ROOT, Threading and Parallelism

Fons Rademakers
CERN PH/SFT

Annual Concurrency Forum Meeting, Fermilab, 4/2/2013.

Current Status of Concurrency in ROOT

- PROOF and PROOF-Lite are available since 1997 and 2008
  - Actively used in analysis, active development program
- The ROOT thread interface classes (TThread et al) since 1997
  - Provide a platform independent interface to pthread, Solaris threads and Win32 threads
- The ROOT implementations for Win32 (1996) and MacOS X (2012) are multi-threaded (device graphics interface runs in a separate thread and the command line input too)
- Several big locks provided (a sense of) thread safety
  - Big CINT lock - only one CINT instance per process
  - Big container lock
  - Big graphics lock
- But also made fine grain threading very difficult or impossible

Enhancing Concurrency in ROOT 6

- Thread safe cling
  - Multiple concurrent cling instances
- Lock free containers
- Concurrency in I/O
  - Multi threaded I/O
  - Asynchronous prefetching
  - Parallel merger
- Concurrency and vectorization in math
  - Possible introduction of the Vc matrix library, // histogramming
  - See Lorenzo's talk
- Concurrency and vectorization in geometry
  - See Andrei's talk
- Concurrent tasks
  - TTask+TThread and TTaskManager

Cling - The New Interpreter

- Replacing CINT by new cling: correctness, full C++
- Cling based on LLVM and Clang compiler libraries
  - "New" open source compiler suite, already default on OSX
- Compiler-grade C++11
- Best diagnostics on the market:

CINT

```
Error: Can't call map<string,const char*,less<string>,allocator<const
string,const char*> > >::operator[]((char*)0x255a9e8) in current scope
err.C:19:
Possible candidates are...
(in map<string,const char*,less<string>,allocator<const string,const
char*> > >)
Error: improper lvalue err.C:19
```

```
err.C:19:15: error: assigning to 'mapped_type' (aka 'const char *') from
      incompatible type 'double'
   myMap["A"] = 12.3;
              ^ ~~~~
```

cling

Cling and Concurrency

- Cling released in July 2011
  - Fully functional C and C++11 interpreter
  - Just-in-Time compilation, i.e. always ACLiC
  - Still a few issues to solve, e.g. reloading of code
- Support multiple interpreter objects (c.f. multi-threaded context)
  - One execution engine per thread
- Separation of dictionary and execution engine
  - Needed for parallel I/O

Collection Classes

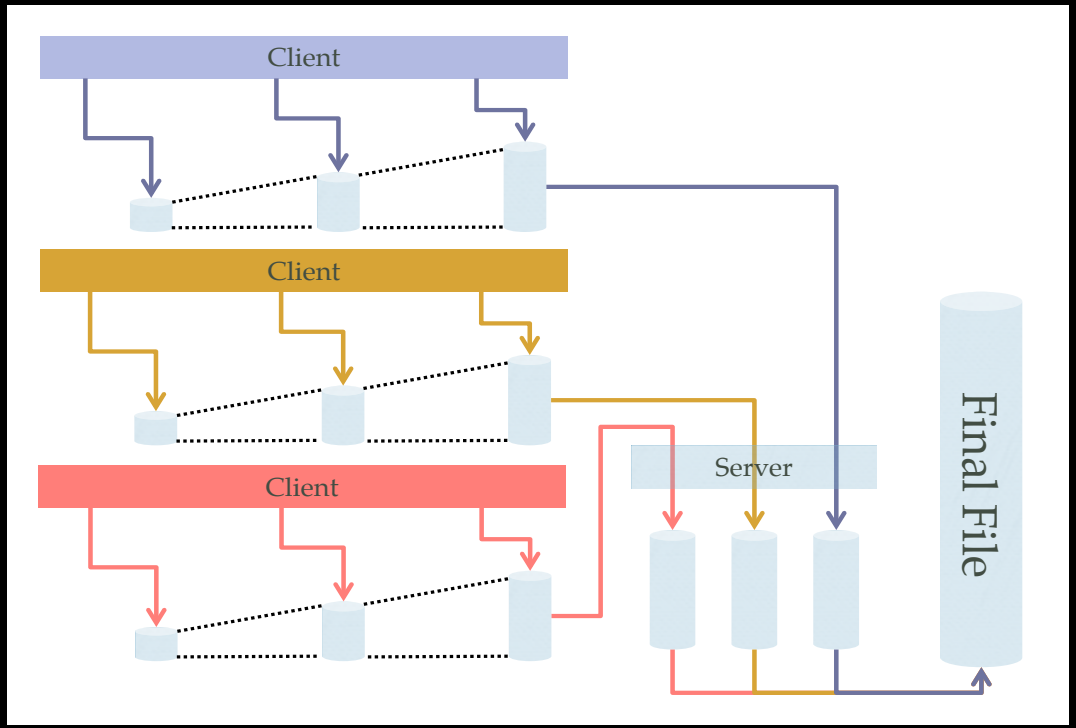
- Work on thread safety issues
  - Lock free containers (a la TBB)
- TThread replaced by C++11 threads or thin layer for backward compatibility
- Focus on C++11 thread, atomicity, tld, concurrency support

