Performance studies for ATLAS workloads in many-core Grid and HPC environments

A. Filipcic[1] F. G. Sciacca[2], M. Weber[2]
1. Jožef Stefan Institute, Ljubljana, Slovenia - 2. LHEP, University of Bern, Switzerland
On behalf of the ATLAS Computing Activity

Improvements in microprocessor technology push the trend towards CPUs featuring higher core densities. These have increased significantly over the past five years for the x86_64 architecture, which is dominating in the LHC computing environment. High core density is not only a feature of large HPC systems, but is also readily available on commodity hardware preferentially used at Grid sites. The baseline multi-core workloads are however still largely based on 8-core jobs across WLCG. The jobs are sized accordingly in terms of number of events processed and memory requirements. In this work, the performance of some ATLAS workloads is investigated when scaling the number of cores used up to whole node where possible and at different job sizes, with the aim of providing input to software developers.

Introduction

In the *Worldwide LHC Computing Grid* (WLCG), physics event processing is serial in nature. Embarrassingly parallel processing is used in order to better scale up to large systems. The use of higher core counts has advantages, including better memory sharing between threads and lower overhead on compute nodes and batch system. Whole node usage is commonly mandated at large HPC centres. In ATLAS each computing job consists generally of one payload, executed across several processing cores (multicore). 8-core dominate with 63%, while 1-core make up for 16% of the processing (1-year statistics).

We have set out to investigate whether the ATLAS code and infrastructure are ready for the **scale-up to higher core counts**. The workloads investigated are:

- **MC reconstruction with minbias Overlay**
- **Event generation (*Madgraph*)**
- **Detector Simulation (*Geant4*)**

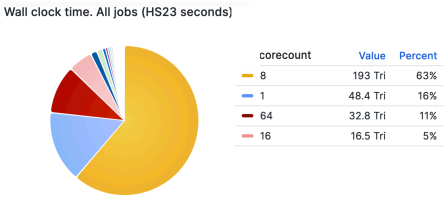

Figure 1. Work completed in HEPScore23 units as a function of the core count

Investigation Strategy

- **Select one task per workload**
- **Select one job out of each task**
- **Re-run each of the 3 jobs on:**
  - a controlled interactive environment
  - Grid/HPC sites
  - for various core counts and nr. of events
- **Performance indicators:**
  - CPU efficiency per core: *CPU time / walltime*corecount*
  - Event throughput

The trends of the metrics collected are largely not dependant of the computing site. The results reported refer to jobs run on the Vega HPC

MC Reconstruction with minbias Overlay

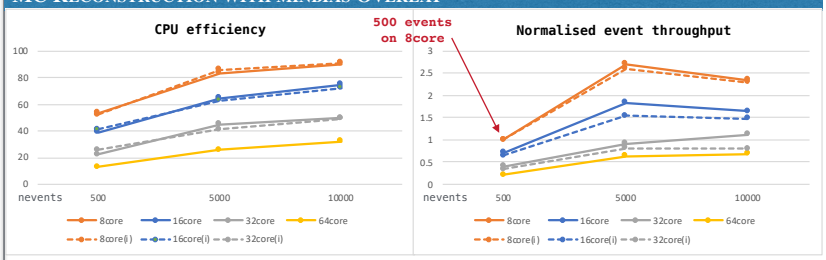

Figure 2. CPU efficiency vs nr. of processed events

Figure 3. Event throughput normalised to the baseline job settings vs nr. of processed events

The normalisation factor is the event rate of the job run with the baseline settings: 500 events on 8 cores

solid – Vega
dotted - - - Interactive

Selected job baseline settings: processing 500 events on 8-core
Re-run: scanning the ranges 8-64 cores and 500-10'000 events
The payload consists of four job substeps:
Overlay, RDOtoRDOTrigger, RAWtoAll, Validation

Findings:

- **We found CPU deadtimes between the first three job substeps**
  - irrespective of core count and nr. of events
  - the **`Overlay`** substep only ever uses 7 cores
    - this degrades cpueff and event throughput at higher core counts
- **Tuning core count and nr. of events can improve overall performance**
  - e.g. can run 10000 events on 32 cores with good event throughput
  - overall performance decays while scaling up to 64 cores

