Timing Review

TRES REID, PETER WITTICH [CORNELL]
BEI WANG [PRINCETON]
PHILIP CHANG, BALAJI VENKAT SATHIA NARAYANAN, SLAVA KRUTELYOV, AVI YAGIL [UCSD]
The purpose of these slides is to restate how our timing is done both as a reminder of how it works and some known issues and for new members.

We might want to rethink how timing is done in the future.

When running timing, use \(-v\) 1 and \(-w\) 0 options

- We don’t want time spent on printing multiplicities or adding objects to the ntuple since this is done in a critical block and takes a lot of time. It will throw off timing measurements.
- Running timing with \texttt{sdl\_timing} handles all of this. There will be a final printout as below which averages all the times over the number of events run.
Timing

Object time: Time to create, clean and add each object
- Reported individually for each event for each stream.

Event time: sum of all listed object times

Short time: sum of the object times we care about (everything but hits and pLS)

Loop time: a inverse throughput measurement. Full time to run over all events divided by the number of events.

Effective time: scales the loop time by the fraction of time spent on the part we care about
- \((\text{Short/ Event}) \times \text{Loop} = \text{Effective}\).

Notable trends: Multistreaming increases object times, short times and event times but decreases the loop time. We also see more streams increases the short time more than the event time, so the ratio we care about increases. This might not be a real effect.
Object timing

Object times are obtained through the following process

- Start timer
- Run create object function
  - Allocate object memory
  - Calculation for thread launch and division of work
    - Memory transfers to host
    - CPU processes
  - Launch create object kernel
  - Duplicate removal
  - Add object to TC (pT5 and TC only)
  - Add object to event
    - This is mostly for printing multiplicities
      Can probably be cut from timing except in a few instances where range values are additionally set
- Stop timer
- Print multiplicities

If something else runs in the middle of this, it can affect the reported object time.
Each event takes a relatively small amount of time
  ◦ Here the highlighted region is ~160 ms

There is a large gap between events. (240 ms)
  ◦ Unclear if this is because of the profiling but I would think not

Each event, and therefore each object time, is untouched by any other
Multiple streams

Multiple streams run multiple events at the same time.

For the most part kernel launches themselves largely don’t overlap
- There can be some overlap if a block for one process finishes early. Then, another process can begin with the available block
- Reported kernel times get slightly drawn out

Other streams might run between processes within the object creation function
- Object times definitely gets drawn out

The gap still exists but it is between sets of events
- A set of events runs in longer time
  - Highlighted = 500 ms over 4 events, so each event takes less time than the 1 stream case
  - Gap between sets is smaller, 170 ms compared to 240ms
- Gap for each stream is about the same but other streams run between that gap
Punchline

Multistream works because you are pipelining the processes and reducing the gap between one calculation and another, not so much running multiple kernels in parallel.

Object times get higher because you can interrupt the create object function in one stream with that of another stream. So, each stream takes longer individually, but the two together are still faster than running two events on the same stream individually.
  ◦ Individual object times for multi streams are not reliable, but overall what we care about is really the full throughput
  ◦ The place this gets fuzzy is an overlap with the Hits (and pLS). Potentially, we can start an object we care about, but be interrupted not by another object we care about but by the hit loading. So, it is unclear if the ratios between the short and event times are accurate and thus the effective time might not be accurate.

We do know that multistreaming reduces the loop time, but we might be overestimating our effective time.

Some improvements:
  ◦ Maybe save a pre-cleaned pLS collection.
  ◦ Include hit loading in our final time measurement and adjust the target time appropriately.
  ◦ Use single stream ratio to calculate effective time
    ◦ This would put our multistreaming for explicit cache in the 27-32 ms range.