Multiprocessing in CERN-ATLAS

Scalability and the Real World:
Lessons Learned Optimizing ATLAS Reconstruction Performance on Multi-Core CPUs.

Mous Tatarkhanov[1]

Sebastien Binet[2], Paolo Calafiura[1], Charles Leggett[1], Wim Lavrijsen[1], David Levinthal[3], Yushu Yao[1]

[1]Lawrence Berkeley National Lab, [2]LAL, [3]Intel

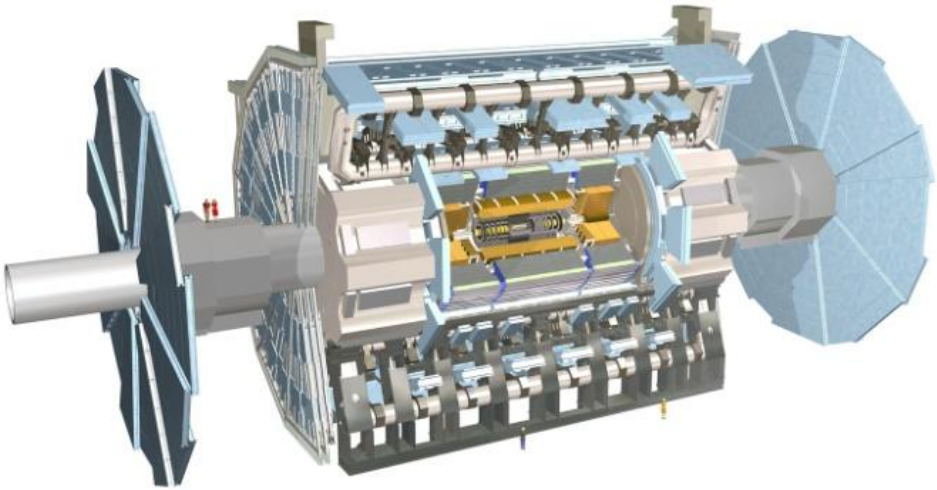
ICCS-2011 – January 26, 2011 –Berkeley, CA

Introduction to ATLAS

ATLAS - Prominent High Energy Experiment at LHC, CERN

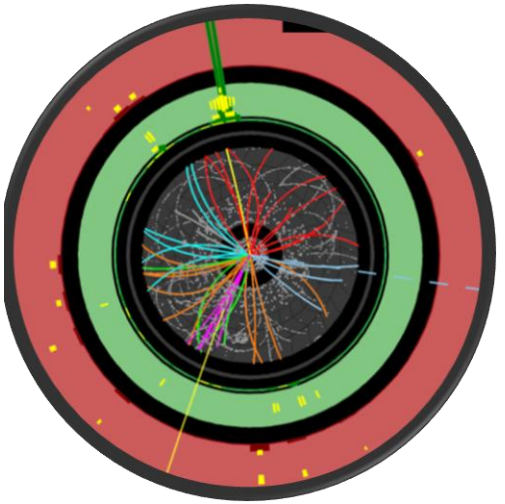
- New discoveries in Fundamental Physics
- Large rate of data:
  - 100 million electronic channels
  - 200 proton-proton collision events/s -> 500MB/s
- ~10000 CPU nodes housed at CERN and around the world

ATLAS Detector



L = 45 m
D = 25 m
M = 7000 tons

Collision Event

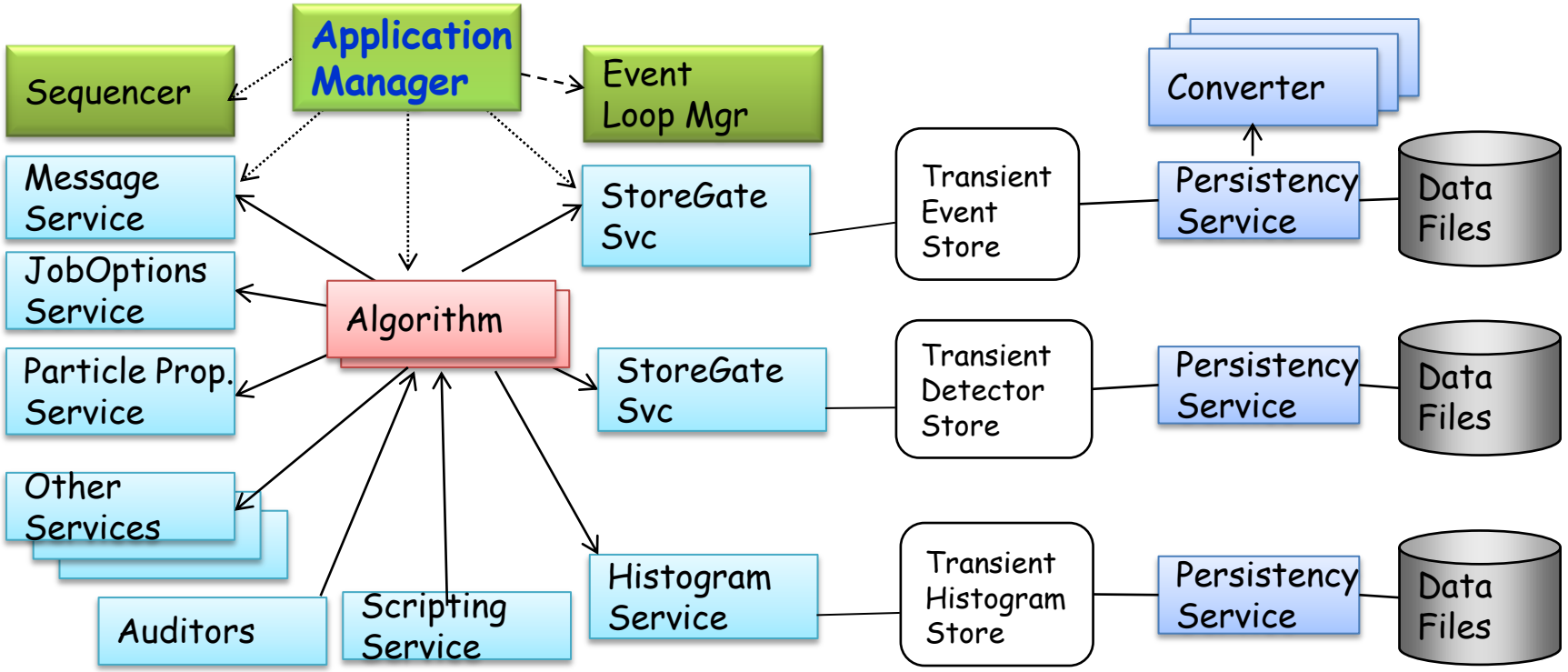


Data = 5 PBytes/year
2500 Scientists

ATLAS software

ATHENA - ATLAS software for Reconstruction, Simulation and Analysis

- Highly robust OO framework based on HEP GAUDI architecture
- Large – 4000 components, 2500 shared libraries
- More than 300 active software developers
- Configured and Steered by Python front-end - **athena.py**
- High application size ~1.5 Gb of real memory for Reconstruction.



Single-threaded design !

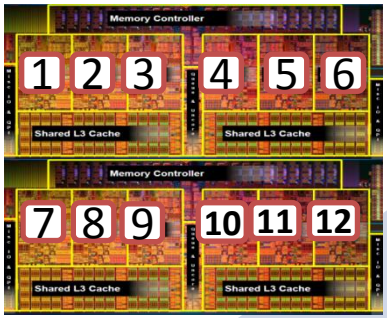
ATLAS multicore hardware

- Typical system: **[Intel-Nehalem, 8 CPU-cores/box, 2Gb/core, ~2.5Ghz]**
- Upgrades: **[Intel-Sandybridge, 12-24 CPU-cores/box, 2-3Gb/core, ~2.5Ghz]**
- In future: **[32 and more CPU-cores/ box , 3 Gb/core, ~2.5Ghz]**

Multicore Era

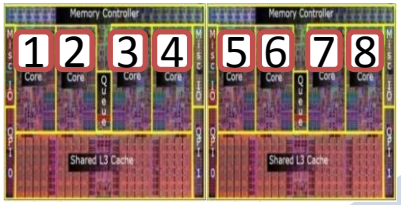
Ghz Era

power wall



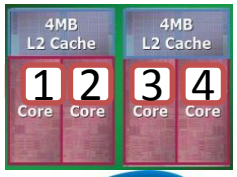
MANY CORES

32 and more

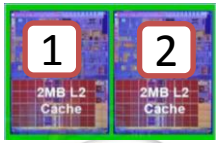


12 cores

8 cores



4 cores



2 cores

HyperThreading
Enables 2 logical cores per 1 physical core

Efficient exploitation of multiple cores on single box is essential!

PART I:
Harnessing multi-core systems in ATLAS

What to expect from parallel solution?

1. **Sharing resources (memory and IO)**
   - Sharing Large part of process memory by cores.
   - Sharing IO and Persistent to Transient conversions

2. **Scalability (parallel event processing throughput)**
   - Close to linear scaling on many-core machines
   - Explore Hyper-Threaded CPU cores.

3. **Transparency**
   - User easily exploits multiple cores
   - Minimum changes to the existing code base

What to expect from parallel solution?

4. Configurability

- Easy and robust control of AthenaMP.

