Running Concurrent Gaudi in Real Life: Status Update on MiniBrunel

D. Piparo for the Concurrent Gaudi Team

Concurrency Forum Meeting 5-5-2013

Content of the Talk

- The goal of the project
- A primer of Concurrent Gaudi design
- Results about physics and code performance for a LHCb reconstruction slice

Part 1
Goal of the Project

Our Goal

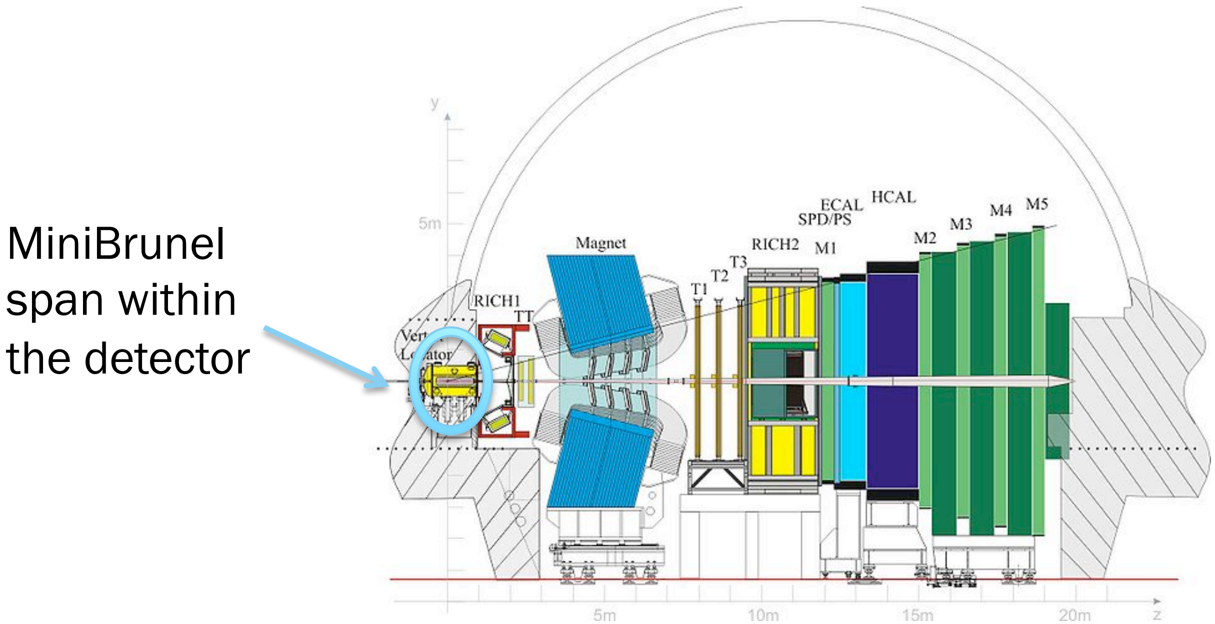
Provide refurbished Gaudi framework which supports

1) Concurrent execution of algorithms

2) Simultaneous processing of multiple events

Pragmatic approach: start from slice of real LHCb reconstruction workflow (called MiniBrunel in the following)

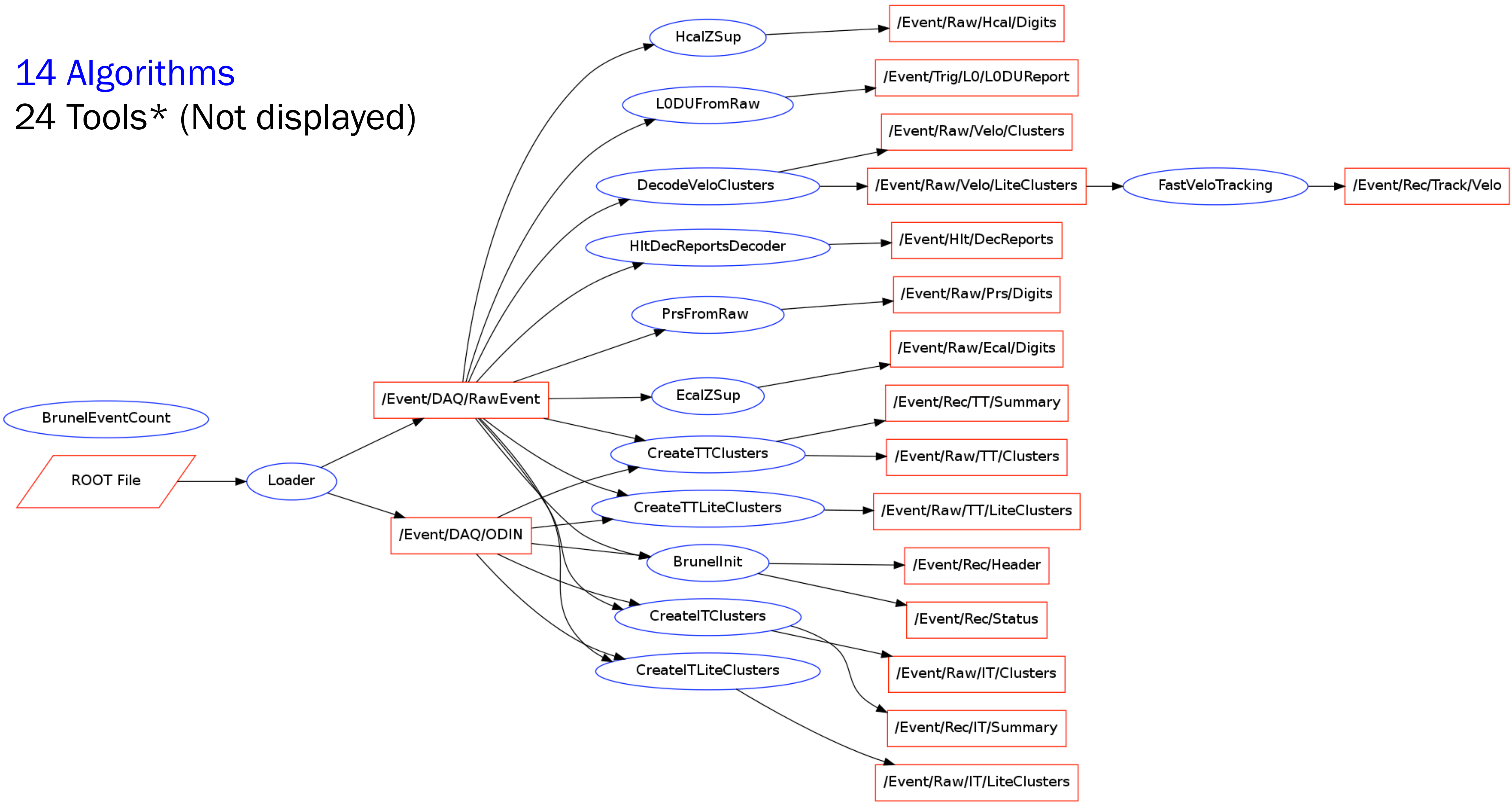
– ~20 algorithms and associated tools: raw decoding and Velo tracking



See backup for useful links about the project.

MiniBrunel: Data Dependencies

14 Algorithms
24 Tools* (Not displayed)



Control flow dependencies not displayed

* Tool: a “lightweight service”, managed by a plugin manager, can be accessible by one or several algorithms, services and tools.

