# Summary of activities in Concurrency Forum

Linear Collider Software Meeting, 1st February 2013
B. Hegner, P. Mato, D. Piparo

# The ‘concurrency’ Forum

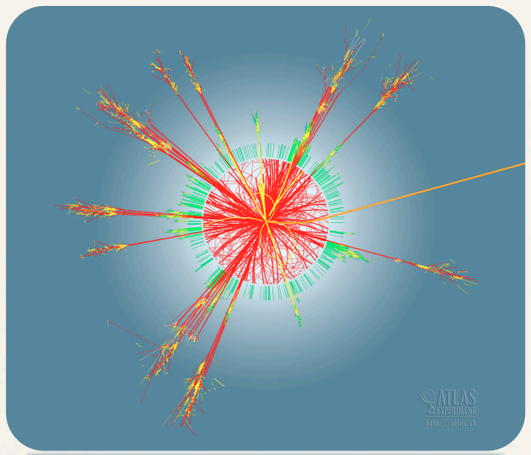
- HEP software needs a paradigm shift
  - Hardware architectures (since years) more and more suited to support parallel programs
- Assist scientists to express parallelism in their applications
  - New programming models
  - Specialized software frameworks
  - Most recent software technologies
- Forum on Concurrent Programming Models and Frameworks
  - Meeting every two weeks
  - Boost knowledge sharing process
  - Find common minimal set of technologies
    - Code sharing, result comparisons

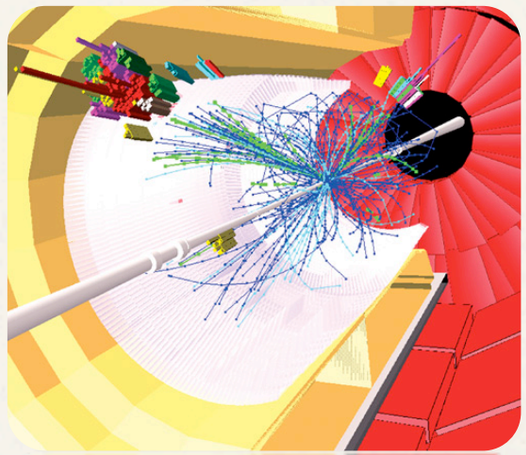
Topics of the Forum for today:

- Parallelism in detector simulation
- Heterogeneous computing
- Memory and parallelism
- Parallelism in algorithms and frameworks
- Whiteboard prototype in Gaudi

http://concurrency.web.cern.ch

# Parallelism in Detector Simulation





# Geant4 MT

- Simulation computationally expensive for the experiments
  - Big weight in a Monte Carlo samples generation campaign
- What is Gean4-MT?
  - 1 event per thread (Posix threads)
- Requested to accommodate parallelism of CMS and ATLAS frameworks:
  - Unit of work: event or single track (ATLAS ISF)
- Geant4-MT will be included in Geant4 10 (end 2013)
- Good scaling achieved up to 40+ workers
  - Hardware Threading: +25% !
  - 1 worker case w.r.t. serial version: 18% penalty
- Validation of results is an interesting challenge!

Example of a large codebase written by physicists going parallel !!

J. Apostolakis et al.

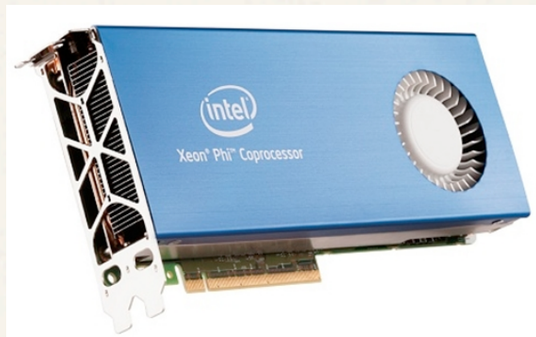
# Geant4 Vector Prototype

- Longer term and ambitious project
  - GPU, accelerators exploited?
  - Data locality and vectorized algorithms
- Idea: lump particles from different events in baskets
  - Within the same volume (same material), transport them together
  - New design of data structures needed
- Non blocking – No communications among threads
  - A lot of intelligence in the design of the scheduler
- Plans:
  - Introduce hits, digitisation and I/O
  - Introduce realistic EM physics
  - Investigate GPUs

- Input collection across events
- Multidimensional Parallelism in the design:
  - Within the core (vectorisation)
  - Among cores (multithreaded)

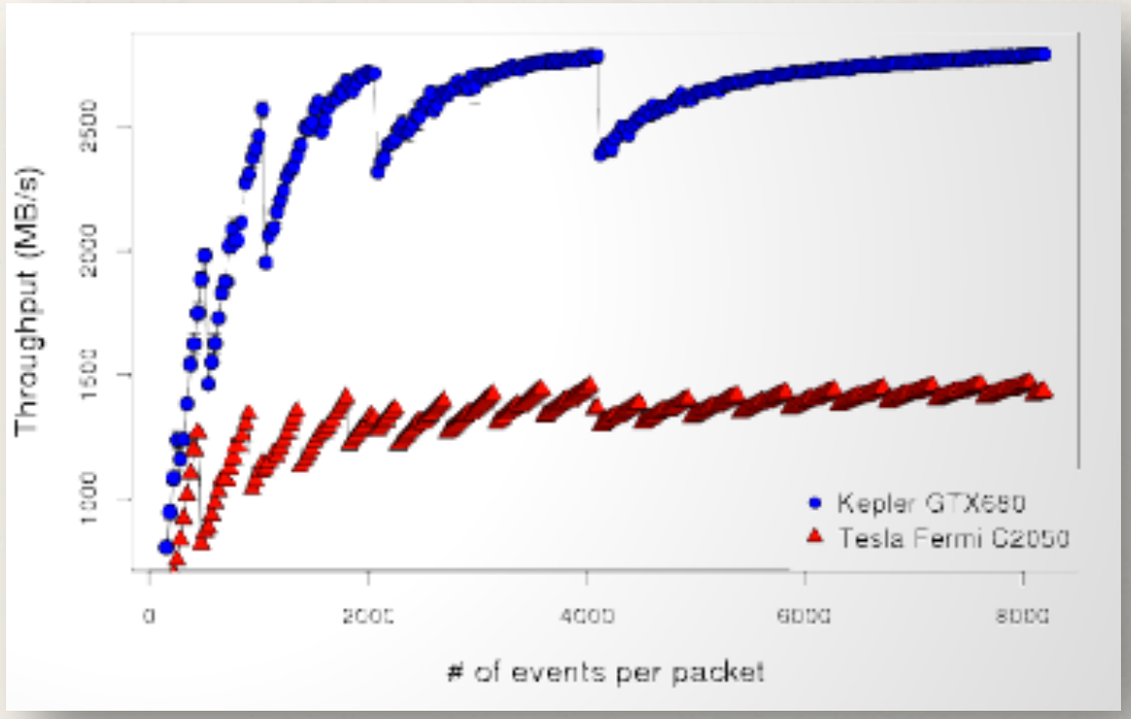
A. Geatha et al.

