Results of performance testing of SuperB/BaBar applications

Vincenzo Ciaschini

4th SuperB Collaboration Meeting
La Biodola, 31 May – 4 June 2012

Applications tested

- SkimMini, BetaMini, Moose release 24.5.6
- FastSim, PacMC release 0.2.7_test
- Bruno, CVS from 29/11/2011

Testing environment

- RAM: 63 GB
- CPU: 4 Intel® Xeon® E7 4870 with hyperthreads disabled (total 40 cores)
- Hard disk: 120 GB
- Data source: CNAF's GPFS file system
- Executable source: NFS mounted partitions
- Output destination: Local hard disk
- OS: Scientific Linux 6

SL6 Adaptation

- No changes to the executables
  - Just install a bunch of -compat libraries and downgrade the TCL ones.
- Why?
  - Older kernels did not recognize the processor as anything more than “i386” thus making collection of processor usage data impossible.

Test scripts and instructions

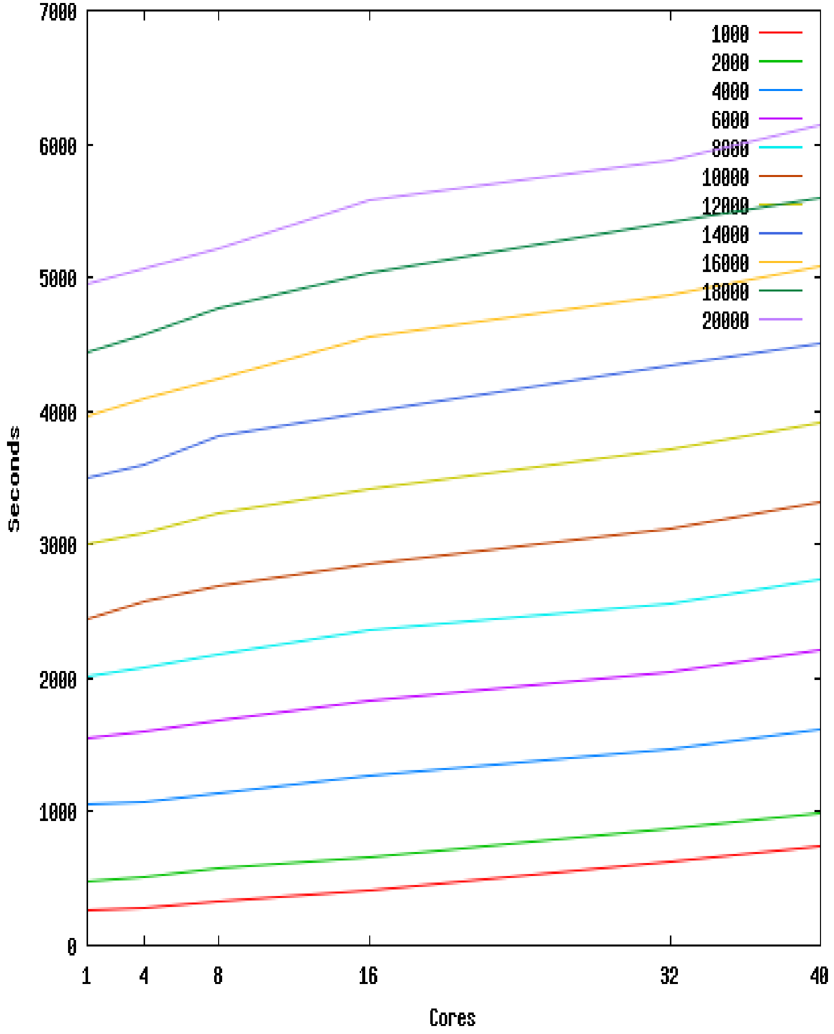
- On CNAF’s public git repo (as soon as it gets official “blessing”)
  - Ask me for it in the meantime

Results

Highlights only

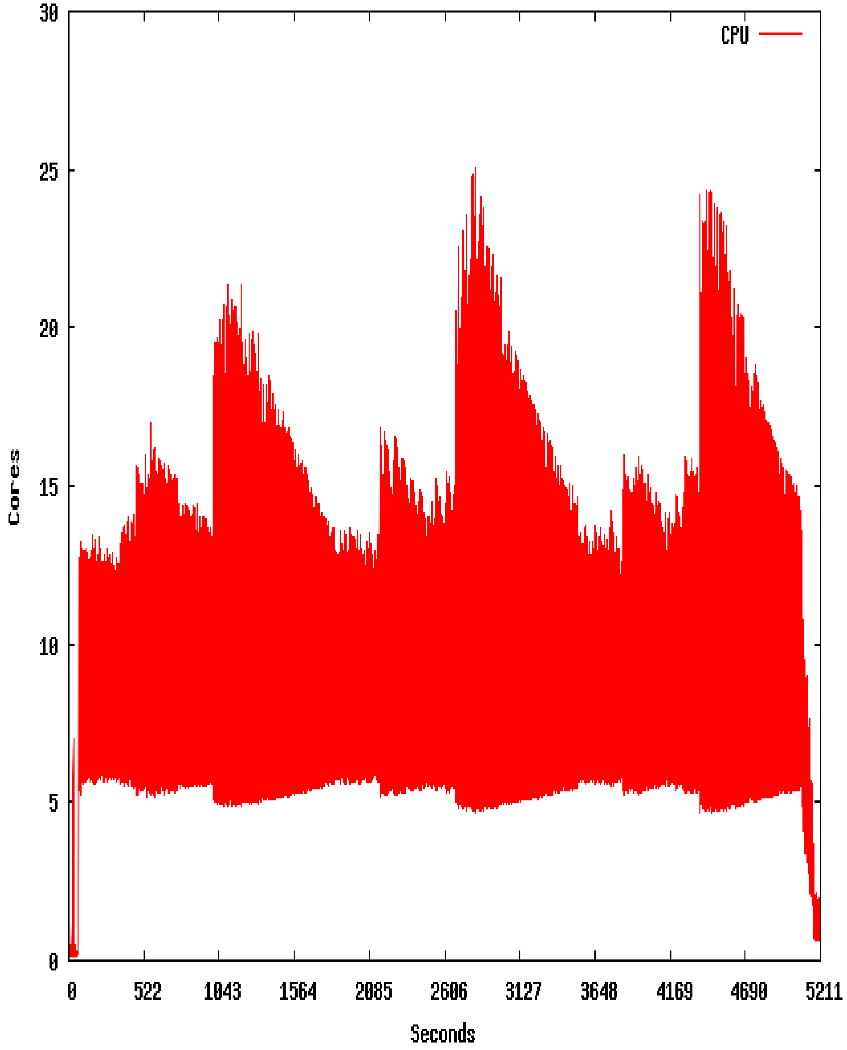
The report, under review, will have the full information

