Parallelism and Control

Hiving complex Algorithms and managing logic flow

Wim Lavrijsen

GaudiHive for ATLAS

- GaudiHive is a multicore version of Gaudi that:
  - Manages multiple events
  - Manages algorithms concurrently
    - Requires re-entrancy or the ability to clone Algorithms
- ATLAS has many Algorithms of some complexity
  - Structured by means of AlgTools through interfaces
    - Duration and resources needed not strictly known until run-time
    - Largely invisible to the framework (modulo creation/destruction)
  - Often not re-entrant → cloning needed?
    - May not be possible and may not be a good idea
      - E.g. public tools with large resources should remain shared
  - Many use memoization strategies
    - Including for (event duration) life-time management

Additional Challenges: Hardware

- Main bottlenecks in Athena codes
  - Memory (strongly dependent on type of job)
  - L1 I-cache: est. loss ~30-50% (OOP, shared libs)
- Solutions (for both today's and future hardware):
  - Lower memory use
  - Greater instruction locality
  - Greater data locality
  - Improve software organization

} strongly related

- Et tu, GaudiHive?
  - Might be able to help lower memory use per event
  - Wrong granularity (too high-level) to fix locality

Why Care About Locality? Example: Xeon Phi (MIC)

- Issue much more limiting than on Xeon
  - Same 32KB L1 I-cache, but shared by 4 threads
  - Half # iTLB entries, again shared by 4 threads
  - Issue of bundled (in-order) instructions
  - No same-thread back-to-back issue
    - Yet, threads still time-muxed: need minimum 2 threads/core
- Limited to max 8GB/60 cores (model-dependent)
  - Yet, 60 x *deep call stack* (x 4 threads) == lots of waste
  - Deployment model takes up on-device memory
    - A single function use can pull in a large, fully mapped, .so

=> Hits every bottleneck for typical Athena jobs hard
=> Not MIC-only: generally true on small-core architectures

Threads?

- Attach themselves to the wrong resources
  - Bottlenecks already exist for *single* thread on *big* core
  - Small core hits even harder on existing limitations
  - Threads compete for the bottleneck resources
- Do not utilize new resources; e.g. for MIC:
  - 512-byte wide registers, vector-, and mask-operations
  - Coherent L2 D-cache for fast data communication
- Instead, good instruction/data locality is needed
  - Once established, threads can follow more naturally

=> Threads *require* clear data and logic flows with good locality for good performance, they do not *provide* them

Hierarchical Solution Needed

- Approach with different solutions on multiple levels
  - Event/Algorithm-level parallelism by GaudiHive
    - For operations on different resources and/or different durations
  - Instruction/data-level parallelism in inner loops
    - For same operations on same data
      - Solve data locality, implement vectorizations
      - Enable fine-grained parallelism
      - Enable off-loading to a co-processor
- Resource management with an overall task pool
  - TBB being the most popular; C++ AMP?
  - C++14? C++11 (on Linux) too close to POSIX

=> Requires restructuring of complex Algorithms, which requires good input to fit components in their proper place

Decisions, Decisions

- Choices for complex Algorithms & their AlgTools:
  - Break up/promote parts into multiple Algorithms
    - Then open for GaudiHive to schedule and clone
  - Leave structurally in-place, but make re-entrant
  - Leave structurally in-place, but make clonable
  - Leave structurally in-place, but control access (locks)
  - Coalesce down into single code sections
    - With fully open/transparent data flow
    - Implement fine-grained parallelism on inner loops
    - Might involve EDM changes

  => Except for the last step, this leaves physics code as-is, with restructuring at the component level only!
- How can the framework help drive decisions?

Utilizing GaudiHandles

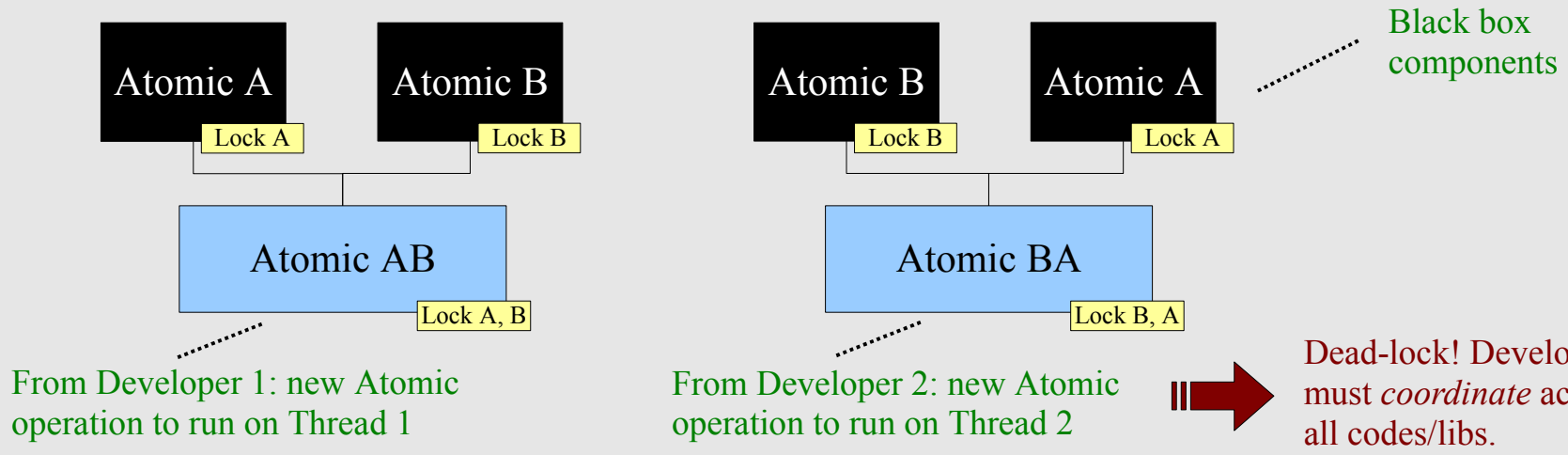
- Framework is rather blind to AlgTool uses ...
  - Only show up on creation/destruction (ToolSvc)
- … GaudiHandles provide a look into logic flow
- *Caveat*: handles can make logic safe, not data flow
  - Data flow usually consist of multiple logic operations
    - E.g. create new container, put into StoreGate, use container
      => Would require *transaction semantics* (another talk ...)

