Chunking the 3D pixel sim

Brett Viren

November 18, 2024

Topics

- Review the pipeline for 3D pixel sim.
- Consideration for introducing **chunking** the pipeline and data.
- Grids defining the problem domains.
- Problems chunking causes and initial solutions at each pipeline stage.

The 3D pixel sim pipeline, chunking and grids

Will consider **chunking** these data:

diffused result of "rastering" a drifted step's charge to a **fine grid**.

currents per-impulse-partition (**fine-grid**) and summed per-pixel (**pixel-grid**) current waveforms.

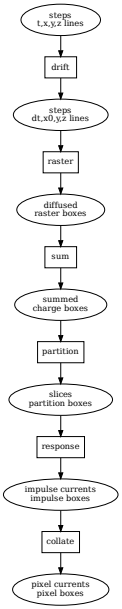
Define intermediate coarse **super-grid**

- Multiple of and aligned with **fine-grid**.
- Aligned to pixel edges, coarser than **pixel-grid**.

Problem decouples into **dense** many operations on smaller 3D chunks.

- Name them "xxx boxes", characterize each and define operations.

Here I give some first conceptual steps, coding likely comes next.



Conceptual basics of chunking

$$A = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] \rightarrow \{A_i\} = C(A) = [(0, 1), (2, 3), (4, 5), (6, 7), (8, 9)]$$

$$F(A) \rightarrow U(F(C(A)))$$

- Apply a **chunking** operation $C(\cdot)$ to get small chunks of large data.
- Map the original operator $F(\cdot)$ over the set of chunks.
- Apply an "unchunking" operator $U(\cdot)$ to "stitch together" the set of processed chunks.

Problems that chunking brings

Overlap

The chunked outputs from $F(\{A_i\})$ can overlap with each other.

- Eg, two independent steps may accidentally "cross" in the post-drift space.
- Outputs of $F(\cdot)$ are larger than inputs and will "grow" overlaps.
- Easy to apply overlaps to neighbors, harder to do it fast/parallel.

Alignment

Unchunking operator $U(\cdot)$ must collate processed chunks across the full domain (in general).

- Allowing arbitrary offsets for a chunk collating harder, especially for parallel.
- We should "pre align" outputs of $F(\cdot)$ in anticipation of downstream unchunking problem.

Stage: raster of diffused charge

Conceptually, a locally-dense grid of charge from a post-drift, diffuse step.

- 2 approaches to charge integral: Gauss-Hermite quadrature or FFT+interpolation.

Given G4 maximum step size, max drift time, diffusion coef and 3D grid definition:

- We can define a maximum box on the grid to contain any rastered step.
- Enlarge this box to **align** to a coarse **super-grid** (eg aligned to pixel edges, larger ticks).
- Require the raster stage to output its data on this super-grid $\rightarrow$ produce a **raster box**.

This is a 1-to-many mapping, some rastered steps will span multiple raster boxes.

We may then apply something similar to PyTorch's `scatter_add()`

- Serial sum over all raster boxes on a common super-grid point,
- Parallellize these sums over the super-grid.

Gives a set of "aligned, summed raster boxes" call it **charge box**.

Stage: partitioning

Each **charge box** is split into 10×10 interleaved partitions.

- Here, rely on the **super-grid** to be defined as an integer multiple of the pixel grid.
- Call this a **partition box** - which is also a chunk.

Stage: Apply response

Each **partition box** chunk is convolved with the FR.

- Output enlarged by size of FR kernel and edges no longer aligned to super-grid.
- Overlaps will form with neighbors on each of 8 sides/edges/corners.

I envision a **contention-free** overlap-sum algorithm.

- Requires $8 \times 3 \times N_{chunk}$ operations. N_{chunk} parallel, 24 serial.
- Requires overlaps to be less than $\frac{1}{2}$ the size of the chunk.

After overlap-sum, we get back to chunks defined on the super-pixel grid.

- Call one an **impact box**.

Stage: Final sum over pixel

A many-to-many sum mapping **impact boxes** to **pixel boxes**.

- A **pixel box** is a region of a pixel vs tick grid - something in between fine-grid and super-grid.
- Each impact box contributes to a set of pixels over some fixed time duration.
- Many impact boxes contribute to the same pixel and time duration.

I've not yet thought carefully how to parallelized this....