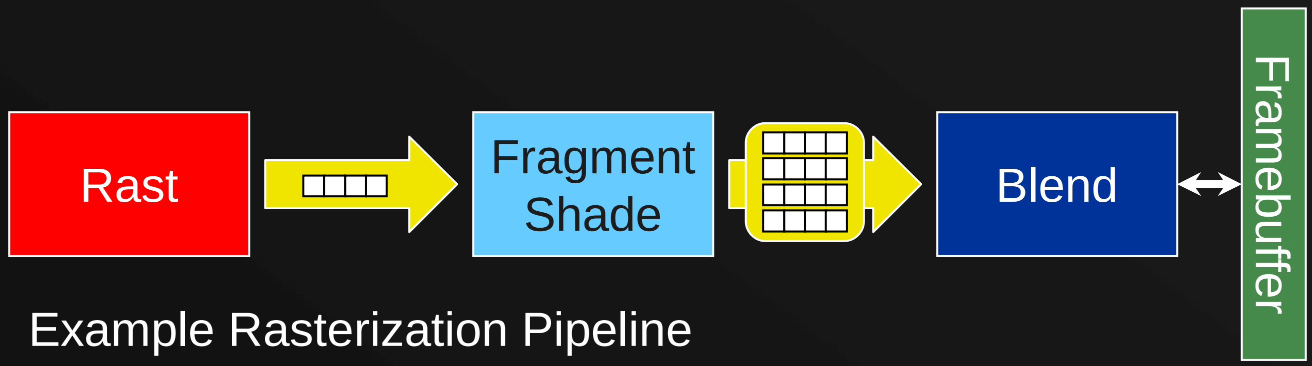
- Past: GRAMPS for building renderers
- This Talk: GRAMPS in two new domains: map-reduce and rigid body physics

GRAMPS Review (1)

- Programming model / API / run-time for heterogeneous many-core machines
- Applications are:
  - Graphs of multiple stages (cycles allowed)
  - Connected via queues
- Interesting workloads are irregular

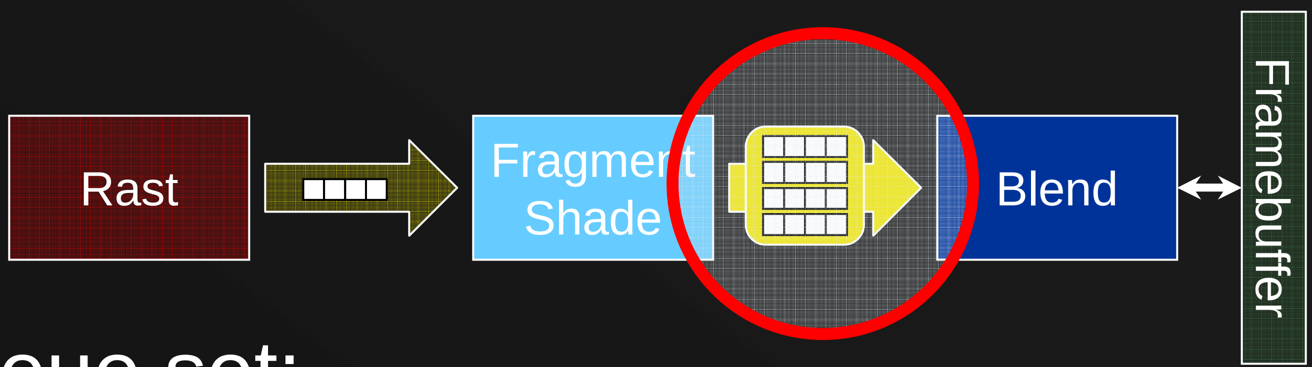
GRAMPS Review (2)

- Shaders: data-parallel, plus push
- Threads/Fixed-function: stateful / tasks



Example Rasterization Pipeline

GRAMPS Review (3)



- Queue set:
  - single logical queue, independent subqueues
- Synchronization **and** parallel consumption
- Binning, screen-space subdivision, etc.

Map-Reduce

- Popular parallel idiom:

Map:

```
Foreach(input) {
   Do something
   Emit(key, &val)
}
```

Reduce:

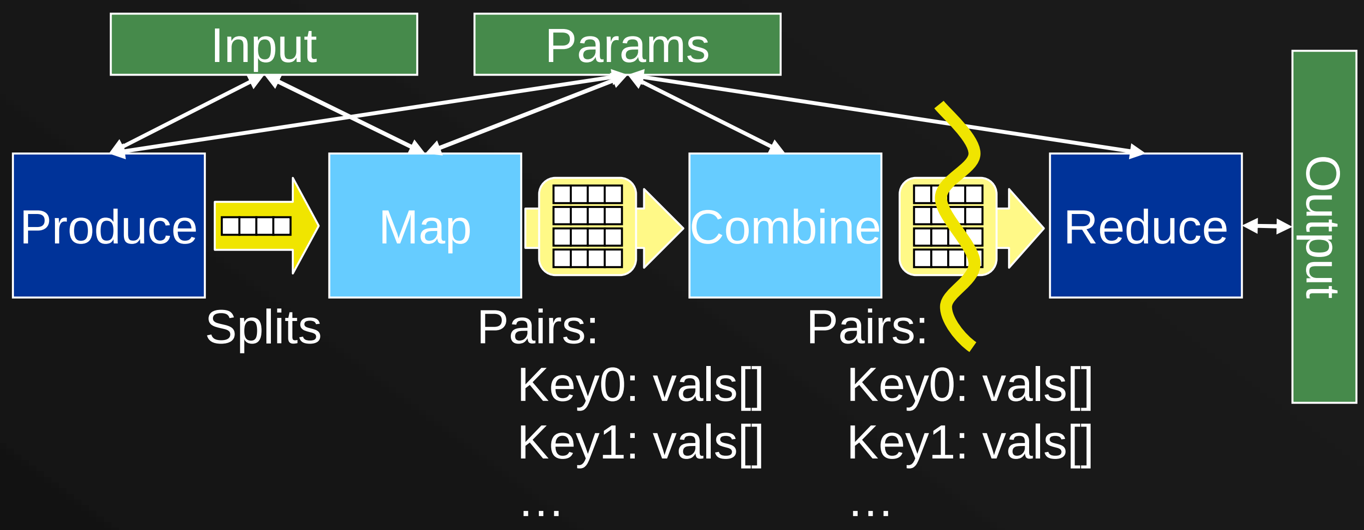
```
Foreach(key) {
   Process values
   EmitFinalResult()
}
```