=> Want to keep *a working application*, while:

- Finding points of congestion and missed parallelism
- Working on that single point of interest only
- Retaining ability to retrace steps or fully revert
  - Think *compilers flags* to go GaudiHive ↔ Athena

Threads and Components: (Non-)Composability

- Complex Algorithm build up with AlgTools
  - Combines AlgTools on declared interfaces only
  - Lock-based atomic operations may not compose



- Deterministically detectable with hierarchical locks
- Solvable by acquiring all locks in order or a priori
  - Proper order; with global lock; or use of std::lock()

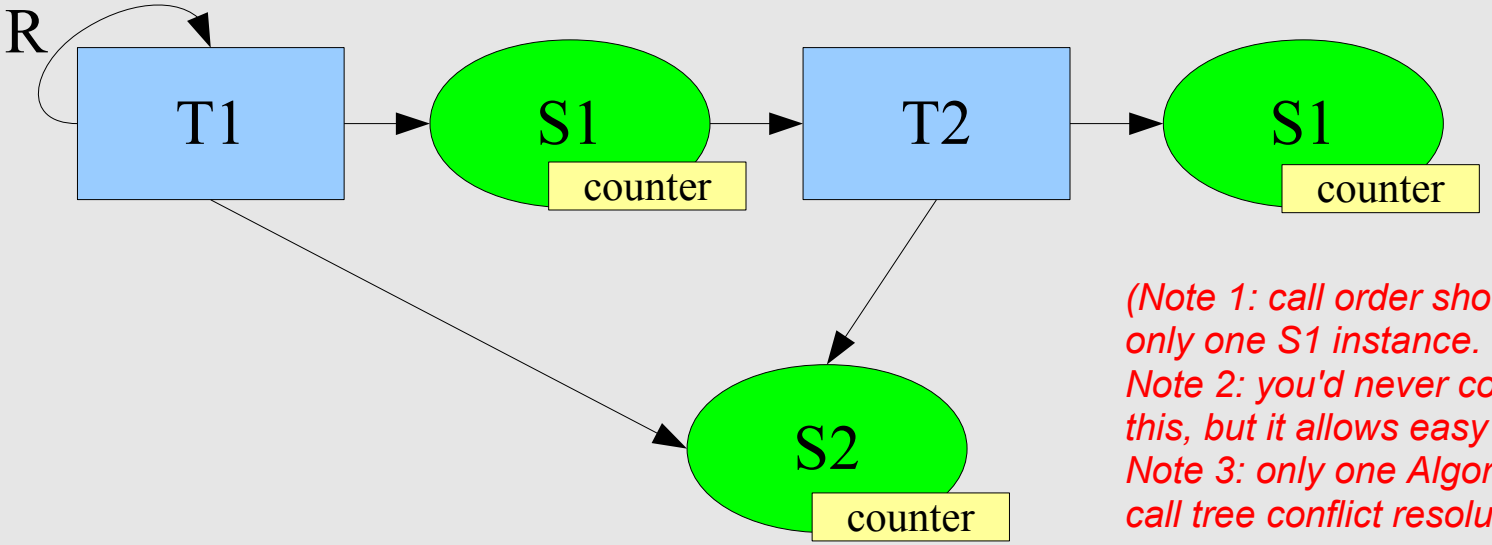
Could the framework do this automatically?

Automatic Fine-grained Locks

- On first use, trace the call-tree(s) of handles
  1. Globally lock framework during tracing
  2. Collect and record all handles seen
  3. Fit results into global order (if possible; diagnose if not)
- On subsequent calls:
  1. If order matches global order, acquire/release lazily
  2. Otherwise, acquire greedily, do not release
  3. Keep following the call tree and if deviates:
     - Accept/release next lock if in global order
     - Re-acquire global lock and start tracing if not
- Composes fine; greedy locking may be expensive
  - Works if many top-level branches and/or shallow stacks

Toy Setup

- C++11 based, AthExHelloWorld derived:
  - CLang++ from trunk, gcc4.7.2 headers and libs
    - Move-semantics needed (concurrency support is limited)



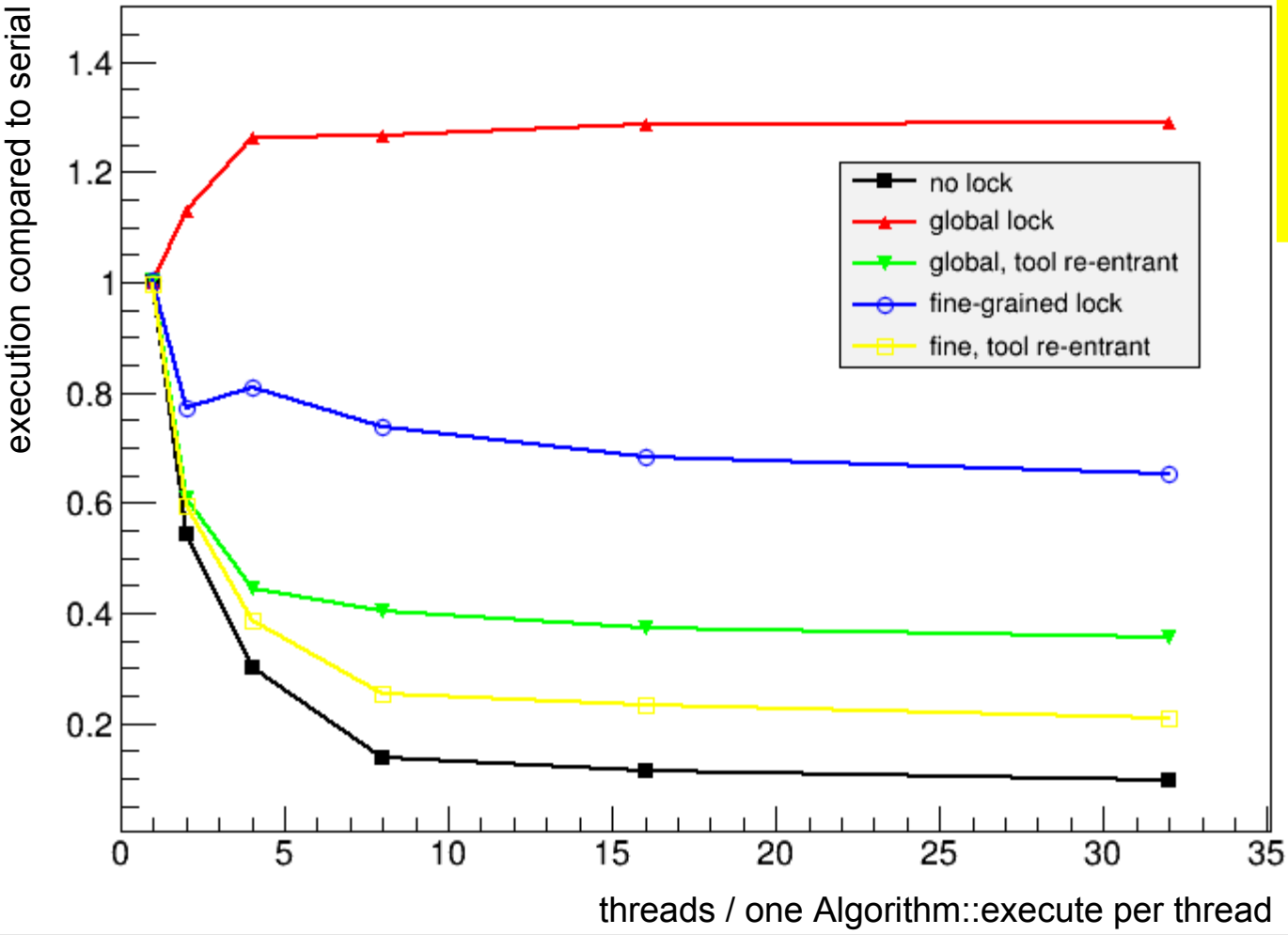
*(Note 1: call order shown, there is only one S1 instance.
Note 2: you'd never count like this, but it allows easy checking.
Note 3: only one Algorithm, no call tree conflict resolution yet.)*