# Heterogeneous Computing

# Real-Time GPU usage in NA62

- Idea: use GPU as L0 RICH trigger
  - Ring finder - Crawford algorithm
  - Nvidia hardware and CUDA
- Pilot project (FPGAs online now)
  - Very promising R&D
- **Memory transfer is an issue**:
  - Use CUDA streams to build a pipeline
- Necessary throughput achieved:
  - At least 2x the max achievable by detector!
- Latency completely under control
- Ready to be tested during Technical Run in November!



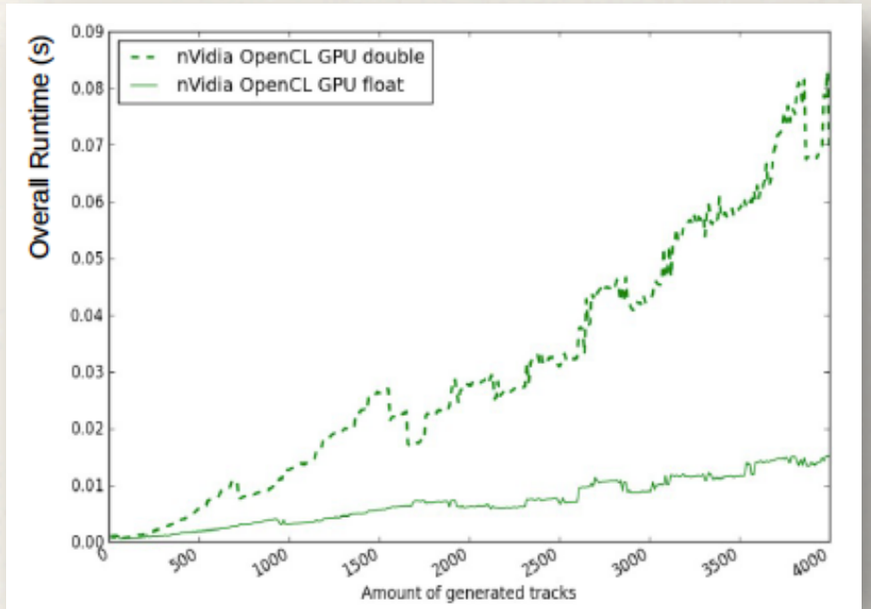

Max Th. Throughput: 0.5 GB/s!

A real (very specialized) system driven by GPU computations ready to be tested in production conditions

F. Pantaleo

# GPUs for CMS Tracking

- Technology: OpenCL
  - Compared with TBB and OpenMP
- Real-life tracking algorithms and inputs
  - Trajectory alteration by multiple scattering
  - Track seeding (simplified geometry)
- Bottom line:
  - Fast code and good scaling
    - Many platforms (Intel,AMD)x(nvidia,AMD)
  - Code is portable
    - Same kernel on all platforms (extrapolate on MIC,..)
  - Design/Development effort required if decide to port CMSSW components
  - Data transfer overhead is an issue:
    - Send to devices input from several events
  - Integration with today's CMSSW not easy




Track seeding

T. Hauth et al.

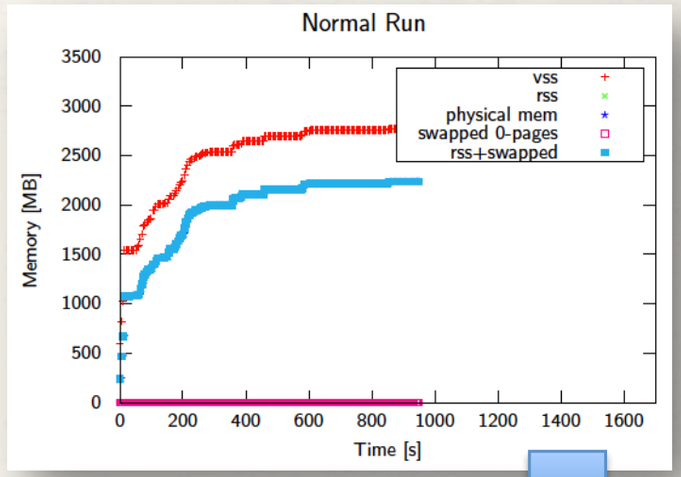
- Real algorithms ported: precious experience gained, good performance, portability.
- Transfer and offload are still open question

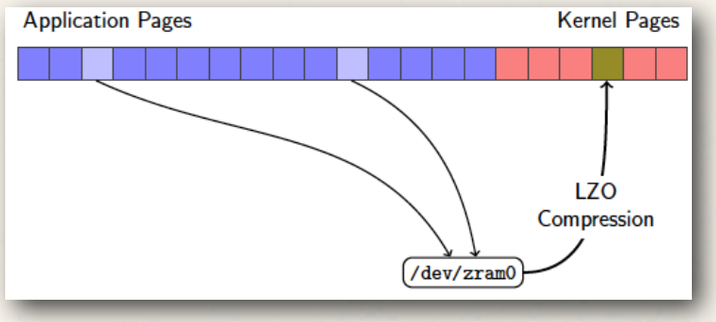
# Memory and Parallelism

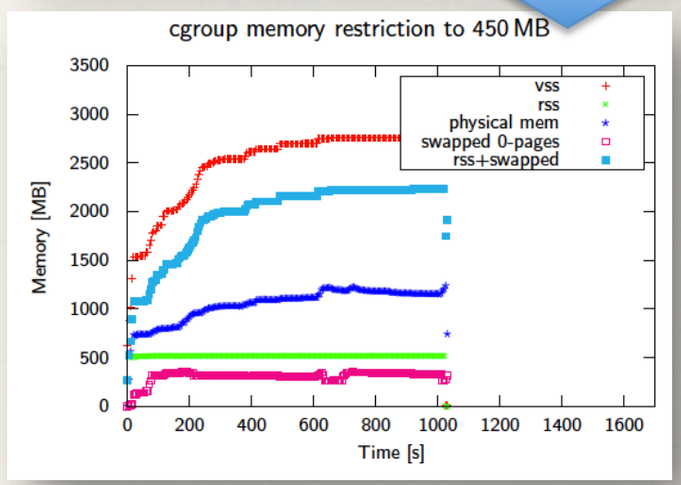
# Memory Saving Techniques

- Large memory footprint of our applications
  - Available and purchased resources idle, e.g. HT
- Idea: compress (part of) memory to drain performance
  - Swap to "compressed block device" whenever possible
  - Trade-off
- Example AliRoot:
  - Reconstruction of pp events
  - Limit RSS from 2.2 GB to 1.2 GB
  - ~8% performance CPU penalty
- Unexpected: large amount of contiguous 0-pages found!
  - Present in CMSSW, AliRoot, DaVinci – Athena?
- Advanced tool developed to track down culprits:
  - Trap all memory allocations
  - Investigation on-going