General Strategy

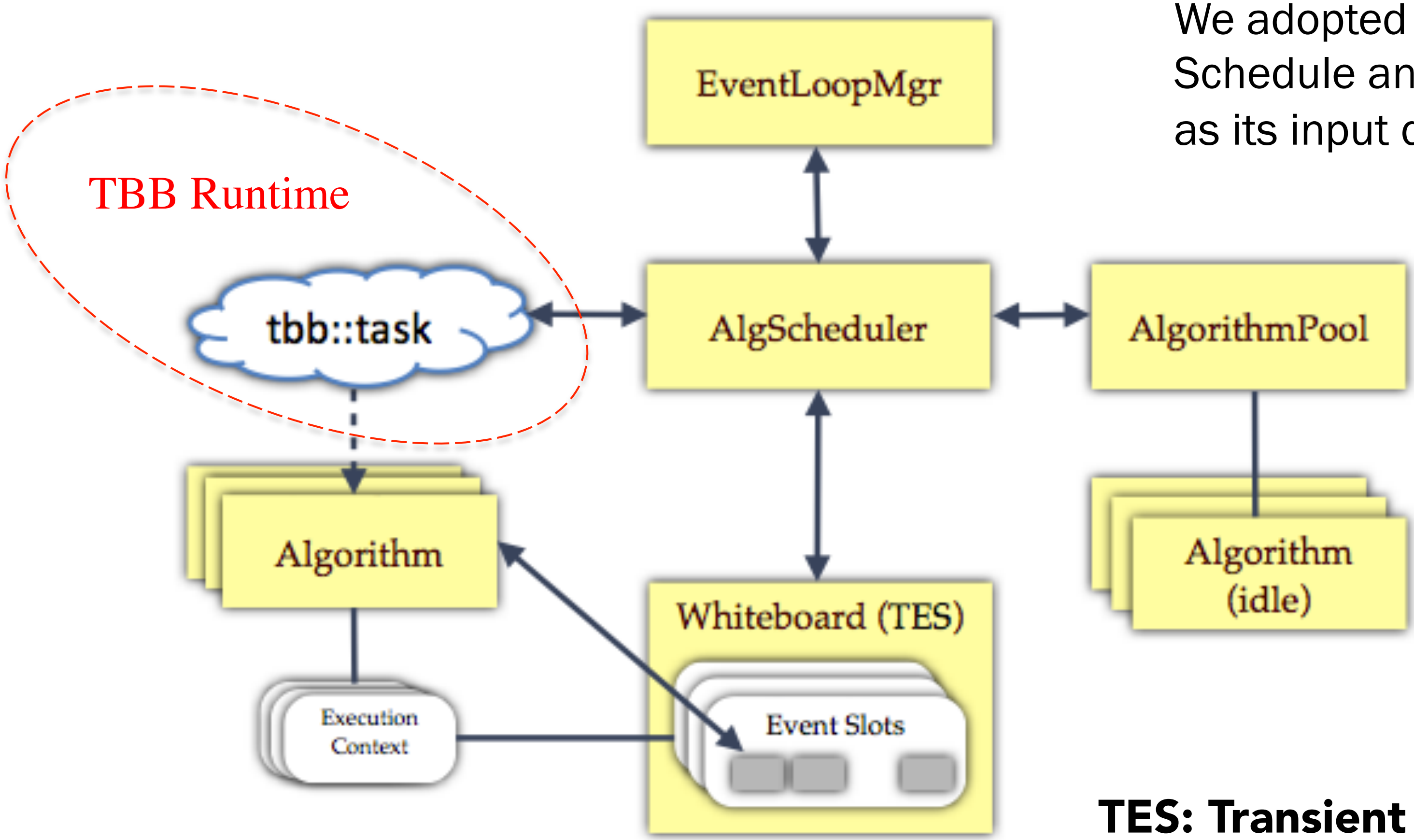
- Classify and document issues encountered during this effort
  - Build a “matrix of costs” - assess the size of the effort that would be required to migrate the complete LHCb stack
- Identify solutions and migration strategies
  - Not only thread safety: assumptions valid in the serial case are broken
  - Operate on existing large codebase
  - Minimal changes of interfaces
  - Provide new components pluggable in the present infrastructure
- Timescale: provide all pieces for MiniBrunel parallel execution by the end of June (the “0.5 Release”, up to now monthly tags provided)

Part 2
Concurrent Gaudi:
a design primer

Components Overview

- New components added to Gaudi to support concurrency
  - E.g. Scheduler, Whiteboard, AlgPool
- Existing components upgraded
  - E.g. ToolSvc, EventLoopMgr

We adopted forward scheduling: Schedule an algorithm as soon as its input data is available

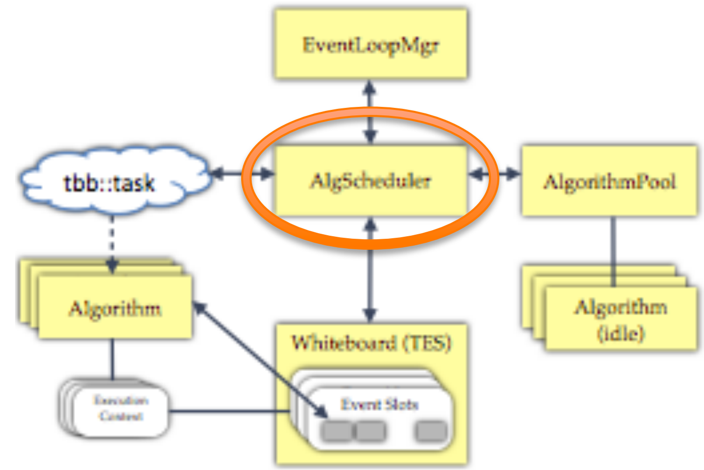


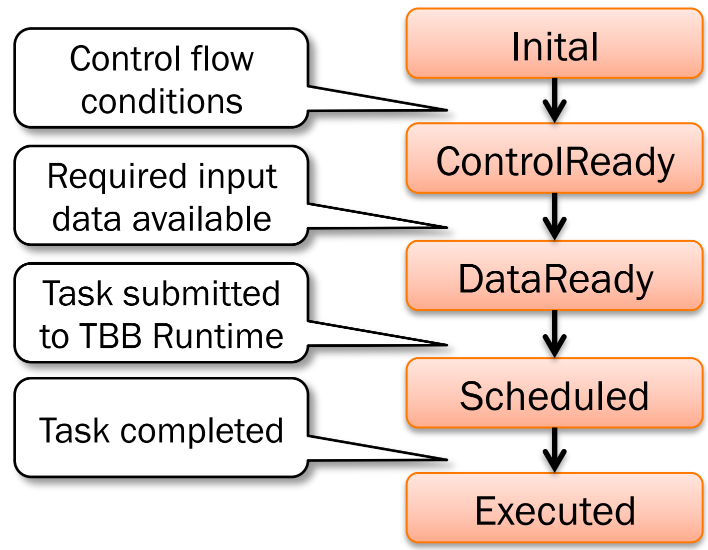
TES: Transient Event Store

The Forward Scheduler

Keeps the state of each algorithm for each event

- Simple finite state machine
- Receives new events from loop manager
- Interrogates whiteboard for new DataObjects
- Pulls algorithms from AlgorithmPool if they are available
- Encapsulate them in a tbb::task for execution
- Absorbs asynchronous events (e.g. arrival of finished tasks) with a thread safe queue of lambda closures (*actions*). Same pattern used for new message svc.





See Backup for more details!

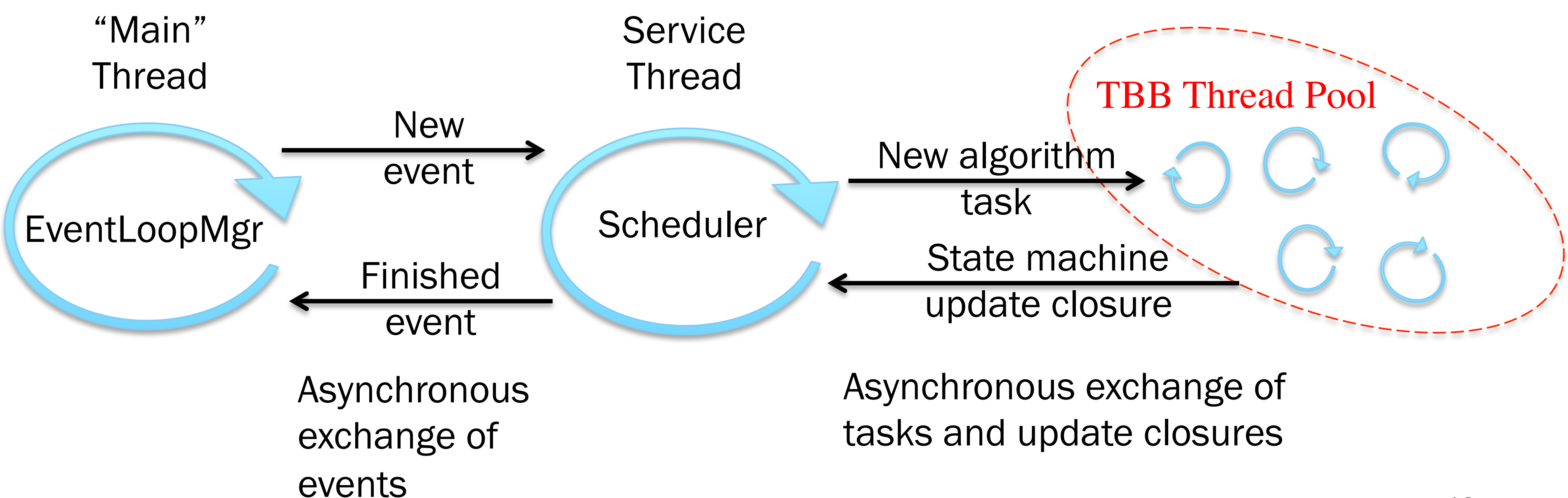
Service Threads

An additional "service" thread (outside the tbb pool, which contains "worker" threads) is spawned:

- Host the scheduler method to update the state machine when an algorithm has run. If no work is available, it sleeps.