SkimMini



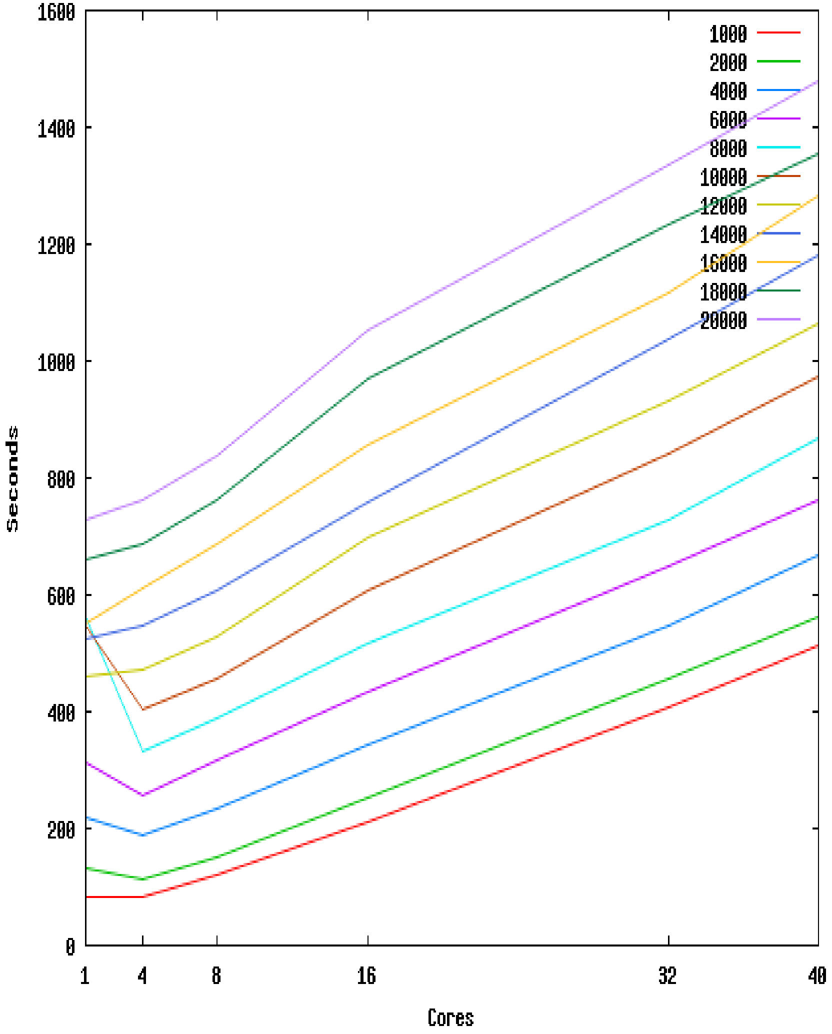
- Does not scale at all
  - Execution time skyrockets with more parallel executions
- Explanation:
  - Executes ~1000000 calls to stat() during startup and first event processing
  - On only 30 different paths
  - Contention pretty much disappears if the stat() time is removed.

SkimMini



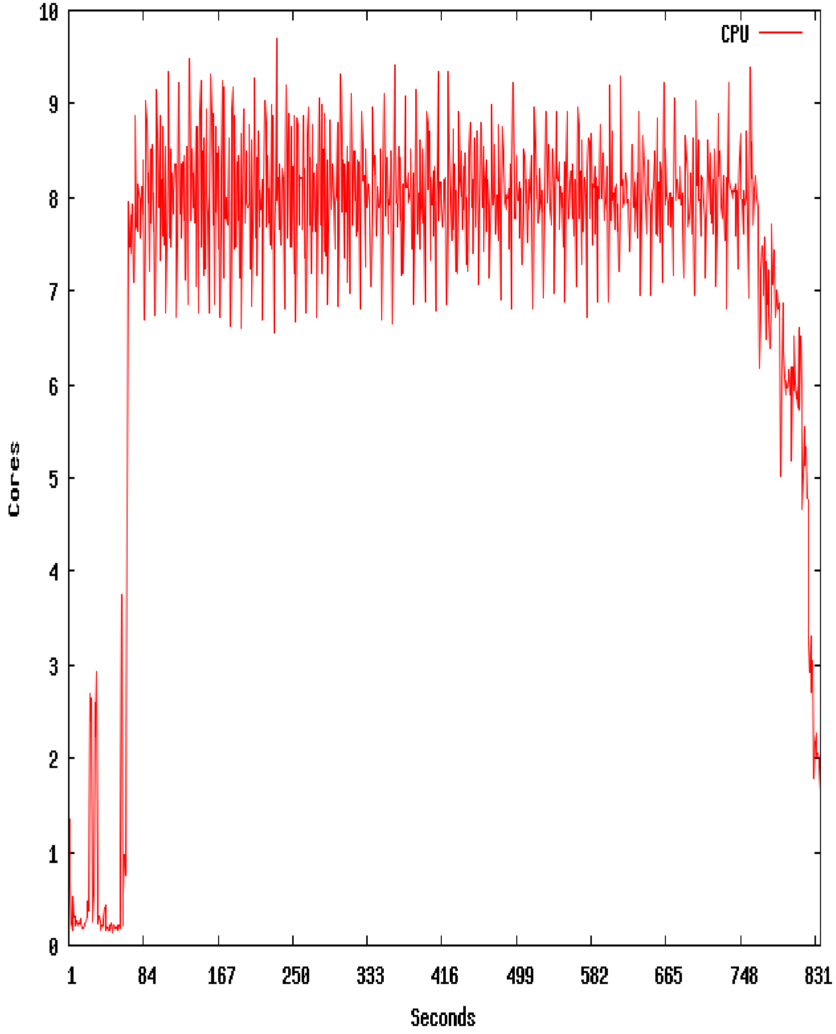
- 20000 events, 8 runs
  - CPU Usage has startup and teardown slowdowns
    - Startup depends on cores → See previous slide.
    - Teardown depends on number of events → writing output.
  - Seesaw pattern: unexplained yet.