Parallelism for I/O

- Parallel Merging via external process
- Support for one TFile per thread
  - Currently requires meta data (TClass, TStreamerInfo) to be fully build before starting parallel operation
  - Planning to lifting this restriction in ROOT 6 (requires Cling)
- Internal use of spare cores
  - Ability to read multiple TBranch data in parallel
  - Top level branches can be uncompressed and un-streamed independently
  - Prefetching of remote file content in a separate thread
  - Asynchronous write

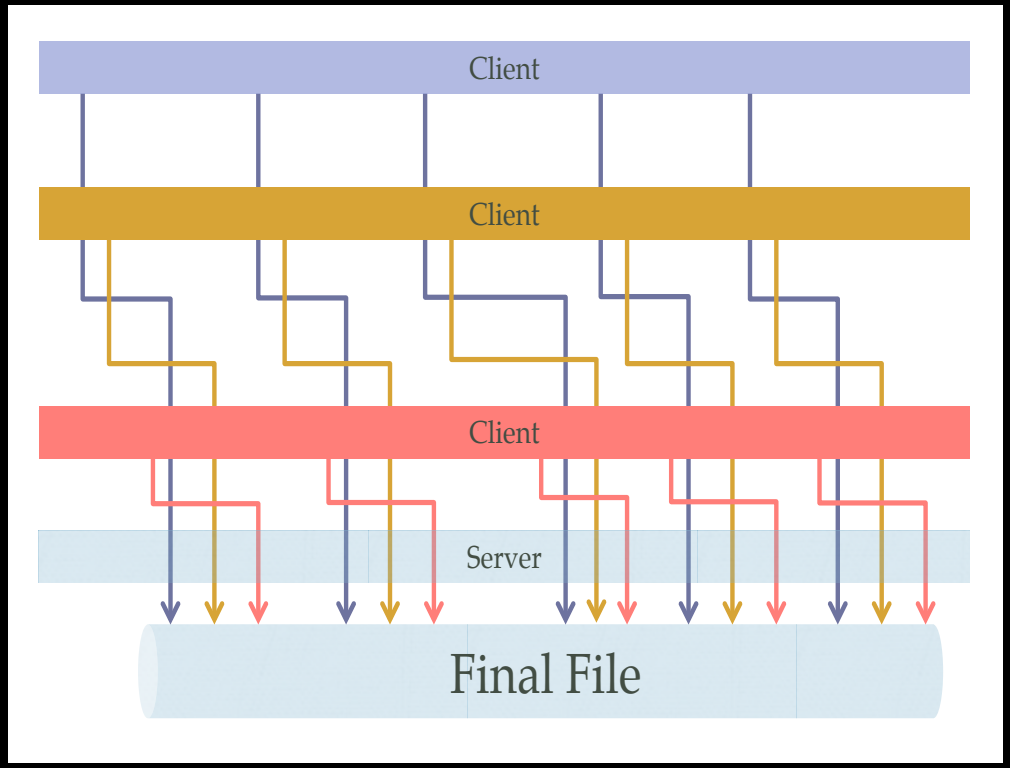
Parallel Merging

- Existing solution:
  - Processing and storing the partial output on each local node and then only at the end of the process, upload to the server and only when all slaves are done, read all files on the server and write to a single output file.



