RooFit Parallelization

W. Verkerke (NIKHEF)
P. Bos (eScience Center)

Introduction - RooFit

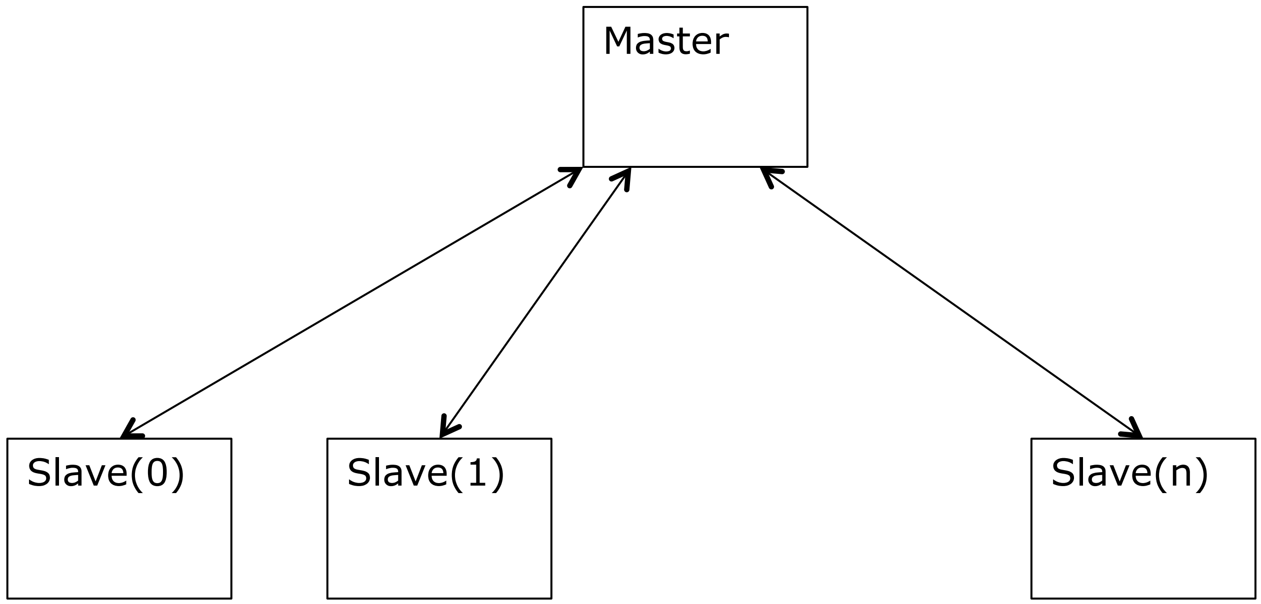
- RooFit is a OO language to write probability models, widely used in HEP.
- User constructs PDF (of arbitrary complexity) as expression tree of RooFit function objects
  - Scales to very complex models (>>10.000 objects for Higgs models)
  - All models provide full functionality for fitting, plotting and toy event generation
- Under the hood, a variety of computational optimizations is applied for potentially CPU-intensive tasks
  - Efficient toy MC generation techniques deployed by pdfs wherever possible
  - Pdfs provide analytical normalization integrals wherever possible
  - Multi-dimensional integrals can by partially numeric, partially analytic
  - Caching and lazy evaluation of integrals and other expensive objects
  - Constant expressions in Likelihood are automatically identified and optimized prior to fits
  - Typical effect of optimization between is a speedup factor 2-20
- Philosophy – Let user concentrate on formulating physics problem
  - Let RooFit worry about optimal computation 'under the hood'
  - Only user input is that PDFs provide analytical integrals and efficient generation algorithms for known cases (will be automatically applied when possible)

Parallelization in RooFit – When is it useful?

- When is parallelization useful in RooFit?
- RooFit is ingredient of 'semi-interactive' final step of physics analysis. Operations should ideally not take more than a few seconds or minutes, to preserve interactive workflow
- True for most uses cases with 'simple' to 'moderately complex' probability models.
- Not true for ambitious use cases
    1. Highly complex likelihood models (Higgs fit to all channels, ca 200 datasets, O(1000) parameter) can take O(few) hours
    2. Unbinned ML fits with very large data samples
    3. Unbinned ML fits with MC-style numeric integrals ('time-dependent angular analysis of Dalitz decays')
    4. Neyman construction of confidence belts ('no asymptotic'). Requires many fits to toy data samples
- Will focus on use cases 1-3), as 4) is easily parallelizable by end-users on batch facilities and is beyond scope of 'semi-interactive' use
- In spirit of 'semi-interactive' use will focus on factor 10-30 speedup on single multi-core machine to reduce O(few hour) operations to O(few minute) operations

Parallelization in RooFit – what is there now

- Likelihood calculations inherently very suitable for parallel calculation as calculation is repetition of N_{event} equal calculations
- RooFit currently has a simple parallel evaluation engine for likelihood objects
- Strategy: divide L(data|param) in N equally sized chunks



RooFit parallelization – work flow

- Essential features
  - Slaves are separate processes started with fork()
  - Inter process communication via combination of pipes and shared memory
  - Sequence of operation during fit