BetaMini



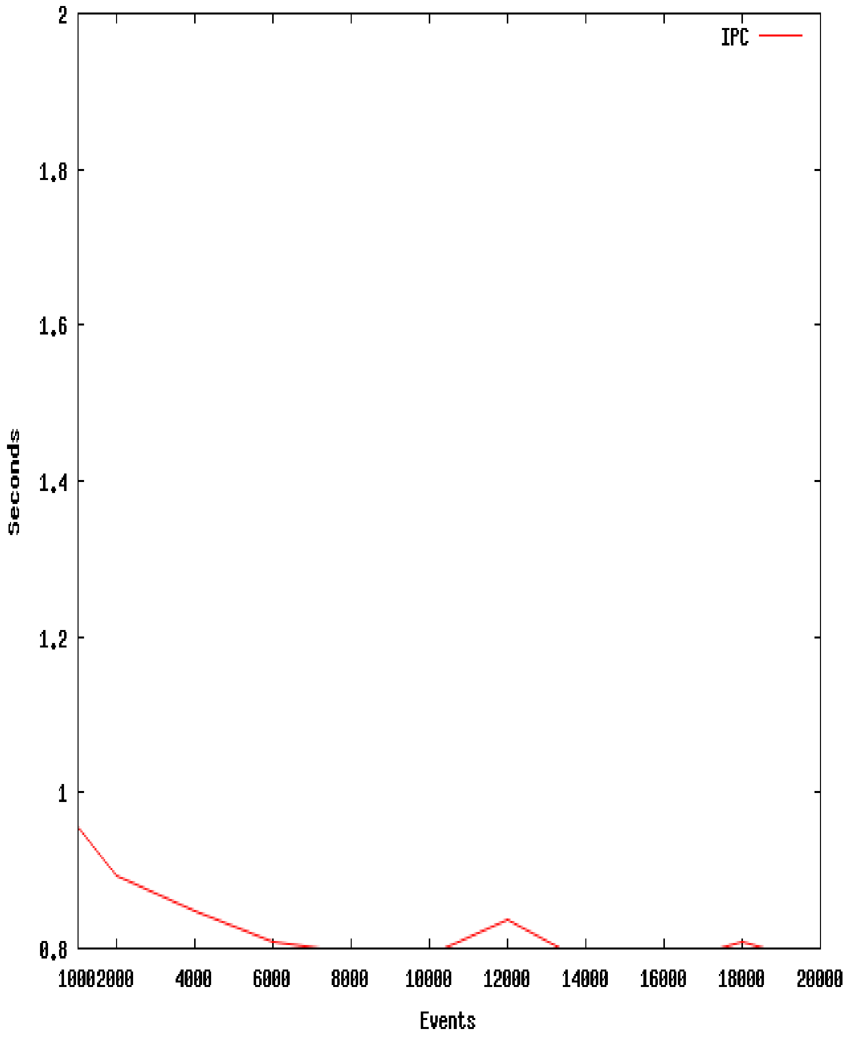
- Contention: exact same issue as SkimMini
  - More pronounced because BetaMini in general is faster than SkimMini

BetaMini



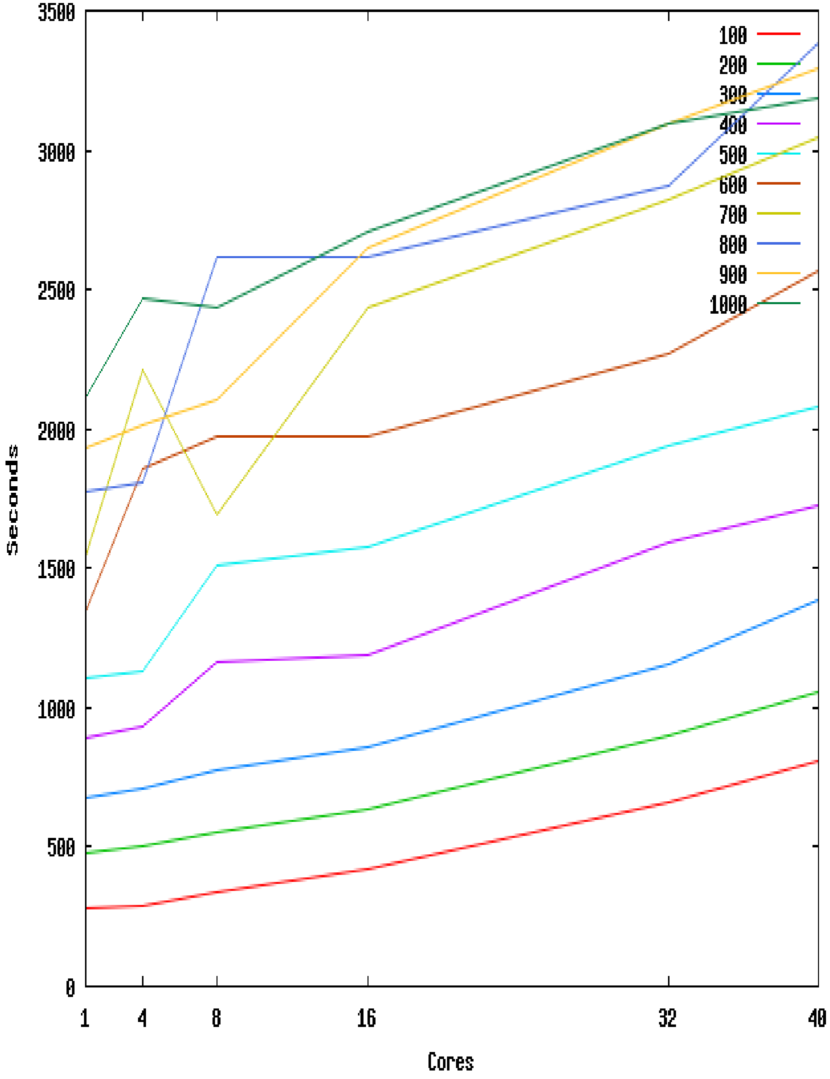
- 20000 events, 8 runs
- Same startup/teardown as SkimMini
  - Same interpretation
- No Seesaw pattern
  - Less time, so I/O time more evident