- 3 policies: unlocked(*), global lock, fine grained

(*) ToolHandles can be created lazily, so a lock on the ToolSvc is needed to prevent crashes when called from a thread

Results

(This is a *toy* setup; only *qualitative* conclusions might be valid!)



- i7, 4 cores, HT on
- thread-safe handles
- no lock →
  wrong results!
- tool re-entrant →
  only service locked

Running Atlas Reconstruction Through GaudiHive

Charles Leggett

Modifications to Athena

- For each Algorithm require:
  - per-event run times
    - modify ChronoAuditor (only does total time)
  - input and output data
    - modify StoreGateAuditor
    - try to differentiate between Sequences and Algorithms/subAlgs
- run normal reconstruction
- massage data into a json file that's used to configure GaudiHive

```
"algorithms" : [
  {
    "name" : "CaloCellMaker",
    "inputs" : ["TileRawChannelCnt", "EventSelector", "LArRawChannels", "MyEvent"],
    "outputs" : ["AllCalo", "MBTSContainer"],
    "runtimes" : [406409, 281193, 383043],
    IsClonable = True
  },
```

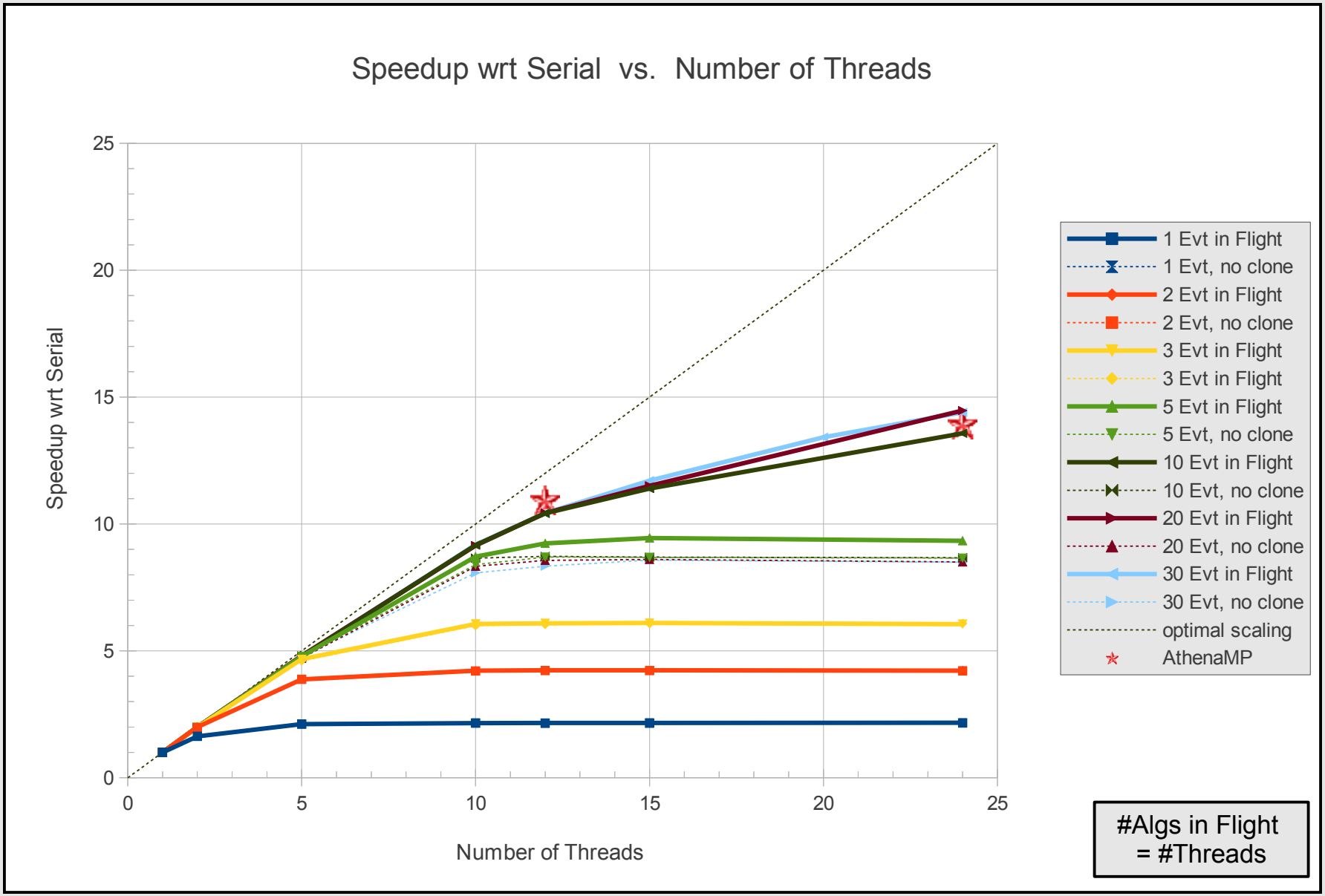
Initial Issues

- Not a drop in replacement:
  - Internal details of Atlas make extracting data flow non-trivial
    - turn off Trigger, or run only on data
  - Lots of cycles in graph of data flow
    - can be removed after understanding source
  - Certain assumptions have to be made as to the source of some Data Objects
- At the very least, will have to modify Athena to make all Algorithms declare *a-priori* their inputs and outputs