- Used at both cluster and multi-core scale
- Analytics, indexing, machine learning, …

Map-Reduce: Combine

- Reduce often has high overhead:
  - Buffering of intermediate pairs (storage, stall)
  - Load imbalance across keys
  - Serialization within a key
- In practice, Reduce is often associative and commutative (and simple).
- **Combine** phase enables *incremental*, *parallel* reduction

Map-Reduce in GRAMPS



- A few API extensions
- A few new optimizations

Extension: Queue Sets

- Make queue sets more dynamic
  - Create subqueues on demand
  - Sparsely indexed ‘keyed’ subqueues
  - ANY_SUBQUEUE flag for Reserve

Make-Grid(obj):

```
For (cells in o.bbox) {
  key = linearize(cell)
  PushKey(out, key, &o)
}
```

Collide():

```
While (Reserve(input, ANY))
  For (each o1, o2 pair)
    if (o1 overlaps o2)
      ...
```

Extension: Instanced Threads

- Automatic instancing of thread stages
  - One to one with input subqueues
  - Only when processing is independent

Make-Grid(obj):

```
For (cells in o.bbox) {
  key = linearize(cell)
  PushKey(out, key, &o)
}
```

Collide(subqueue):

```
For (each o1, o2 pair)
  if (o1 overlaps o2)
    ...
```

Extension: Fan-in Shaders

- Enable shader parallel partial reductions
  - Input: One packet, Output: One element
  - Can operate in-place or as a filter
  - Run-time coalesces mostly empty packets

Histogram(pixels):

```
For (i < pixels.numEl){
  c = .3r + .6g + .1b
  PushKey(out,c/256,1)
}
```

Sum(packet):

```
For (i < packet.numEl)
  sum += packet.v[i]
packet.v[0] = sum
packet.numEl = 1
```

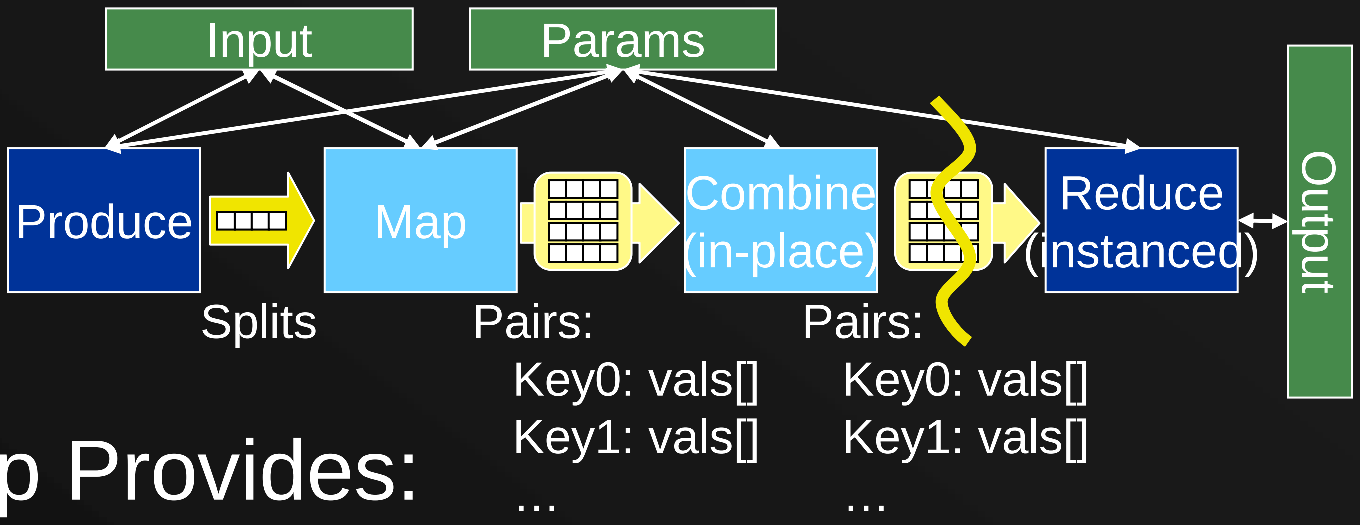
Digression: Combine is a builtin

- Alternatives:
  - Regular shader accumulating with atomics
  - GPGPU multi-pass shader reduction
  - Manually replicated thread stages
  - Fan-in with same queue as input and output
- Reality: Messy, micro-managed, slow
  - Run-time should *hide* complexity, not export it