5. Generalization

- Works for all Atlas domains:
  reconstruction, simulation, digitization, pileup

- Works for all Athena run types:
  jobos, job-trfs

- Works for all i/o data types and storages:
  data types: **bs, raw, rdo, esd, aod, tag, histos, perfmon, monitor**
  storages: **pool, root-storage, root-collections, root, custom-types.**

Possible Parallelization Approaches

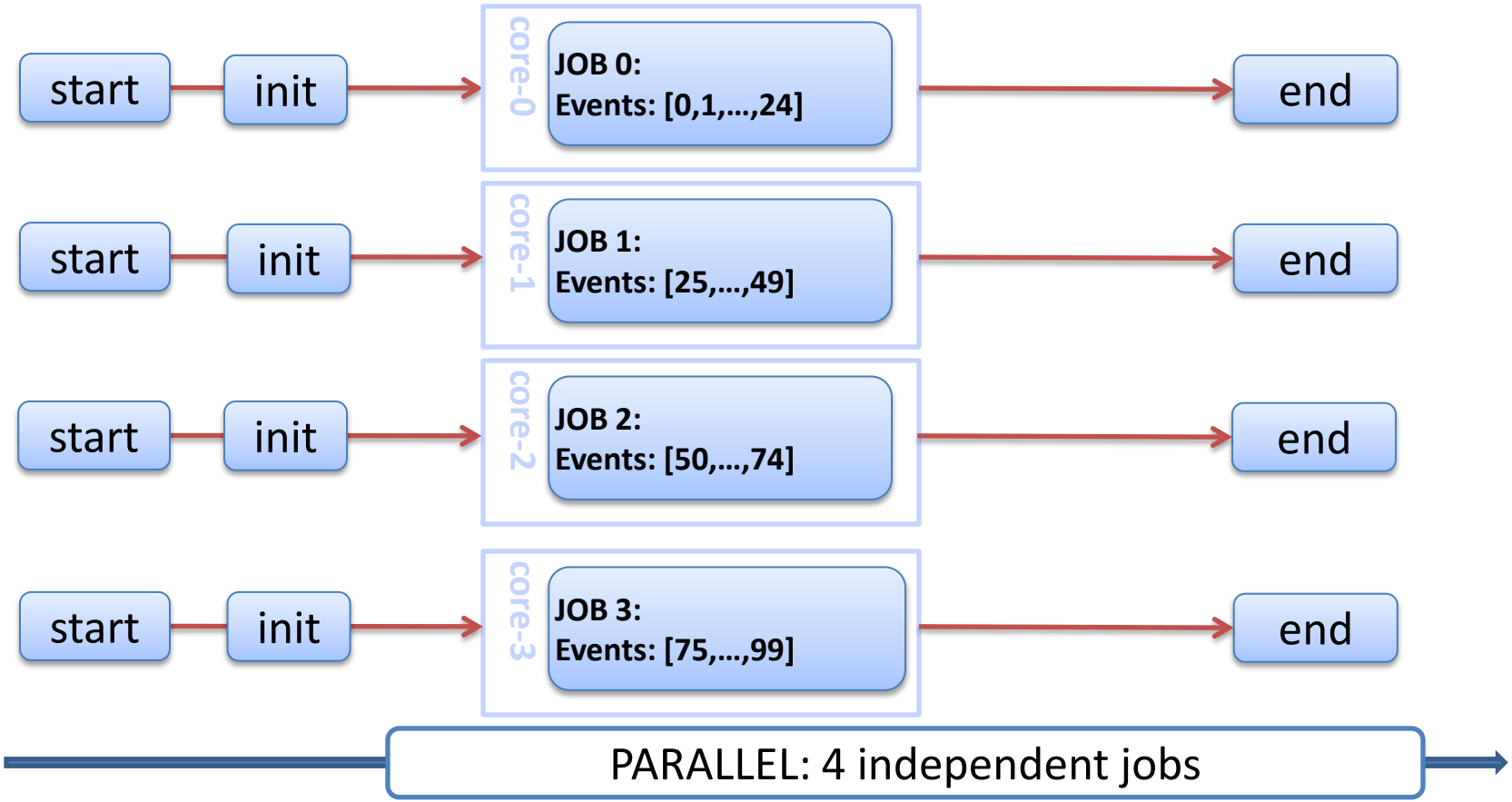
- **Jobs in parallel (job parallelism)** — **Athena MJ**
  - Simplest, no code rewriting
  - No sharing of resources
- **Events in parallel (event parallelism)** — **Athena MP**
  - Minimal changes to the existing code
  - Share Memory (code, configuration, detector data, etc.)
  - Requires: Parallelization of I/O
- **Sub-event parts in parallel (fine-grained parallelism)**
  - Parallelize tasks and "regions of interest" within Atlas domain and event using multithreaded approach.

Athena MT , GPU (future plans)

Athena jobs in parallel

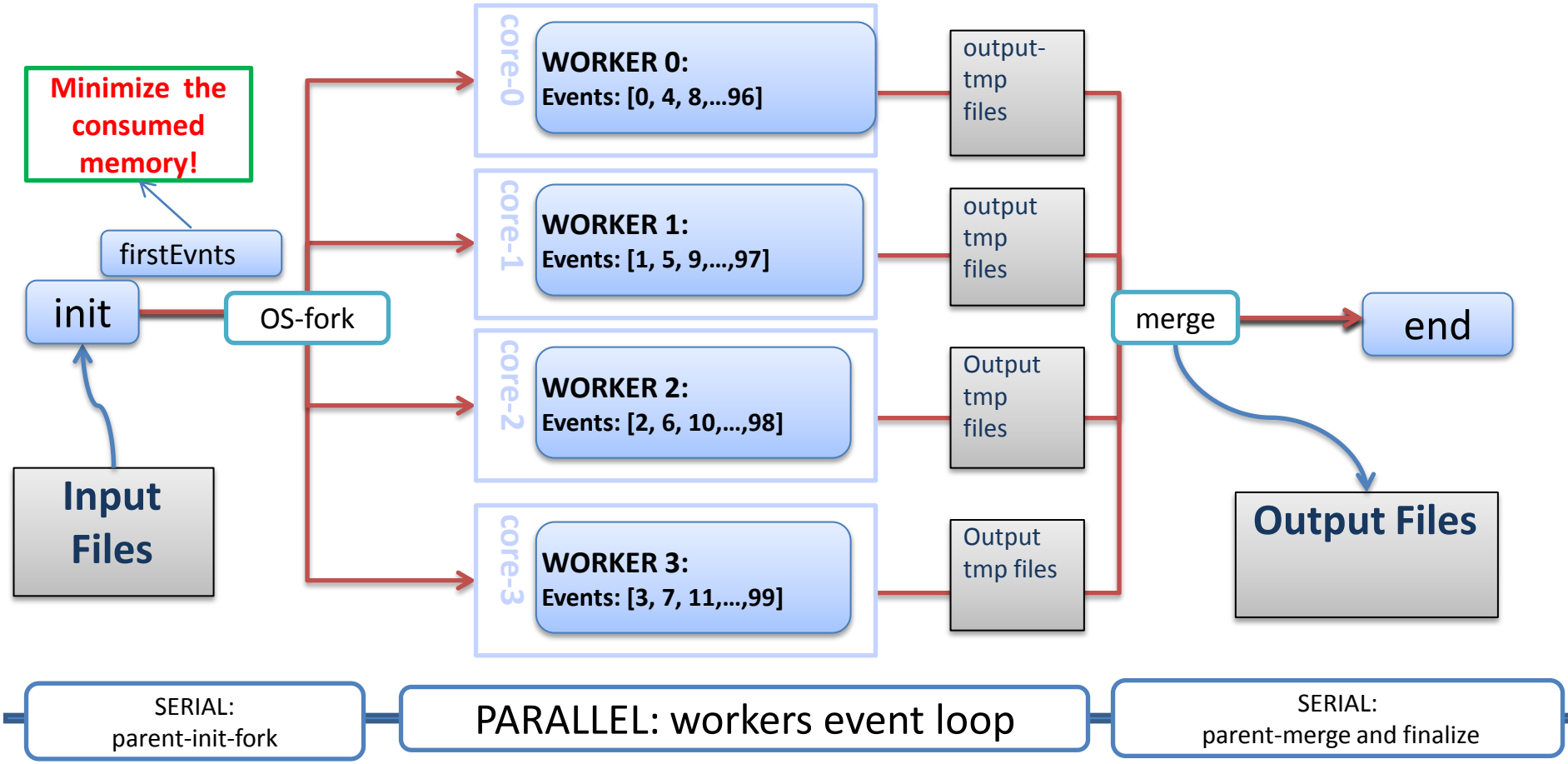
Athena MJ

```
for i in range(4):
    $> athena.py  -c  “EvtMax=25; SkipEvents=$i*25”  Jobo.py
```



AthenaMP – transparent approach to process events in parallel

$> athena.py --nprocs=4 -c EvtMax=100 Jobo.py



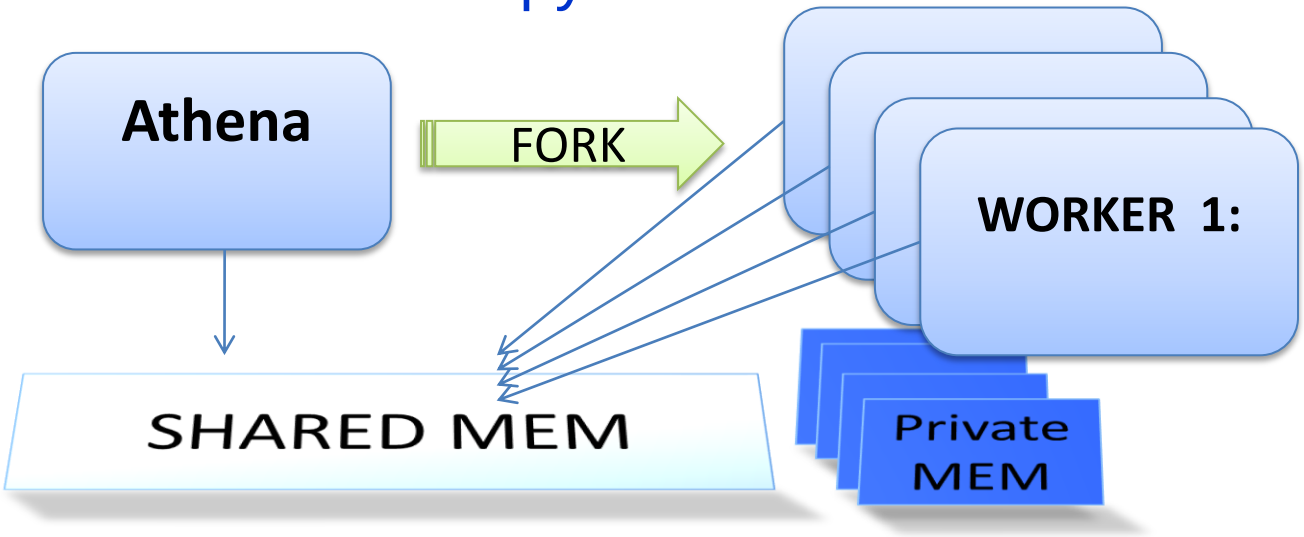
WORKER - forked clone of serial process

- Master process memory is used by workers
- Processes see the same physical memory while reading

How AthenaMP Works?