BetaMini



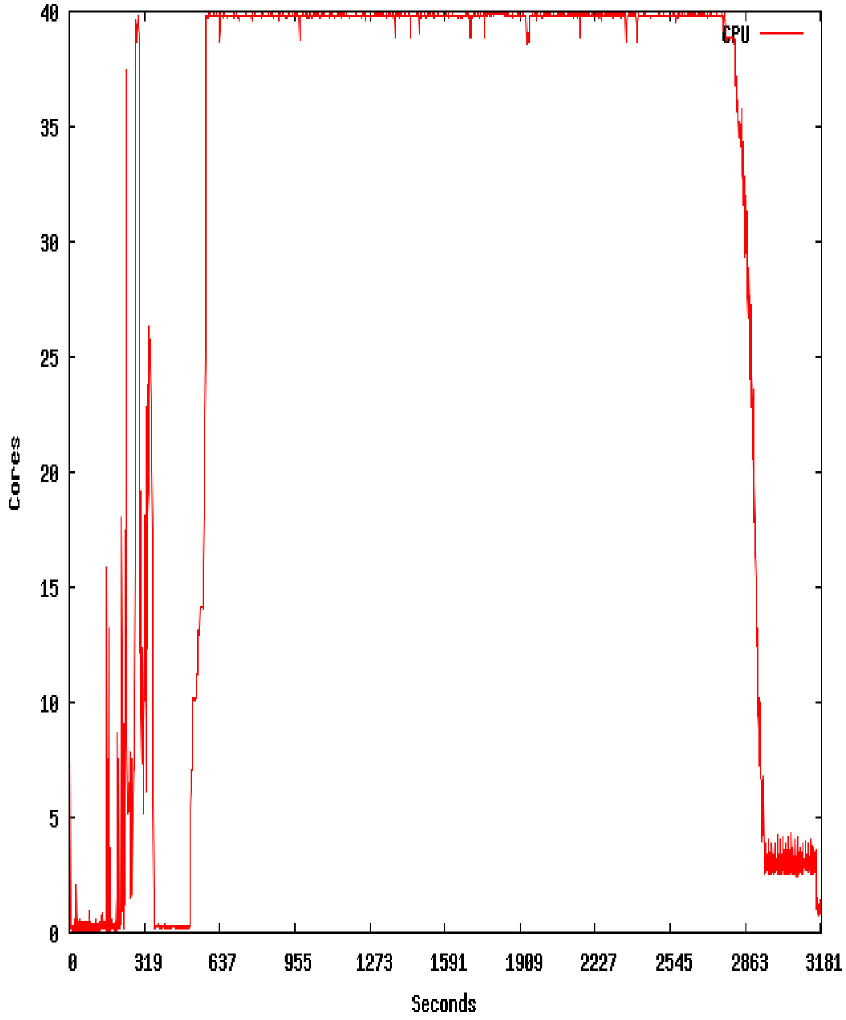
- IPC (Instructions per cycle)
  - Always less than 1
  - Often less than 0.8
  - The processor is doing nothing but waiting, for large amounts of time!
    - Worse and worse as the number of events increases
    - This with only one instance running
  - The algorithms used are in sore need of optimization.

Moose

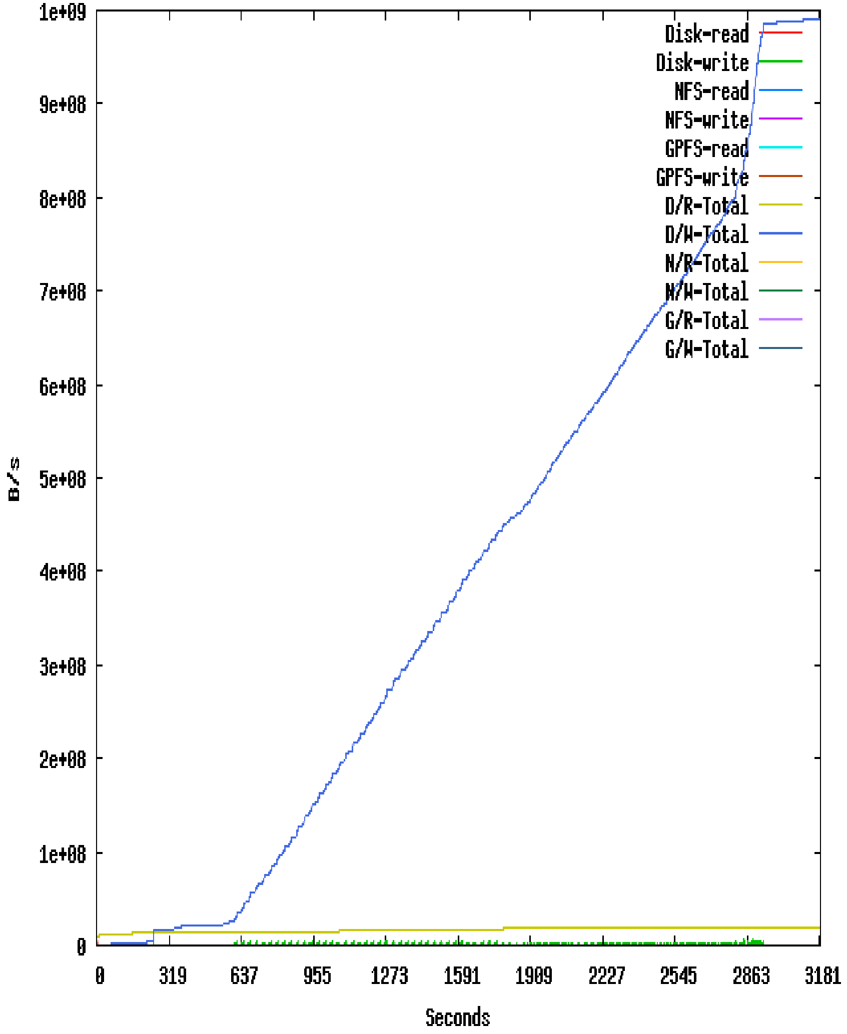


- Suffers from same issues as BetaMini and FastSim
  - But that is not all
    - Greater irregularities at high number of events.
    - Data not sufficient for explanation
      - But see next slide