Parallel Merging

- New TFile implementations:
  - TMemFile: a completely in-memory version of TFile.
  - TParallelMergingFile: a TMemFile that on a call to Write will upload its content and reset the TTree objects.
- New solution:
  - Increase parallelism by having the slaves start uploading the TTree clusters directly to the server which immediately starts saving them in the final output file.



Parallel Tasks

- TParallelTask
  - Basically a TTask combined with a TThread
- TParallelTaskManager
  - Management of sub tasks and sub managers
  - It can access the parent manager
  - Dependency management
  - Runs TParallelTasks in parallel or one-by-one depending on dependencies
- Prototype implementation available

New PROOF Features

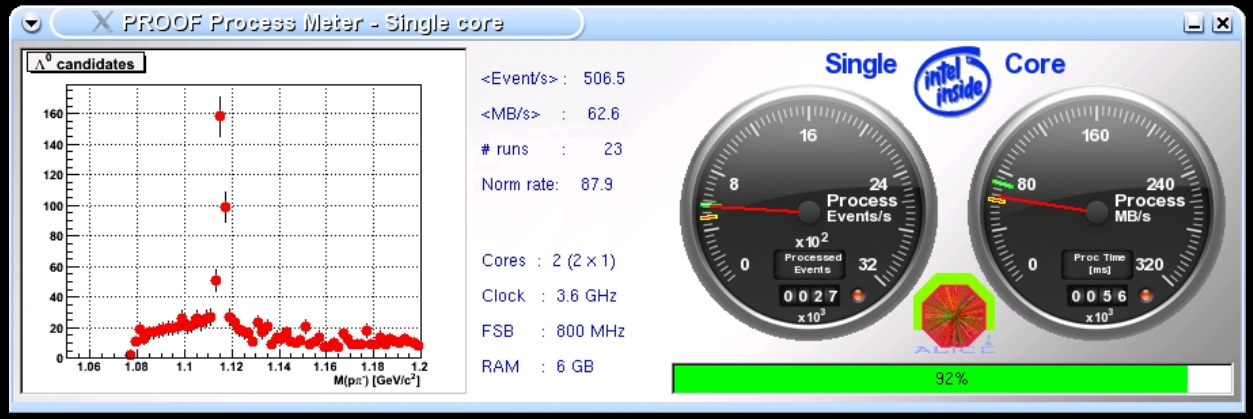
- Increasing number of sites and users deploying PROOF
- Deployment on ad-hoc PROOF clusters is made easy by PROOF-on-Demand (PoD)
- The setup of a dedicated PROOF cluster including file management and file access using xrootd has been made easy by the PEAC package
- An extensive PROOF bench suite has been developed allowing to test many aspects of PROOF related parameters (network, disk, cpu, etc.)
- Many stability improvements

