RooFit: Performance studies & Gradient parallelization

Patrick Bos
Performance studies (chronological)

Single core
- Profiling
- Caching methods (memoization)

Multi-core
- TensorFlow
- Likelihood fits
- Numerical integrals
- Gradients
Multi-core

- TensorFlow
- Likelihood fits
  - Unbinned
  - Binned
- Numerical integrals
- Gradients
TensorFlow

- Fits on identical model & data (single i7 machine)

<table>
<thead>
<tr>
<th></th>
<th>RooFit (MINUIT)</th>
<th>TensorFlow (BFGS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unbinned fit</td>
<td>0.1s</td>
<td>0.01 - 0.1s (dep. on precision)</td>
</tr>
<tr>
<td>Binned fit</td>
<td>0.7ms</td>
<td>2.3ms (3x)</td>
</tr>
</tbody>
</table>

- TensorFlow: No caching!
  - Major advantage of RooFit for binned fits (e.g. morphing histograms)
  - (feature request for memoization https://github.com/tensorflow/tensorflow/issues/5323)

- N.B.: measured before CPU affinity fixing (next slides)!
  - RooFit now even faster (but limited to running one machine)
Multi-core likelihood fits

RooRealMPFE / BidirMMapPipe

- Custom multi-process message passing protocol
  - POSIX fork, pipe, mmap
- Communication “overhead” (delay between sending and receiving messages): ~ 1e-4 seconds
  - serverLoop waits for message & runs server-side code
  - messages used sparingly
  - data transfer over memory-mapped pipes
Multi-core *unbinned* likelihood fits

**Run-time vs N(cores)**

**Before:** max ~2x

**Now (with CPU affinity fixed):** max ~20x (more for larger fits)

---

**Actual performance**

**Expected performance (ideal parallelization)**
Multi-core *binned* likelihood fits

Run-time vs N(cores) in *binned fits*

- Actual performance
- Expected performance (ideal parallelization)
- CPU time (single core)

Room for Improvement

WIP
Numerical integrals

• “Analytical” integrals

• Forced numerical (Monte Carlo) integrals (Higgs fits didn’t have them)
Numerical integrals

Individual NI timings (variation in runs and iterations)

Maxima

Minima

Sum of slowest integrals/cores per iteration over the entire run

(single core total runtime: 3.2s)
Gradient parallelization

✔ 0th step: get Minuit to use external derivative:
  • Testing class RooGaussMinimizer (Vince & me)

✔ 1st step: replicate Minuit2 behavior
  • NumericalDerivator (Lorenzo)
    • Modified to exactly replicate Minuit2
    • ... almost working
  • RooGradMinimizer

... Next: calculate partial derivative for each parameter in parallel
Single core

- Profiling
- Caching methods (memoization)
Profiling: Callgrind, Cachegrind, Instruments

- **Higgs ggf & 9 channel fits (Lydia)**
- Callgrind/Instruments – time spent on:
  - RooVectorDataStore::get() (4% / 32%)
    - RooNLLVar::evaluatePartition → RooDataSet::get → RooAbsData::get → RooVectorDataStore::get
  - Logarithms: 12% (ggf)
  - RooStats::HistFactory::FlexibleInterpVar (in ggf 10%); interpolation in general costs a lot
  - Dynamic casts (ggf): 1% with optimization level 2 (10% & 13% with opt.levels 0 & 1)
  - RooLinkedList::At took considerable time in Gaussian test fit (Vince); std::vector lookup → 1.6x speedup! WIP
- **Cachegrind – memory access:**
  - 0.3% LL cache misses (expensive!) on RooVectorDataStore::get()
    - Row-wise access pattern on column-wise data store (and std::vector< std::vector >)
  - RooLinkedList::findArg: ~ 5% of instructions
Profiling: meta-conclusions

• profiling functions & classes
  • valgrind
  • gprof
  • Instruments
  • … etc.

• profiling objects (e.g. call-trees, e.g. RooFit…)
  • … DIY?
Caching

• Memoization works very well, no obvious optimization targets

• E.g. cache based on indegree
  (~ common sub-tree elimination)
  – Implicitly included by memoization
    • though explicit disabling of sub-trees may be better
  – Network analysis call-trees Higgs fits
    • ggf: no common sub-trees