J. Blomer et al.




- Useful for farm management
- Progress in understanding memory allocations

# KSM Studies with GaudiMP

- KSM kernel module tries to merge identical (virtual) memory pages into a single (physical) one across processes
- Nice memory savings in realistic LHCb applications

| | serial mode | 2 workers | 4 workers | 8 workers |
|---|---|---|---|---|
| Gauss | 183 MB ( 22 %) | 623 MB (33 %) | 1275 MB (42 %) | 2659 MB (48 %) |
| DaVinci | 190 MB (10 %) | 600 MB (17 %) | 1577 MB (24 %) | 3315 MB (27 %) |
| Brunel | 94 MB ( 10 % ) | 465 MB (23%) | 1112 MB (32 %) | 1900 MB (31 %) |

Memory savings in absolute (percentage w.r.t no KSM)

- Caveats
  - Merging rate must be adapted otherwise high CPU consumption by KSM-thread
  - KSM does not work on the level of virtual memory
  - pages_volatile (changes too fast) becomes likely a bottleneck
  - madvise-call inside application

# Transactional Memory (TM)

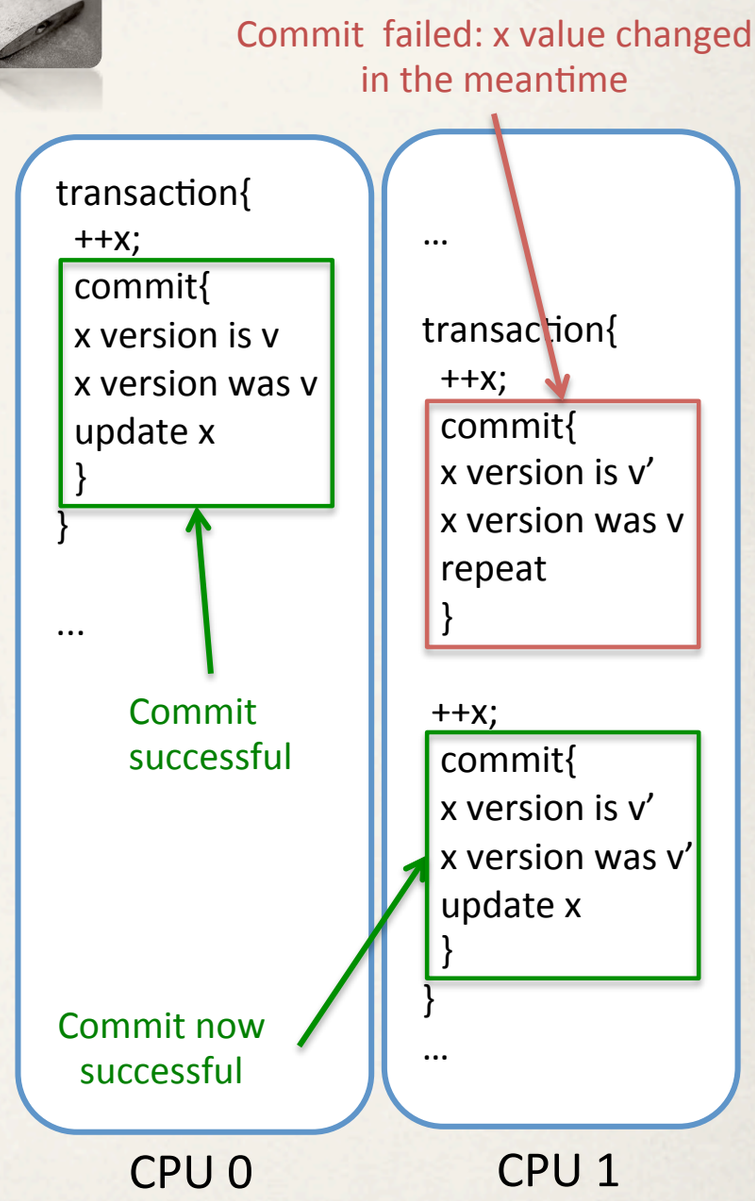
- TM tries to address 'locks' pathologies: deadlocks convoying , priority inversion, do not compose…

  - Supported by GCC 4.7

```
void f(){
__transaction_atomic{ ++a;}
}
```

- Idea: promote code sections to transactions. If state of variables changes during transaction (~collision), roll-back and retry
- Concurrent queue implementations studied:
  - Home made using TM (by a summer student)
  - Intel TBB
- Out of the box performance:
  - Already comparable!