PROOF Developments

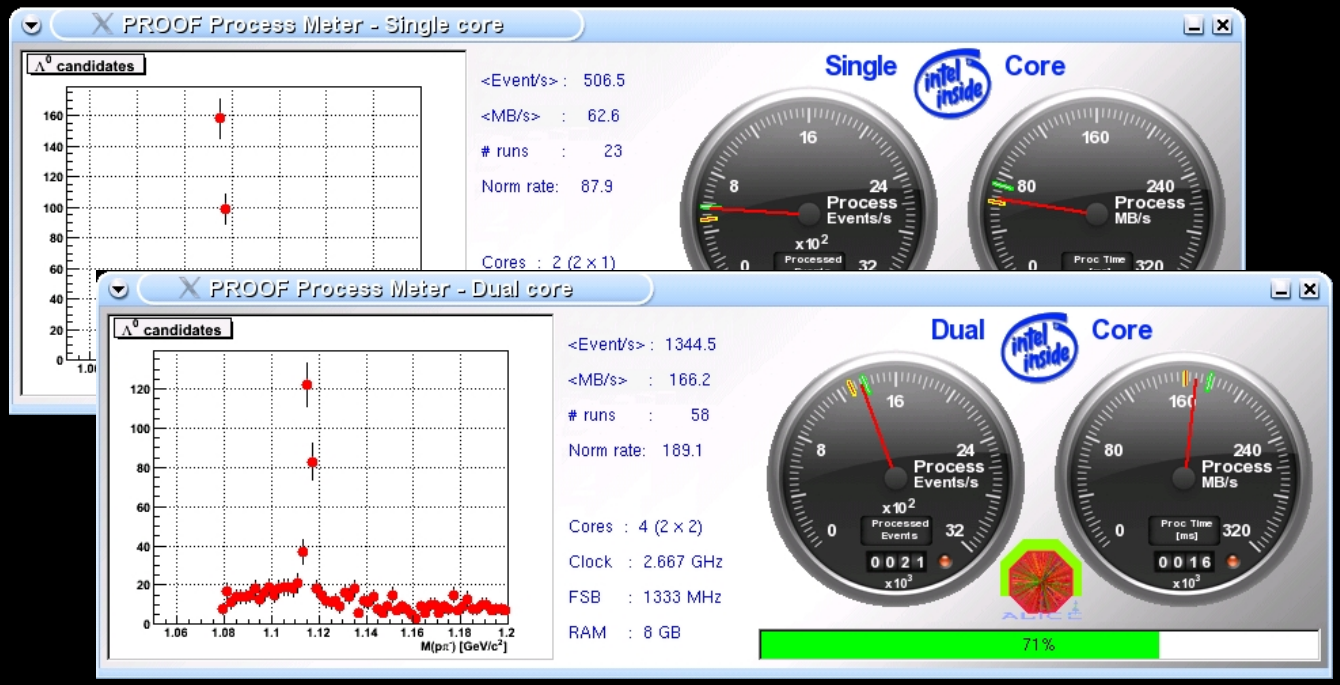
- Improvement of the connection layer and of the work distribution engine (packetizer)
- Result merging optimization (see also parallel tree merging in I/O)
- Improvement of memory management (in merging of results, in sharing common data)
- Support for dynamic addition of workers
- Towards a fully-functional multi-master setup
  - Dataset management across clusters
  - Dynamic load-balancing across sub-masters
  - Dynamic re-organization of a set of workers in multi-tier structures to scale up to O(1000) cores
  - Needed for extreme platforms like IBM's BG/Q