Optimizations

- Aggressive shader instancing
- Per-subqueue push coalescing
- Per-core scoreboard

GRAMPS Map-Reduce



- App Provides:
  - Produce, Guts of: map, combine, reduce
- Run-time Provides:
  - GRAMPS bindings, elems per packet

GRAMPS Map-Reduce Apps

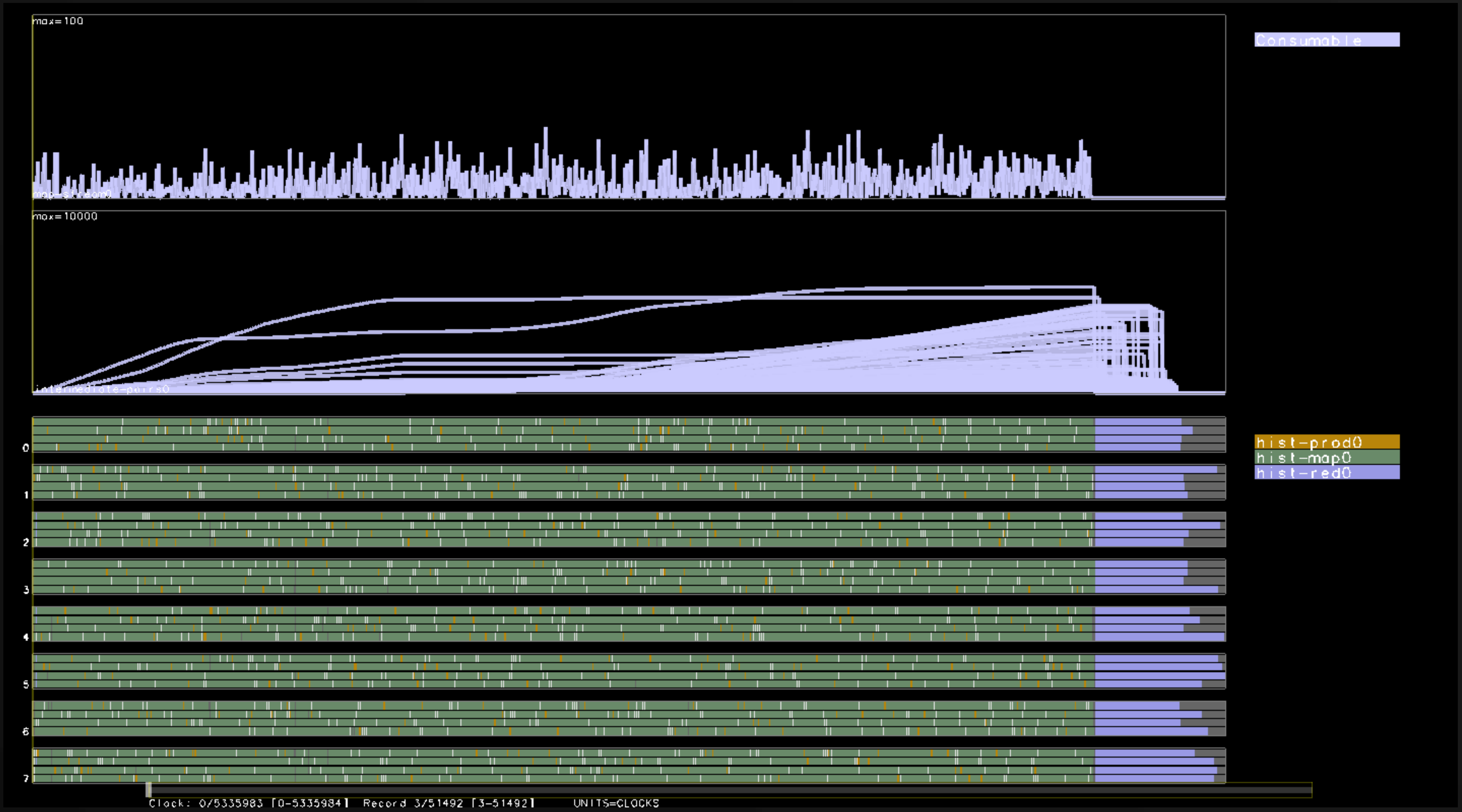
Based on Phoenix map-reduce apps:

- Histogram: Quantize input image into 256 buckets
- Linear Regression: For a set of (x,y) pairs, compute average x, x^2, y, y^2, and xy
- PCA: For a matrix M, compute the mean of each row and the covariance of all pairs of rows