Moose

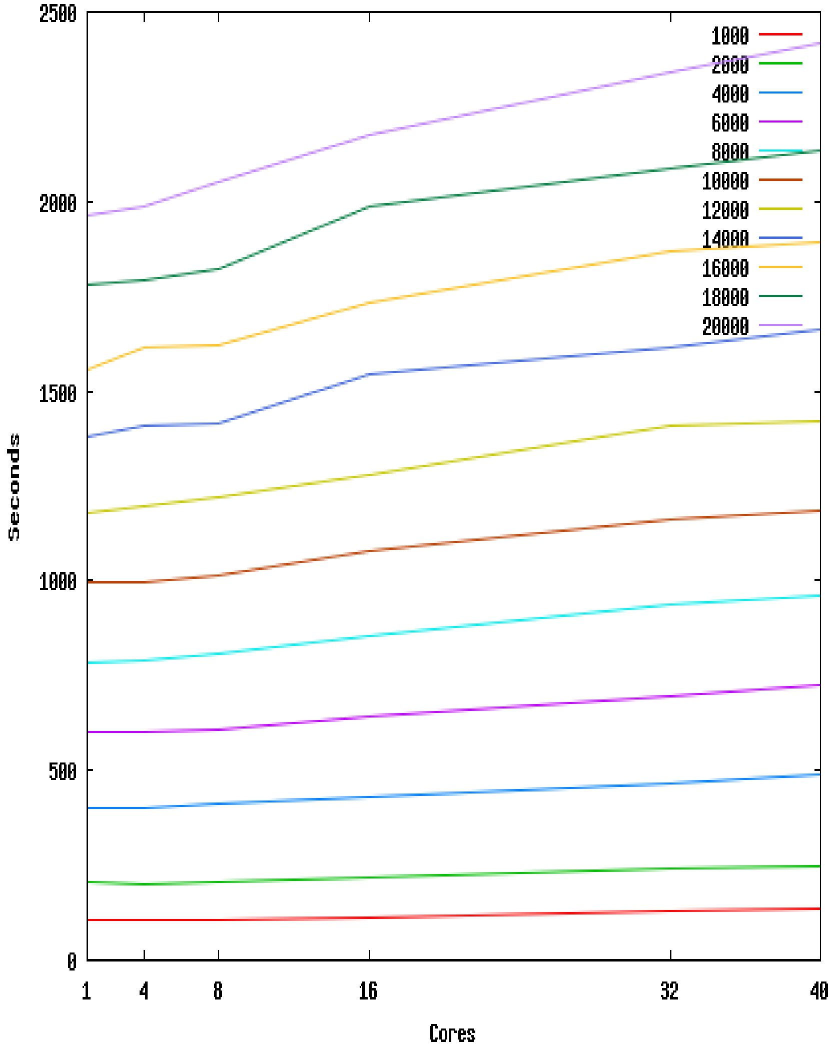


- 1000 events, 40 runs
- Four clear phases
  - Initialization: Around 470 seconds regardless of number of events
  - Computing
  - Partial teardown: calculation still ongoing
  - Final teardown
- Interpretation:
  - Race for resource access during teardown
  - Cannot get more details because reporting tools like strace alter the pattern and make it disappear



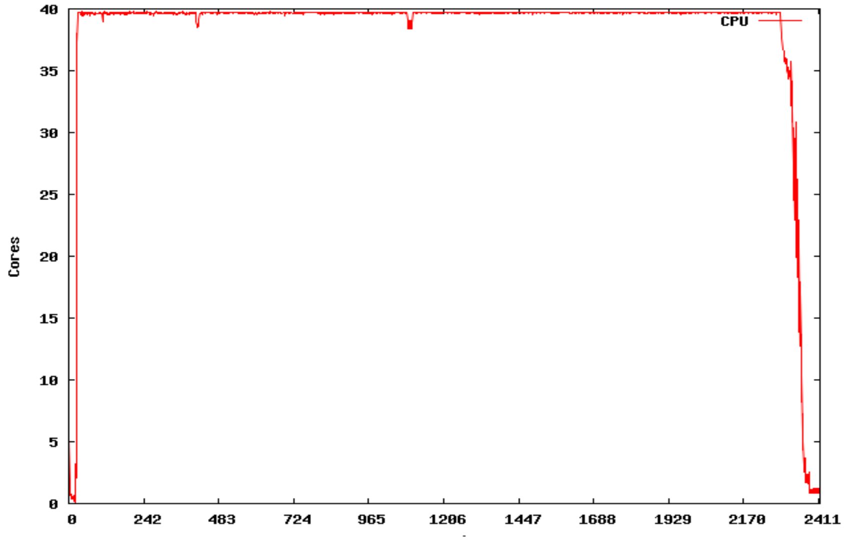
- 1000 events, 40 runs
- Shows slowdown corresponding to third phase of CPU usage graph
  - Probably I/O related issues.

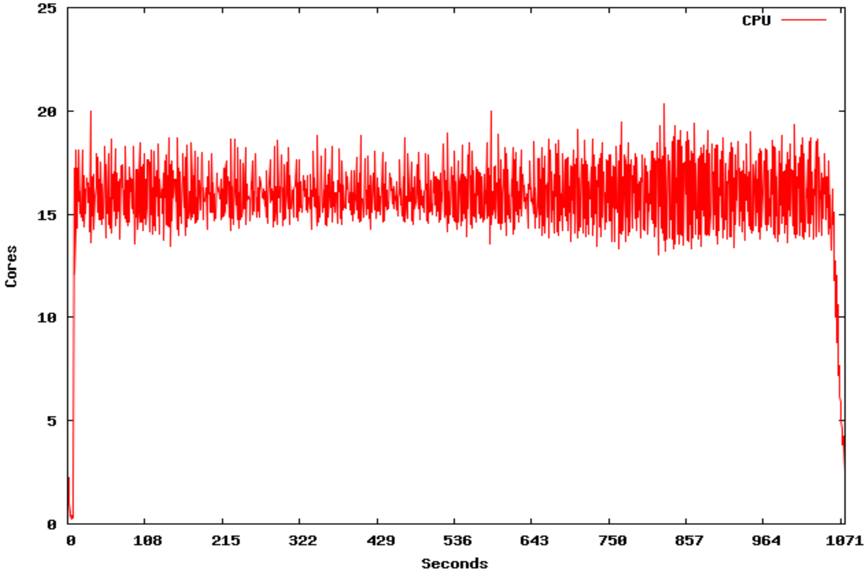
FastSim



- Contention still there
  - But related to number of events rather than parallel executions
  - **Not** CPU-related (see next slide)
  - Probably caused by event generation
- External info:
  - FastSim generation creates some events much slower than others by orders of magnitude
    - More events → More slow events