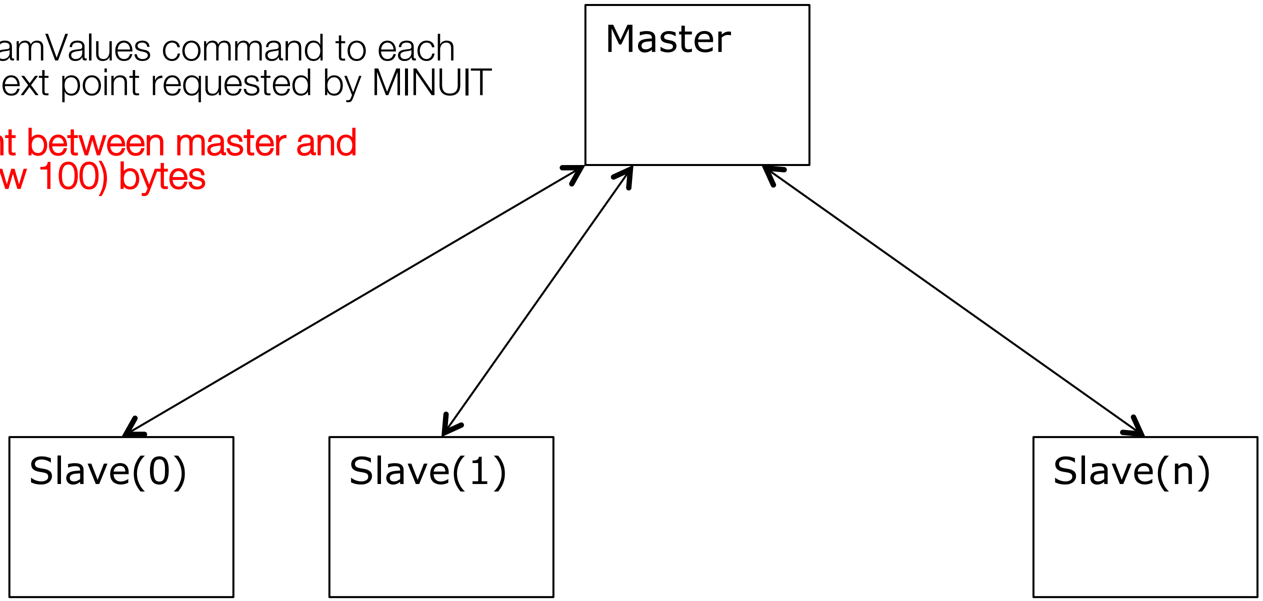
    1) Send Calculate command to each Slave (non-blocking)
    (all slaves perform calculation in parallel)

    2) Send Retrieve command to each Slave (blocking)

    3) If evaluation errors were detected, request full details from slaves

    4) Combine Slave results and feed result to MINUIT

    5) Send UpdateParamValues command to each Slave reflecting next point requested by MINUIT
  - Amount of data sent between master and slaves typically O(few 100) bytes



RooFit parallelization performance

- Consider a simple scenario (for parallelization): fitting a single pdf to an unbinned dataset – how well does it scale with Ncpu?
- T_{CPU}(fit) = T_{CPU}(Lcalc)/NCPU + T_{CPU}(protocol-overh) + T_{CPU}(comp-overh)
  - Protocol overhead is extra time spend passing information from/to slaves.
  - Computation overhead is extra time spent in duplicate calculations due to splitting.
- Protocol CPU overhead is typically very small, as little is done (adding back likelihood components together).
- Computational overhead varies by situation. In current implementation PDF normalization integrals are calculated fully by each Slave process.
  - For *analytical integrals* this is a small (neglibible) overhead.
  - For *numeric integrals* this may take as much time as likelihood calculation as is a potentially killer issue. (One of the issues to be addressed. Will come back to this)

RooFit parallelization performance

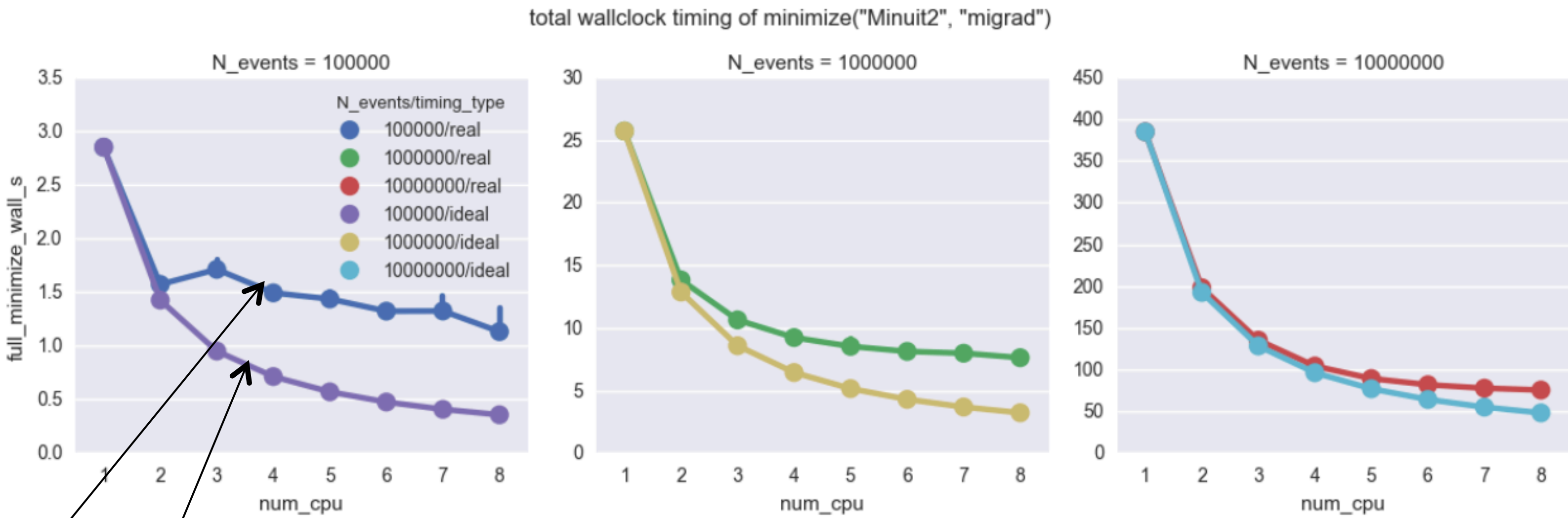
- But *key metric is wall-time performance*, so wall-time overhead (waiting, idle times) are also important
- $T_{wall}(fit) = T_{CPU}(fit) + T_{wall}(protocol\text{-}overh) + T_{wall}(scheduling) + T_{wall}(imbalance)$
  - Protocol wall-time overhead is wall-time spent in inter-process communication
  - Scheduling wall-time overhead occurs if Slave process don't all exactly start at the same time due to OS-related scheduling timing
  - Imbalance wall time occurs if length of calculation (wall) time of the slaves varies
- Protocol wall-time overhead must be small for parellelization to be efficient at large NCPU (next up: performance tests of current protocol)
- Scheduling wall-time overhead can be a real problem! Size of an individual slave task can be quite small → frequent start/stop of calculations.
- Imbalance wall-time overhead is strongly dependent on nature of the model: for a single model/dataset, an (almost) perfectly balanced workload is easy to achieve, but for a 500-dataset Higgs combination likelihood with datasets varying between few(Kevt) unbinned and 1-bin counting experiments, this can be extremely difficult.