The "main" thread manages the event loop ("little more than an event factory"). While the scheduler processes the events, it sleeps.

Other service threads existed and continue to exist (e.g. conditions watchdogs)

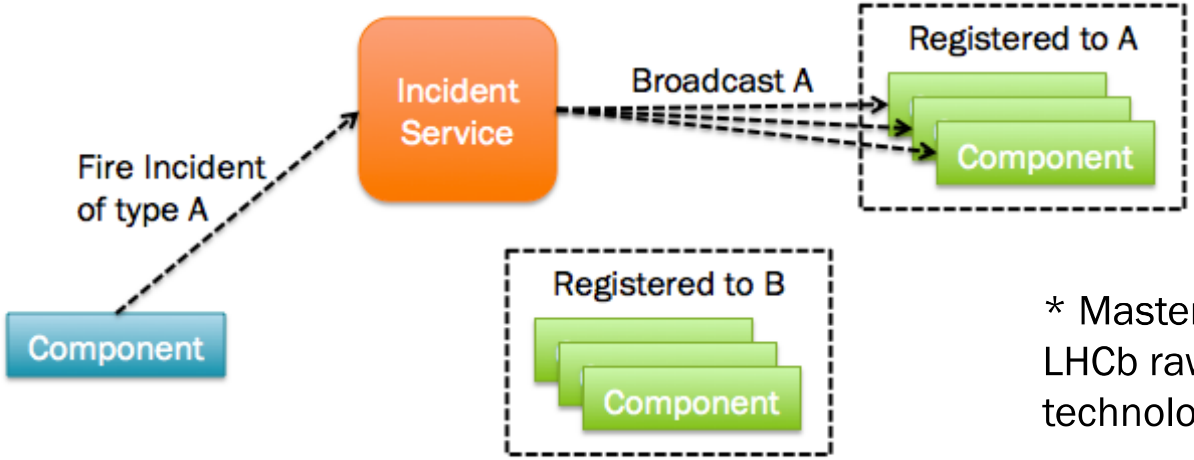


Other Code Changes: Executive Summary

- Algorithm dependencies
  - Data dependencies: announced by the algorithms themselves
- Tools
  - A few tools served as back-door communication channels bypassing the official (event data) channel
- Incidents
  - Meaning of many global incidents radically changed (e.g. BeginEvent)
- MDF* Conversion
  - Support multiple events in flight

See Concurrency Forum meeting on April the 24th
https://indico.cern.ch/conferenceDisplay.py?confId=248560

An incident:

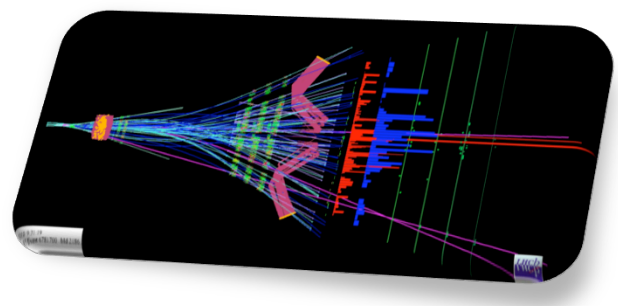


* Master Data Format
LHCb raw banks persistency technology

Part 3
Minibrunel-Physics and Code Performance

Concurrent execution of Minibrunel works!

- Real algorithms running on real data
  - January 2013 software stack, 2011 collision raw data
- Tested with various scenarios
  - Different number of events in flight
  - Several algorithms in parallel
- Assumption: no change of detector conditions during run

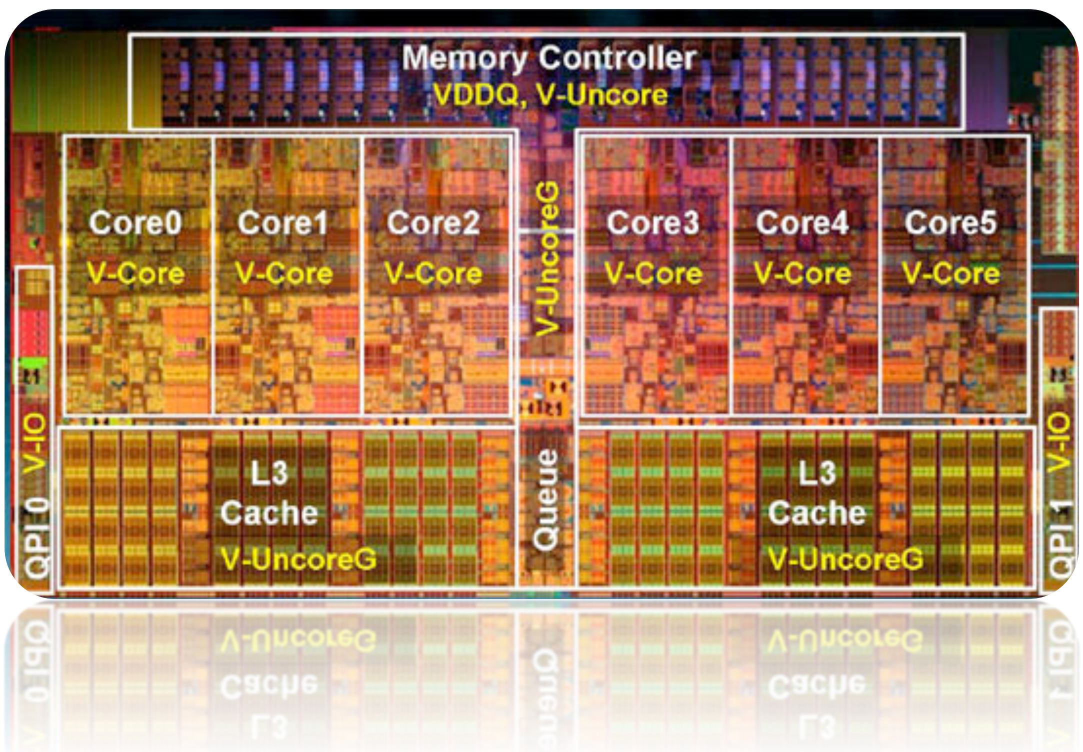


Apects of MiniBrunel Discussed Today

1. Physics performance
2. Overheads due to multievent-multialgorithm execution mode
   - Runtime
   - Memory
3. Scaling on one multicore processor

The Testbed

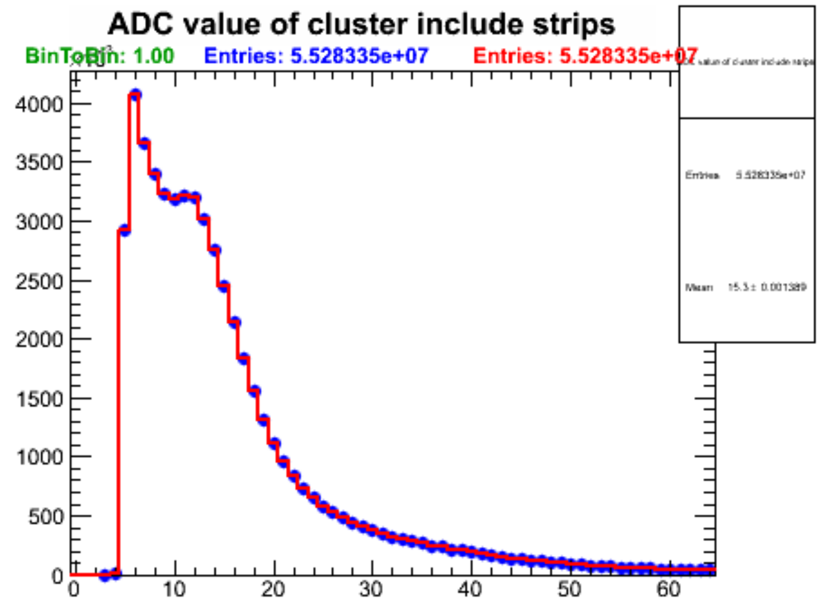
- 10k events (60k for the physics performance estimation)
- SLC6, gcc46
- TCMalloc
- Xeon L5640 @2.27 GHz
- 2 sockets 6+6 HT Cores each (Westmere)



Physics Performance

- It is crucial to obtain the same physics output
- Our physics output estimator: LHCb standard set of data quality monitoring histograms
- Necessary but not sufficient to guarantee production quality results
- Check histograms for serial and concurrent version (high number of simultaneous events and algorithms)

Example of data monitoring istogram: adc counts.



