Profiling of LArSoft code

Gianluca Petrillo

LArSoft stakeholders' meeting, March 12th , 2014

Profiling tools

For CPU profiling:

`Timing` (`art` service): per-event, module-level information

`gperftools` (Google) quick snapshot of where time is spent with full call history

`callgrind` (`valgrind` tool) count of each call and used cycles

For memory profiling:

`SimpleMemoryCheck` (`art` service) mostly useful to detect large memory leaks per-event, module-level information

`massif` (`valgrind` tool) complete *heap* map with allocating functions

`procfs` (Linux) complete memory map (`/proc/PID/maps`) and statistics

Profiling time

Currently, I can run with `e4:prof` code:

- plain run: 100-event samples (thousands should not be a problem)
- statistical CPU speed profiling: few percent overhead respect to plain
- complete call profiling: 5-event chunks (takes 40'/1h each; 10 possible)
- memory profiling: roughly as complete call profiling (it's `valgrind`)
- memory and stack profiling: 3-event chunks (takes longer than just memory)

A 8 GB memory machine would help making this quicker (virtual memory is deadly for me and for the people on my same machine).

Stack profiling has shown to be not necessary (fortunately!).

Currently on air

I have been focusing on μBooNE code, due to the coming MC challenge and retreat:

- profiling the full chain: GEANT simulation, detector digitization, reconstruction
- on top of a “busy” event: cosmic ray plus beam activity

Both the techniques and the code I am learning will be instrumental to the optimization of LBNE as well.

GEANT simulation

Worked on input data generated by
`prodgenie_bnb_nue_cosmic_3window_uboone.fcl`:

- identified the places where most of the time is spent (MicroBooNE configuration)
  - message facility dispatch: fixed (I've seen $\times 2$ speed gain)
- memory allocation mapped:
  - moving from dynamic to static allocation where proper
  - reduce the occurrences of data copy and duplication
  - use advanced allocation to avoid memory usage spikes
- identified (and fixed) a couple of memory leaks

Detector digitization

Worked on input data generated by `prodgenie_bnb_nue_cosmic_3window_uboone.fcl` and `standard_g4_uboone.fcl`.
Ongoing work:

- helped Matt Toups and Kazuhiro Terao to fix a leak they found
- memory allocation mapped
- large “anonymous” memory pages... fragmentation?

Memory footprint is *not only LBNE problem*!
MicroBooNE code is, in the current status, very troublesome in a 2+2 GB memory (physical+virtual) environment.

What's next

In general, I have (or can get) a precise map of what uses memory.
Possible developments:

- investigate memory fragmentation (hints point to that being a major problem)
- reduce memory usage by ROOT (it's a trade off)
- optimize the flow of data to avoid unnecessary copies and duplication
- analyse and optimize the scope of variables
- reengineering of loops

After μBooNE retreat, I'll also get back to LBNE GEANT simulation.

This work would be much more expedite if unit tests were already in place.