Atomics + transactions + scheduler:
Locks usage need greatly reduced

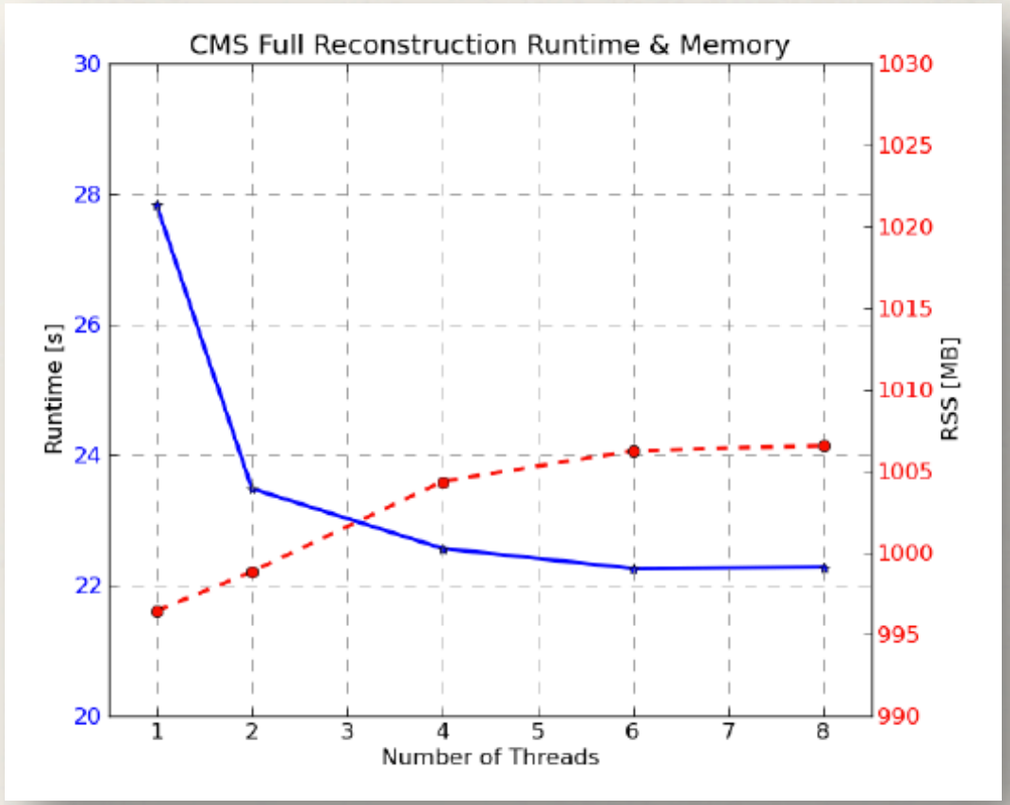



A. Pakalniskis et al.

# Algorithms and Framework Parallelism

# CMS Parallel Track Seeding

- Track seeding: from hits pairs, find triplets
  - Start then track building
- Cost: 10% of reco time with 40 pile-up evts
- Within algorithm
  - Parallelize for loop on pairs
  - Preserve ordering
  - Threading Building Blocks (TBB) technology
- Changes needed in the Framework:
  - Add simple TBB service to hold thread pool
  - Atomic reference counting
- Ready for production: fully validated
  - Good scaling for speedup
  - Almost no memory footprint increase
  - Way to get back unused resources
  - Shrink runtime of long-running algorithm

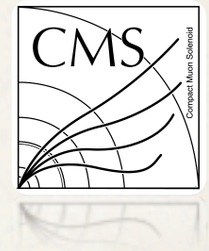



- Production ready parallel algorithm
- No cost in RSS
- Few 'framework' changes needed

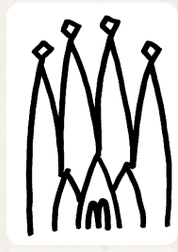
T. Hauth et al.

# ‘Concurrent’ Frameworks

- Three products:
  - SuperB FW
  - CMSSW
  - GaudiHive (‘Whiteboard’ prototype integrated in Gaudi)
- Common denominator:
  - TBB technology
  - **Event level parallelism**
    - Multiple events processed simultaneously
  - **Algorithm level parallelism**
    - Multiple algorithms running concurrently
  - “Task-oriented” programming model
    - Central scheduler managing algorithm execution
    - Allows transparent execution of parallelized algorithms
    - Adopted by CMS and GaudiHive

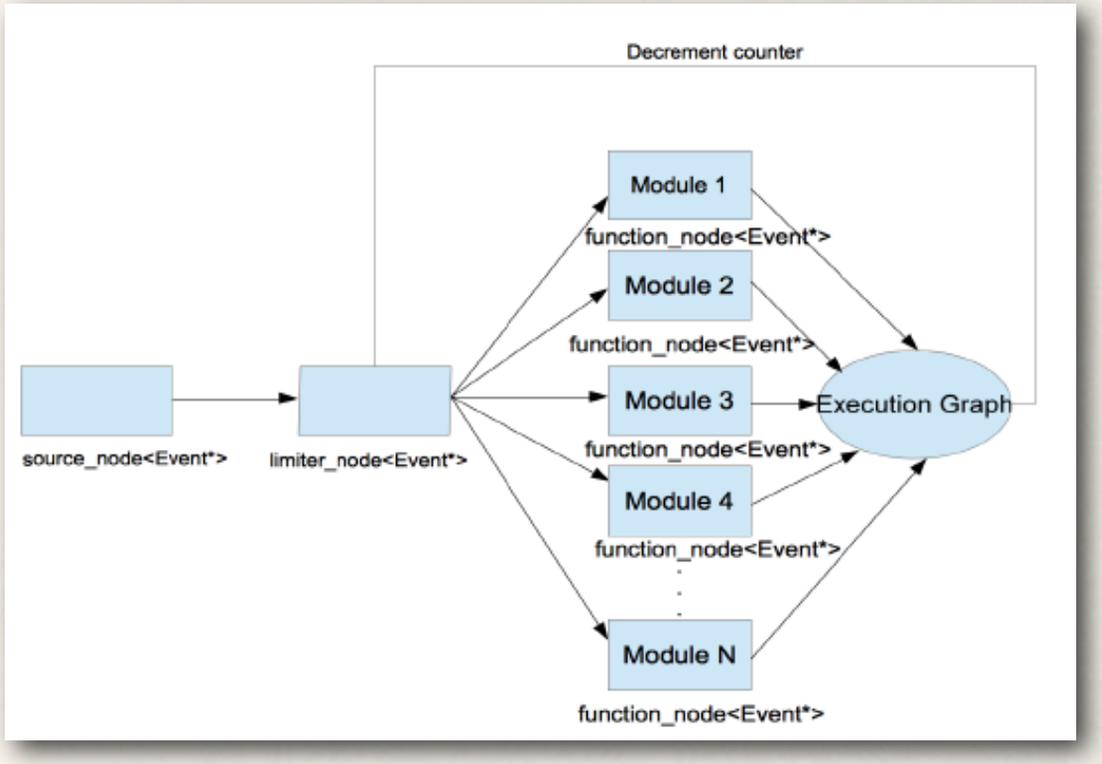



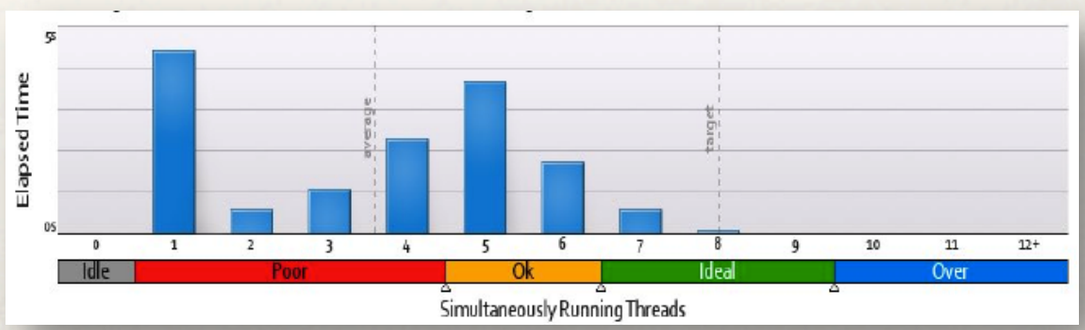


# Concurrent Frameworks: SuperB

- Prototype built on top of serial BaBar framework
- Fastsim worflow considered
  - 10% of time spent in modules concurrently runnable
- No scheduler
  - Use a priori computed flow graph
- Protect modules with locks
- Bottom line:
  - Real simulation code ran
    - Speedup achieved
  - Limited parallelism
    - Locks
    - Nature of the workflow
- Leave prototype
  - Move to natively parallel framework