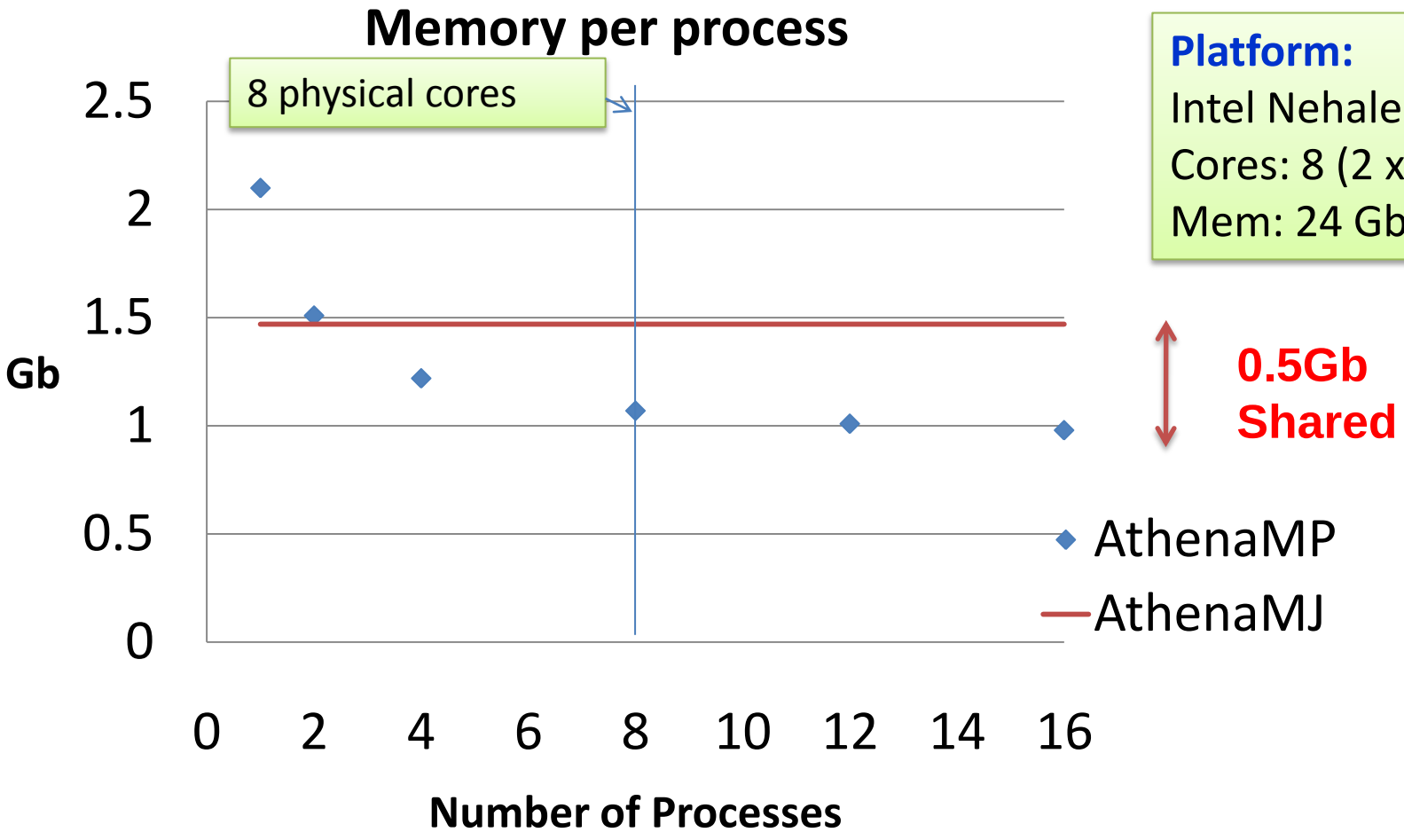
1. Initialize serial version of Athena.
   - Wrap athena EventLoopMgr with mp version during initialization.
   - Configure athena job, initialize athena, load necessary libraries.
2. Create pool of event processing Workers
   - Fork workers using *multiprocessing* package in python
   - Processes see the same physical memory initially
   - New allocations and touched pages created in private memory

WORKER = Copy-on-Write forked clone



3. Merge Output:
   - Input: Each worker processes separate set of events from common event file.
   - Output: Each worker produces separate set of output files.

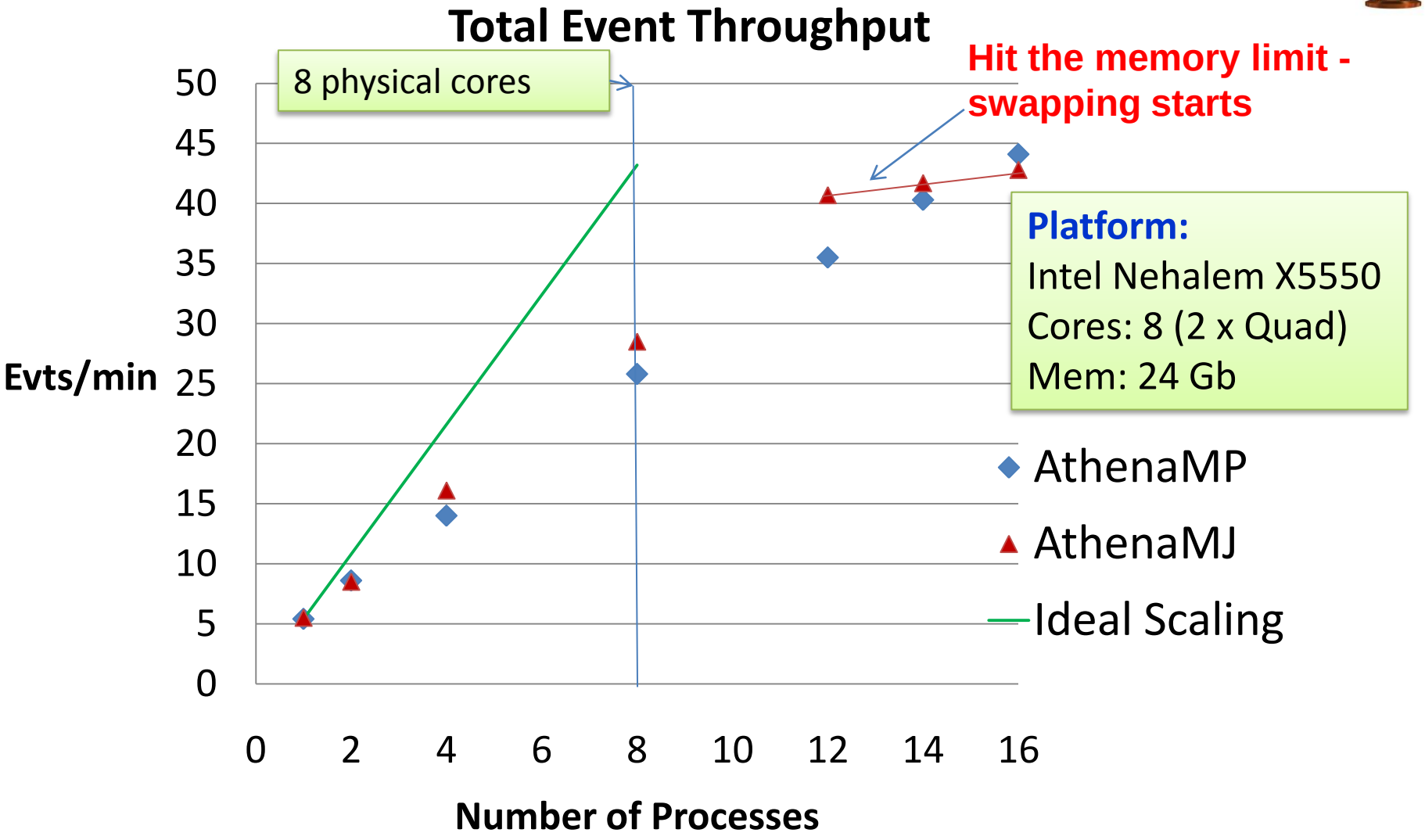
1. Resource sharing in AthenaMP



Platform:
Intel Nehalem X5550
Cores: 8 (2 x Quad)
Mem: 24 Gb

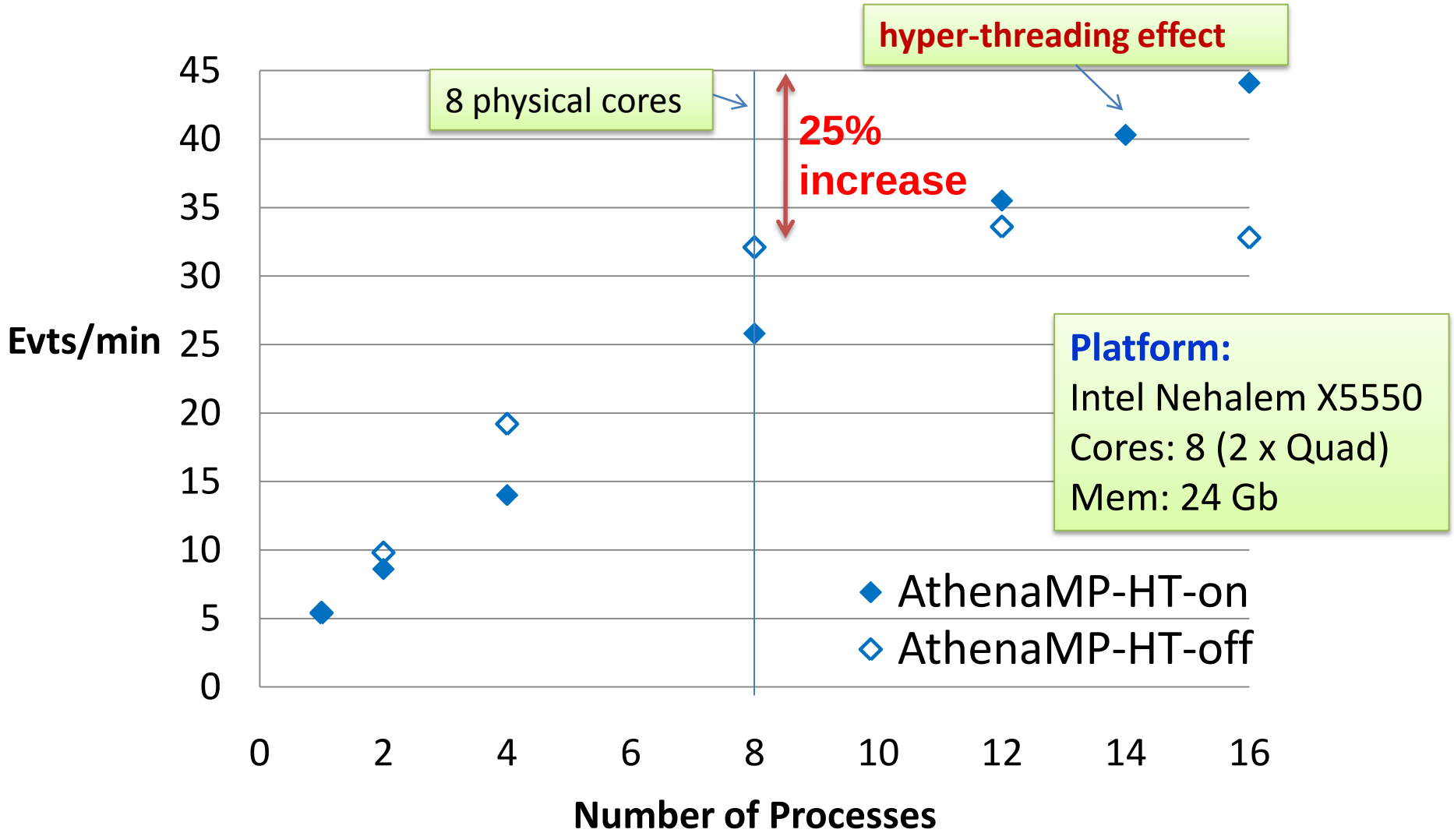
AthenaMP 0.5 Gb physical memory shared per process