All standard histograms identical bin by bin

Runtime Overhead

There is an overhead when using new components designed for concurrency with one worker thread only in the TBB Pool (as ~expected)

Timing for the event loop only (no initialisation/finalisation):

Serial Gaudi (no new components) ….. 72.9 s

Concurrent Gaudi 1 evt in flight ……… 97.7 s

Concurrent Gaudi 2 evts in flight ……. 73.9 s

Concurrent Gaudi 10 evts in flight …… 72.3 s

1 algorithm running at the time

Work needs to be poured. One MiniBrunel event only is not enough: this is linked to the presence of 1 thread for the EventLoopMgr and 1 thread for the Scheduler.

2 events in flight: enough to get rid of the overhead

Memory Overhead

Running mode:

- 1 clone per event in flight of 3 longest running algorithms
- Full tbb thread pool (24 threads)
- Limit algorithms in flight to 6

Resident Set Size at the end of the event loop (no finalisation):

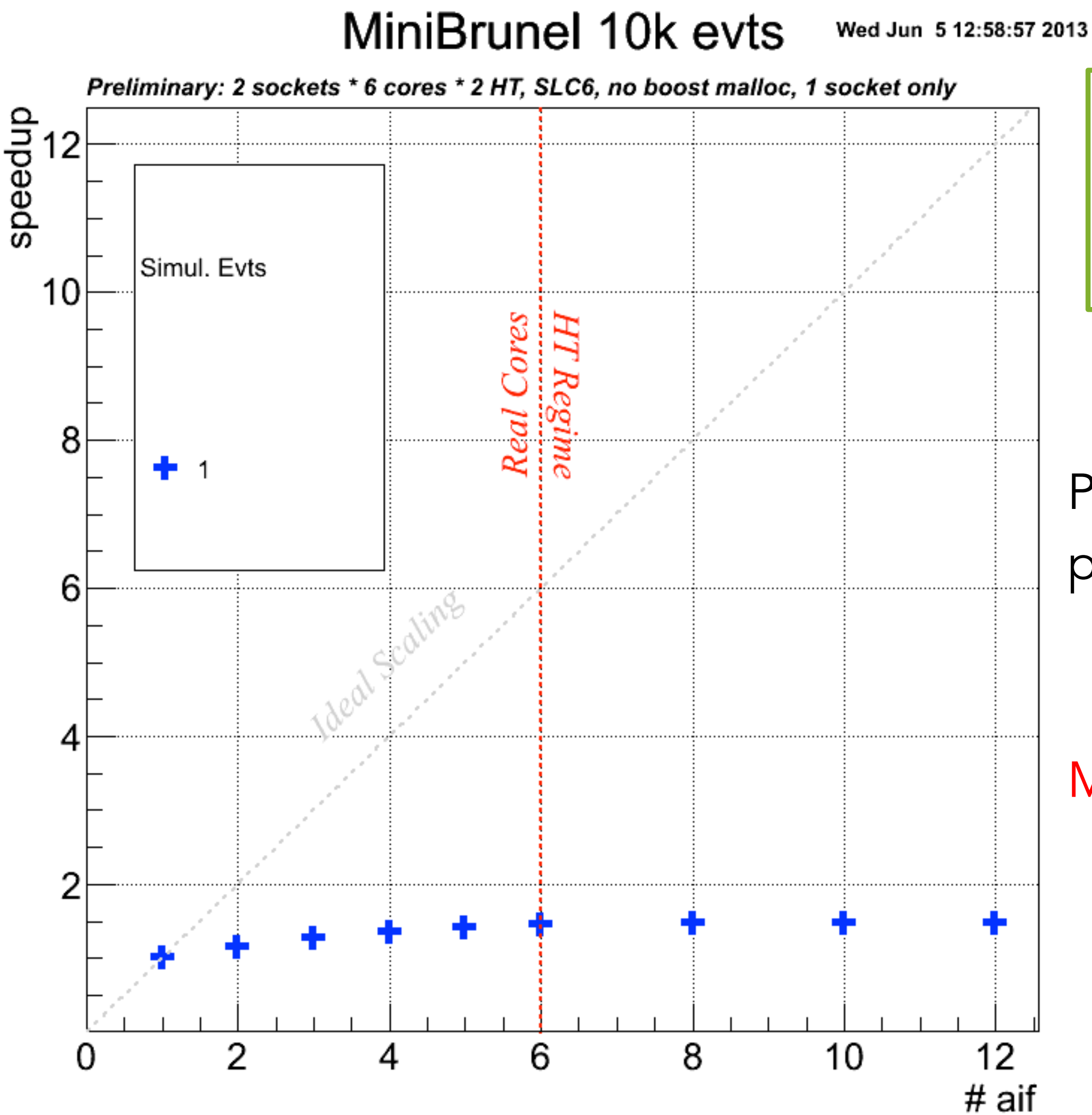
Serial Gaudi (no new components) ….. 478 MB
Concurrent Gaudi 1 evt in flight ……… 480 MB
Concurrent Gaudi 2 evts in flight ……. 485 MB
Concurrent Gaudi 10 evts in flight …… 514 MB

6 algorithms running simultaneously

Note: Not full LHCb events but Minibrunel events.

Memory: multithreaded solution is cheap!