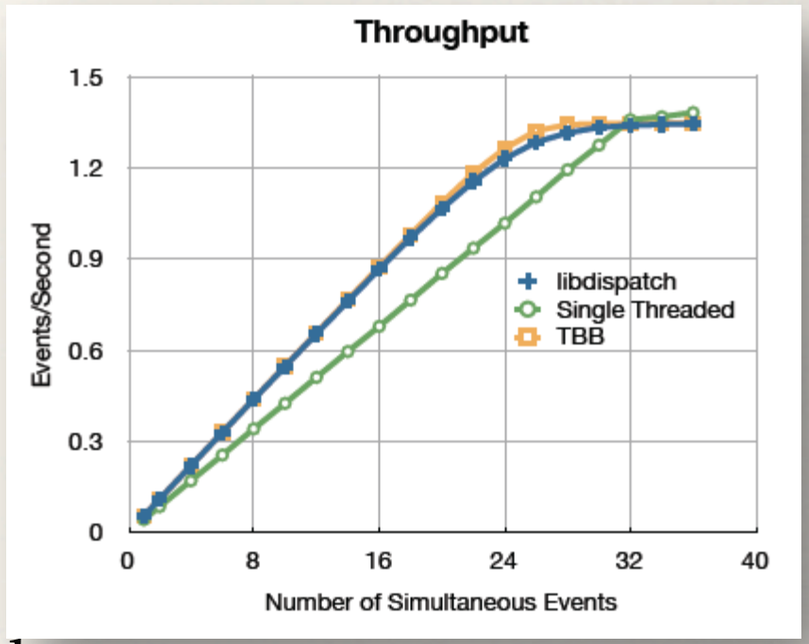


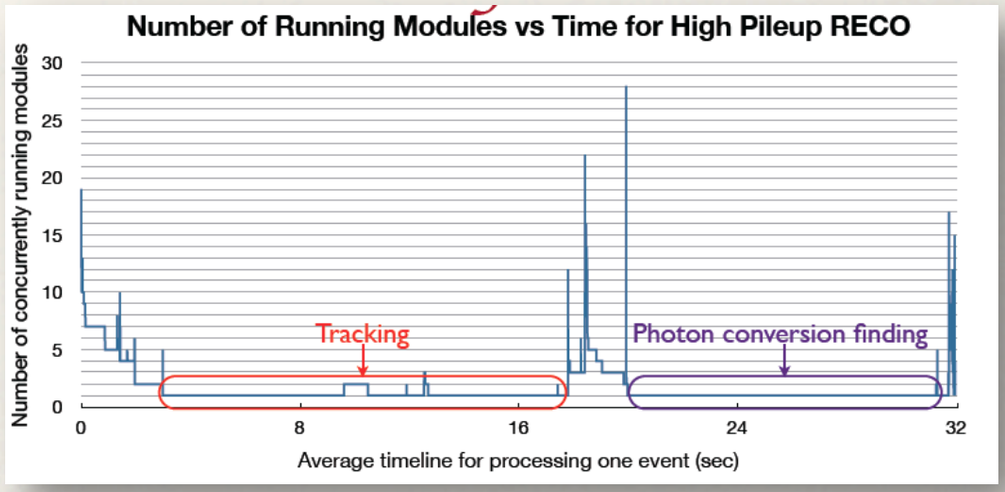




M. Corvo

# Concurrent Frameworks: CMS

- Toy framework
  - No real algorithms but CPU crunchers
  - Timing of real workflows reproduced
- Data on demand driven scheduling:
  - Input needed by algorithm triggers algorithm scheduling
- Large portions do not allow parallelism among modules:
  - Shorten them with parallelism within modules!
- Campaign to make CMS code thread safe already started
  - Clang static analysis
  - Spot thread unsafe constructs