2. Scaling of AthenaMP and AthenaMJ



Gain from Hyper-Threading

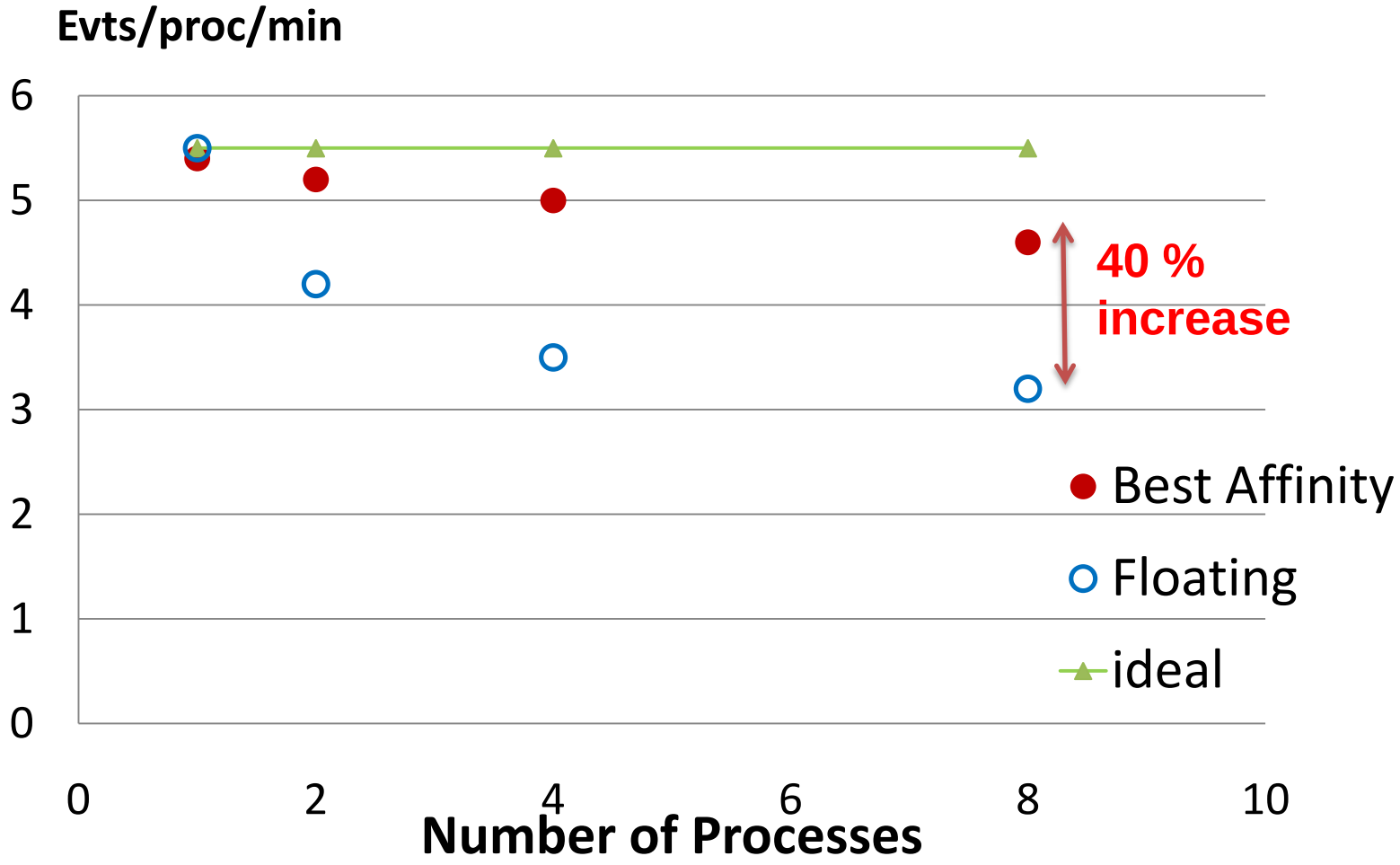
AthenaMP total event throughput



Confining workers to cpu-cores

Affinity: pinning each processes to a separate CPU-core

Floating: each process scheduled by OS; frequent task switching



3. Transparency of AthenaMP

Turn any standard ATLAS job into mp-version by:

```
athena.py   --nprocs=4   rdotoesd.py …

reco_trf.py --athenaopts="--nprocs=2" …
```

4. Configurability of AthenaMP

Configuration of AthenaMP is done using the jobproperties:

```
from AthenaMP.AthenaMPFlags import jobproperties
```

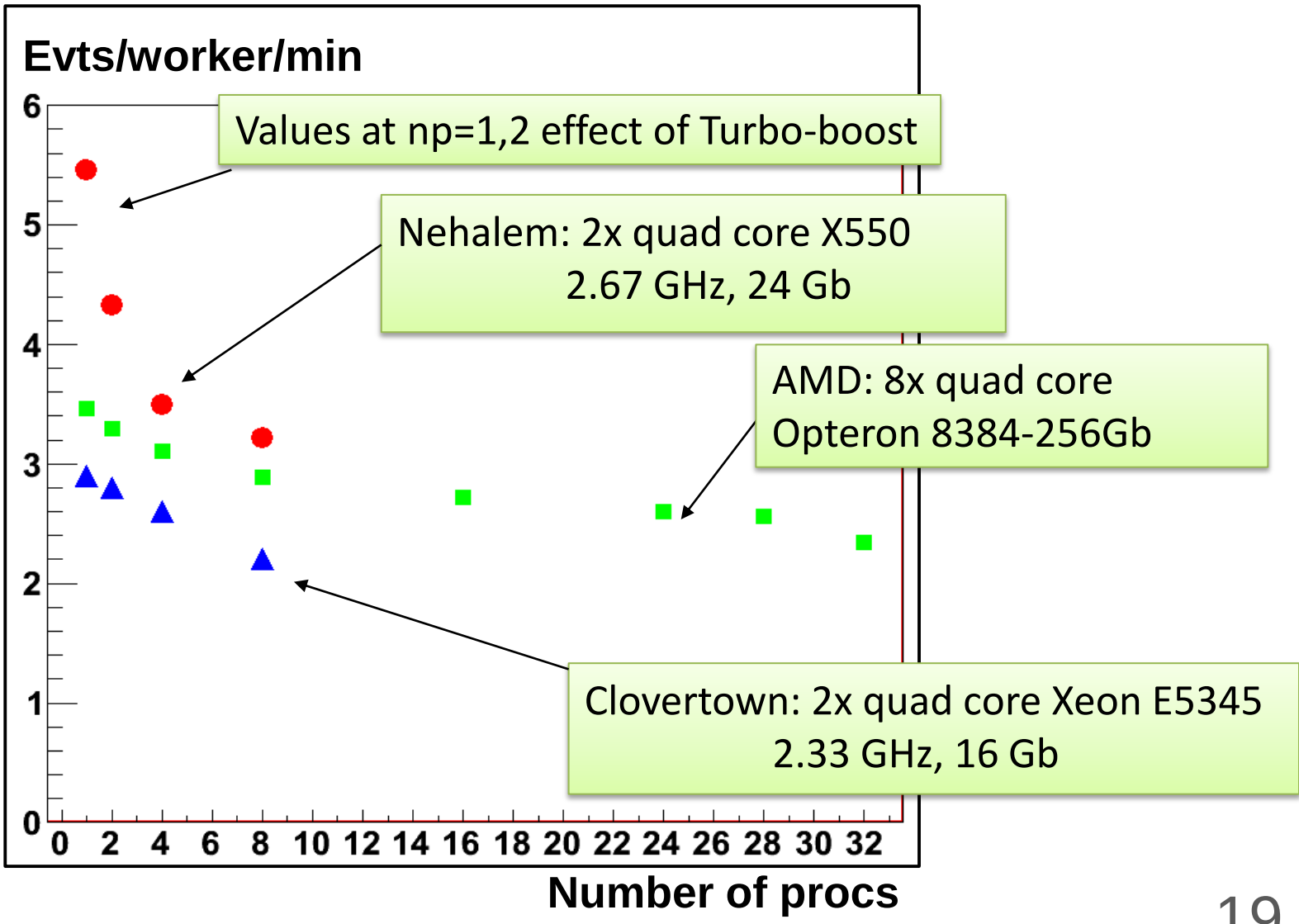
Various switches and properties should allow user-friendly control.

```
# AthenaMP/python/AthenaMPFlags.py
from AthenaMP.AthenaMPFlags import jobproperties as jps
mpjp = jps.AthenaMPFlags

mpjp.EventsBeforeFork = 0
mpjp.AffinityCPUList = [0,1,2,3,4,5,6,7]
mpjp.TmpDir = "/tmp/tmp_dir"
mpjp.doFastMerge = True
mpjp.doRoundRobin = False
# more will be added as needed
# example: AthenaMP/share/mp_rdotoesd.py
```

5. Challenges of Generalizing AthenaMP

- **RECONSTRUCTION**
  - Job option recos
  - Job transform recos, chained job transforms (rdo2esd2aod-tag)
  - Large nbr. of different output files
  - Problems with PoolFileCatalog
- **ByteStream-RECONSTRUCTION**
  - Seeking didn't work
  - Problems with PoolFileCatalog
- **SIMULATION**
  - Random generator seeding
  - Outside framework files
- **DIGITIZATION**
- **PILE UP**
  - Uses it's own PileUpEvtLoopMgr
  - Multiple inputs and files.

Scaling of AthenaMP on multi-core machines



AthenaMP Progress

Achievements:

- AthenaMP reduces memory requirement by 35%.
- Good mp-scaling on current hardware
  - Affinity settings exploit CPUs better than Linux CPU scheduling.
  - MP-Queue balances workers arrival times.
- Hyper-Threading gives 25-30% gain on throughput.