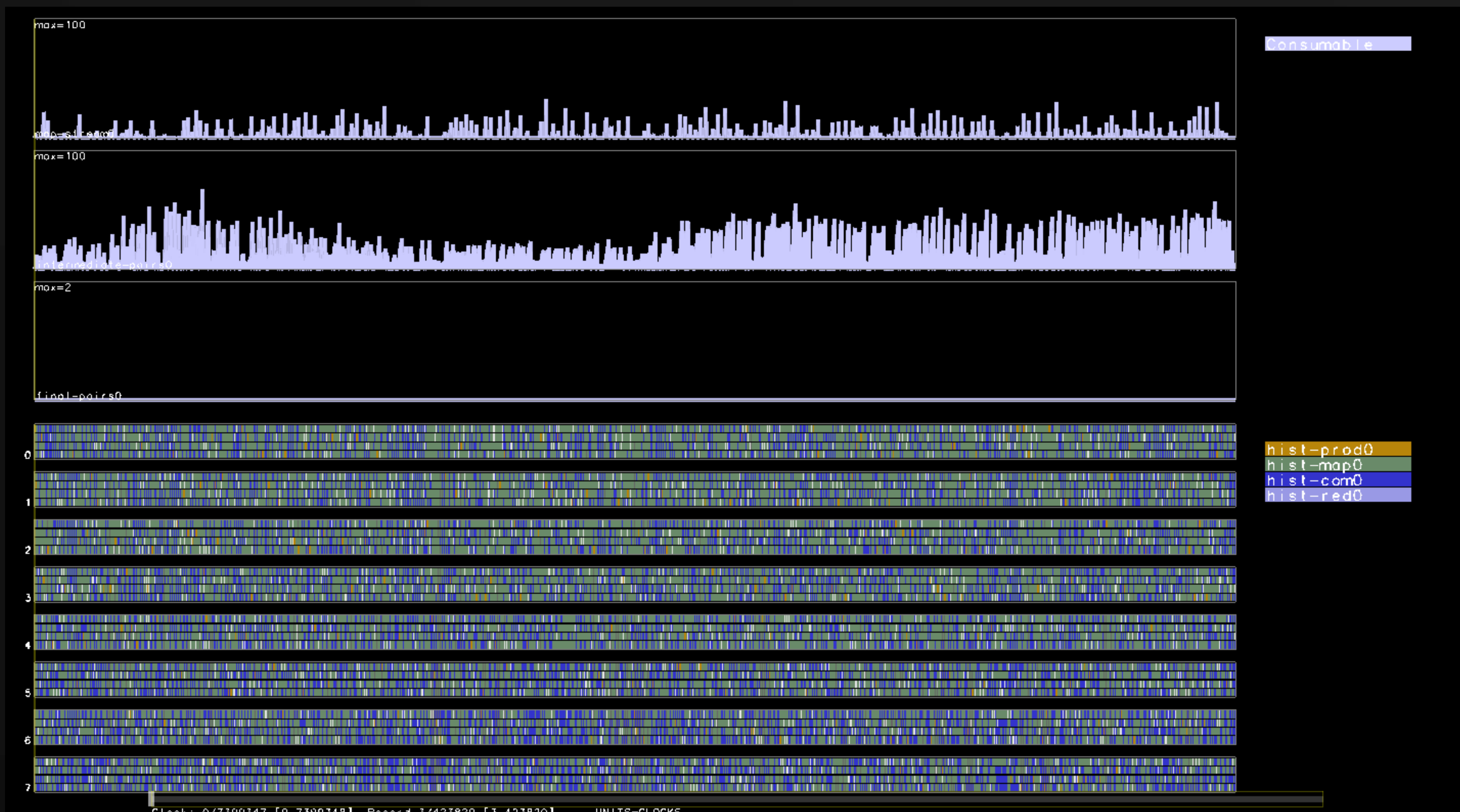
Map-Reduce App Results

	Occupancy (CPU-Like)	Footprint (Avg.)	Footprint (Peak)
Histogram-512	97.2%	2300 KB	4700 KB
(combine)	96.2%	10 KB	20 KB
LR-32768	65.5%	100 KB	205 KB
(combine)	97.0%	1 KB	1.5 KB
PCA-128	99.2%	.5 KB	1 KB