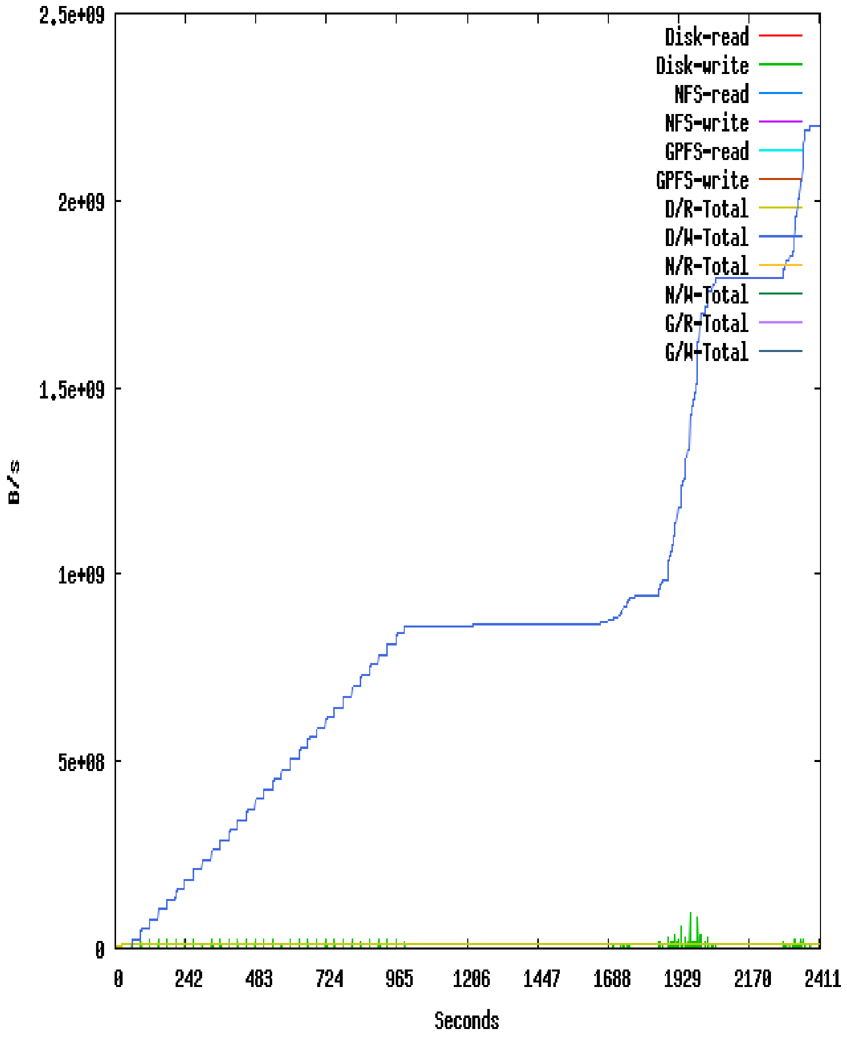
FastSim





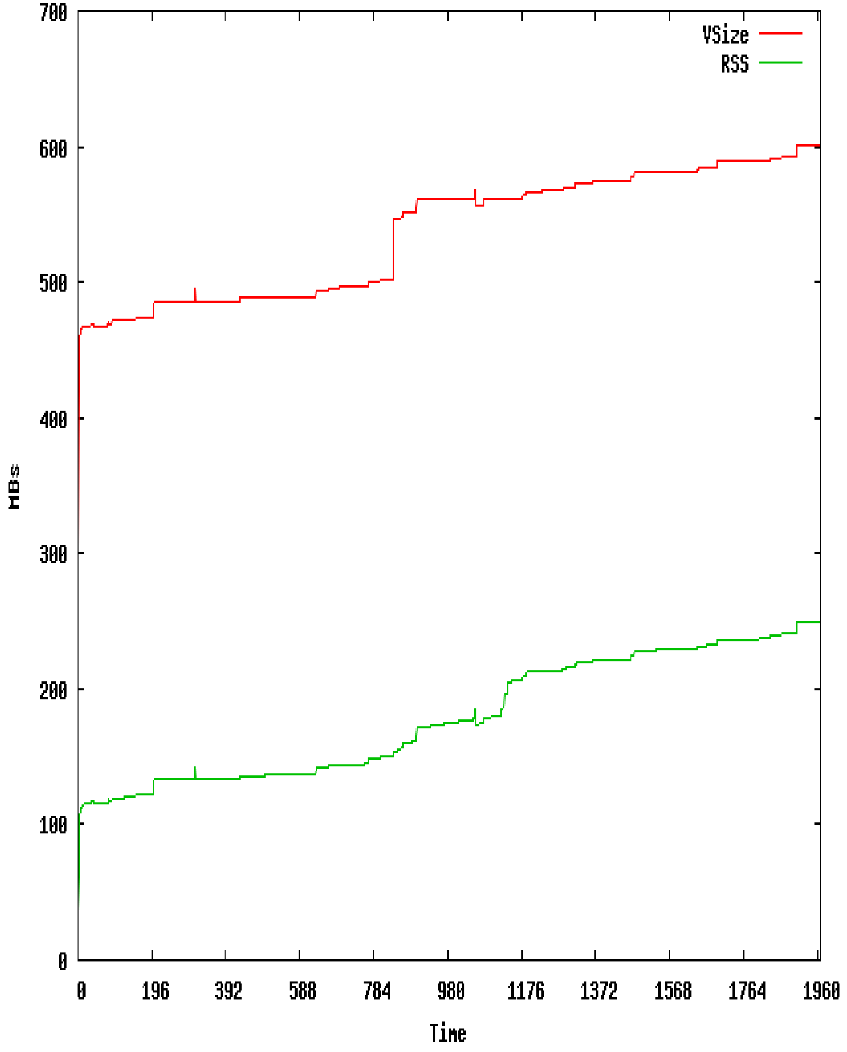
- 20000 events, 40 runs
- 10000 events, 16 runs
- No evidence of significant CPU problems

FastSim



- 20000 events, 40 runs
- Very particular I/O usage:
  - Writing data suffers from periodic “stalls”
  - With less events, stalls are not reached → stalls are the cause of scaling problems
  - Must be investigated by source code experts

FastSim

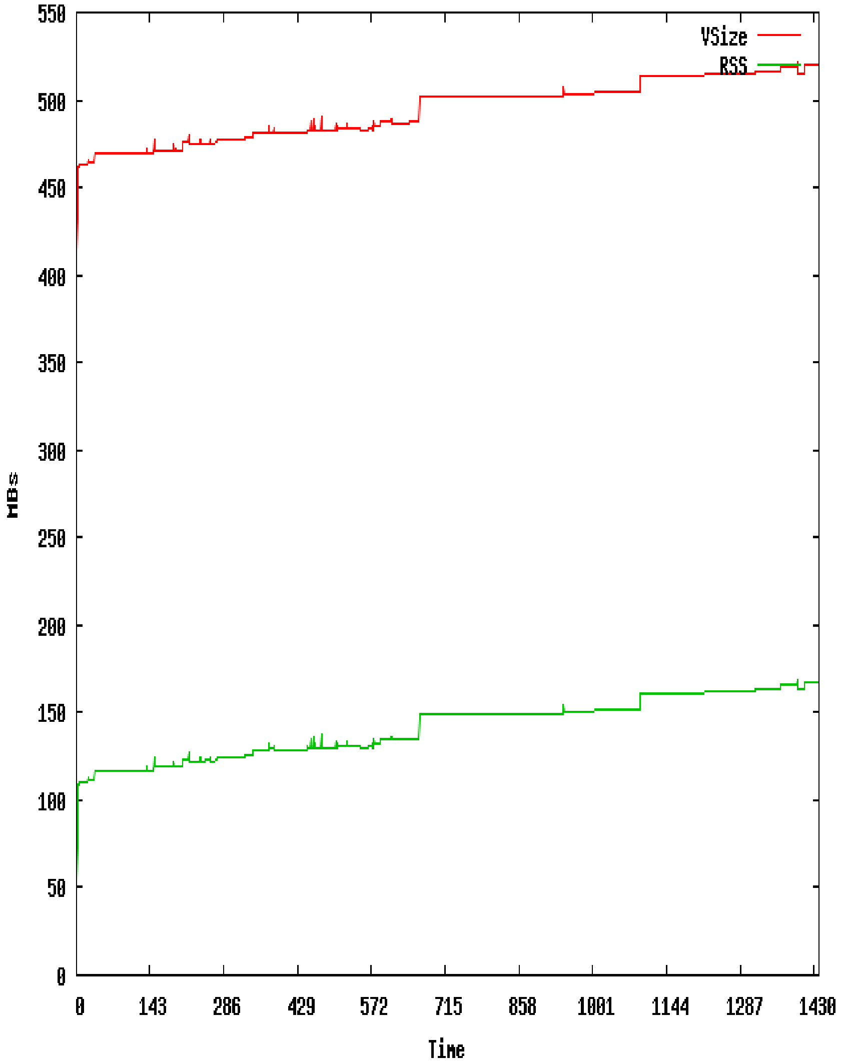


- Memory usage keeps increasing with time
  - Hints at memory leaks in event generation/handling code.
  - Greater offender: PacTrkHitMeas::createHots
    - Per-event leak of around 2800 bytes
    - Not the only cause
  - Freeing memory at end of execution and not at end of event **is** a memory leak for practical purposes