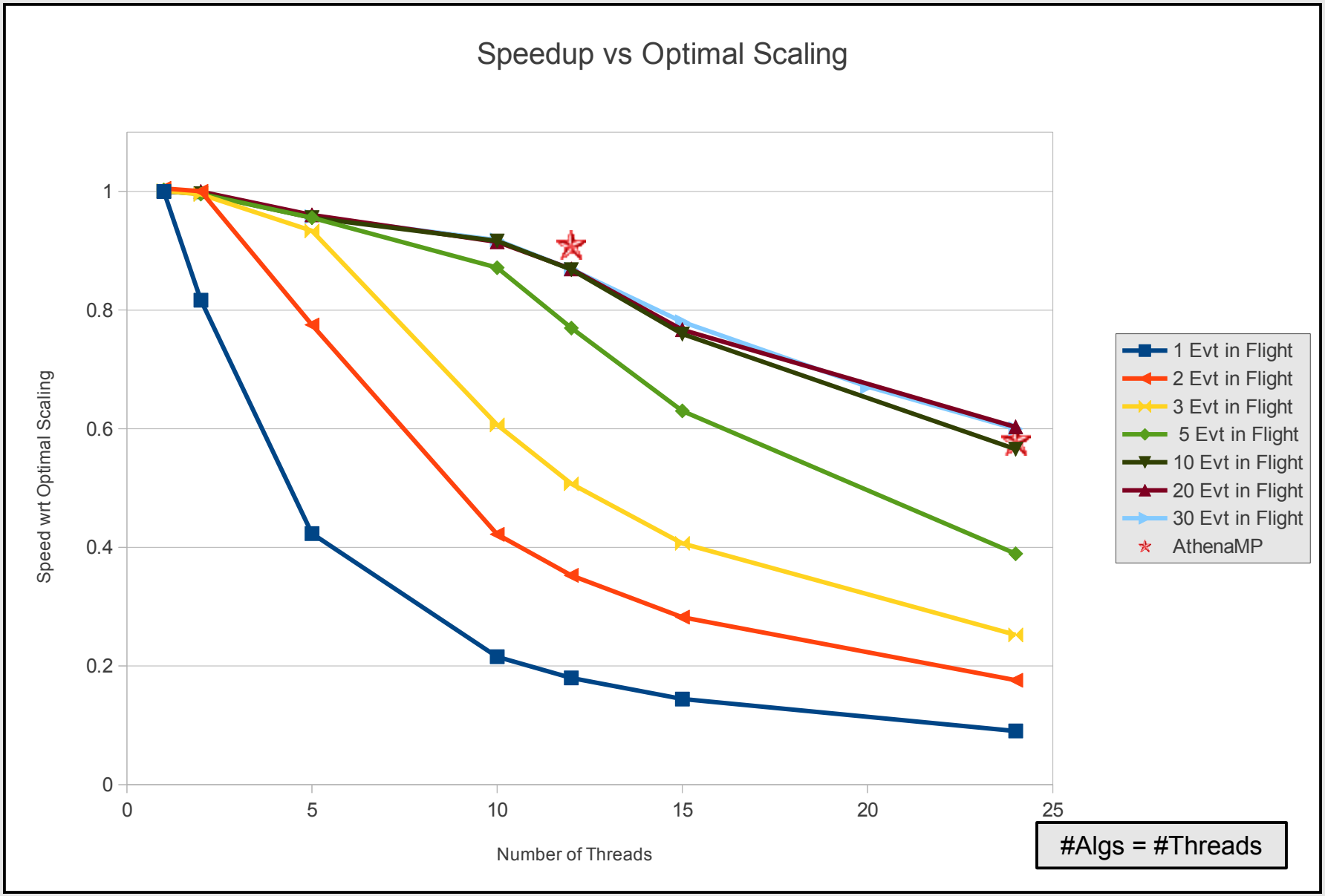
Initial Results

- Configuration options
  - #Algorithms in flight
  - #Events in flight
  - #threads
  - cloning
    - phase space is large
- Test platform:
  - 12 CPUs with hyperthreading = 24 virtual cores
- Data set
  - Standard Atlas Reco from t-tbar MC, no Trigger
  - 100 events
  - 161 Algorithms, 317 DataObj