Remaining Challenges:

- IO will become limiting factor for multiple workers: Optimize parallel IO that currently relies fully on OS and requires additional job of merging the output .
- Mitigate NUMA effects on multi-core machines
- Exploit the finer grained parallelism by using threads.

PART II:
Performance and optimization study of Athena

Tools Used

- Linux tools:
  - sar (I/O to disk and system-CPU, Memory, IO loads)
  - vmstat: memory performace
  - IPM: time spent in I/O vs computation
  - numastat/numactl: reports/controls NUMA memory settings

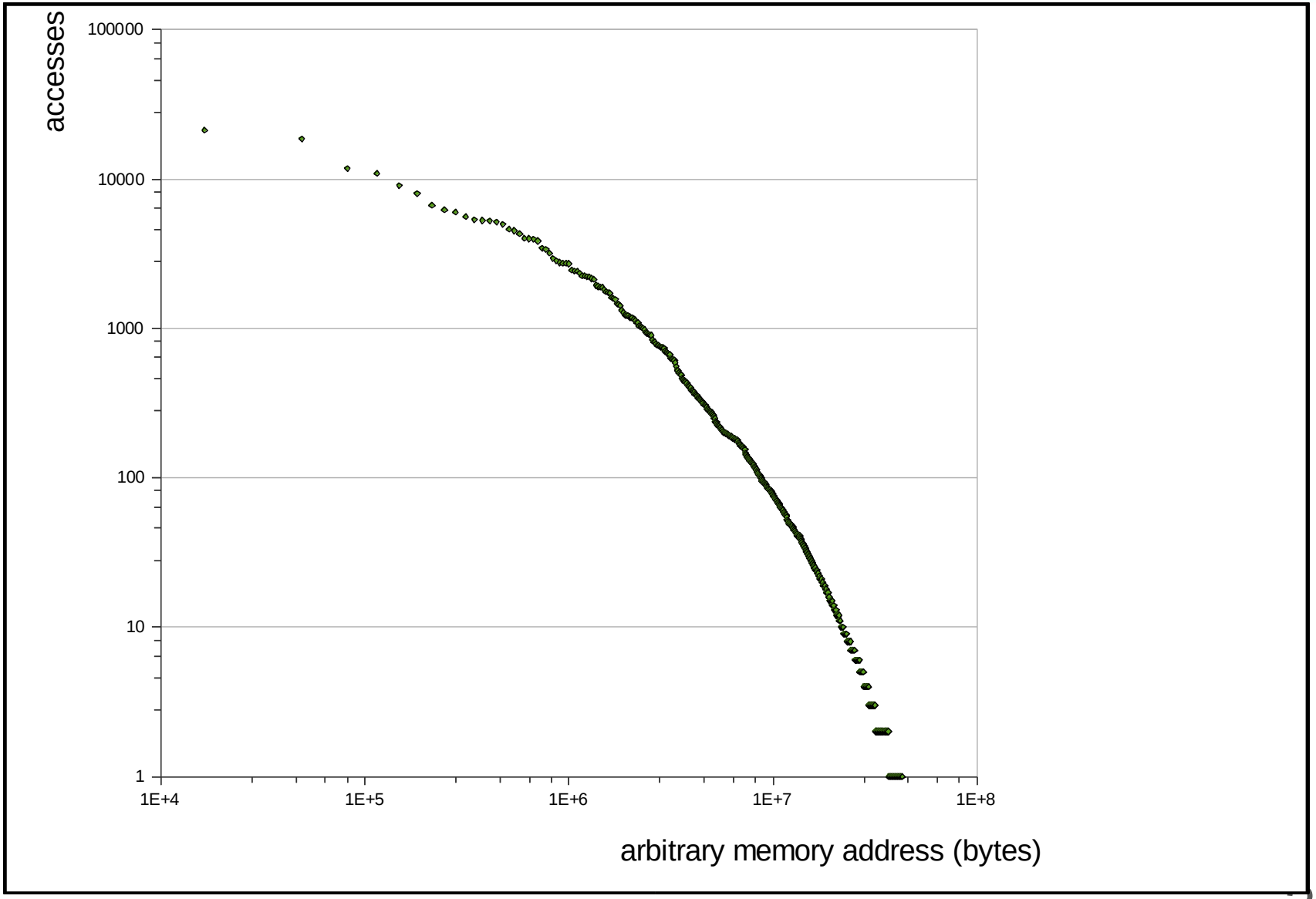
- Intel Performance Tuning Utility (**PTU**)
  - Uses linux kernel module to provide a sampling profiler
  - Captures information from hardware counters available on Intel chips
  - Most accurate tool to understand what's going on at the hardware level
  - HUGE number of counters available

Initial Assumptions

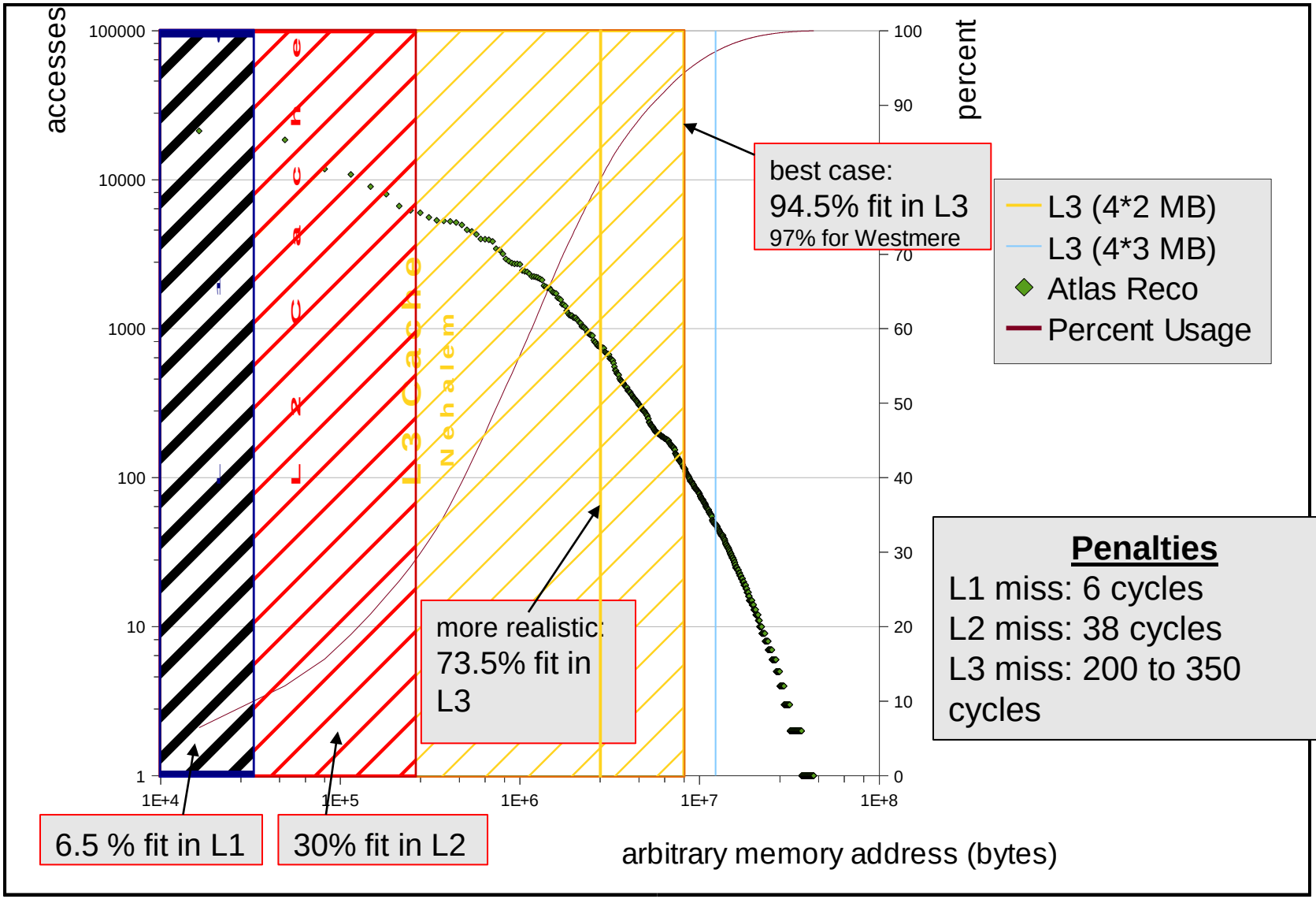
- Our initial assumption was that we were I/O and memory bandwidth limited

we were WRONG

Size of Atlas Reconstruction



Cache (un)friendliness of Athena



Issues with large OOP Code Bases

- Function calls result in added instructions
  - Call and return
  - Runtime address resolution (trampolines) required for position independent code/ shared object cross invocations
    - Indirect branches can be more costly
  - Freeing & restoring registers for local use
  - Setting and reading function arguments
- Virtual function calls (function pointers) increase indirect call instructions and associated pointer loads
  - Virtual functions can't be inlined!
- Atlas code has 2500 shared libraries!

Observations from Intel-PTU

- In Atlas code, functions are on average only 33 instructions long
- Overhead for function calls is anywhere between 6 and 12 instructions
  - We can have up to 35% overhead!
- We also see instruction starvation of about 20%