PacMC

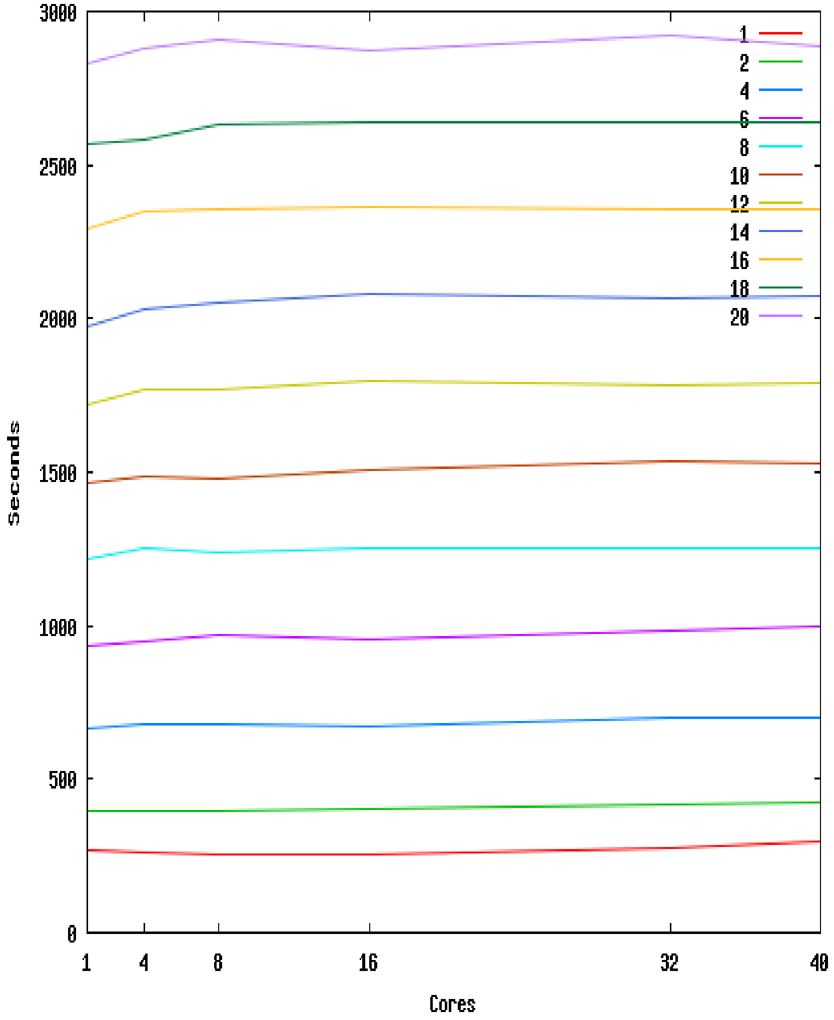
- Execution time and CPU usage are completely analogous to FastSim
  - Therefore not shown here