Scaling on One Processor

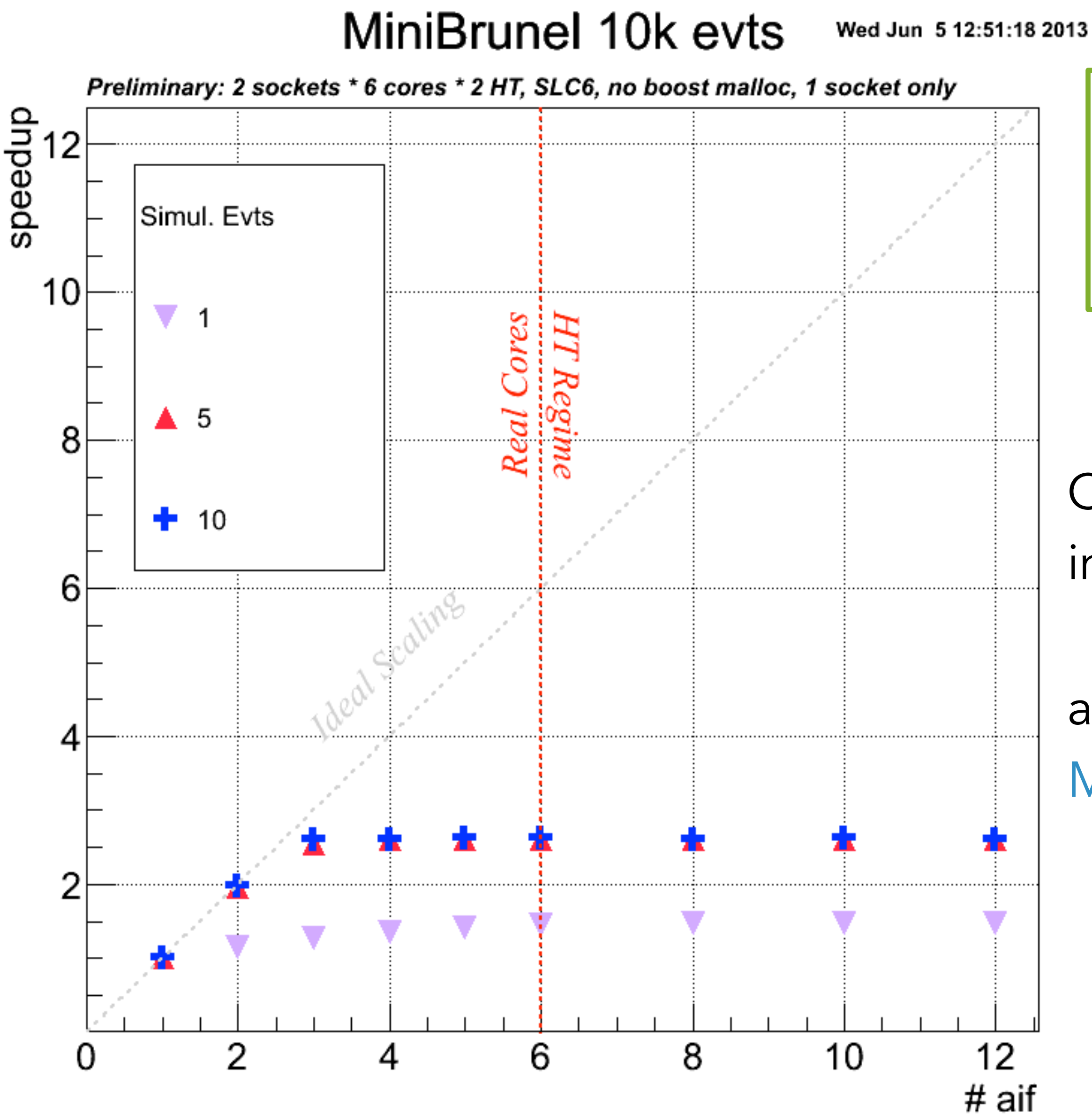


1 Event in Flight
1-12 algorithms simultaneously

Poor opportunities for parallelism

- Data dependencies
- Control flow

Maximum Speedup: ~30%



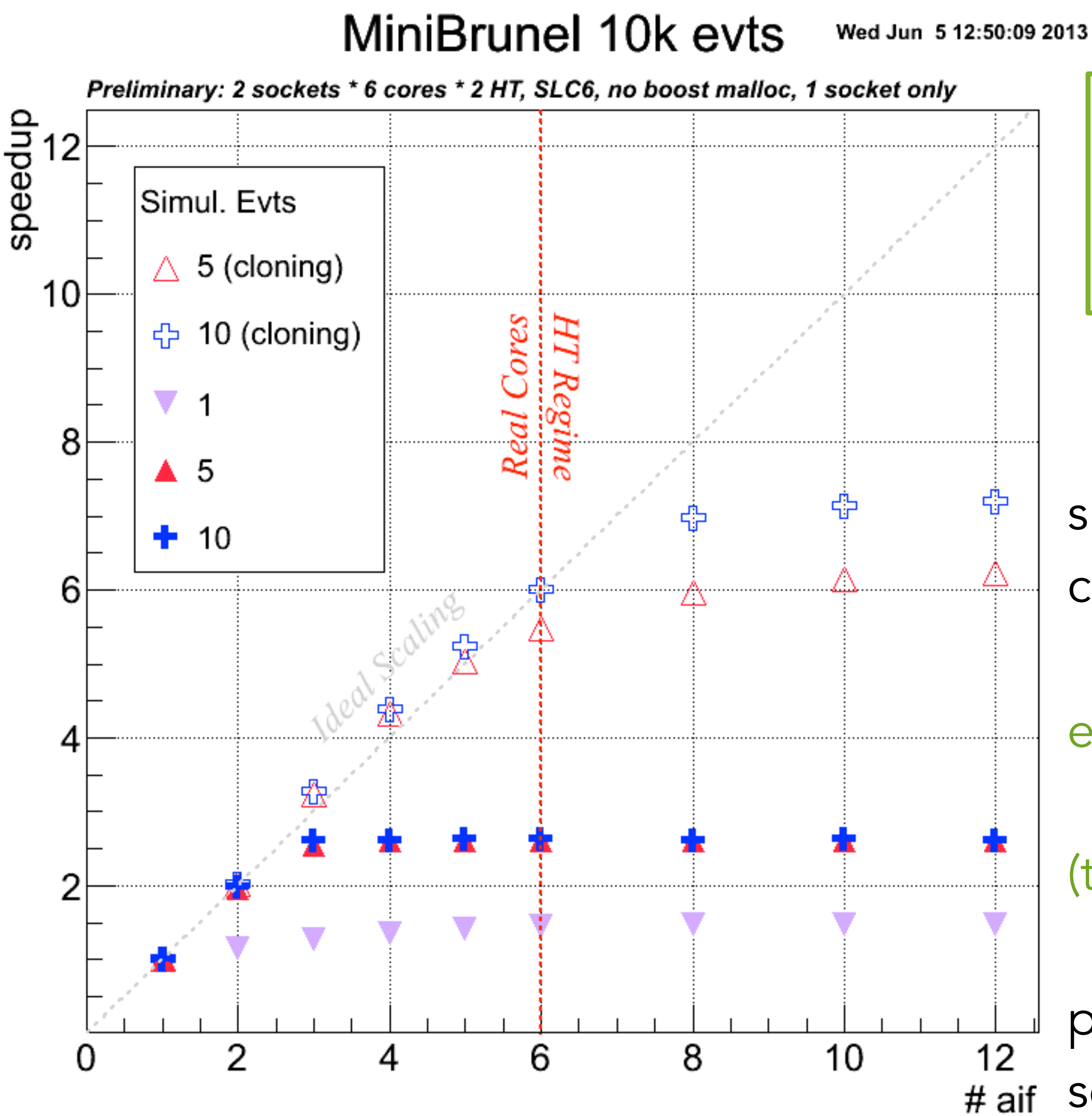
1,5,10 Event in Flight
N algorithms simultaneously

Opportunities for parallelism increase

- Probability to schedule an algorithm is bigger

Maximum Speedup: 2.5x

Scaling on One Processor



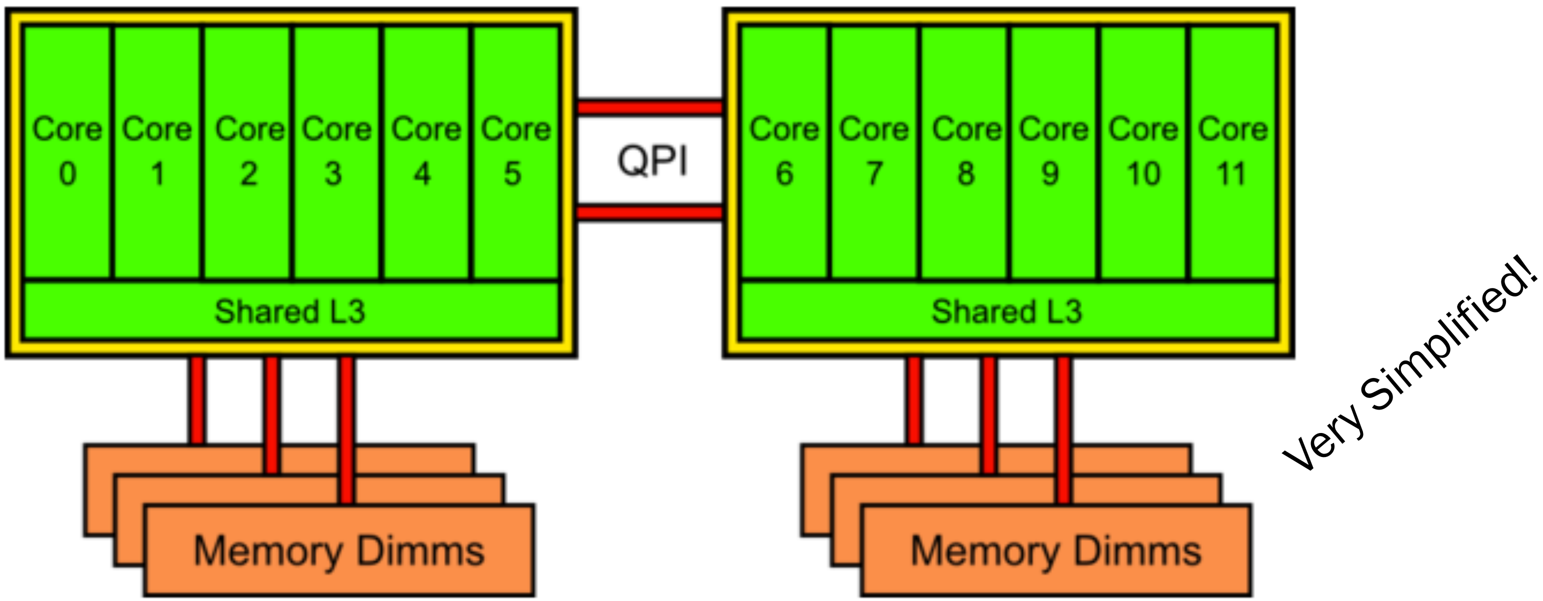
1,5,10 Event in Flight
12 algorithms simultaneously
Clone 3 most time consuming algs (1 copy per event in flight)

- Linear scaling up to 6 algos simultaneously (number of real cores)
- 10 events in flight already enough for peak performance*
- Speedup of ~7x reached (thanks to HT)
- Successful test of "one job per socket" deployment scenario.