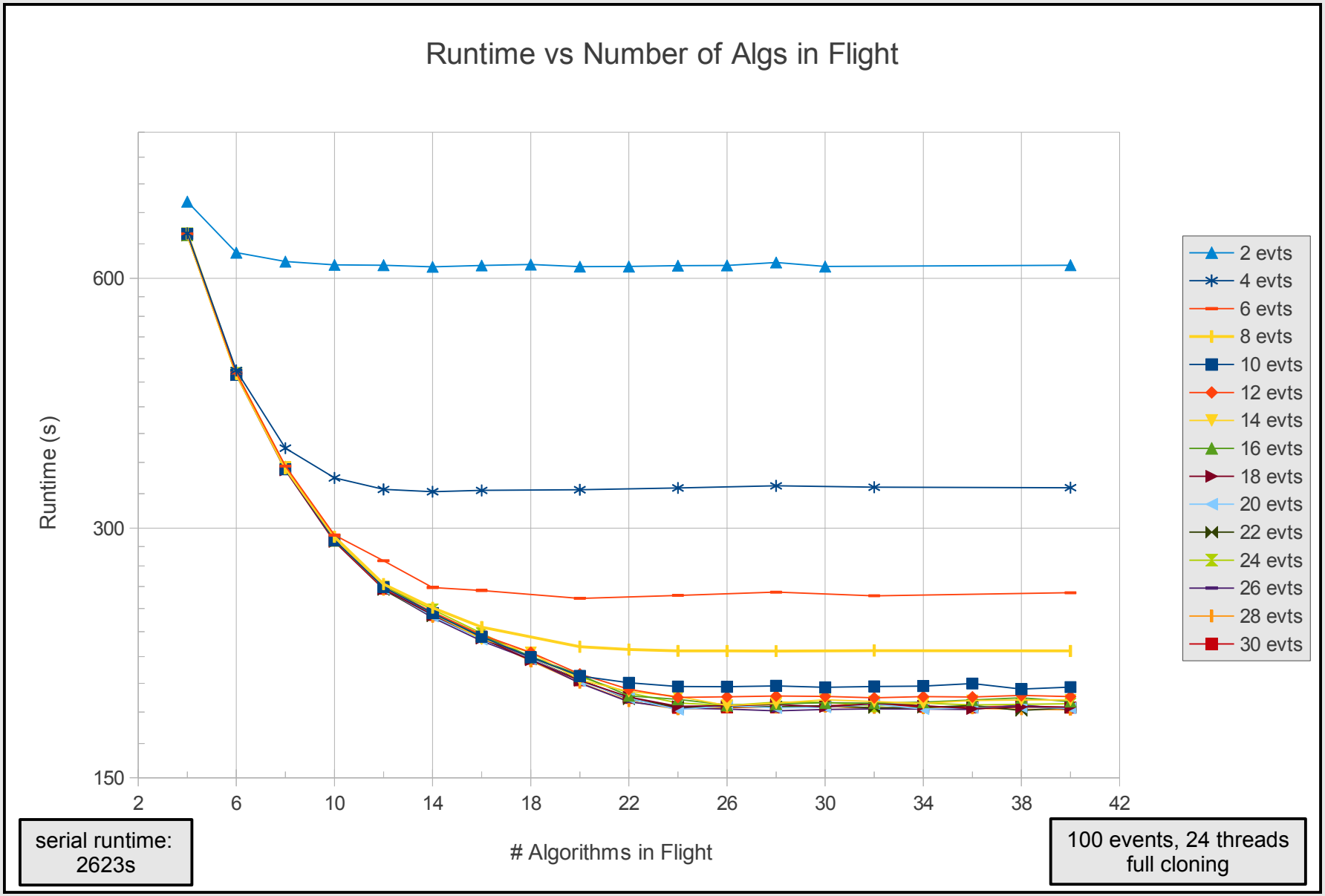
Timing Results



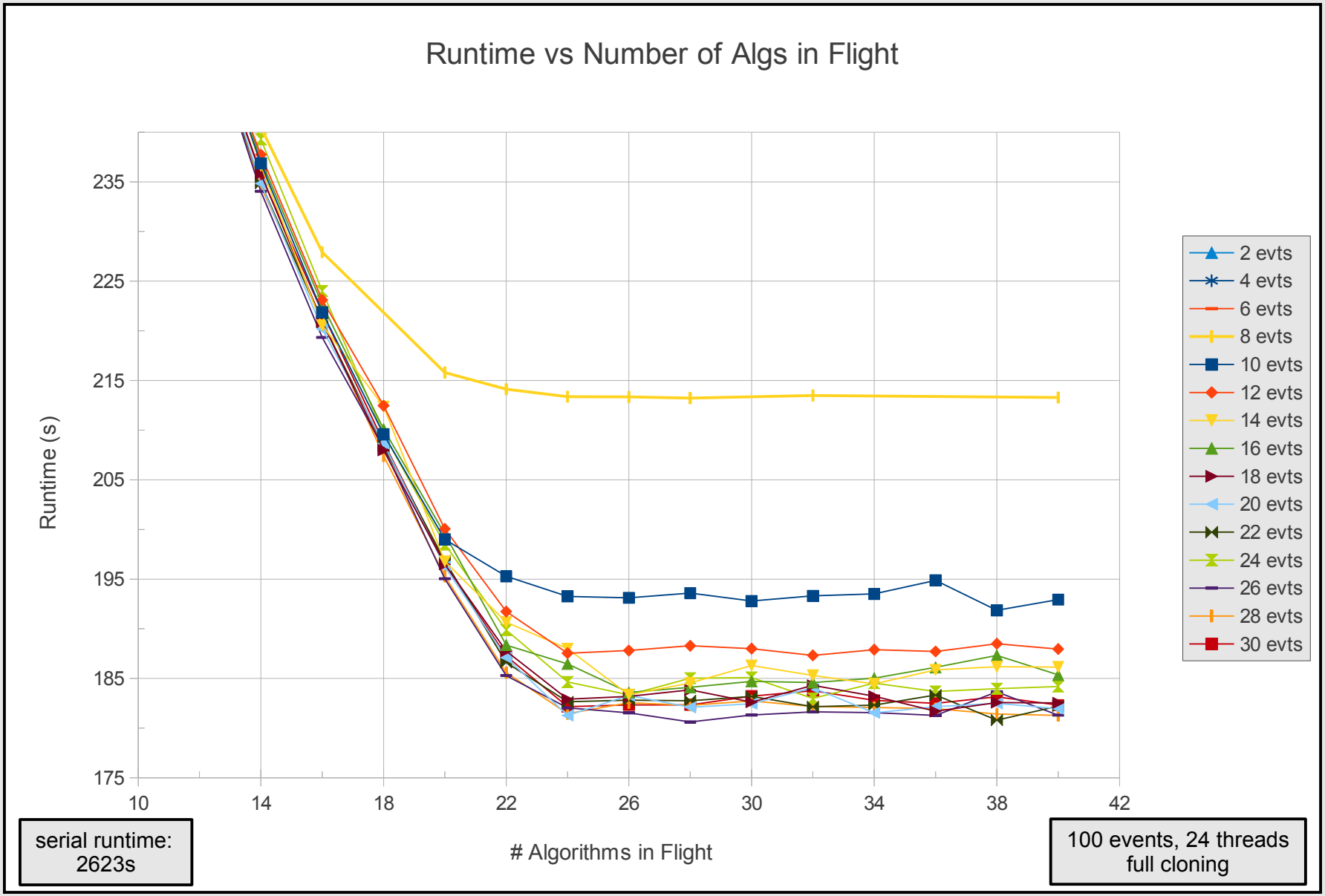
Timing Results



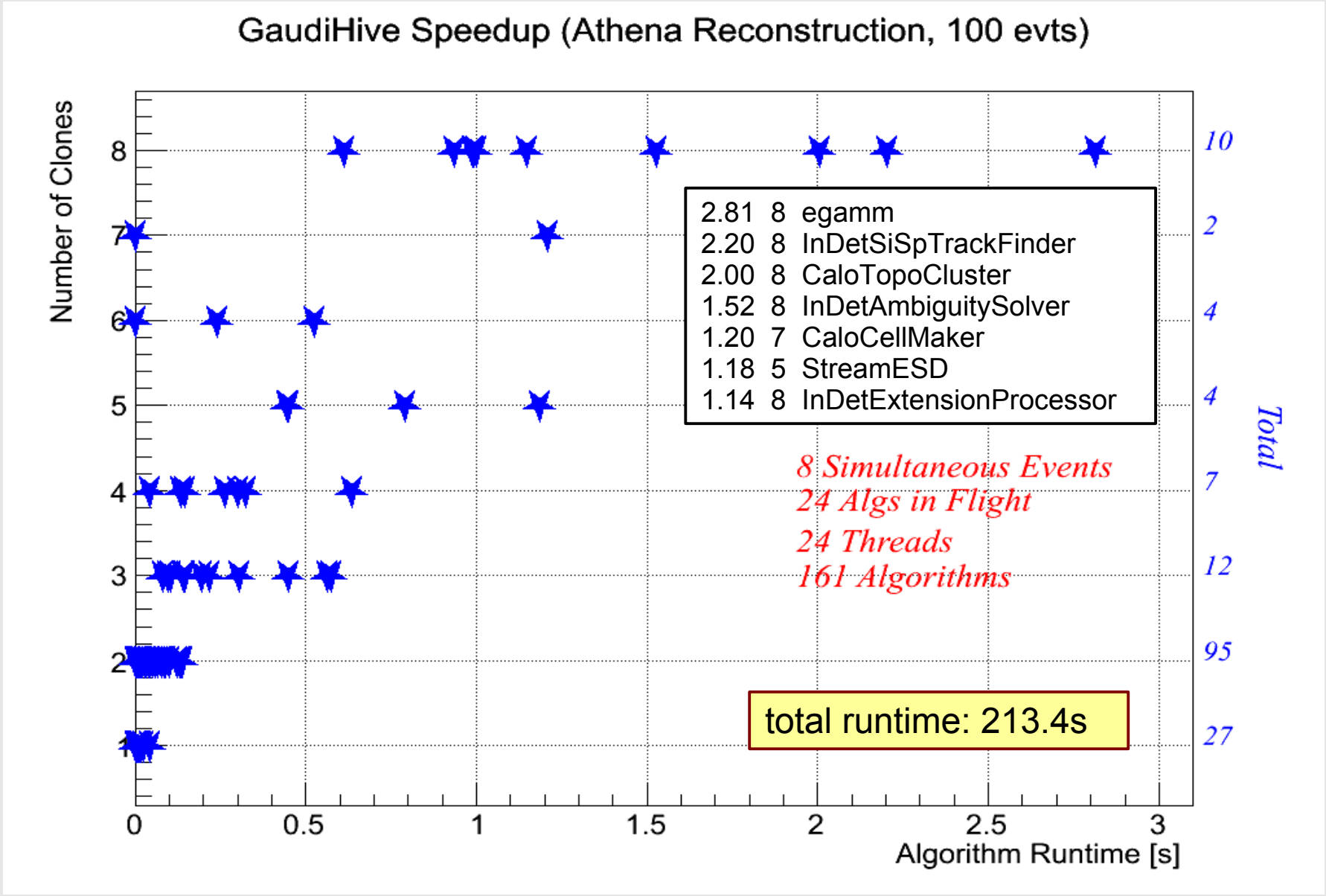
Timing Results



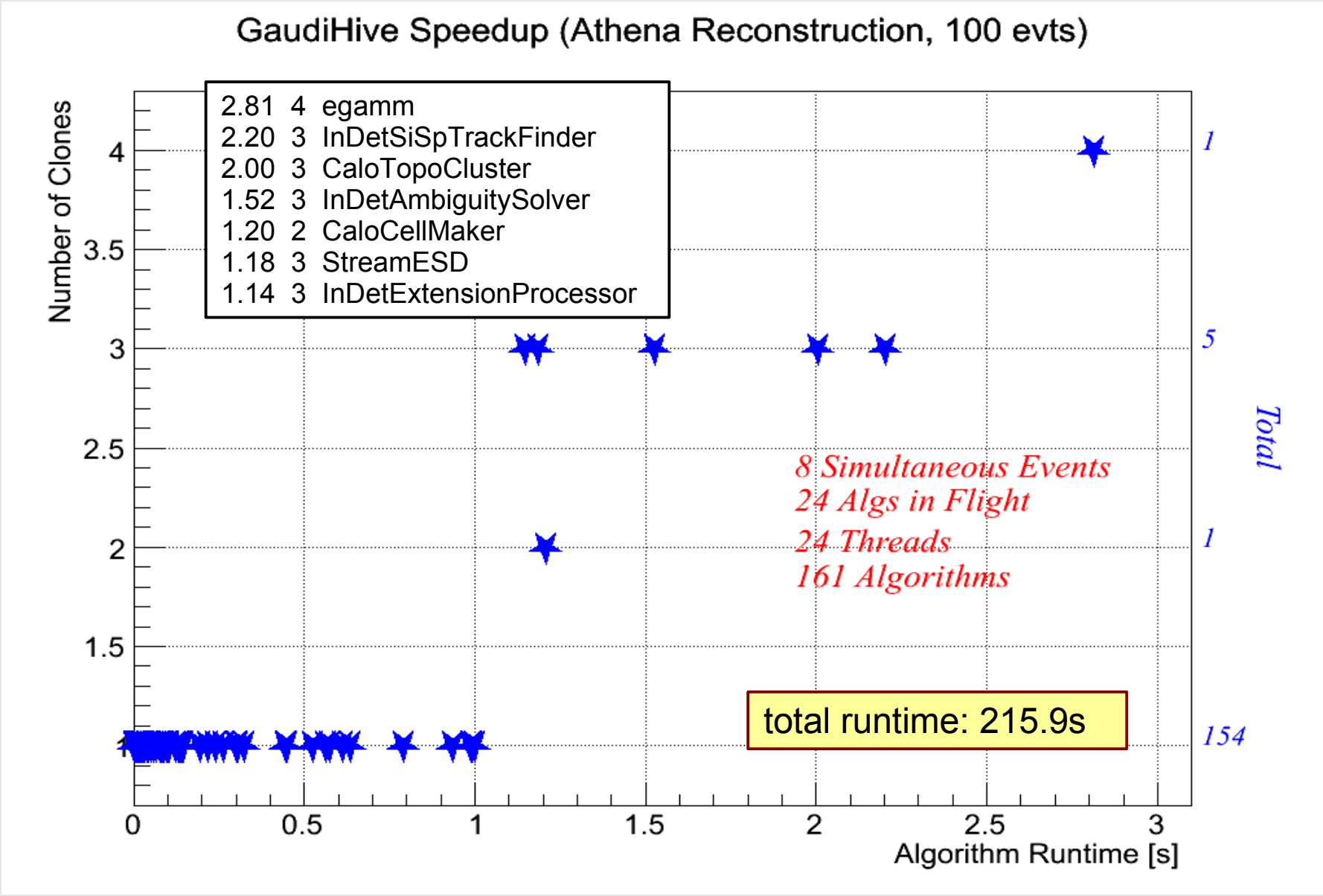
Timing Results



Full Cloning

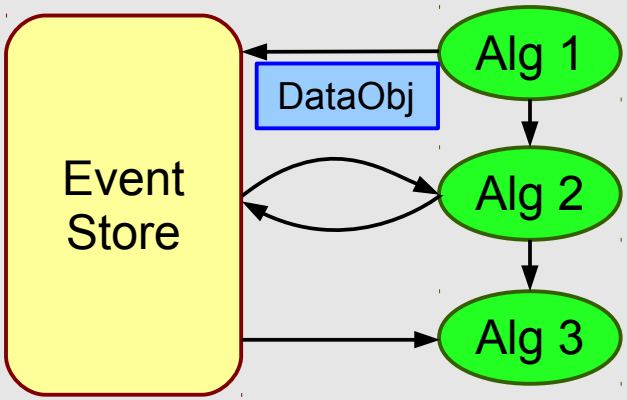


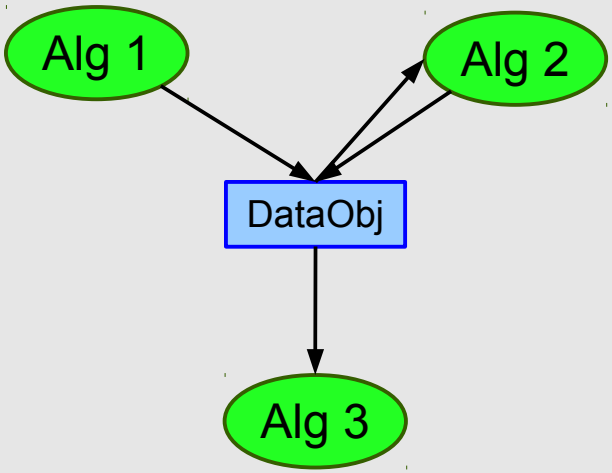
Limit Cloning



Other Issues

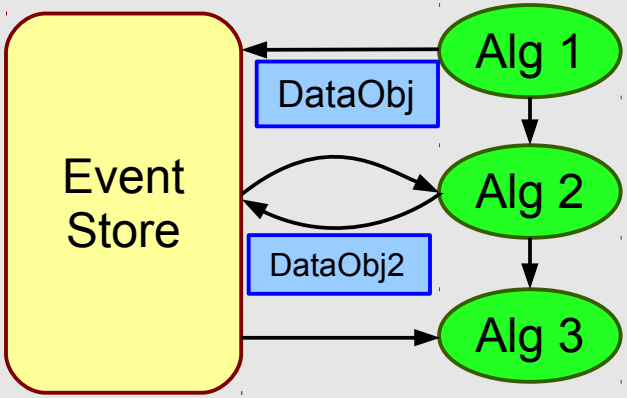
- How to handle Sequences?

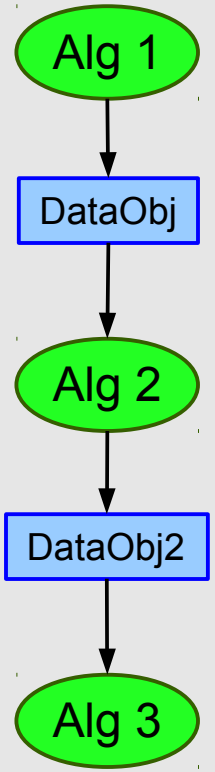




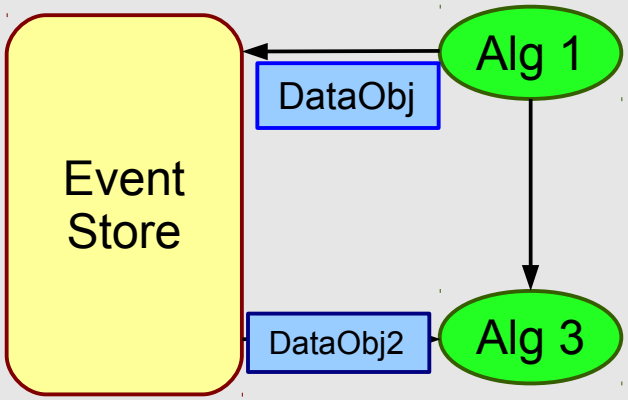
- Looking at just the data flow graph, the scheduling order of Alg 2 and Alg 3 is ambiguous

- Could solve in by renaming/versioning the DataObj





- But with a squence, the execution of Alg2 may be set at runtime via a configuration option.



- Need to be able to mark a sequence of Algorithms as being purely serial

Other Issues

- How to implement Filters?
  - Task scheduler must be told to stop executing current event across all Algorithms and threads.
- Determining data flow for Trigger is non obvious
  - no communication with StoreGate
- No idea of what are the memory implications of cloning Algorithms - obviously depends on the Algorithm
- Need to experiment with user initiated multi-threading (eg a parallel_for) inside Algorithm or AlgTool
- Even if we only make the top few algs cloneable, how much are we going to have to modify the code?

What's Next

- These results are an idealized, best case scenario - no memory issues, no L1/L2/L3 cache, I/O, locking, *etc*.
  - we will never see this in real life - how much worse will it be?
- Can we obtain information at runtime from Components?
  - re-entrancy, cloning-safety, resources required
  - tool author should declare, rather than user find out
- Figure out how to handle Trigger
- Build infrastructure to allow easy collection of statistics
  - find points of contention and missed parallelism
  - graph for GaudiHive can also serve to improve code organization
- Explore ramifications of user level parallelism in Algorithms
  - get a MIC sometime soon!
  - interactions with TBB/off-loading
- Ultimate goals will depend on decisions at this workshop