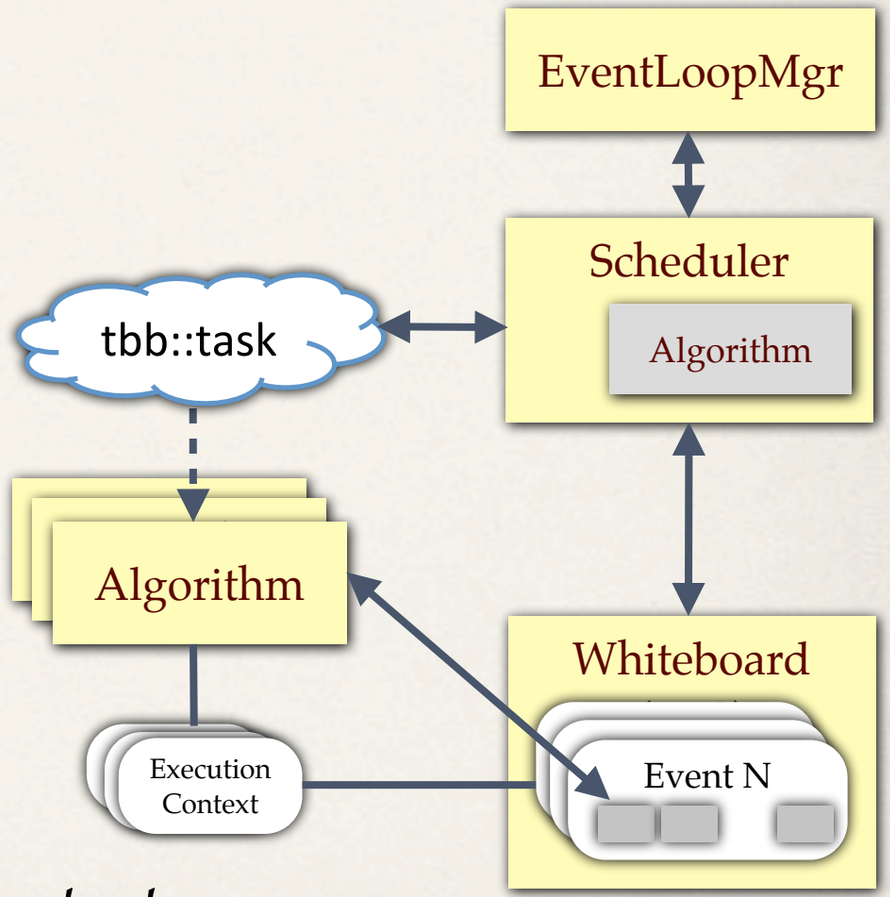




- CMS is moving towards a parallel framework
- Forking is not enough

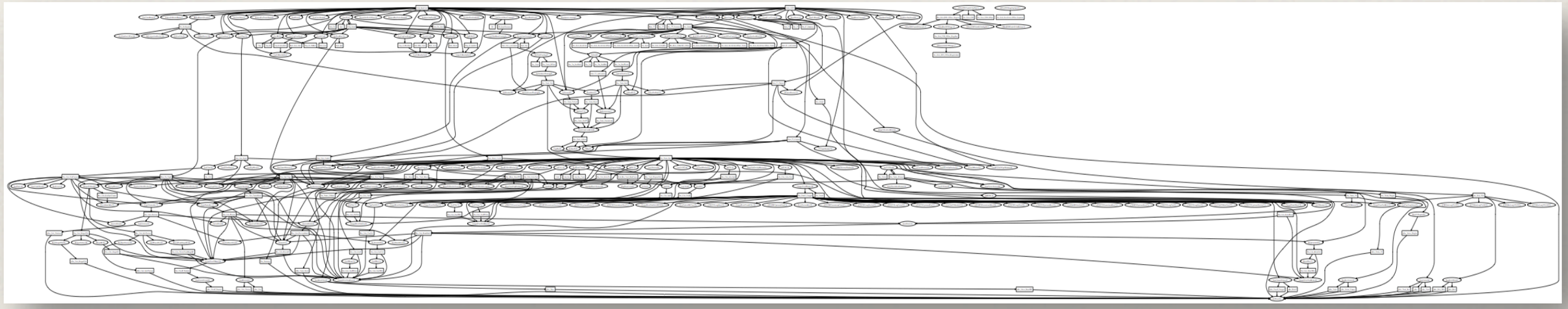
C. Jones

# Concurrent Frameworks: Gaudi

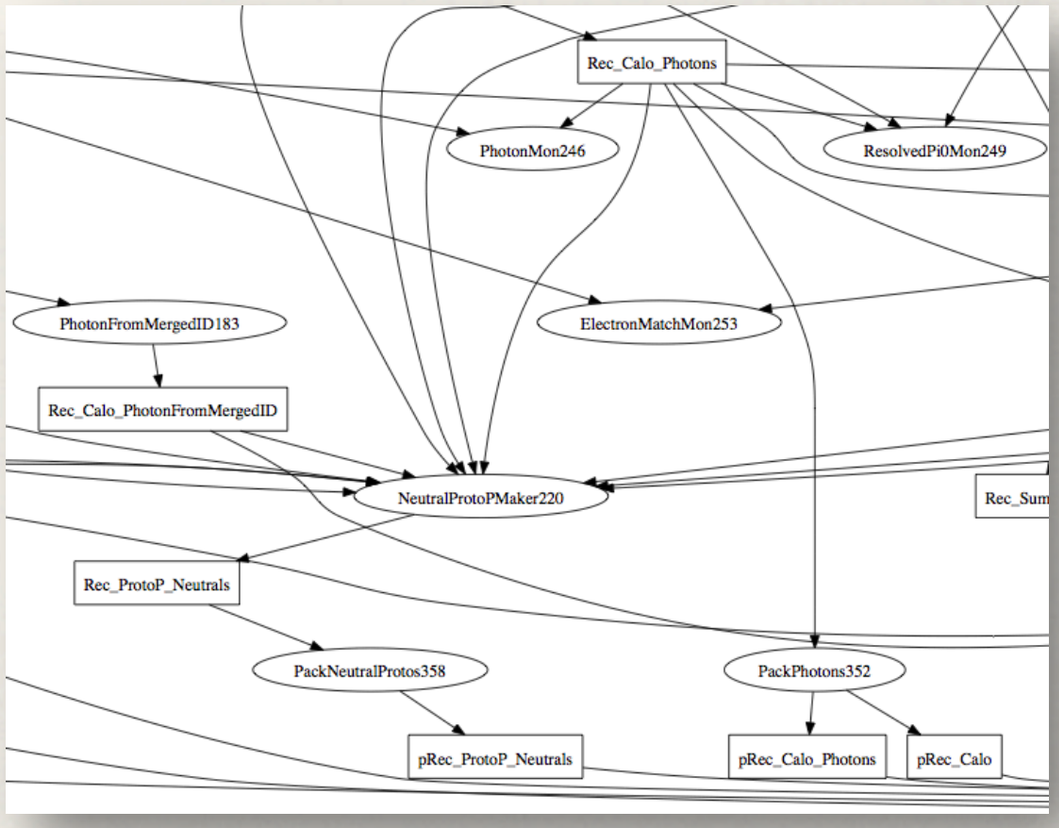
- Toy framework
  - No real algorithms but CPU crunchers
  - Timing of real workflows reproduced
- Schedule algorithm when its inputs are available
  - Need to declare *Algorithms'* inputs
- Multiple events managed simultaneously
  - Bigger probability to schedule an algorithm
  - Whiteboard integrated in the DataSvc
  - Which was made thread safe
- Several copies of the same algorithms can coexist
  - Running on different events
  - Responsibility of AlgoPool
- Data specific to execution stored in the *execution context*
- Used *TBBMessageSvc* for logging