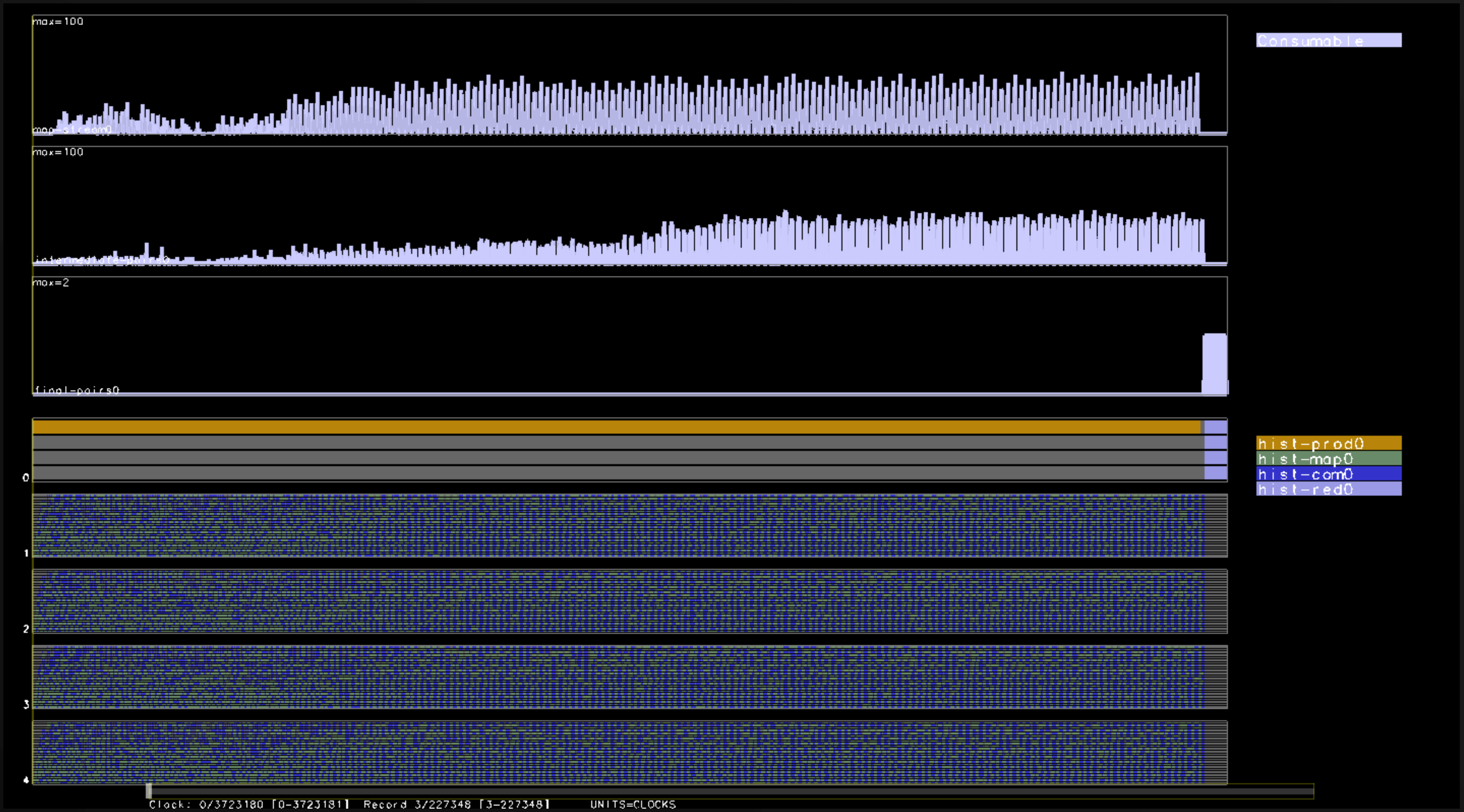
Histogram 512x512



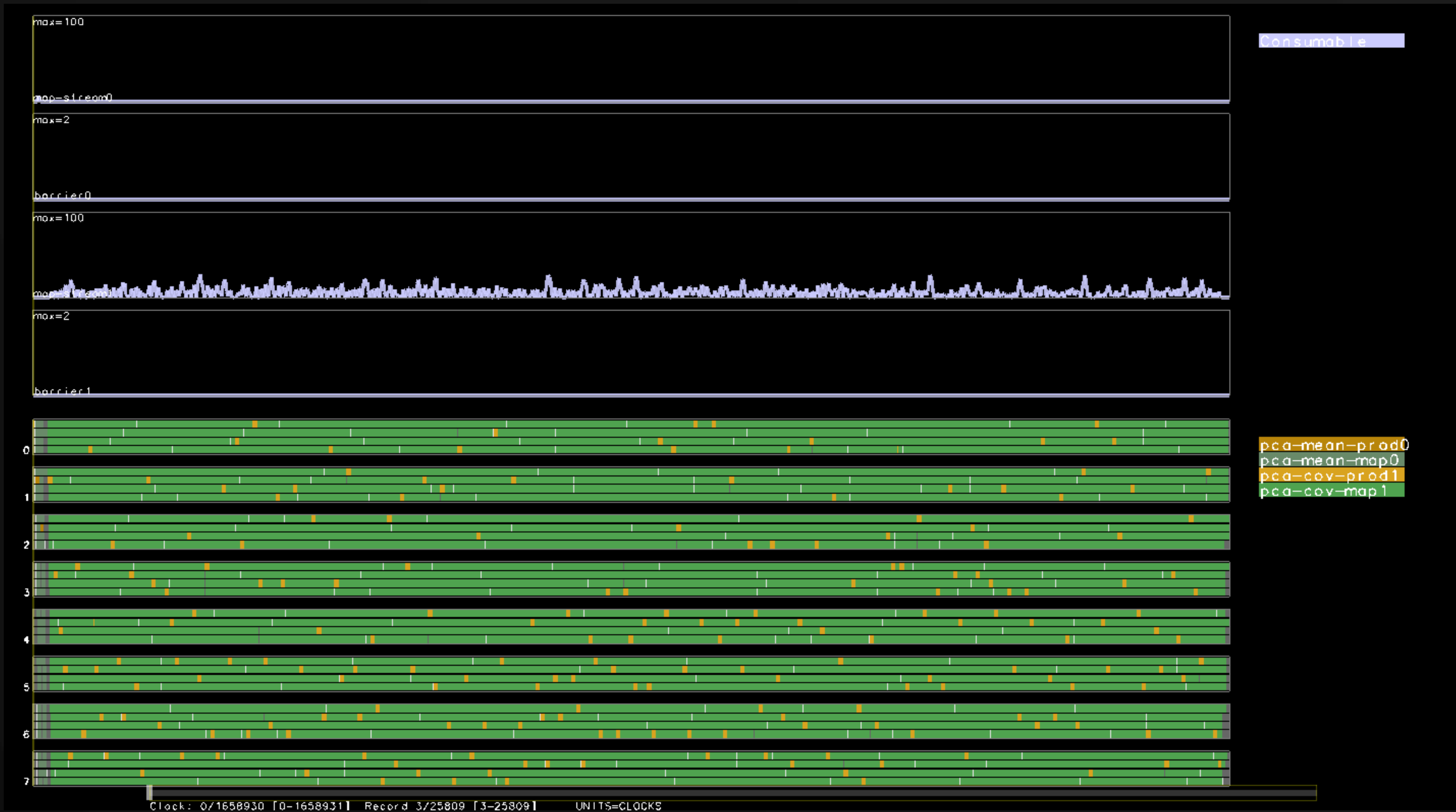
Histogram 512x512 (Combine)