Short Term Solutions

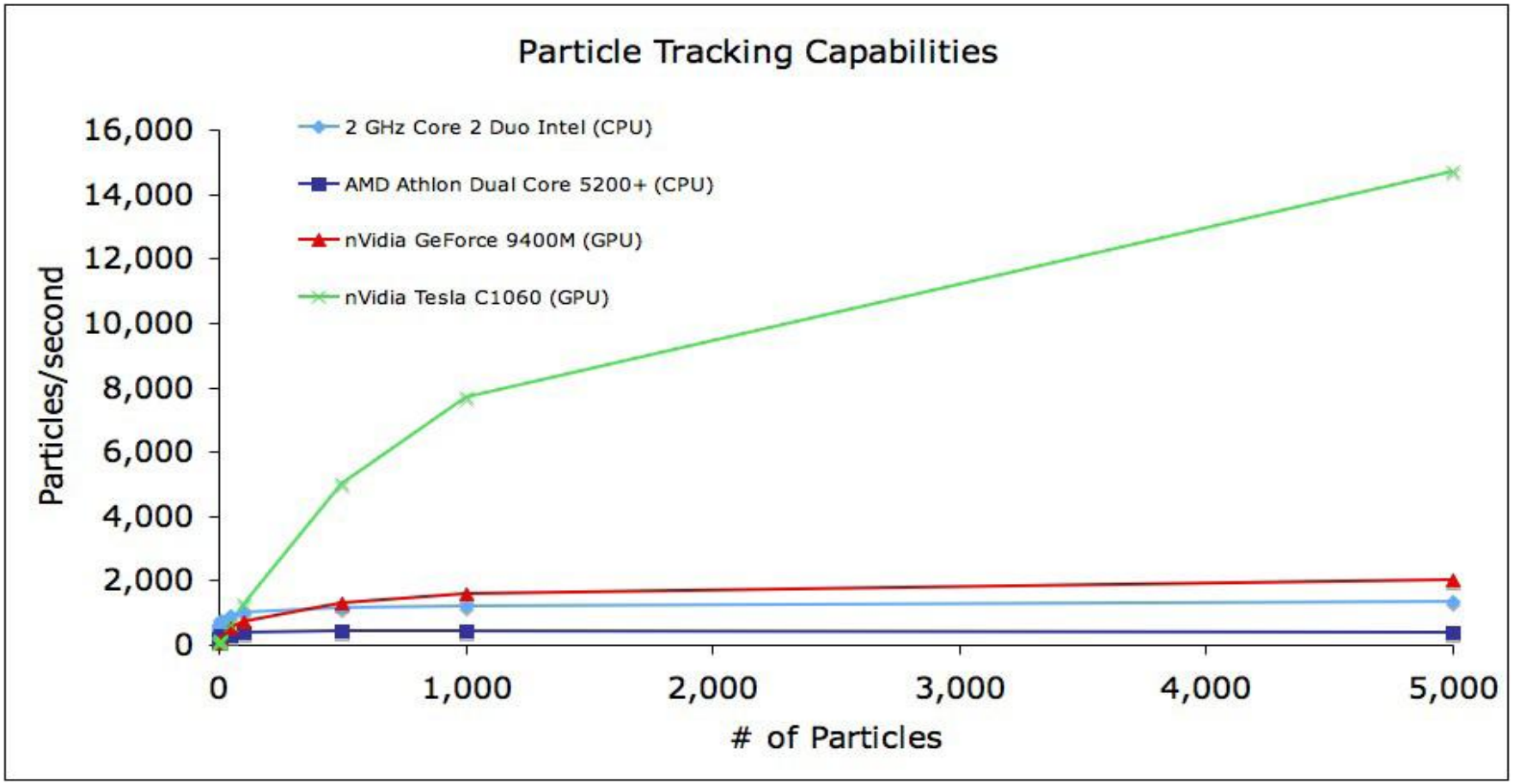
- Use social network analysis to identify clusters of active, costly function call activity
- Order clusters by total time and/or total “cost”
  - Split time of functions shared between clusters by call counts
  - Calls have a direction
  - Utility functions must not be viewed as bridges
- Manually reduce function count in hot clusters by explicit code inlining
- Prioritize work by call overhead cost to be gained
- Reduce cross shared object call counts

Part III:
GPU computing in ATLAS:

Particle tracking in magnetic field simulation.

Solve the differential equation with 4th order RungaKutta Integration

A. Washbrook, P.J. Clark – ATLAS-Group @ University of Edinburgh, UK



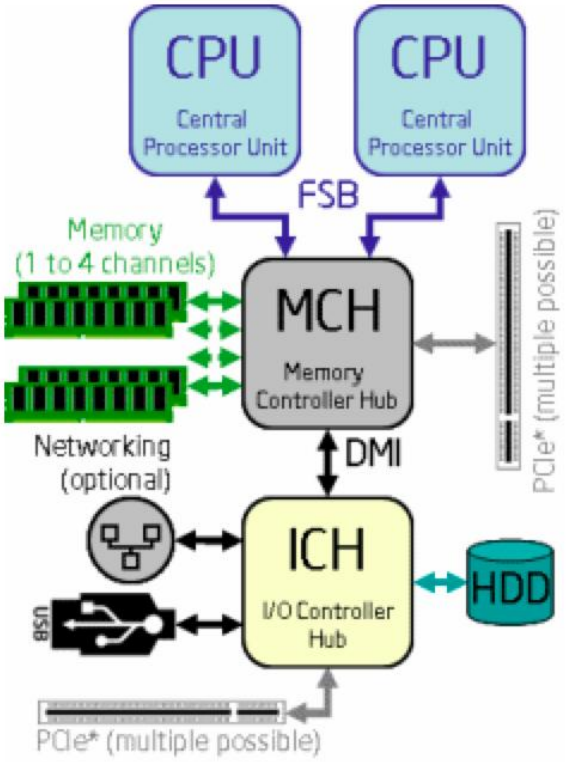
Preliminary results (Tesla C1060): achieved a factor 32 speedup !

THANK YOU!

CPU Architectures used in ATLAS

Intel sub-Nehalem

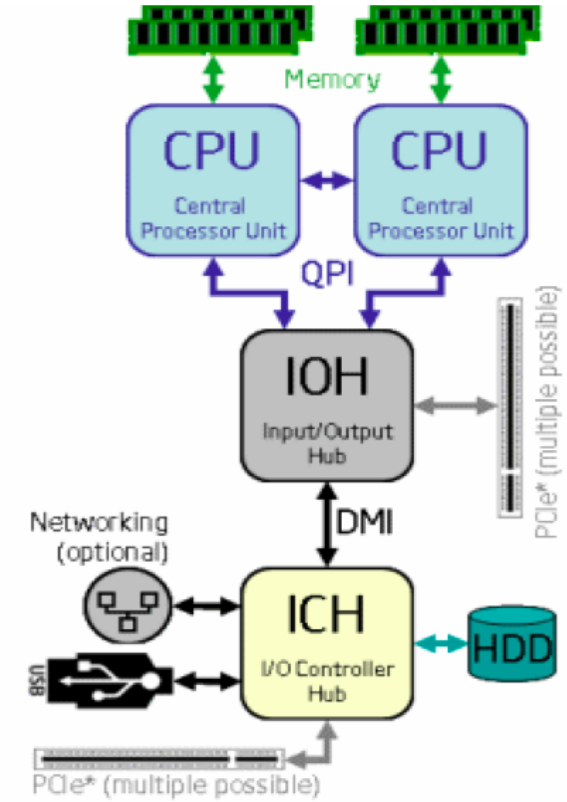
most of LXPLUS machines: Voatlas91,lxplus250,lxplus251



CPU-Memory symmetric access

Intel Nehalem

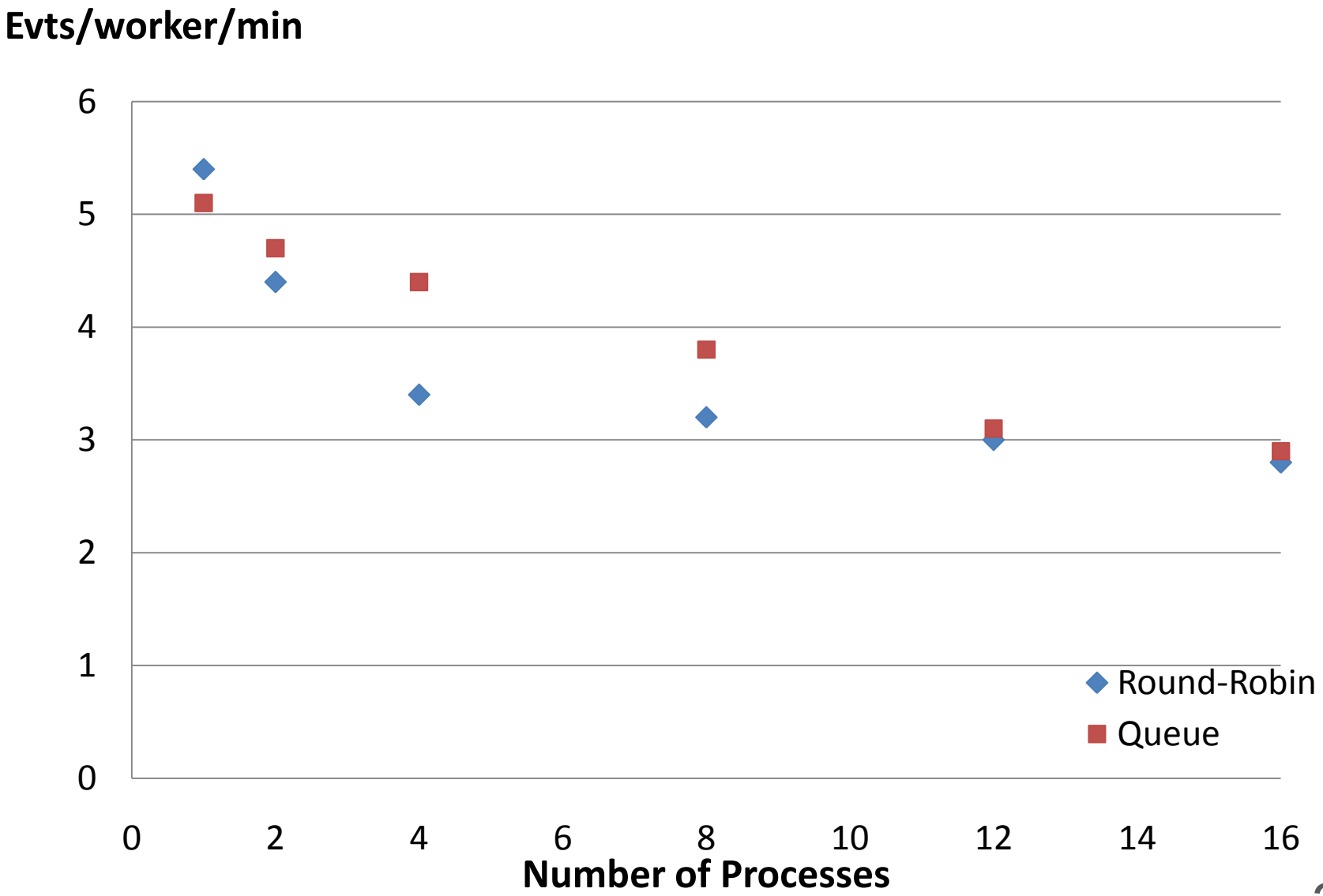
coors.lbl.gov, rainier.lbl.gov



- Hyper Threading ->two logical cores on physical one
- QPI Quick Path from CPU to CPU and CPU-to-Memory
- Turbo Boost -> dynamic change of CPU-frequency