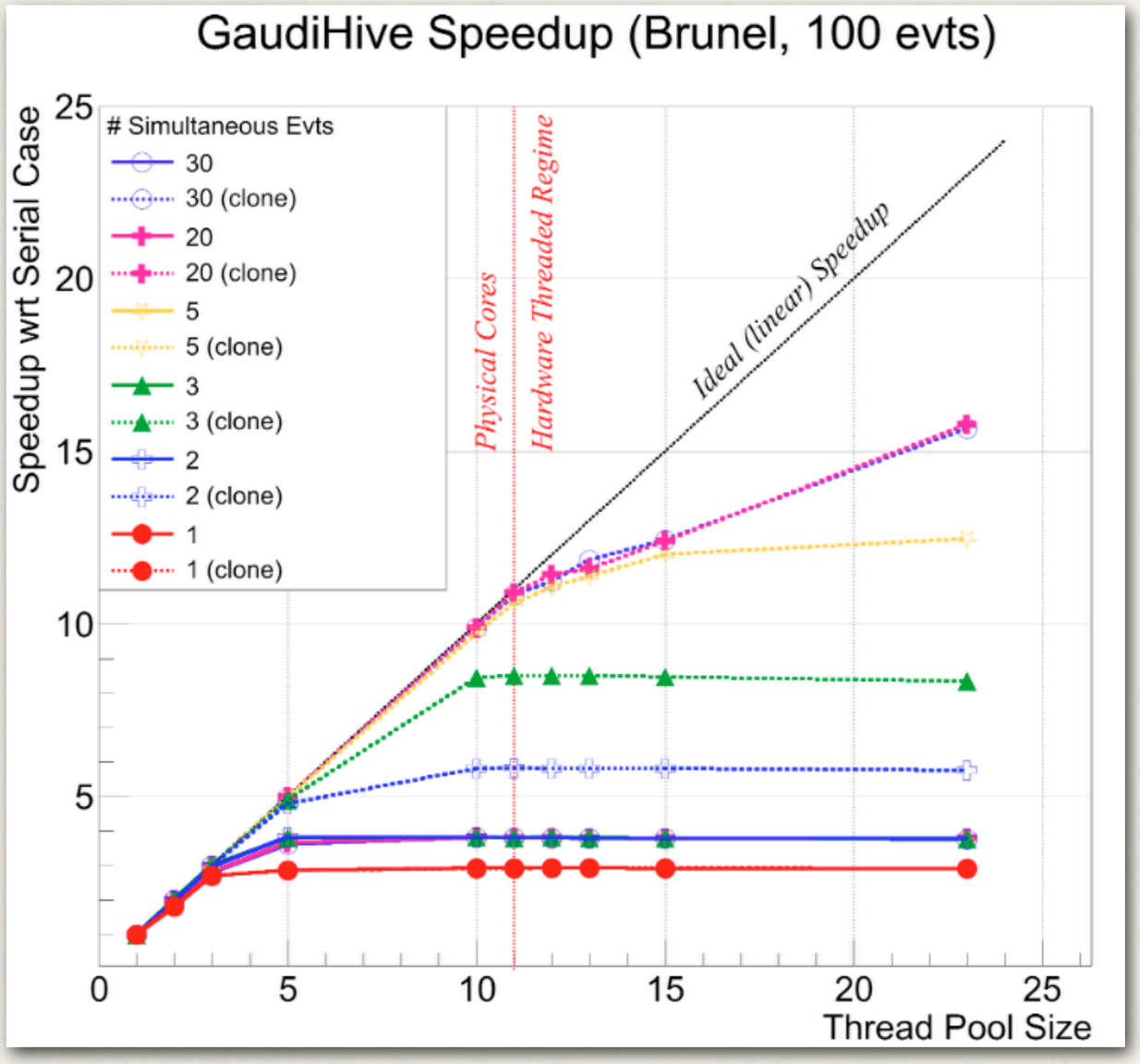
# Test On Brunel Workflow



- DAG of Brunel (214 Algorithms)
  - Obtained from the existing code instrumented with 'Auditors'
  - Probably still missing 'hidden or indirect' dependencies (e.g. Tools)
- Can give an idea of potential for 'concurrency'
  - Assuming no changes in current reconstruction algorithms



# Test On Brunel Workflow

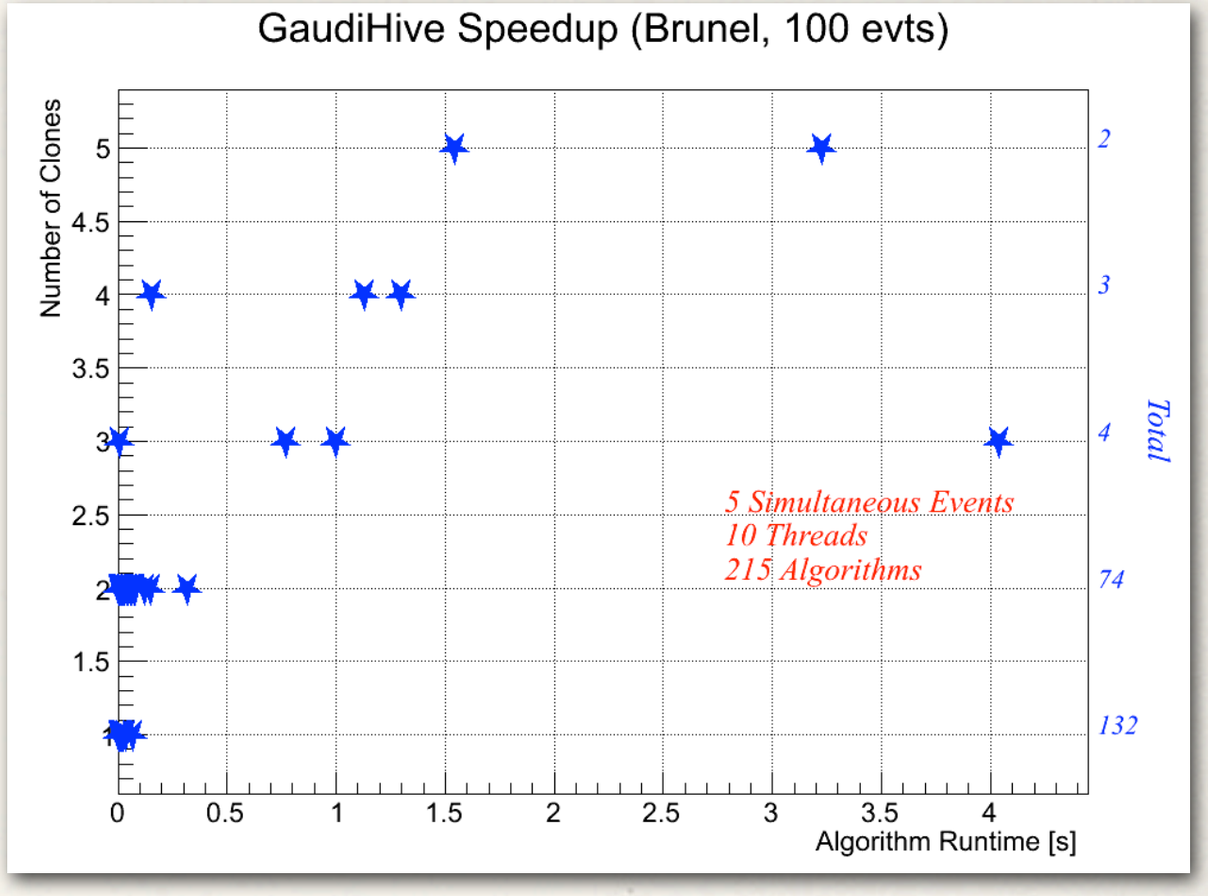



Test system with 12 physical cores x 2 hardware threads (HT)