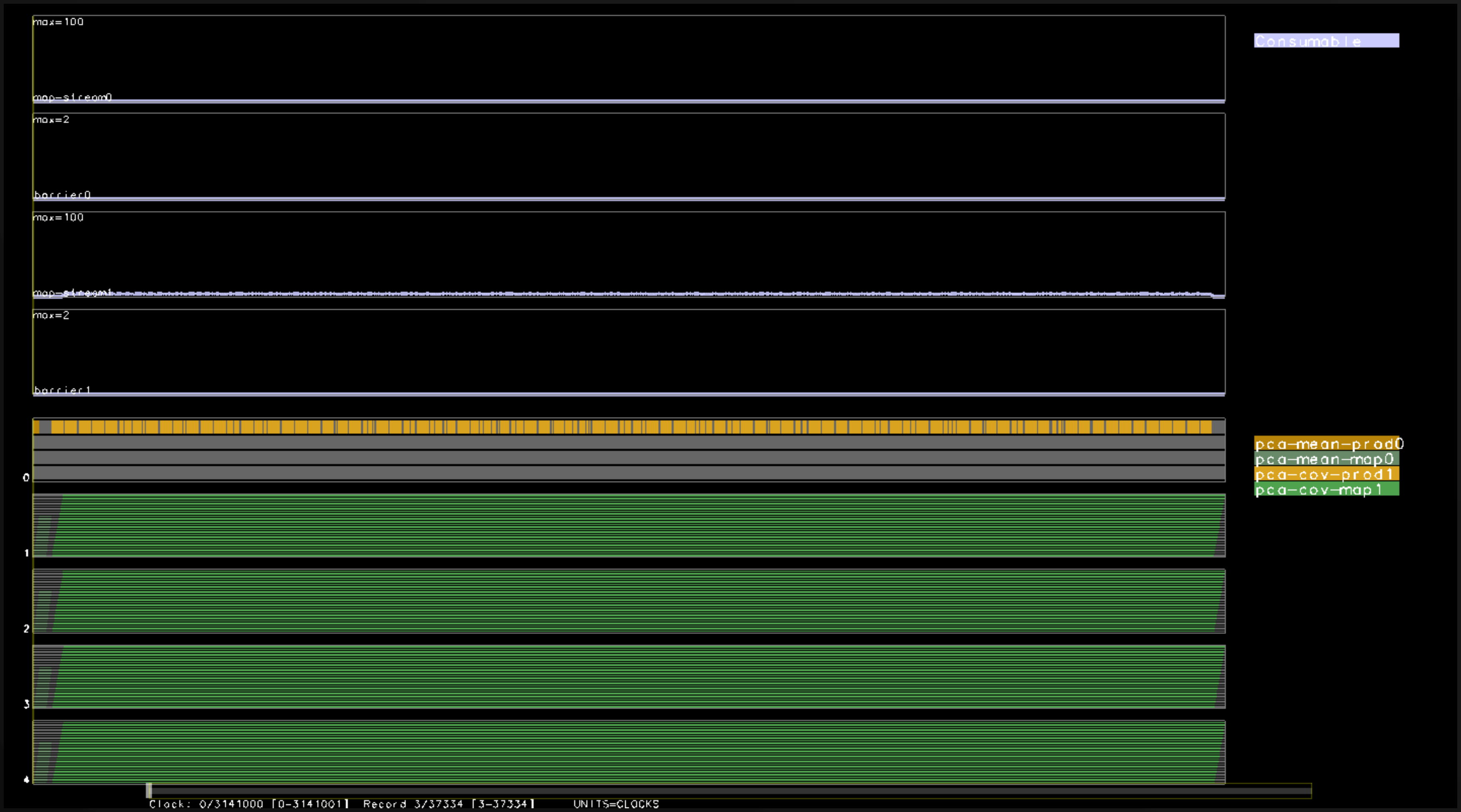
Histogram 512x512 (GPU)