RooFit parallelization performance

- Demonstration model: unbinned ML fit to sum-of-Gaussians pdf (with fully analytical normalization)
- Comparison of wall-times vs N(cores) uses

Short fit (3-sec)

Long fit (400-sec)



Expected performance
(ideal parallelization)

Actual performance

Performance quite horrible for NCPU>2 for short fits (comparatively large fixed-time overhead),

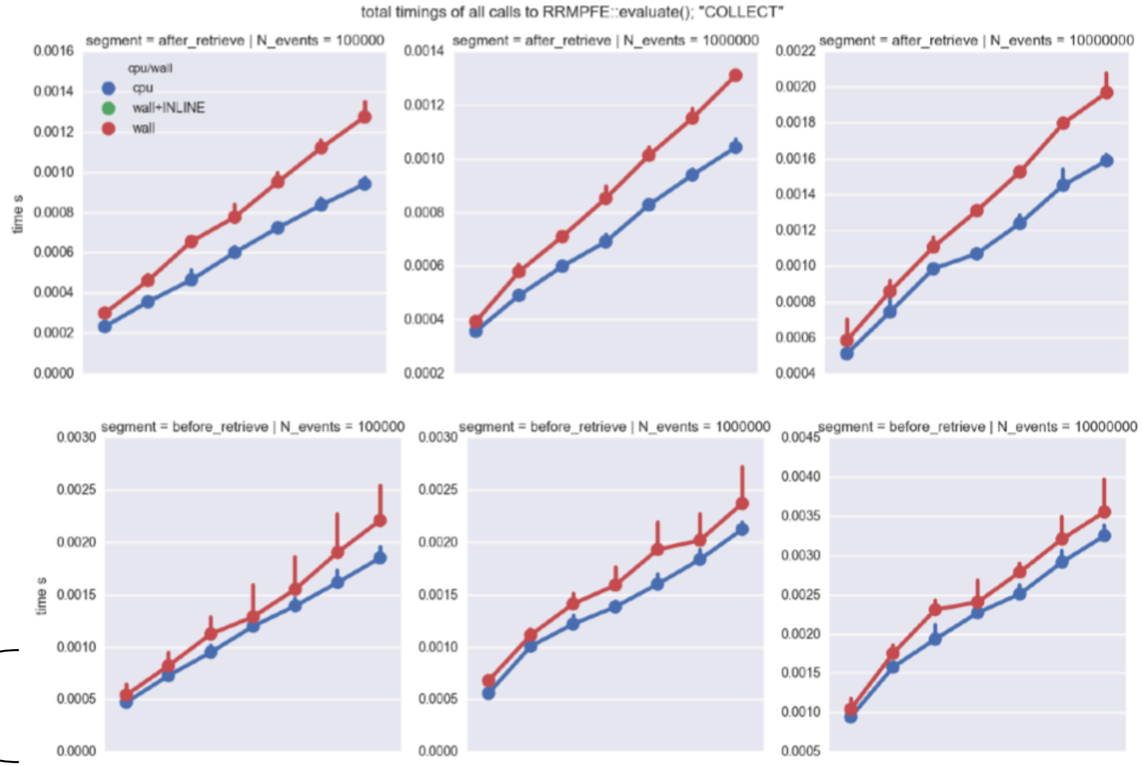
Overhead becomes less dominant for longer fits

RooFit parallelization performance

- Have been investigating various contributions of protocol, OS, imbalance etc to overhead
- Communication protocol overhead increases with NCPU, but is tiny in size and cannot explain effect