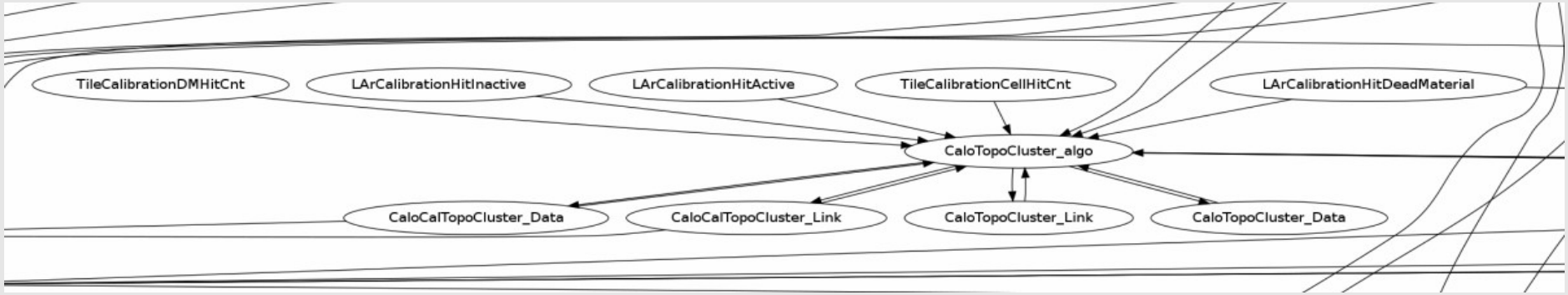
Extras

Data Flow in Atlas Reconstruction

- GaudiHive builds a directed, acyclic graph to determine which algorithms to schedule
  - nodes are algorithms and data objects
- In order to schedule an alg, it must have inputs and outputs
  - Atlas reco has 1892 Algorithms, of which 709 either read from or write to StoreGate
- This is due to the Trigger
  - Trigger algs don't use StoreGate, but rather communicate with Trigger Tokens
- For now, let's turn off Trigger, and rerun
  - 161 Algs, 161 with StoreGate info
  - 317 Data Objects

Cycles in Atlas Reconstruction

- Atlas flow graph has lots of cycles



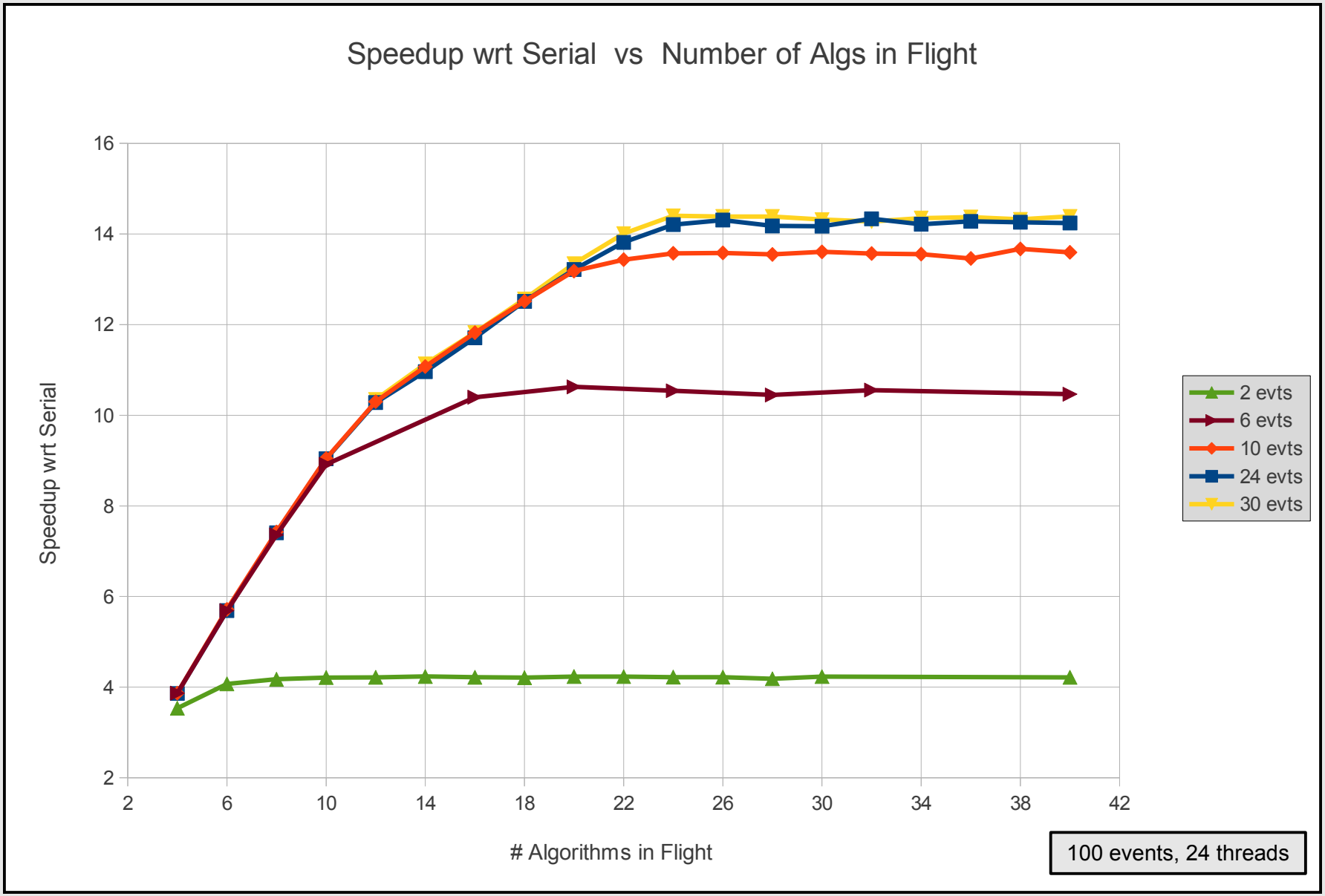
- usually due to a “contains” followed by a “record”

```
if (evtStore()->contains<Muon::RpcCoinDataContainer>(m_outputCollectionLocation)) {
     msg( MSG::FATAL) <<"Muon::RpcPrepDataContainer not found while “
                      <<”Muon::RpcCoinDataContainer found in Event Store"<<endreq;
     return StatusCode::FAILURE;
}

m_rpcCoinDataContainer->cleanup();

StatusCode status = evtStore()->record(m_rpcPrepDataContainer,
                                       m_outputCollectionLocation);
```

- remove all cycles


Speedup wrt Serial vs Number of Algs in Flight
Speedup wrt Serial
16
14
12
10
8
6
4
2
2
6
10
14
18
22
26
30
34
38
42
Algorithms in Flight
2 evts
6 evts
10 evts
24 evts
30 evts
100 events, 24 threads