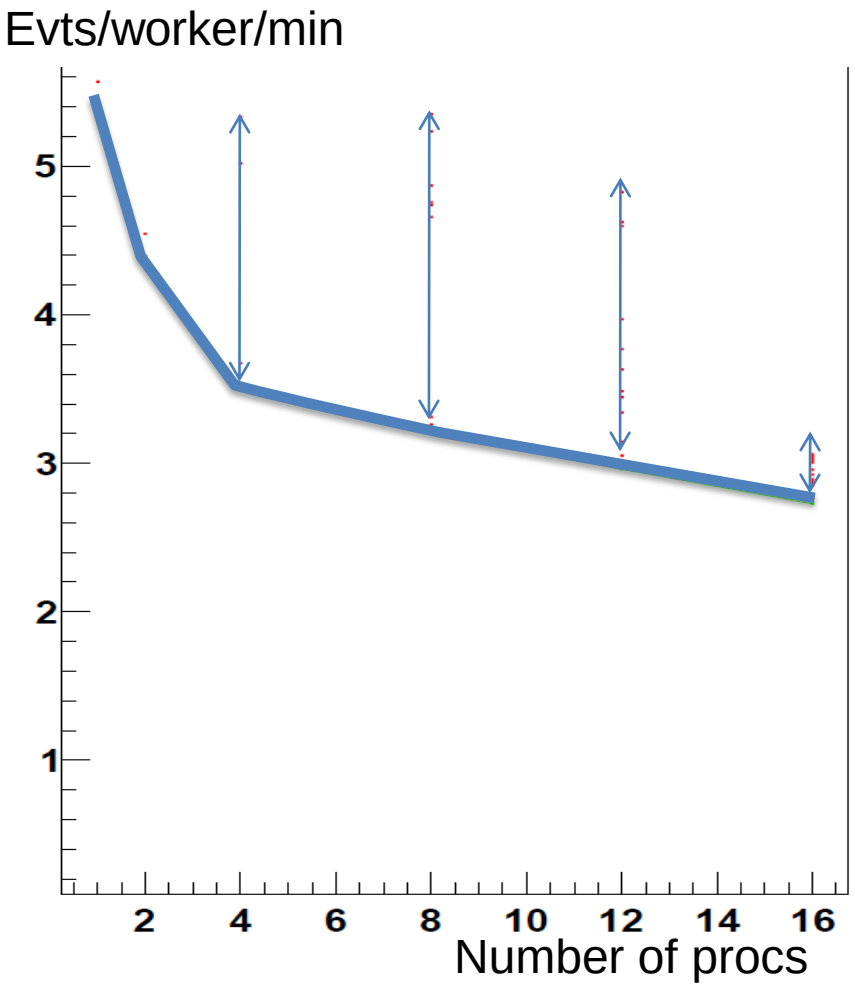
- CPU-Memory non-symmetric access (NUMA)

Event processing for Queue vs. Static

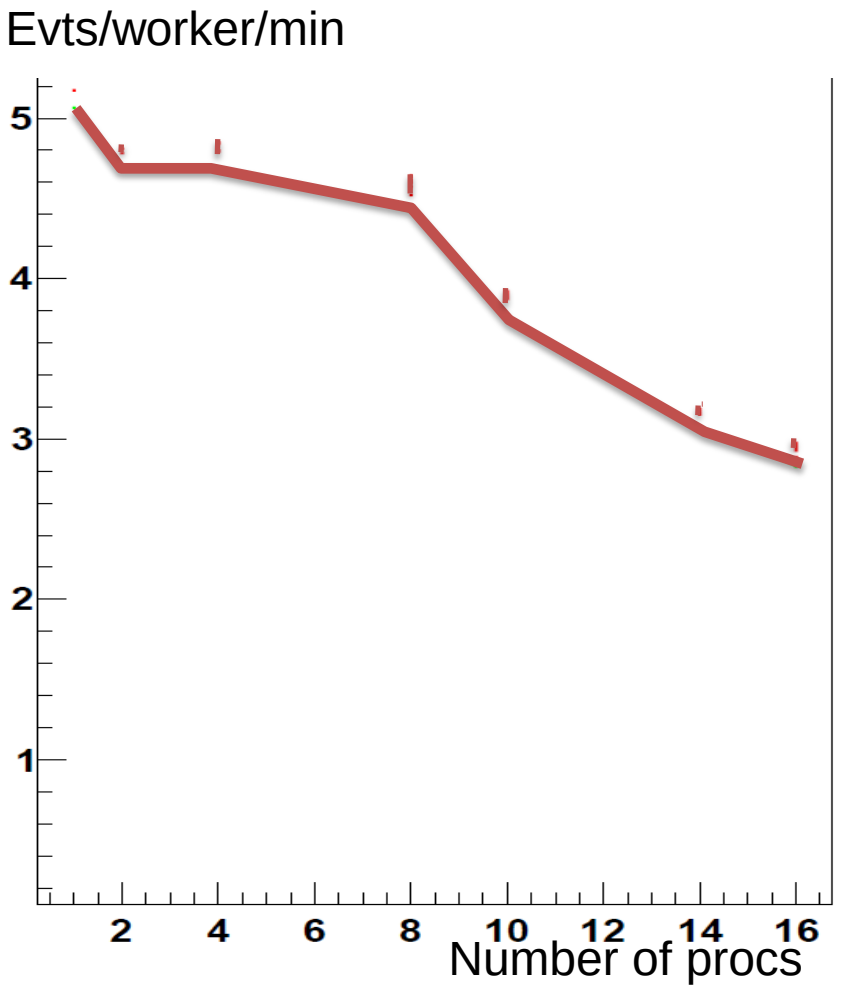


Workers throughput for Queue vs. Round-Robin

Round-robin event distribution



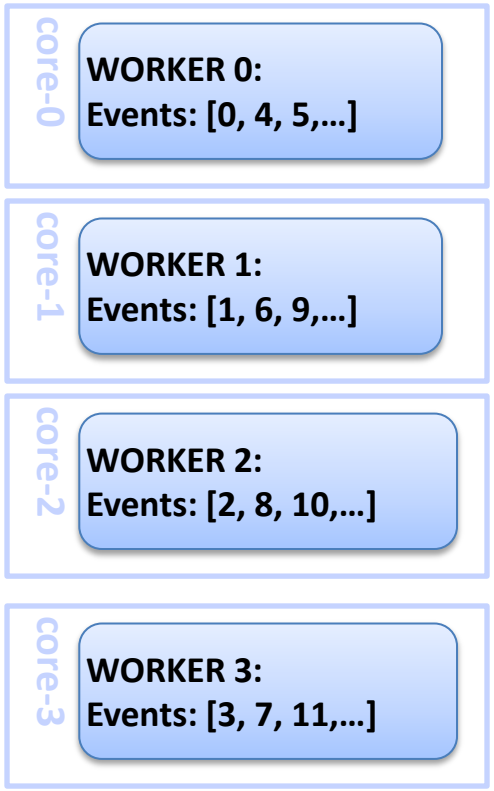
Queue event distribution



Balance the arrival times of workers!

Event distribution using Queue…

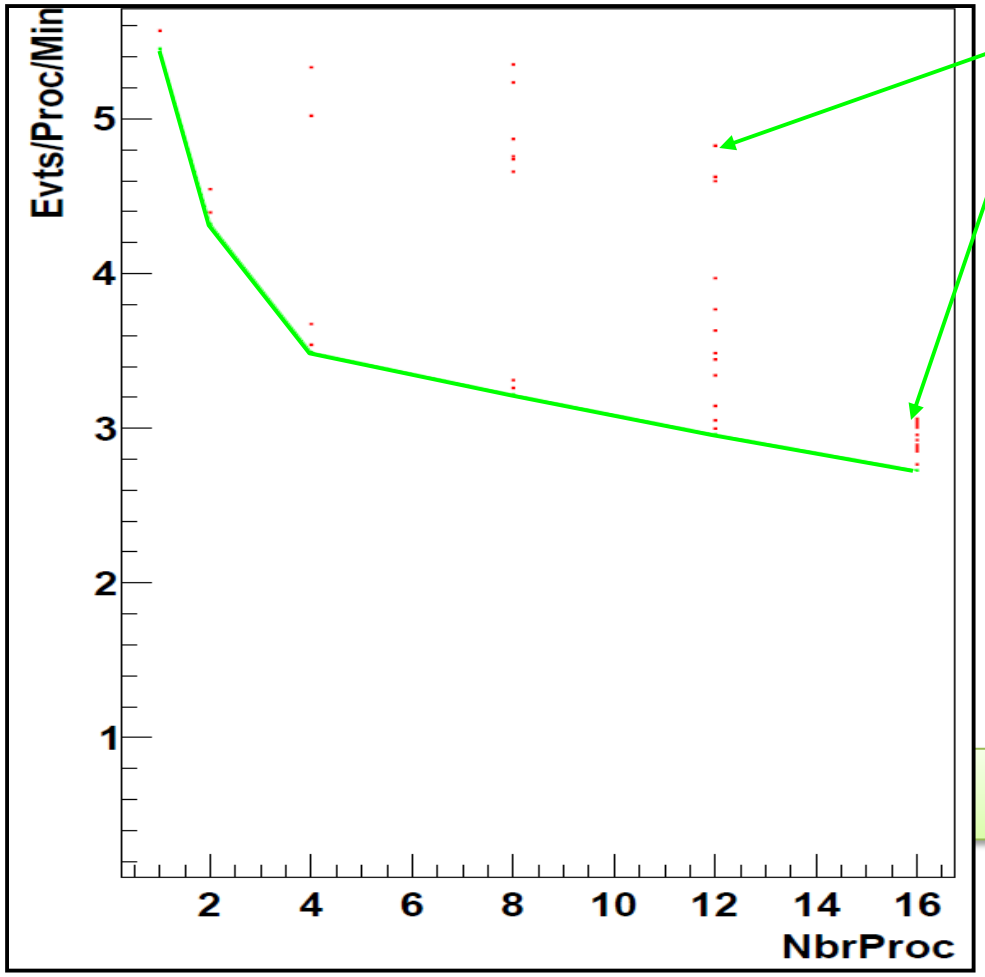
```
events = multiprocesssing.queue(EvtMax+ncpus)
events = [0,1,2,3,4,…,99, None,None,None,None]

…

evt_loop(evt=events.get(); evt != None):
     evt_loop_mgr.seek (evt_nbr)
     evt_loop_mgr.nextEvent ()
```



Balance the arrival times of workers!