PROOF Lite

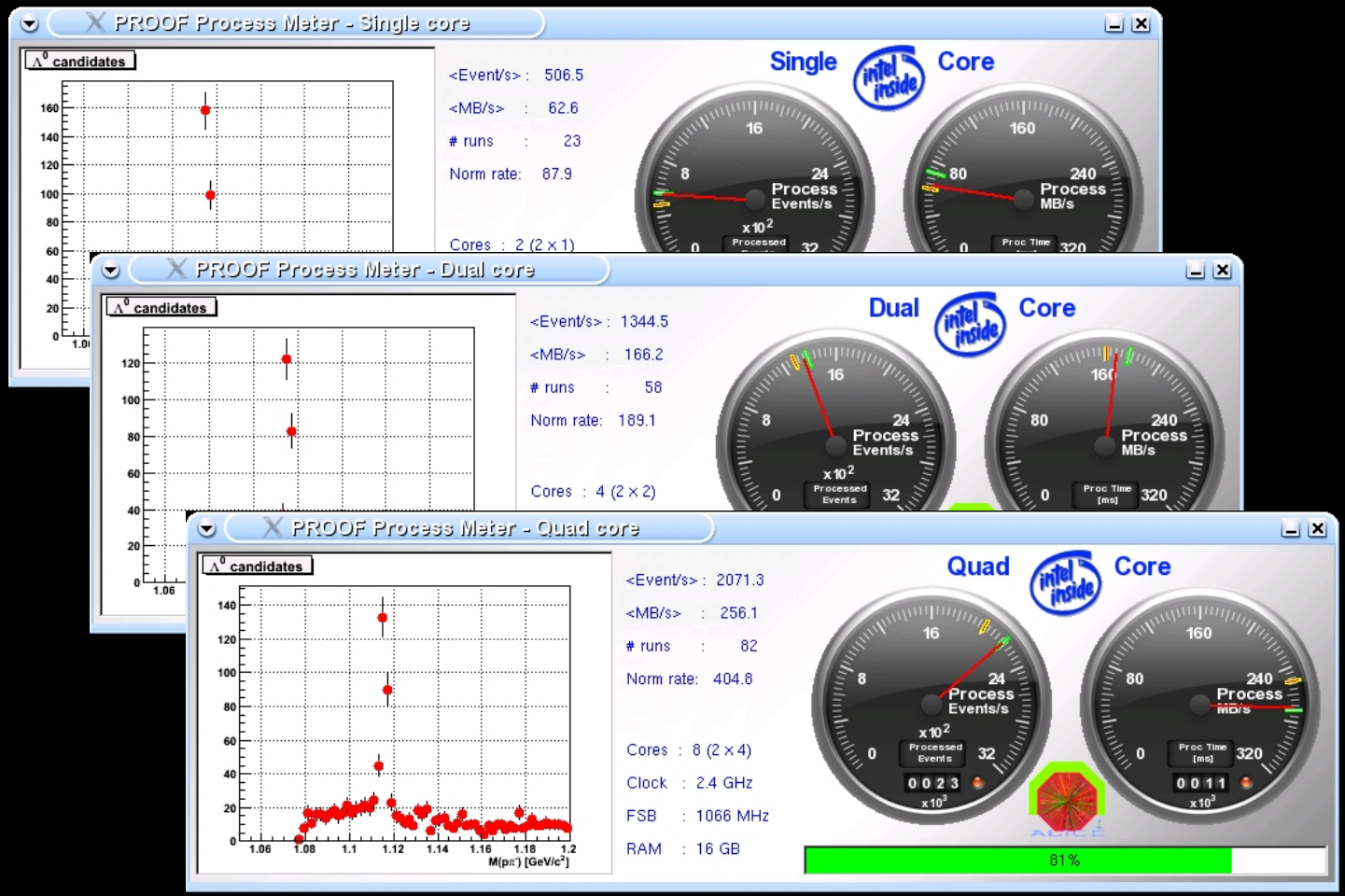
PROOF Lite


PROOF Process Meter - Single core
Λ0 candidates
M(pπ) [GeV/c²]
<Event/s> : 506.5
<MB/s> : 62.6
runs : 23
Norm rate: 87.9
Cores : 2 (2 x 1)
Clock : 3.6 GHz
FSB : 800 MHz
RAM : 6 GB
Single
Core
Process Events/s
Processed Events
Process MB/s
Proc Time [ms]
92%