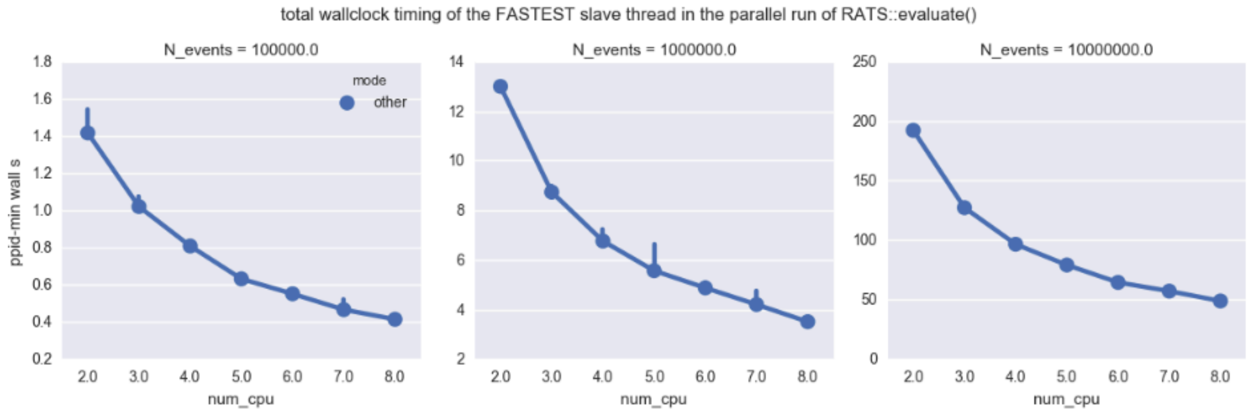
=0.1%
of total
wall time

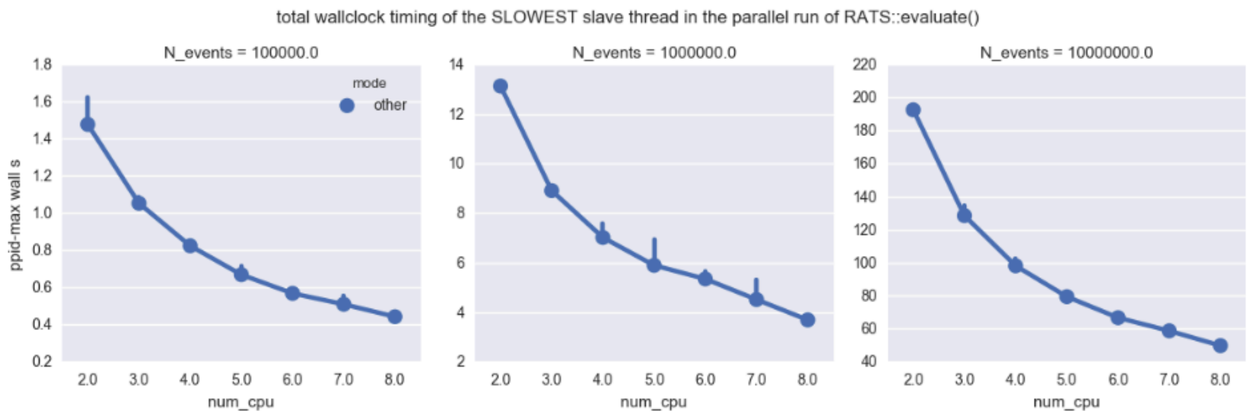
=0.1%
of total
wall time



RooFit parallelization performance

- Next investigated load-balancing overhead. What is difference between slowest & fastest slave process?

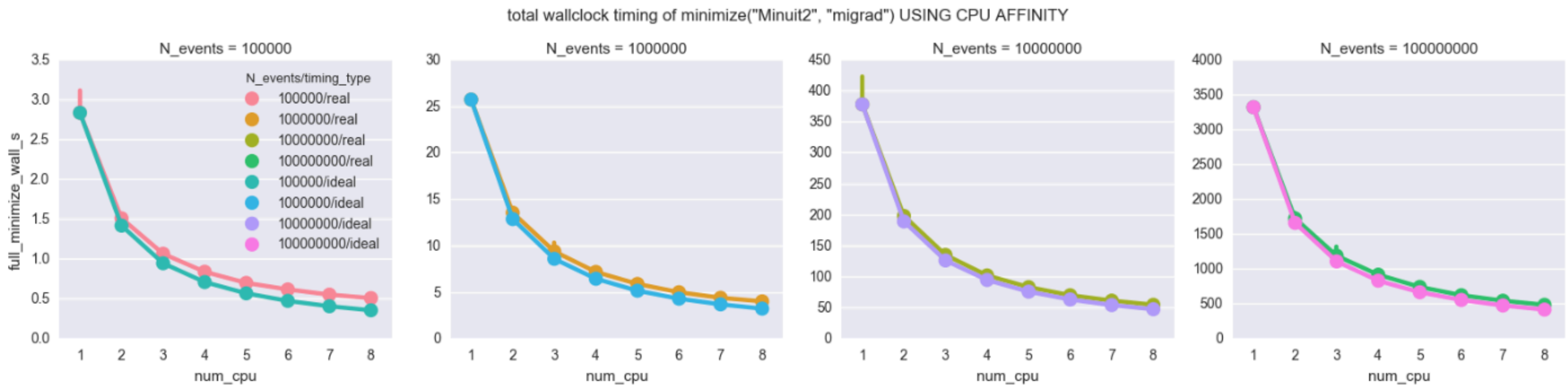




Differences tiny – but also expected here since this was the easiest use case (single unbinned PDF with many events)

RooFit parallelization performance

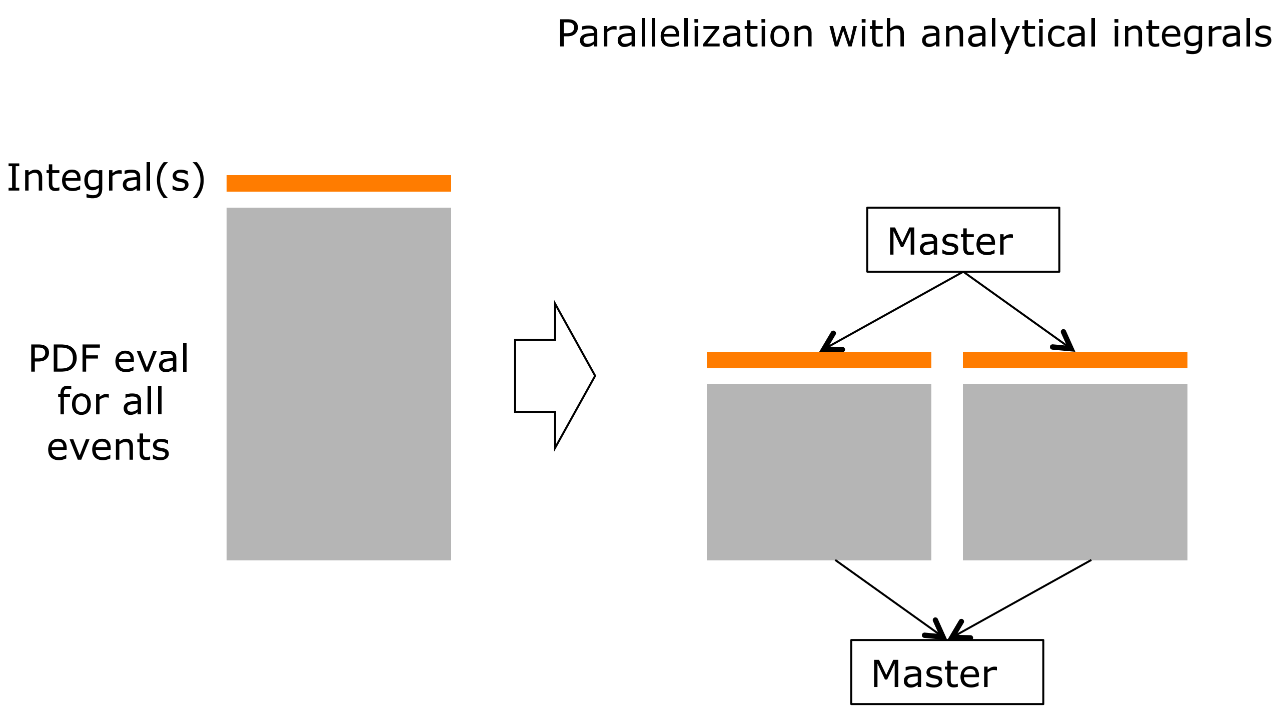
- Finally – looked into OS scheduling overhead. This turns to be the dominant effect of wall-time overhead!
- Can be improved by setting CPU affinity: if each slave process is pinned to a designated core, overhead reduces drastically!