* See backup for a complete study

Scaling on Two Sockets

- Behaviour of the application on a full Numa node is not trivial
  - Eg: remote DRAM access, cross-socket caches synchronisation…

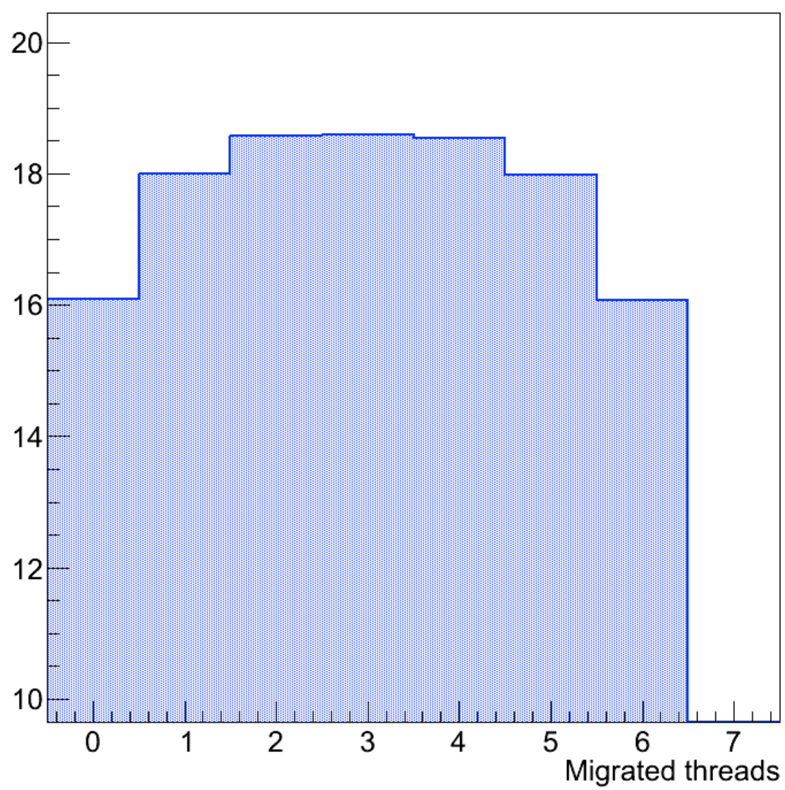


Numa Effects

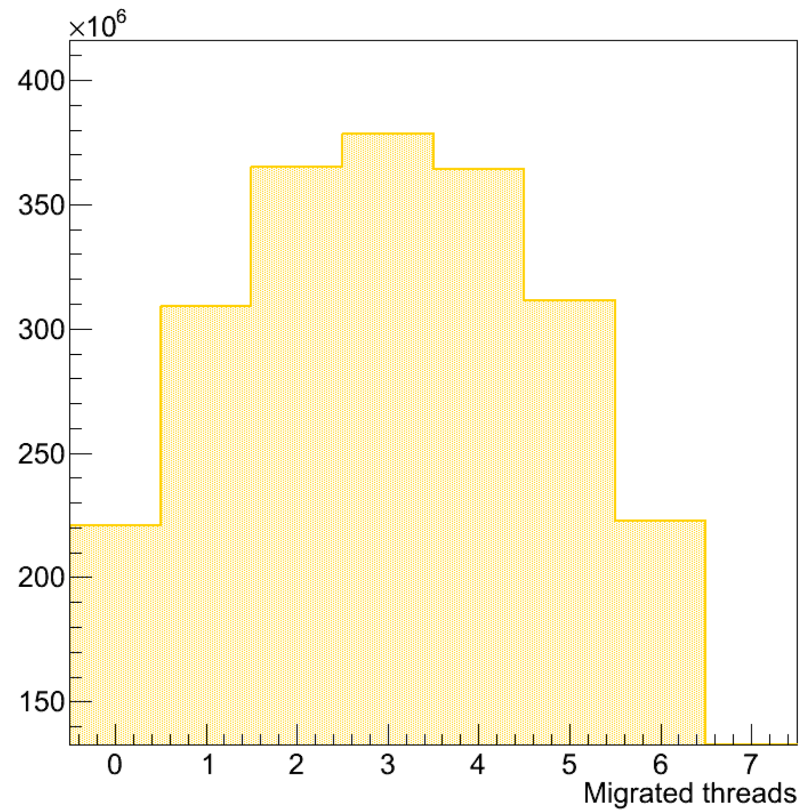
Thanks to Vincenzo I. for the fruitful discussions!!

- Run with 10 events in flight and 6 threads
- Use the "taskset" command to assign cpus to a process
- Start with 6 cpus on one socket, move them one by one to the other
- Measure event loop time and use perf to count cache misses

Evt. Loop Time [s]



Cache Misses



See backup for nice measuerements of uncore events!

Scaling on Two Sockets

- We don't understand completely the behaviour of the application (performance degradation) yet

But using the full numa node with 2 sockets is not the only possible deployment scenario!

- Runtime of one full-socket job alone on the machine and two simultaneous one-socket jobs was verified to be identical.

Along the lines of the "one job per cpu" philosophy behind our data processing since years, but with *much* less memory (even HT cores usable!)

One job per socket deployment scenario: successful

Conclusions

All required developments necessary for the Minibrunel exercise finished

- Framework: components for MT execution (Scheduler, EventLoopManager) and integration with TBB runtime
- Usercode: input declaration, thread unsafety removal, compatibility with >1 event simultaneously processed

Successfully supports concurrency within and across events

Serial and concurrent Minibrunel yield identical physics output

Concurrent MiniBrunel scales linearly on a single die (on the test machines available)

- Negligible increase of memory consumption processing multiple events simultaneously and cloning algorithms

NUMA effects not yet fully understood

- But one job per socket solution successfully tested

Plans (not a complete list)

We did a lot, but there is quite some work ahead!

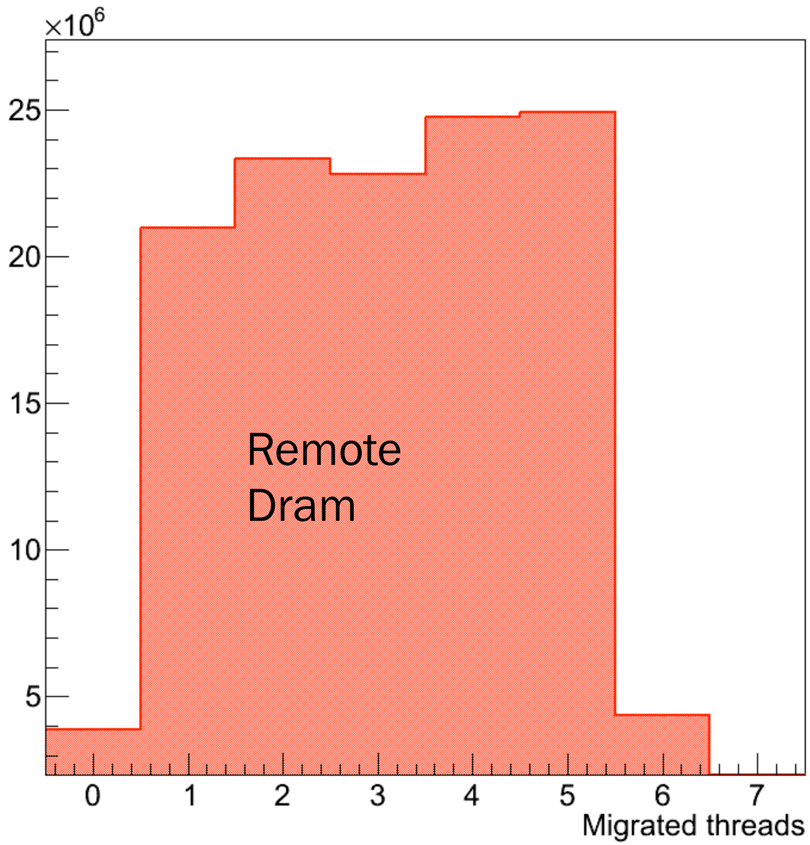
- Consolidate code and documentation for release at the end of June
- Better understand scaling on yet bigger cpus / machines
- Address all points not covered up to now: conditions changes,
- Capitalise on the accumulated knowledge: support ATLAS in setting up a reconstruction slice with concurrent Gaudi
    - Dedicated sprint 2nd half of June