PacMC



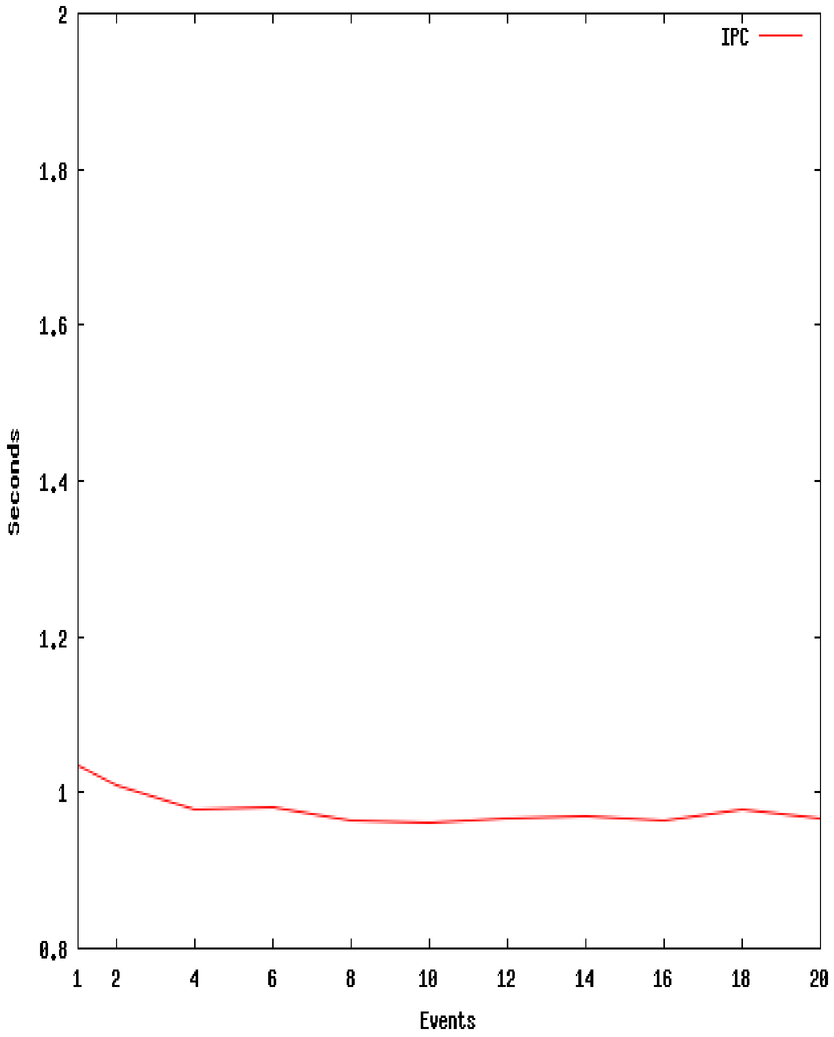
- 20000 events
- Same issues as FastSim

Bruno



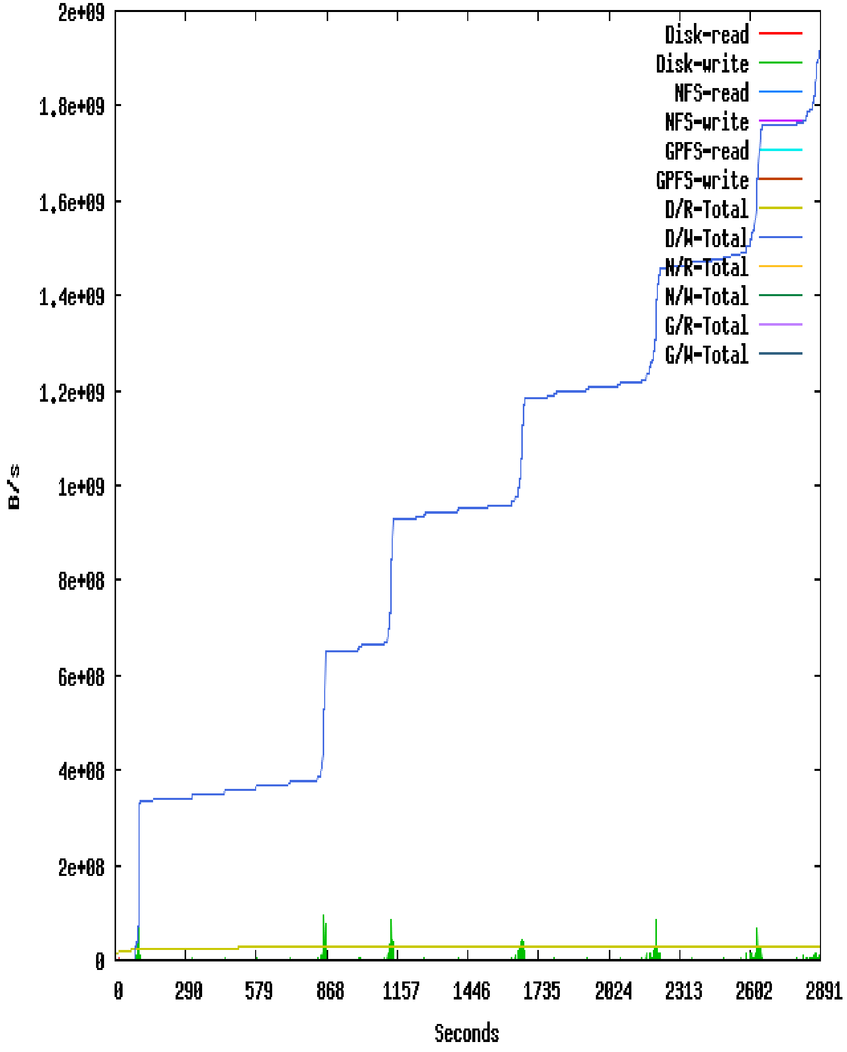
- Almost no evidence of contention
  - Maybe because of low number of events
  - Still, by far most scalable program
- CPU analogous to FastSim and PacMC, therefore not shown

Bruno



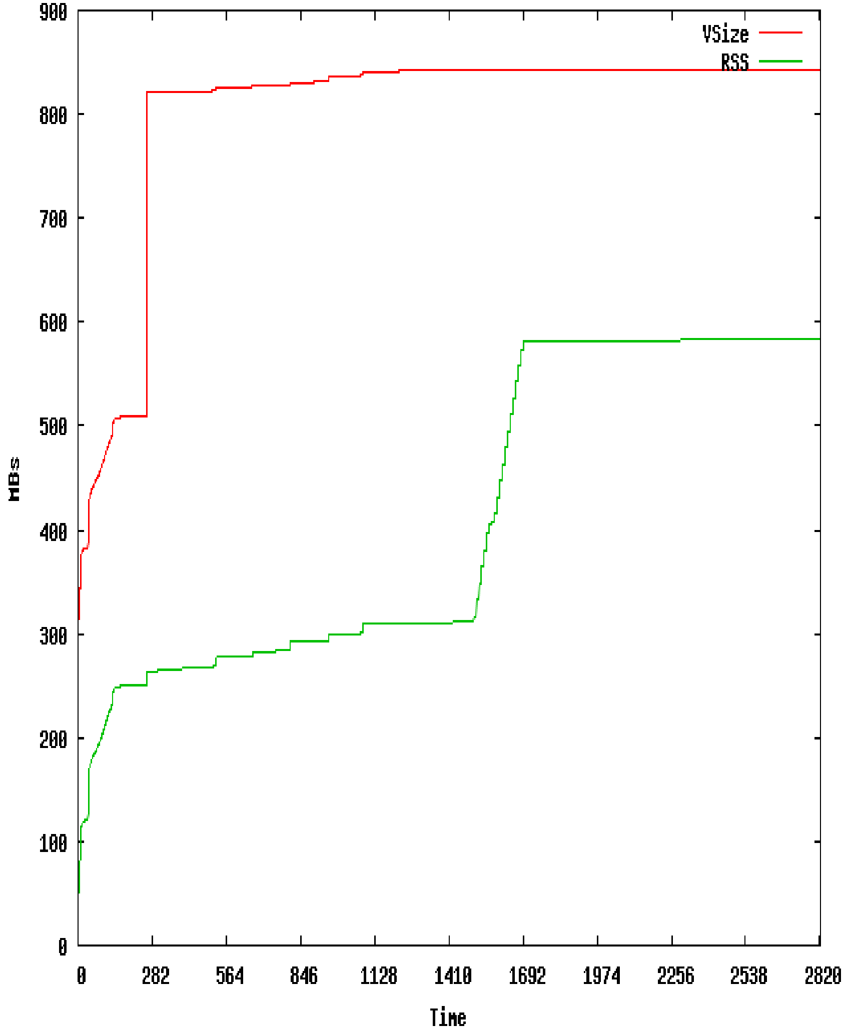
- Unsatisfactory number of instructions per cycle
  - Drops to less than 1 per cycle
    - CPU is waiting for something
- Code needs rewrite/optimization

Bruno



- 20 events, 40 runs
- Data is written “in batches”

Bruno



- 20 events
- Very clear “stepping stones”
- Actual virtual memory usage stabilizes quickly
- But “in memory” virtual memory increases sharply at the middle. This is not understood.

Summary cache info

- BaBar
  - Cache misses around 4%
  - Ranging from 5% to 9% of actual time spent waiting for memory

- SuperB
  - Cache misses < 1%
  - Ranging from 0.2% to 0.6% of actual time spent waiting for memory

Conclusions: Babar

- Clean up the init-phase stat() shenanigans
  - While runs with more events reduce the impact, runs with more cores augment it
  - Preliminary analysis points to ROOT being the culprit
- Generally clean up I/O
- Optimize code
- No significant statements on parallelism can be made until these issues are cleared

Conclusions: SuperB

- Generally in much better shape
  - But memory issues present
    - Should be fixed.
  - I/O issues are still present
    - I/O seems to be generally problematic with exp. Software

- Again, optimization and fixing should be done before statements on parallelism can be made.