- With CPU affinity fixed, total wall-time overhead of RooFit parallelization w.r.t ideal is O(5%) for 8-core fit for 400-CPU second fit (and better for longer fits)
- Consider this 'good enough for now' and move on to address much more O(100%) scaling issues that arise with use of numeric integrals and Likelihoods with >>1 dataset/model

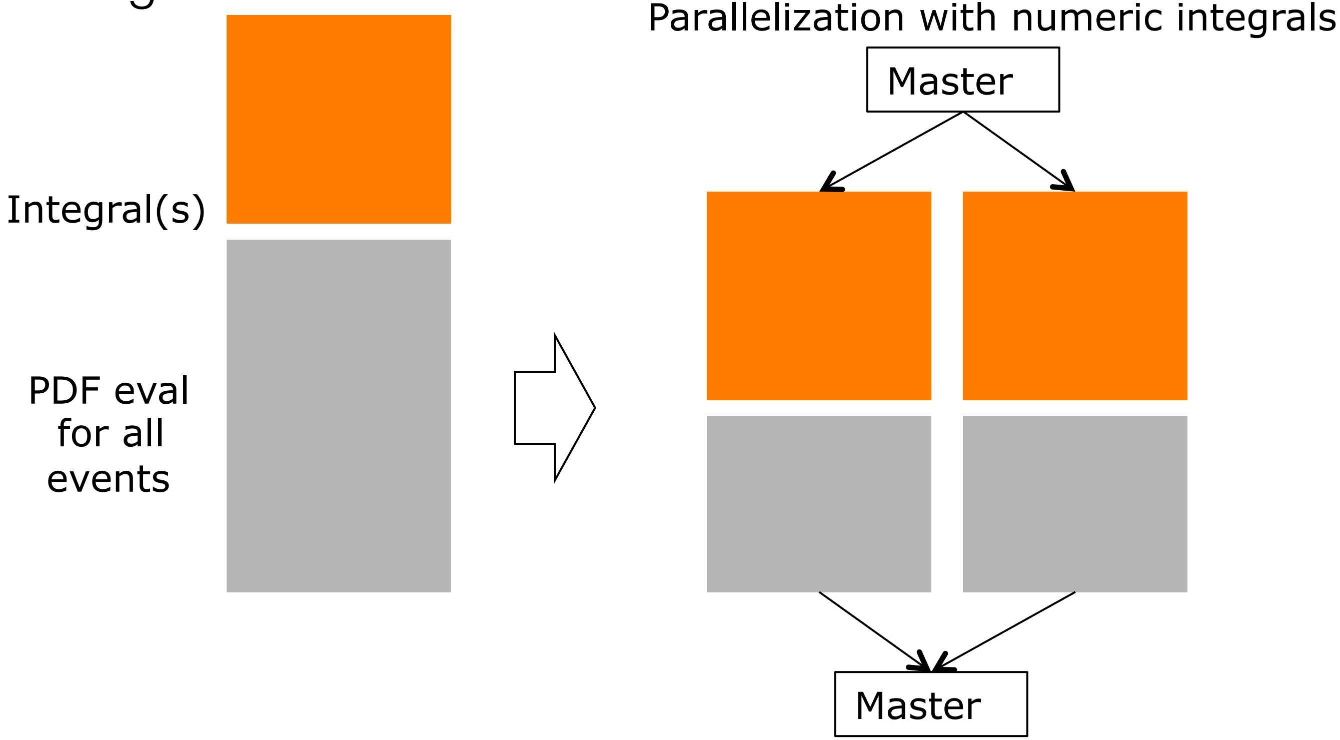
RooFit – timing & parallelization of numeric integrals

- (Everything from here on is work in progress)
- Project 1 – Distributed calculation of numeric integrals
- Problem: If normalization integrals in a pdf expression are not (all) analytical, per-slave overhead of numeric integration will spoil scaling