Backup

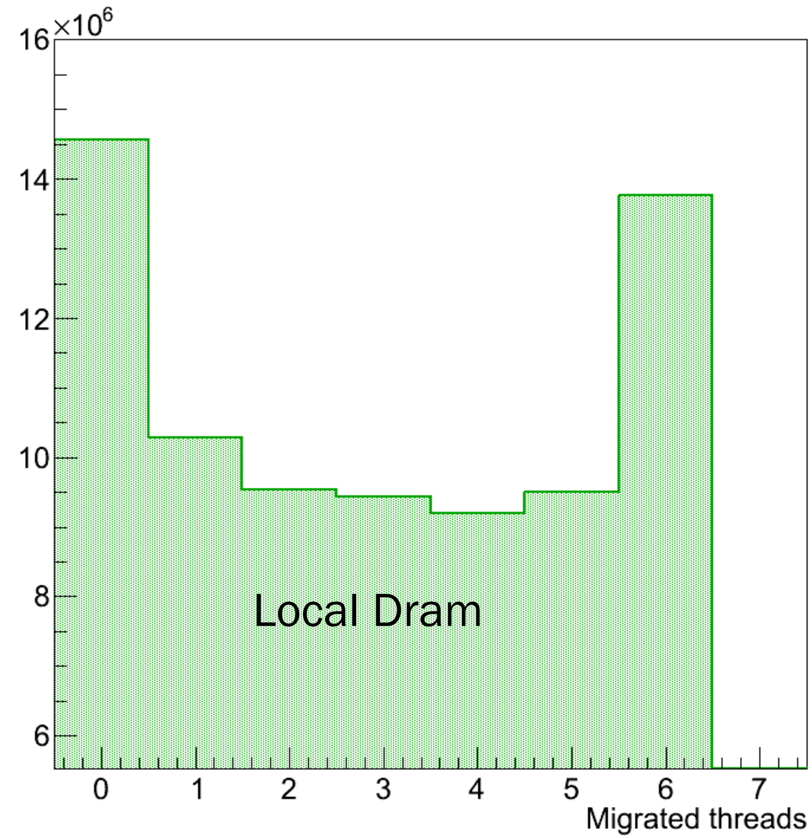
Performance Counter Analysis

REMOTE_DRAM	0x10	Load instructions retired remote DRAM and remote home-remote cache HITM (Precise Event)	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and LLC caches and was remotely homed. This includes both DRAM access and HITM in a remote socket's cache for remotely homed lines.
LOCAL_DRAM	0x20	Load instructions retired with a data source of local DRAM or locally homed remote hitm (Precise Event)	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and LLC caches and required a local socket memory reference. This includes locally homed cachelines that were in a modified state in another socket.

r100F Counter



r200F Counter



http://software.intel.com/sites/products/documentation/doclib/iss/2013/amplifier/lin/ug_docs/reference/pmbk/events/mem_uncore_retired.html

Useful Material

Project Page on the Concurrency Forum Site:

http://concurrency.web.cern.ch/GaudiHive

Main Twikipage:

https://twiki.cern.ch/twiki/bin/view/C4Hep

Git Repository Web Interface:

http://lcgapp.cern.ch/git/GaudiMT/

Jira:

https://sft.its.cern.ch/jira/browse/CFHEP

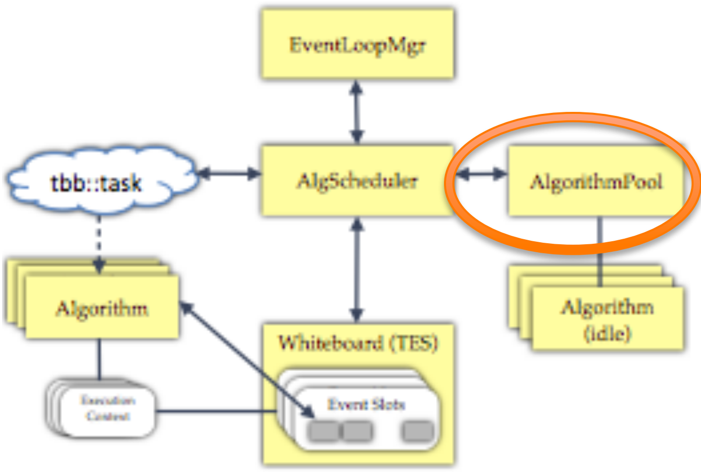
Weekly (Thursday 10:30 a.m., with phoneconf) Working Meeting Minutes:

http://sync.in/k5XvRql9y9

Algorithm Pool

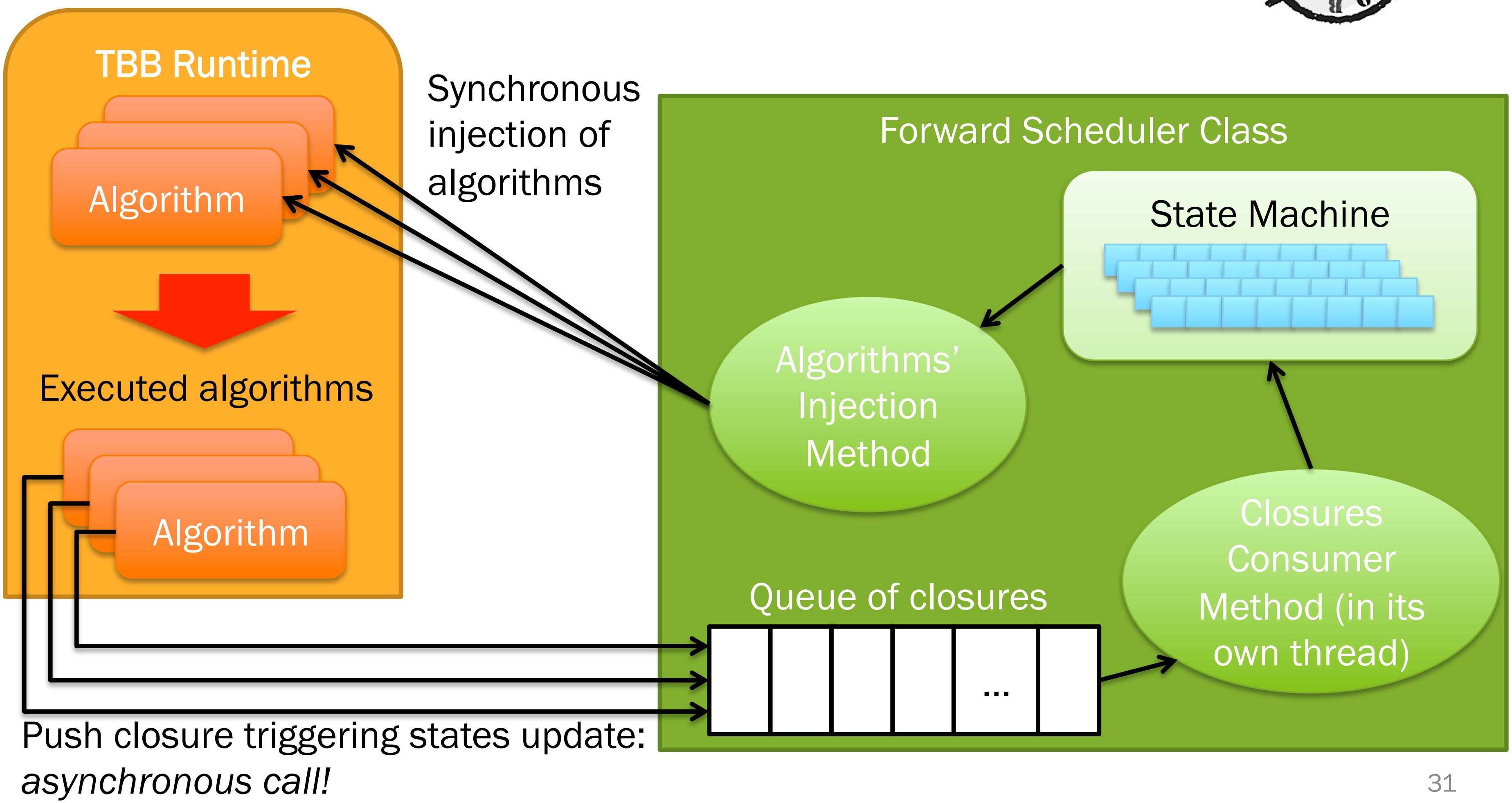
Contains algorithms and coordinate them

- Gives away instances to run, retrieves ran algorithms
- Clones algorithms (via AlgManager)
  - Number depends on code re-entrancy: non re-entrant (1 copy only), non re-entrant (use n copies), fully re-entrant (re-use same instance n times)
- "Flattens" sequencers
- Allow for exclusive resource checking: e.g. if 2 algos using a non re-entrant external library, only one at the time can run.



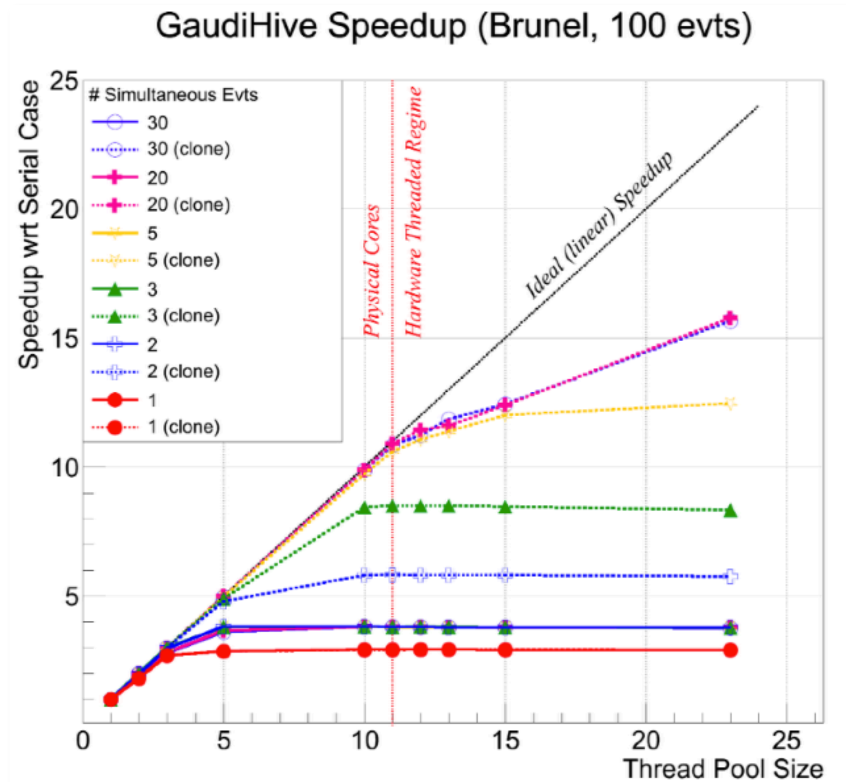
Forward Scheduler

- Component that submits to TBB runtime algorithms according to their data and control flow dependencies
- Absorb the asynchronous finishing of submitted tasks
- Update internal algorithms' state machine accordingly



The Past (2012): CPUCrunchers Demonstrator

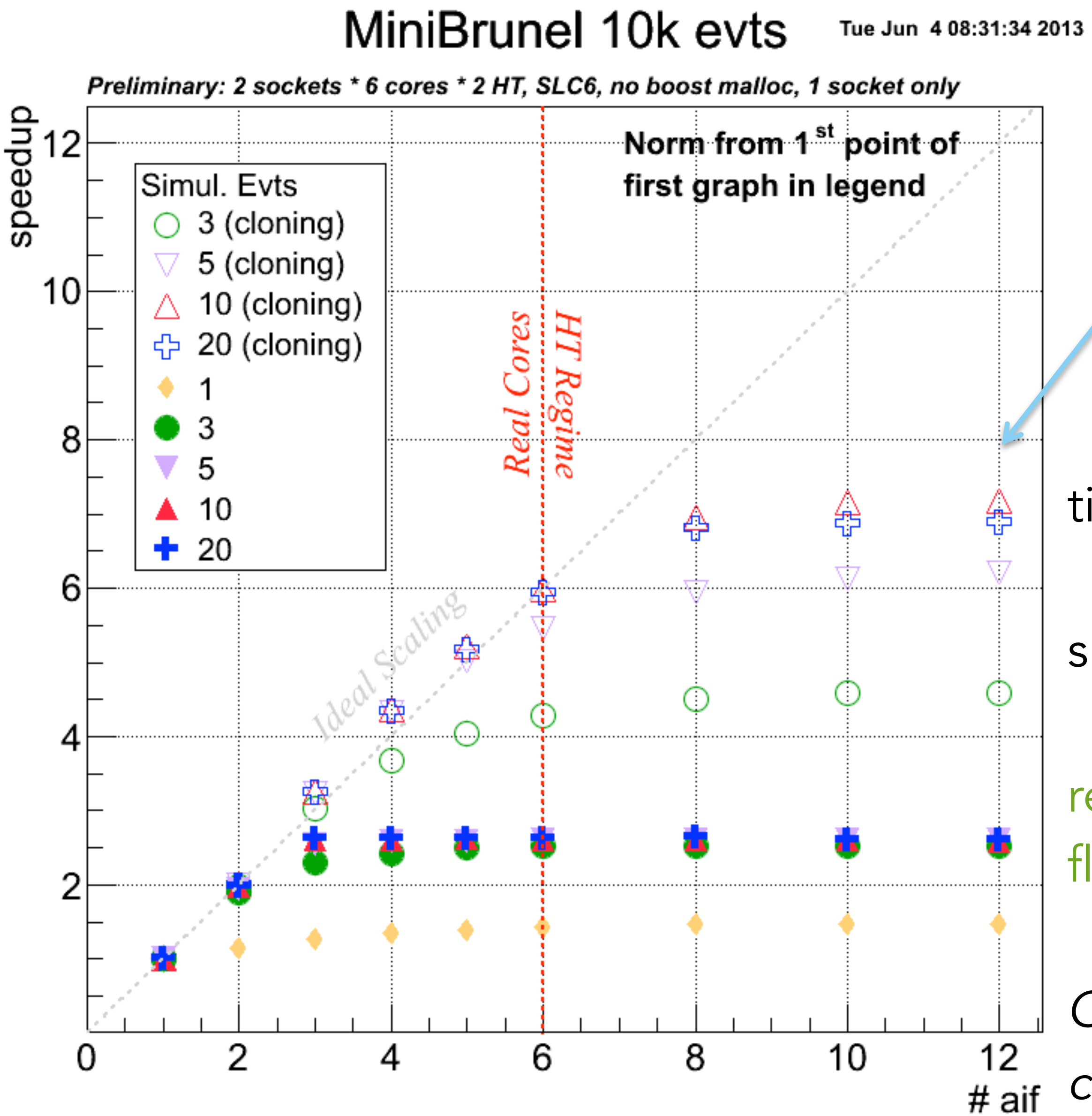
- Emulate an LHCb full reconstruction workflow with CPUCrunching algorithms (no real work done, just keep cpus busy)
- Explore expected behaviour
- Demonstrate potential of the multithreaded approach



~8 Months ago

Evolving LHC Data Processing Frameworks for Efficient Exploitation of New CPU Architectures
B. Hegner at al, IEEE-NSS 2012

Scaling on One Processor



12 simultaneous algorithms: run the application occupying the full socket

One event processed at the time: ~30% speedup

No cloning: saturate at a speedup of 2x

Cloning: ideal (linear) scaling reached with ~10 events in flight

Cloning of the 3 most time consuming algs only