Slower worker doesn't get left behind

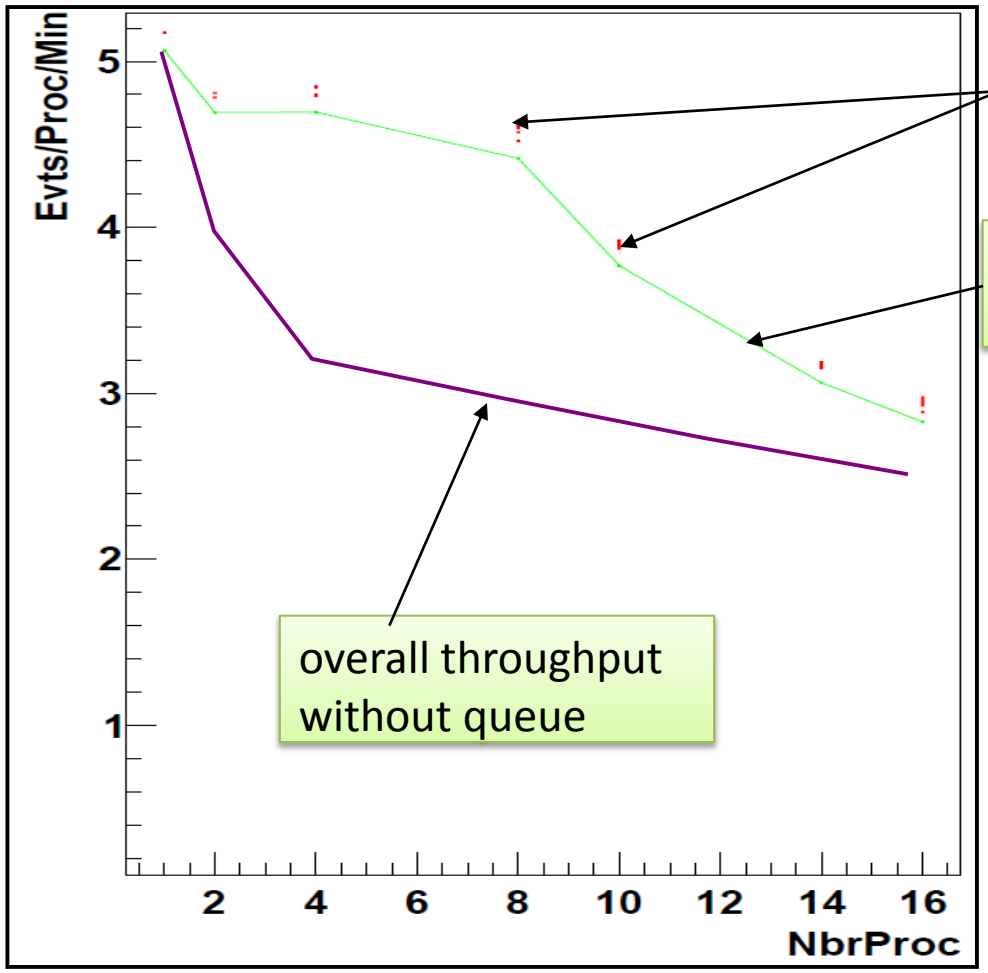
Worker Throughput, No Event Queue



individual worker event rates

- 8 core HT machine

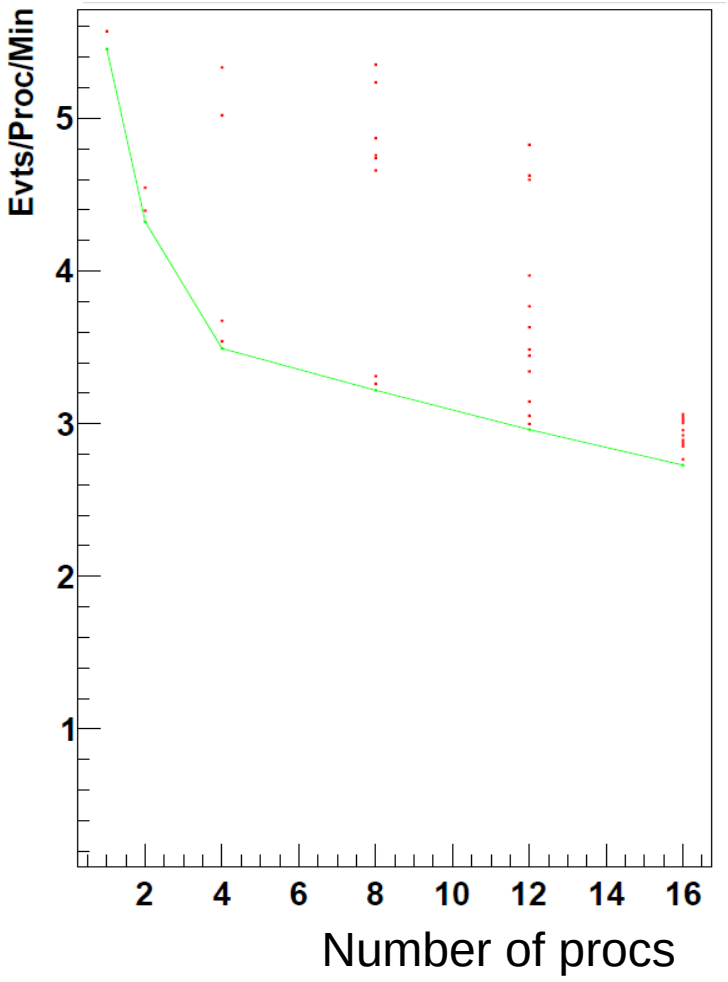
Worker Throughput with Event Queue



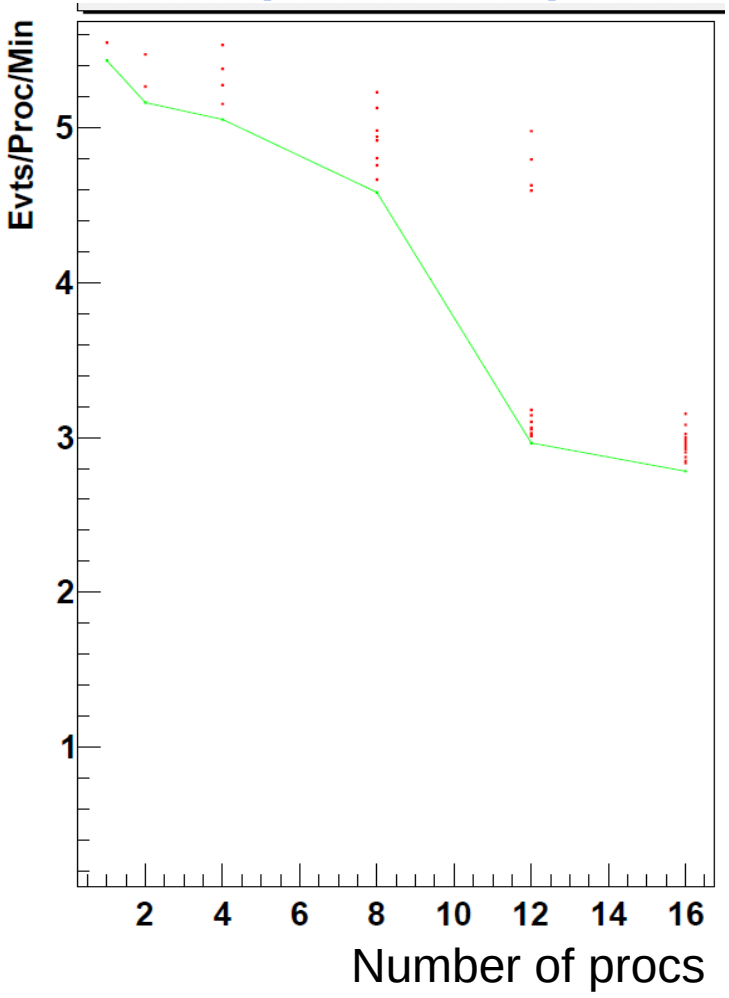
- 8 core HT machine

Effect of Affinity Settings

Workers floating



Workers pinned to cpu-cores



Maximizing AthenaMP performance

1. Externally available performance gains (without touching the athena code)

- Architectural gains: HyperThreading, QPI, NUMA etc.
- OS gains: affinity, numactl, io-related, disks, virtual machines, etc.
- Compiler, Malloc, etc.

2. Gains from Athena/AthenaMP design improvements:

- Faster initialization…
- Faster distribution of events to workers...
- Faster merging: merging events processed by workers instantly by one writer on a fly, without waiting for workers to finish…
- Faster finalization…

endless ground for improvements :)

Running AthenaMP jobs.

Tests:
/share/mp_genevt_test.py
/share/mp_basic_test.py

Running Reco:
athena.py –nprocs=$ncpus reco_jobo.py

#example of AthenaMP configuration for reco job
/share/mp_reco_fast.py

Running Job Transform:
==========
csc_atlasG4_trf.py --athenaopts="--nprocs=2"
inputEvgenFile=evgen-105145.pool.root
outputHitsFile=HITS.pool.root maxEvents=2 skipEvents=1 randomSeed=3945
geometryVersion=ATLAS-GEO-10-00-00 physicsList=QGSP_BERT
conditionsTag=OFLCOND-SIM-BS7T-02 IgnoreConfigError=False AMITag=s765
==========

The scripts for mp-scaling measurements in Athena

mpMon.py - mp-scaling measurement script that runs automatically athenaMP and collects system data using sar (CPU, IO, Memory) + numa activity during the run; and extracts worker-time-statistics from the parent and workers logs. The script allows to vary numa bindings, affinity settings.

```
$>  mpMon.py --jobo 'athena.py --nprocs=$NPROCS rdotoesd.py -c EvtMax=$MAXEVT'  \
            --np [1,2,4,8,12,16,18]     --ne 100    --comments "HTon.TBon"   --doPlots    --output   mpMon.log
```

mjMon.py - mp-scaling for athena multi jobs (Athena MJ), similar to mpMon.py

```
$> mjMon.py –jobo ‘athena.py –c EvtMax=$MAXEVT rdotoesd.py’ \
            -np [1,2,4,8,12,16,18]  –ne 100     –comments “Hton.Tbon”  –doPlots     –output mpMon.log
```

#mp-scaling of the job-transform

```
$> mjMon.py --np [1,2,4] --ne 100  --se 10 --output mjMon.log --comments “trf“  --doPlots  \
 --jobo 'csc_atlasG4_trf.py inputEvgenFile=Evt.pool.root  outputHitsFile=HITS.pool.root  maxEvents=$MAXEVT  \
skipEvents=$SKIPEVT  randomSeed=54298752 geometryVersion=ATLAS-GEO-08-00-01 physicsList=QGSP_BERT  jobConfig=NONE’
```

both scripts come with Control/AthenaMP package.

MP-scaling reports of rdotoesd for both AthenaMP and AthenaMJ on different machines here:

https://docs.google.com/leaf?id=0B1Yjr6DOl_RaOTM0OTUyNGItMjk4Ni00NmRmLTg4NjctMDc5ZWRmM2I0NDA5&hl=en