RooFit – timing & parallelization of numeric integrals

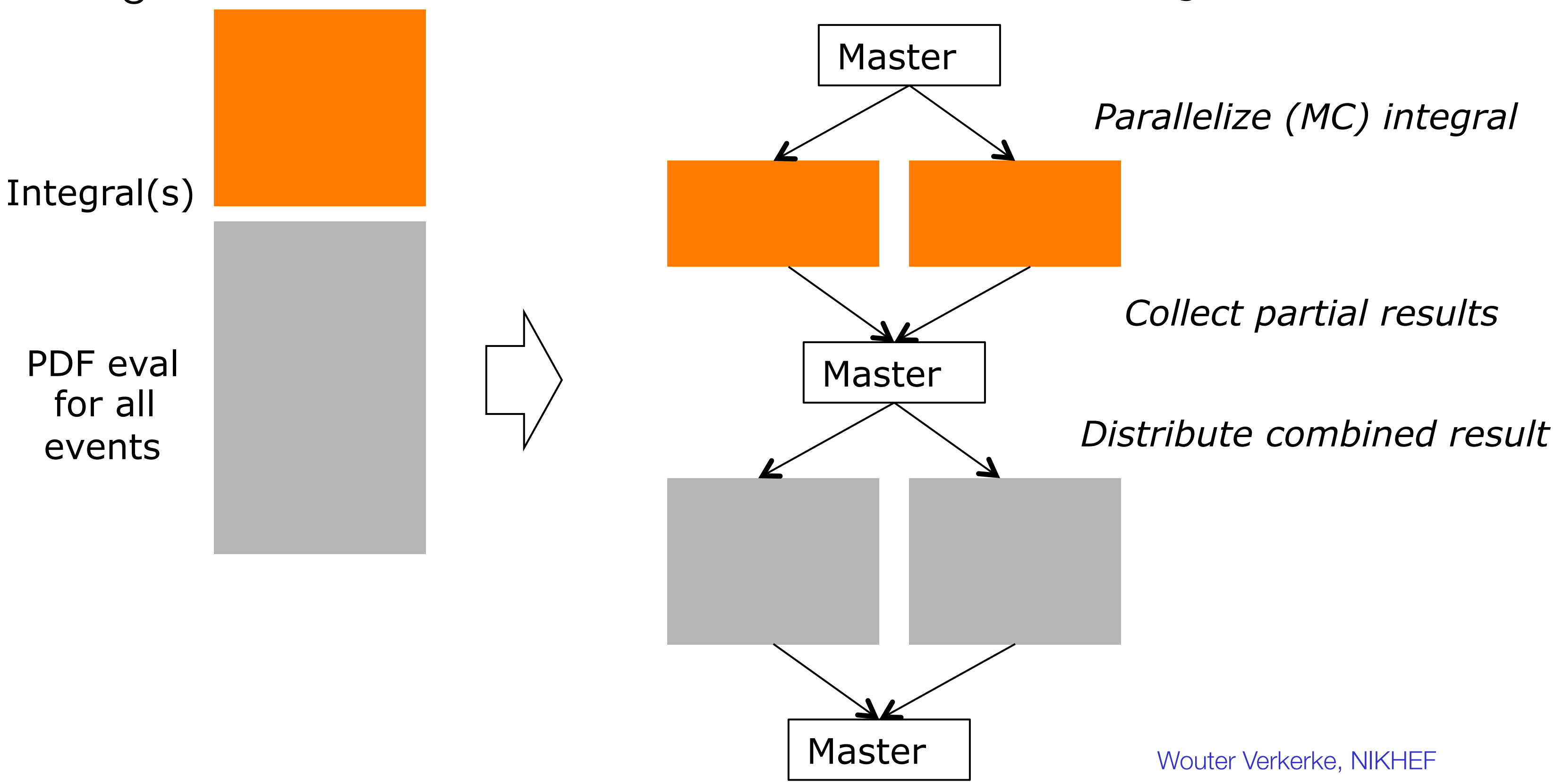
- (Everything from here on is work in progress)
- Project 1 – Distributed calculation of numeric integrals
- Problem: If normalization integrals in a pdf expression are not (all) analytical, per-slave overhead of numeric integration will spoil scaling



RooFit – timing & parallelization of numeric integrals

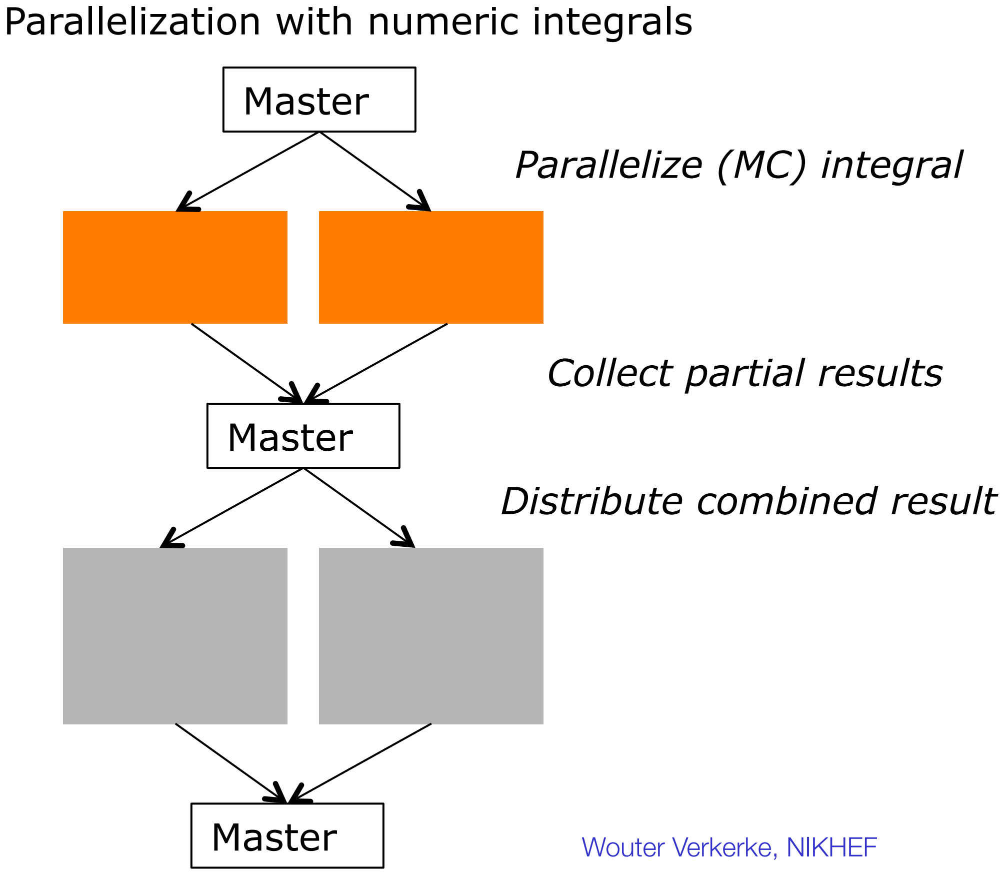
- (Everything from here on is work in progress)
- Project 1 – Distributed calculation of numeric integrals
- Problem: If normalization integrals in a pdf expression are not (all) analytical, per-slave overhead of numeric integration will spoil scaling