- 214 Algorithms, real data dependencies, (average) real timing
  - Maximum speedup depends strongly on the workflow chosen
- Adding more simultaneous events moves the maximum concurrency from 3 to 4 with single *Algorithm* instances
- Increased parallelism when cloning algorithms
  - Even with a moderate number of events in flight

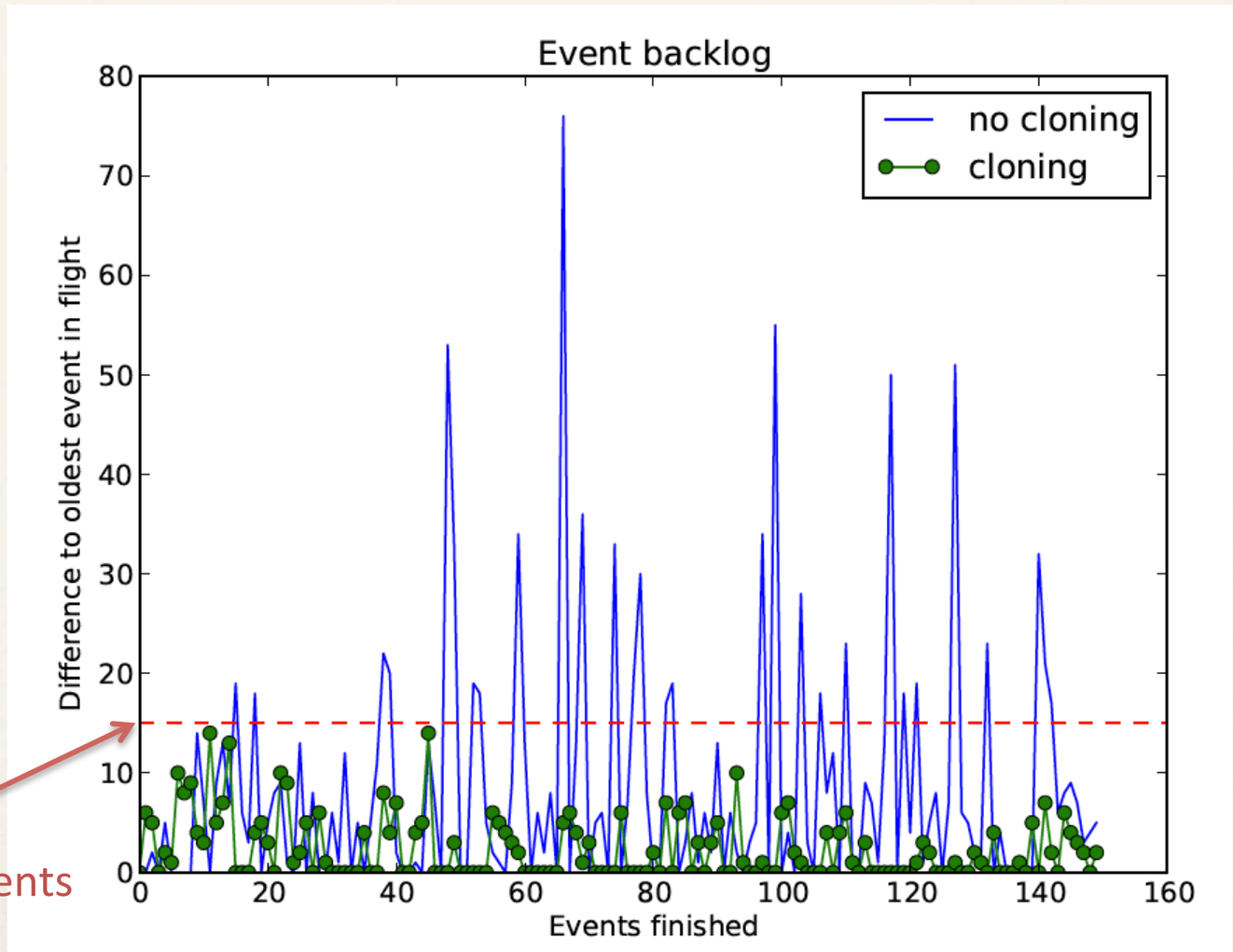
# # Clones vs. Runtime

- Tested strategy
  - Cloning an Algorithm each time can be scheduled and all instances busy on other events
- Long running algorithms end having multiple clones
  - Easy solution but we need to worry about statistical outputs (counters, histograms, etc.)
  - Alternatively, these are candidate algorithms to be parallelized
- A high number of short algorithms have 2 copies
  - Probably we should forbid multiple copies for those



# Event Backlog

- Event backlog: difference between latest event put in flight and oldest event being processed
- Cloning helps maintaining a little event backlog.




Number of events in flight

# Concurrent Gaudi: Status

- A prototype of a concurrent Gaudi (**GaudiHive**) has been developed as an evolution (new branch in the git repository)
  - Able to schedule and run **algorithms concurrently**
  - Able to run **multiple events simultaneously**
  - Friendly with **sub-event parallelism** if using TBB (not really tested yet)
- So far has been tested with `fake' BRUNEL reconstruction workflow:
  - Important speedup already been obtained, not yet the optimal
  - *Algorithm* cloning increase parallelism, keeps events backlog low
- Test bench to exercise timings and dependencies for other applications:
  - CMSSW reconstruction workflow (already there)
  - ATLAS (waiting for inputs)

# Concurrent Gaudi: Plans

- Continue the investigation on problematic Gaudi elements
  - For example *Services*, public *Tools*, *Incidents*, etc.
- Provide options for their upgrade to be thread-safe
  - Multiple copies+merge?
  - Locked-gateway?
  - Synchronizing queues?
- Strategy: start running real 'physics' algorithms
  - Start with subset of LHCb reconstruction (~30 algorithms)
  - Understand and find solutions to the problems
  - Validate results
  - Extend to full workflow later

# Conclusions

- The Concurrency Forum:
  - Stimulated big enthusiasm of the community!
  - Infrastructure started to deliver
- Important results achieved and knowledge shared in the field of:
  - Parallel simulation
    - Present: Geant4-MT, future Geant Vector Prototype
  - Heterogeneous computing
    - "Different" multicore systems
  - Studies of memory in the field of multicore applications
    - Potential of TM, various memory saving techniques investigated
  - Common technologies
    - TBB is an example, tools and procedures (not shown here for brevity!)
- A clear trend emerged for the future of HEP data processing
  - Parallelism within the algorithms
  - Parallelism among algorithms
  - Parallelism among events
- CMSSW and Gaudi already evolving in this direction