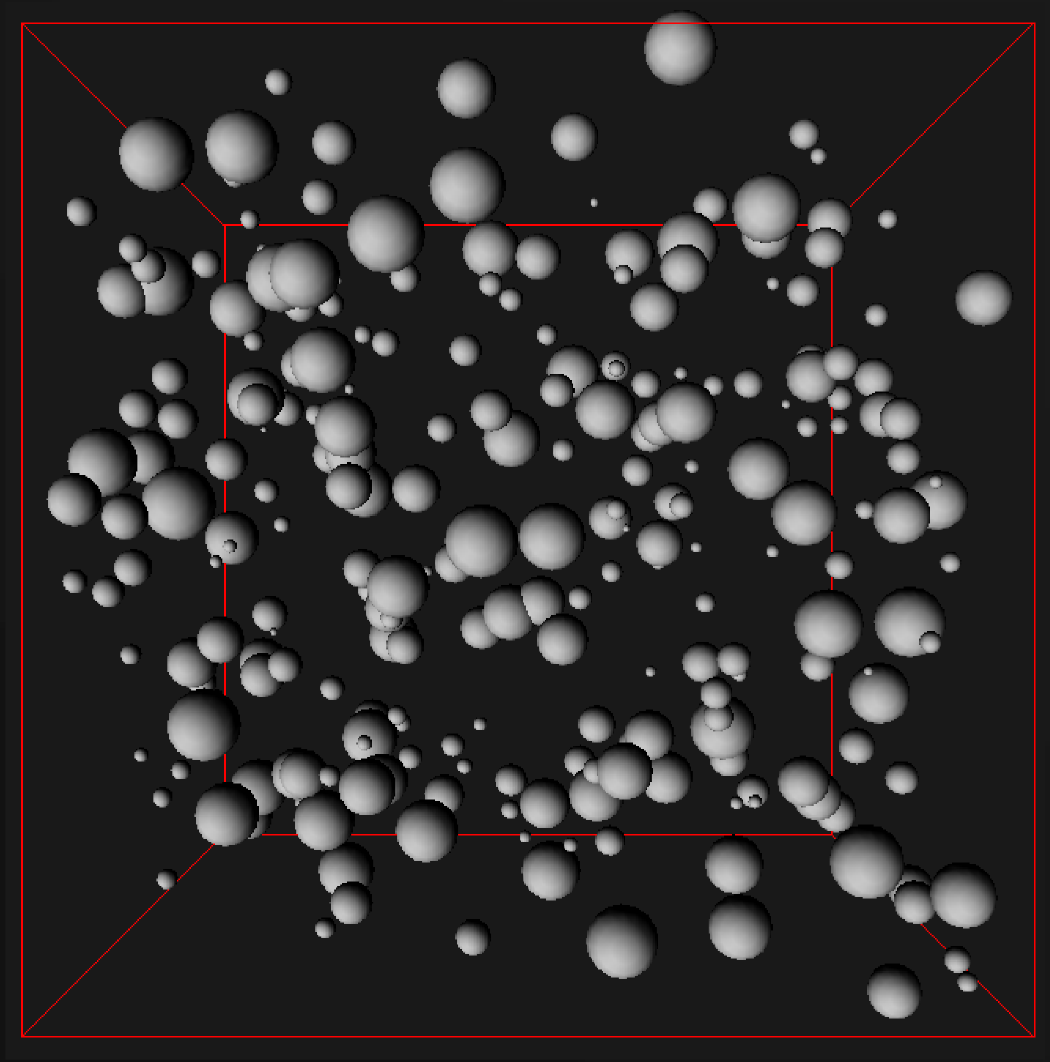
PCA 128x128 (CPU)



PCA 128x128 (GPU)



Sphere Physics

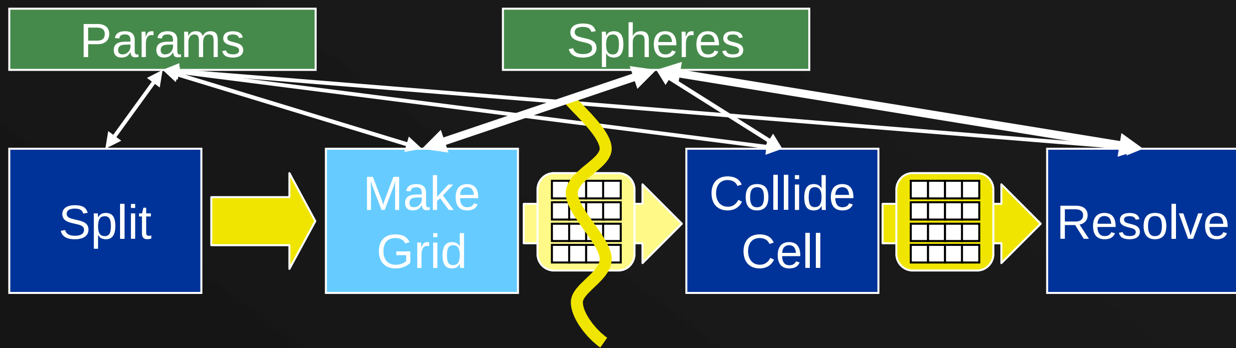


Sphere Physics

A (simplified) proxy for rigid body physics:

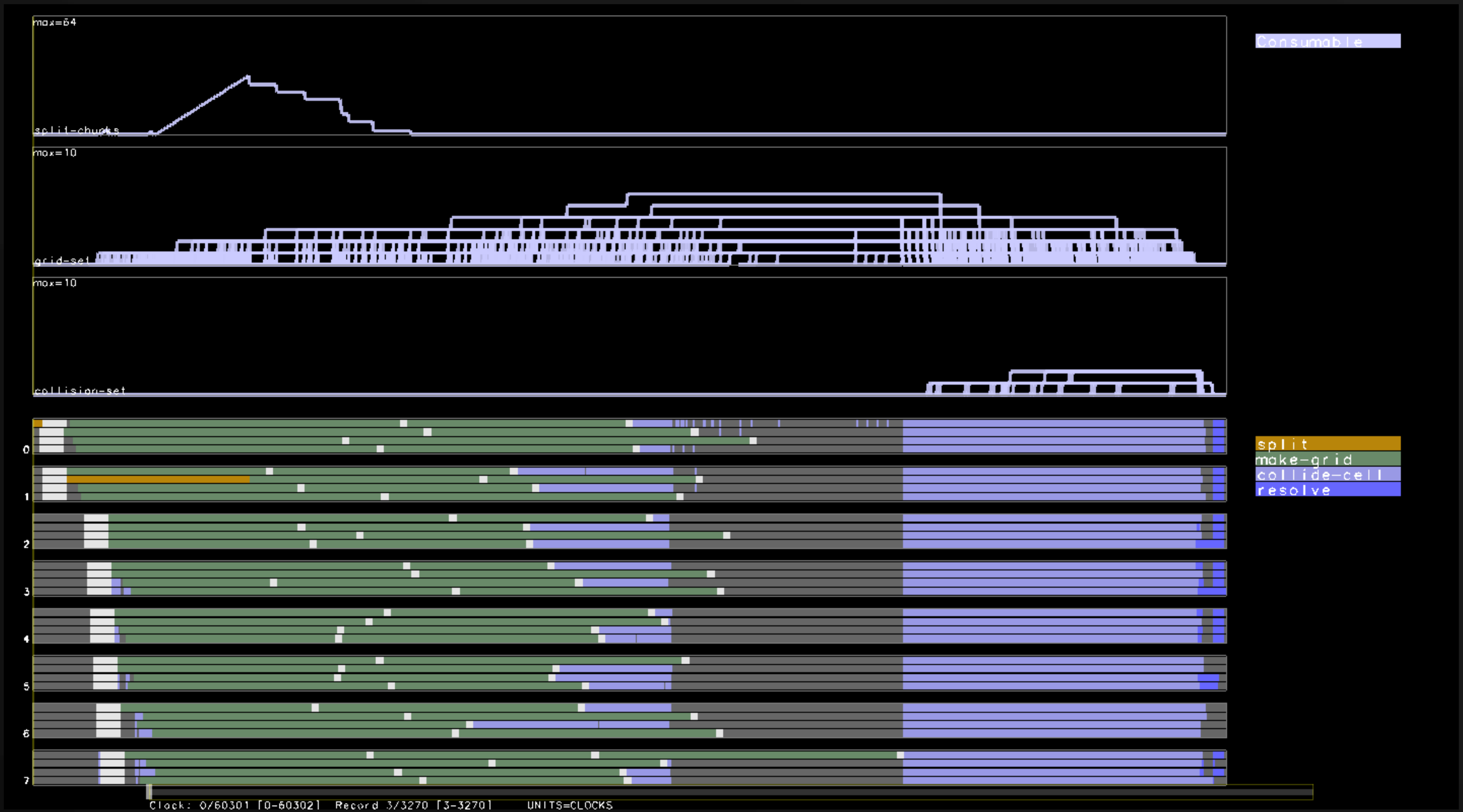
```
Generate N spheres, initial velocity
while(true) {
    • Find all pairs of intersecting
      spheres
    • Compute Δv to resolve collision
      (conserve energy, momentum)
    • Compute updated result velocity and
      position
}
```

GRAMPS: Sphere Physics



1. Split Spheres into chunks of N
2. Emit(cell, sphereNum) for each sphere
3. Emit(s1, s2) for each intersection in cell
4. For each sphere, resolve and update

CPU-Like: 256 Spheres

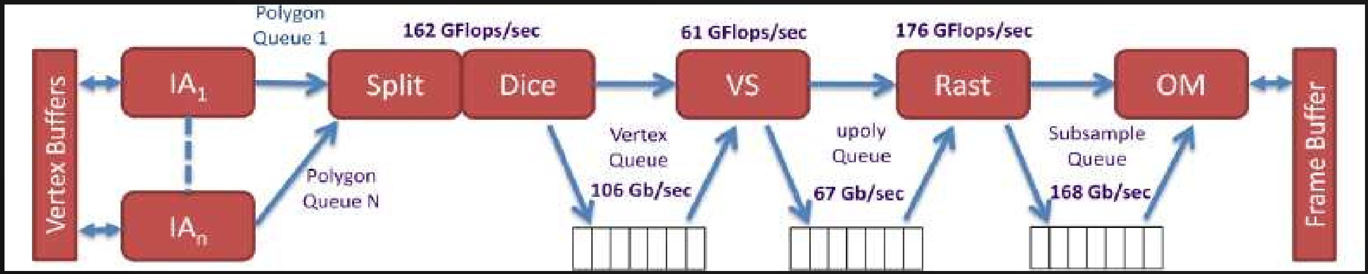


Future Work

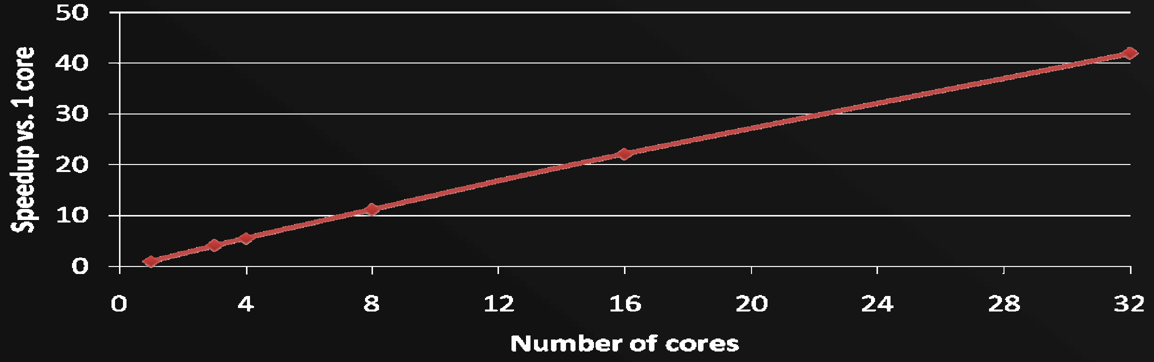
- Tuning:
  - Push, combine coalesce efficiency
  - Map-Reduce chunk sizes for split, reduce
- Extensions to enable more shader usage in Sphere Physics?
- Flesh out how/where to apply application enhancements, optimizations

Other People's Work

- Improved sim: model ILP and caches
- Micropolygon rasterization, fixed functions



- x86 many-core:



Thank You

- Questions?