Parallelization with numeric integrals



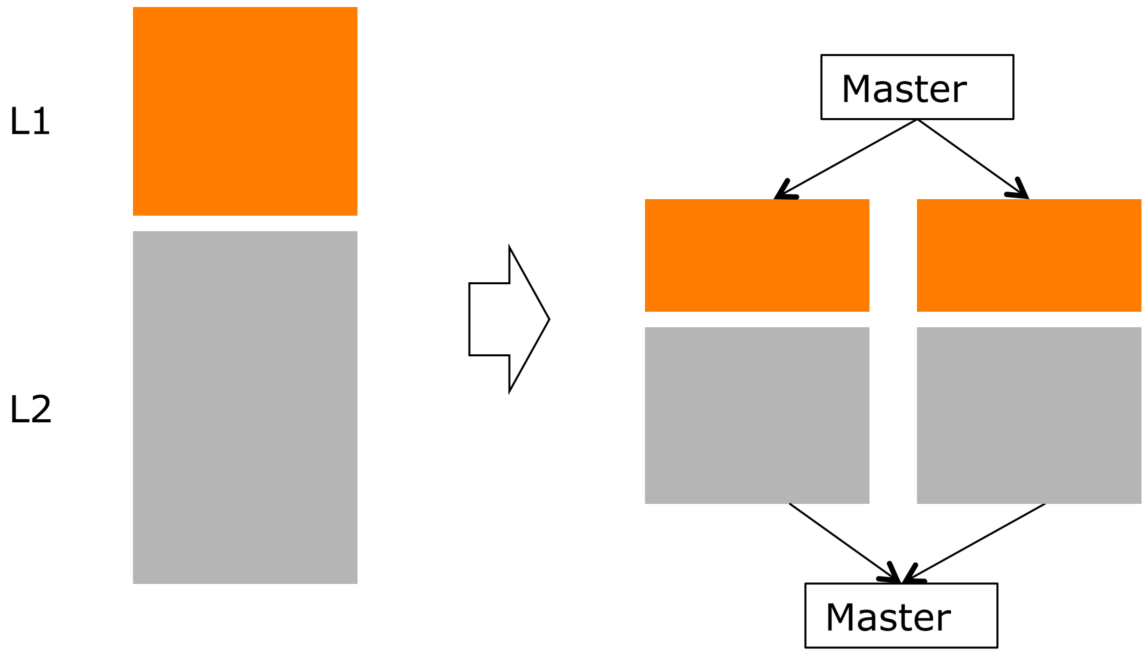
RooFit – timing & parallelization of numeric integrals

- (Everything from here on is work in progress)
- Project 1 – Distributed calculation of numeric integrals
- Problem: If normalization integrals in a pdf expression are not (all) analytical, per-slave overhead of numeric integration will spoil scaling
- Can be solved by distributing calculation of integral
  - Modify RooMCIntegrator (but would then work transparently for all pdfs)
  - Modify Master/Slave architecture to be able to distribute more type of tasks (now only 1/Nth of L)
  - Threshold to decide if any given integral is expensive enough to merit overhead of distributed calculation



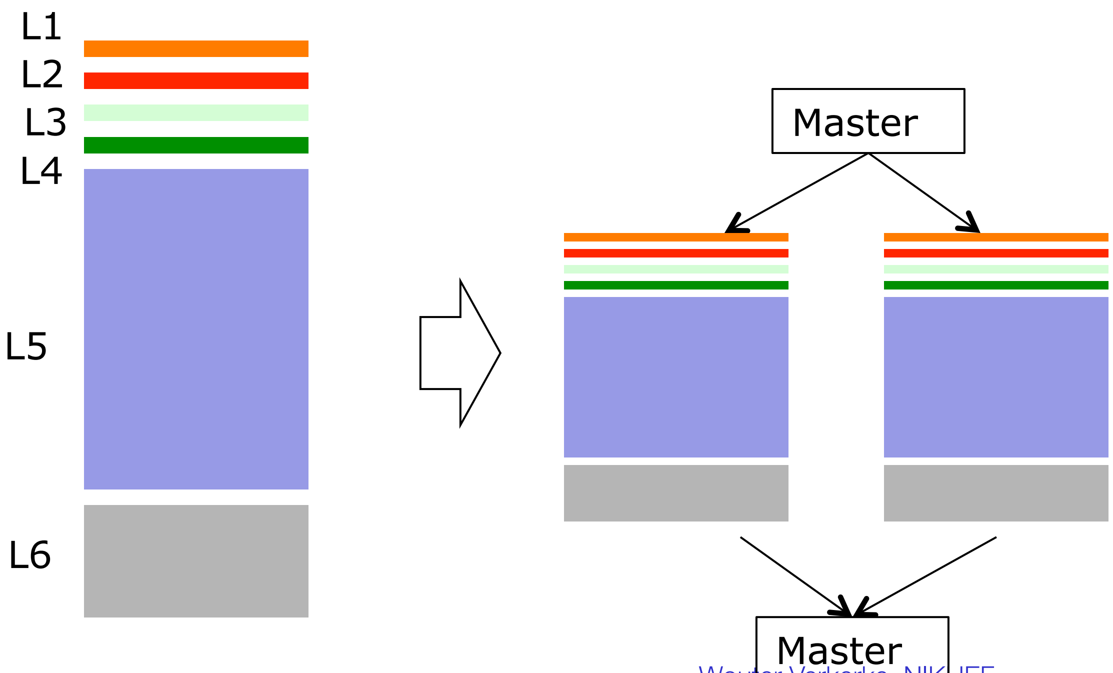
RooFit – timing & parallelization of multi-part Likelihoods