Event generation (*Madgraph*)

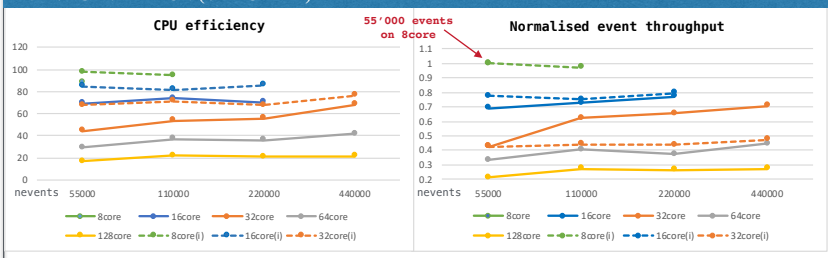

Figure 4. CPU efficiency vs nr. of processed events

Figure 5. Event throughput normalised to the baseline job settings vs nr. of processed events

The normalisation factor is the event rate of the job run with the baseline settings: 55'000 events on 8 cores

solid – Vega
dotted - - - Interactive

Selected job baseline settings: processing 55'000 events on 8-core
Re-run: scanning the ranges 8-128 cores and 55'000-440'000 events
The payload consists of three job substeps: Init, Generate, Merge

Findings:

- **`init`** and **`merge`** have consistent duration for all job settings
- in **`generate`** winding down threads is very inefficient above 8-core
- **cpueff and event throughput decrease as core count increases**
  - overall performance decays while scaling up to 128 cores
- **cpueff and event throughput fairly flat as function of nr. of events**
  - minor improvement observed with 32-core jobs with increasing nr. of events e.g. when processing 440'000 events on 32 cores
  - otherwise the overall performance is much worse than 8-core jobs

Detector Simulation (*Geant4*)

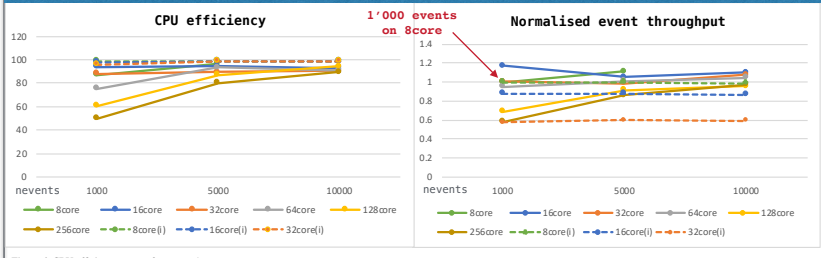

Figure 6. CPU efficiency vs nr. of processed events

Figure 7. Event throughput normalised to the baseline job settings vs nr. of processed events

The normalisation factor is the event rate of the job run with the baseline settings: 1'000 events on 8 cores

solid – Vega
dotted - - - Interactive

Selected job baseline settings: processing 1000 events on 8-core
Re-run: scanning the ranges 8-256 cores and 1'000-10'000 events
The payload consists of three job substeps: Init, Generate, Merge

Findings:

- **Up to 32 cores the trend is flat on Vega**
- **Beyond 32 cores, cpueff and event throughput decline on Vega**
  - Inefficiencies are recovered when increasing nr. of events to 10'000
- **On the interactive platform a slightly different trend is observed**
  - (*could not test on the interactive environment beyond 32 cores*)
  - cpueff is flat at any setting
    - event throughput decays while scaling from 8 to 32 cores, and is not recovered with increasing the nr. of processed events

Conclusions

The performance of the most used ATLAS workloads has been evaluated while increasing the number of cores used by the payload from the current grid baseline of 8 up to 256, the size of a state of the art compute node. It has been found that the advantages of increasing the core count are partially offset by some performance degradation measured for CPU efficiency and event throughput. Some of the inefficiencies can be partially recovered by tuning the job size, while others would require changes at the code level.