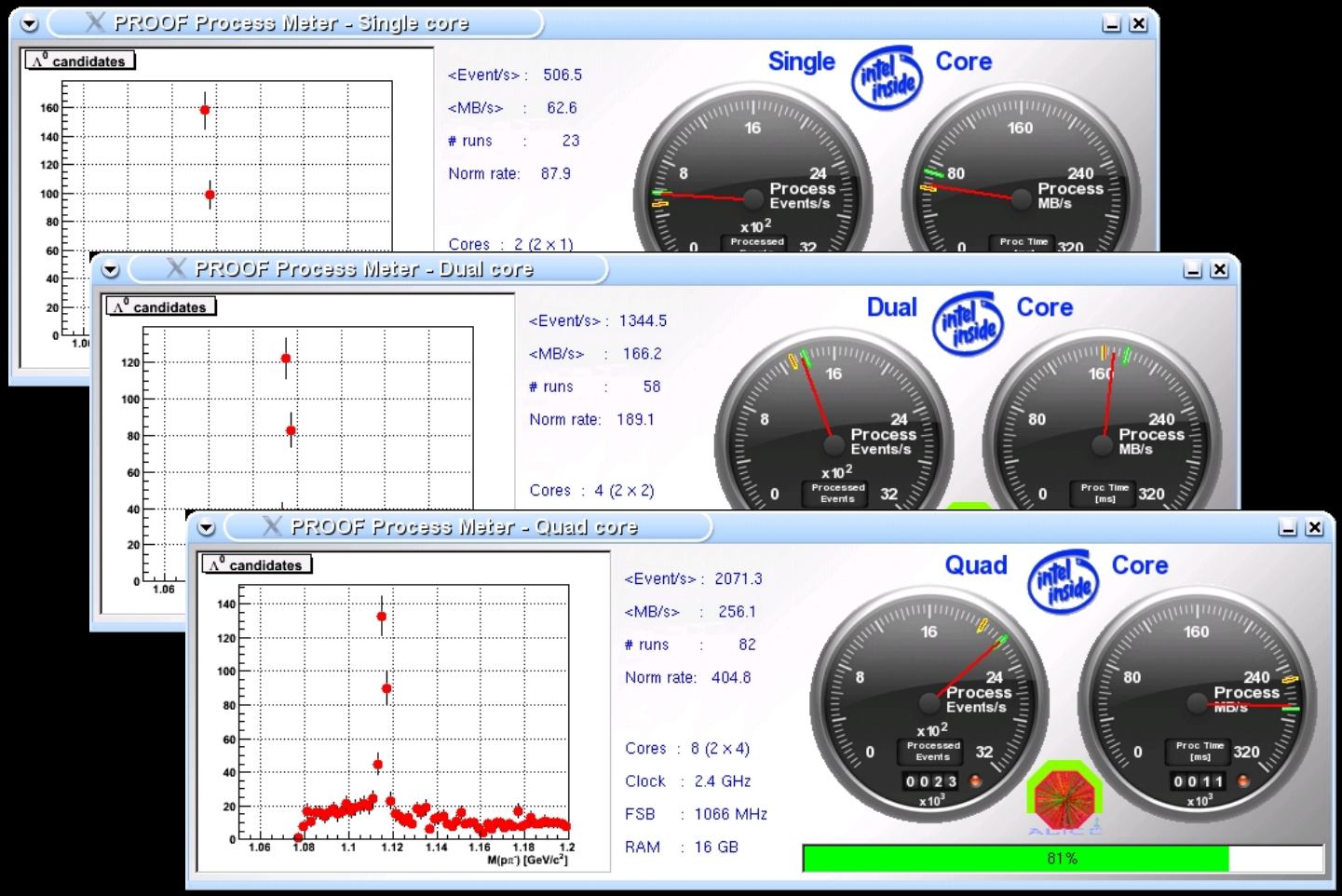
PROOF Lite


PROOF Process Meter - Single core
Λ0 candidates
<Event/s> : 506.5
<MB/s> : 62.6
runs : 23
Norm rate: 87.9
Cores : 2 (2 x 1)
Single Core
Process Events/s
Process MB/s
PROOF Process Meter - Dual core
Λ0 candidates
<Event/s> : 1344.5
<MB/s> : 166.2
runs : 58
Norm rate: 189.1
Cores : 4 (2 x 2)
Clock : 2.667 GHz
FSB : 1333 MHz
RAM : 8 GB
Dual Core
Process Events/s
Process MB/s
71%

PROOF Lite


PROOF Process Meter - Single core
Λ0 candidates
<Event/s> : 506.5
<MB/s> : 62.6
runs : 23
Norm rate: 87.9
Cores : 2 (2 x 1)
Single Core
PROOF Process Meter - Dual core
Λ0 candidates
<Event/s> : 1344.5
<MB/s> : 166.2
runs : 58
Norm rate: 189.1
Cores : 4 (2 x 2)
Dual Core
PROOF Process Meter - Quad core
Λ0 candidates
<Event/s> : 2071.3
<MB/s> : 256.1
runs : 82
Norm rate: 404.8
Cores : 8 (2 x 4)
Clock : 2.4 GHz
FSB : 1066 MHz
RAM : 16 GB
Quad Core
Process Events/s
Process MB/s
81%

PROOF Lite



Running on MacPro with dual Quad Core CPU's.

ROOT 6 2013 Release Plan

- Release ROOT v6-00 end of March
- Release ROOT v6-02 end of November
- As many v5-34 patch releases as needed

Conclusions

- ROOT had threading and parallelism support since its inception
- However, it was not engineered from the ground up for multi-threading
- With ROOT 6 and Cling, we enter a new era
- Next development cycle is focused on
  - Adding finer grain thread safety
  - Multi-threading for I/O, math, fitting, geometry, ...