- Project 2 – Intelligent balancing of component likelihood over slaves
- Problem: if a Likelihood consist of n>>1 component likelihoods based on separate pdfs/datasets, load balancing becomes a problem
- Current strategy: parallelize each component likelihood



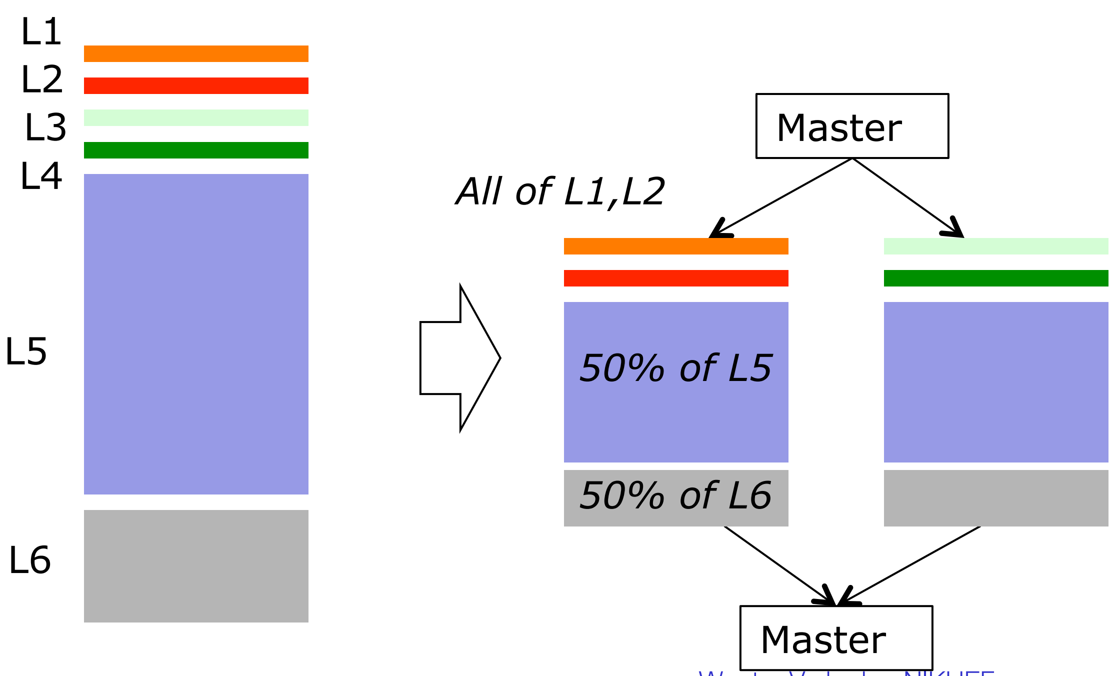
RooFit – timing & parallelization of multi-part Likelihoods

- Project 2 – Intelligent balancing of component likelihood over slaves
- Problem: if a Likelihood consist of n>>1 component likelihoods based on separate pdfs/datasets, load balancing becomes a problem
- Current strategy: parallelize each component likelihood
  - Will break down if small components exist that cannot be evenly divided (e.g. Nevents/bins < Ncpu)
  - Typical Higgs combination likelihood (L1…L250) will only achieve 2x speedup with NCPU=6.



RooFit – timing & parallelization of multi-part Likelihoods

- Project 2 – Intelligent balancing of component likelihood over slaves
- Problem: if a Likelihood consist of n>>1 component likelihoods based on separate pdfs/datasets, load balancing becomes a problem
- New strategy – time components and distribute dynamically
  - Will break down if small components exist that cannot be evenly divided (e.g. Nevents/bins < Ncpu)
  - Typical Higgs combination likelihood (L1…L250) will only achieve 2x speedup with NCPU=6.



Short-term plans

- Step 1 – Implement collection of timing information of numeric integrals and Likelihood components (now in progress)
- Step 2a – Augment existing Master/Slave scheduler to be able to distribute calculations of multi-component likelihoods in arbitrary ways: i.e. can specify calculation fraction of each component likelihood individually
- Steb 2b – Implement a new load balancing algorithm that divides work between N slaves intelligently based on timing information and dataset size information
- Step 3a – Implement parallel calculation strategy for RooMCIntegrator
- Steb 3b – Augment existing Master/Slave schedule identify expensive integrals for parallel calculation and execute them in that way.

Medium-term plans

- Complete rewrite of Master/Slave scheduler
  - Current class structure not very suitable for new plans, but first need to test some of the new concepts before converging on a good practical design for the next scheduler
  - Foresee ability to have different back-end implementations for actual calculations: now only multi-core on single hosts. Other useful backends could be multi-host, GPU-based, or backends that aim for a complete re-expression in another tool (Theano, tensorflow).
  - Need a bit of thinking on how to best design this: tie scheduler implementation to a particular back-end architecture (i.e. one scheduler for multi-core, one scheduler for GPU etc…), or decouple those and have another interface layer in between (that would e.g. allow hybrid calculation strategy: big likelihood components on paralellized on GPU, counting measurements on CPU).