Evaluating HPX as a Next-Gen Scheduler for ATLAS on HPCs

Paolo Calafiura[1], Julien Esseiva[1], Xiangyang Ju[1], Charles Leggett[1], Beojan Stanislaus[1], and Vakho Tsulaia[1], on behalf of the ATLAS Collaboration

[1]Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA

Abstract. Experiments at the CERN High-Luminosity Large Hadron Collider (HL-LHC) will produce hundreds of petabytes of data per year. Efficient processing of this dataset represents a significant human resource and technical challenge. Today, ATLAS data processing applications run in multi-threaded mode, using Intel TBB for thread management, which allows efficient utilization of all available CPU cores on the computing resources. However, modern HPC systems and high-end computing clusters are increasingly based on heterogeneous architectures, usually a combination of CPU and accelerators (e.g., GPU, FPGA).

It is increasingly desirable to utilize High Performance Computing facilities for ATLAS data processing, and this work explores the possibility of using HPX as a basis for a next-generation scheduler for Athena running on HPCs.

1. Introduction

The ATLAS experiment operates one of the two large general-purpose detectors at the Large Hadron Collider at CERN. It carries out physics analysis in a broad range of areas in particle physics, from searches for supersymmetry and other exotic physics models to precision measurements of Standard Model parameters to investigating quark-gluon plasmas.

Much of the data collection and processing for the experiment is centralized into a framework called Athena [1], building on the Gaudi [2] framework shared with the LHCb experiment and others. Most data processing is carried out on the Worldwide LHC Computing Grid, consisting of a large number of High Throughput Computing sites distributed around the world.

Distribution of work to the grid is coordinated by PanDA [3], which submits jobs into the local batch system (e.g. HTCondor) at each site. PanDA is designed to intelligently direct work by considering the resources available at each site and each site's proximity to the necessary input data (given the large size of the datasets in question). PanDA submits single node jobs to sites, with Athena handling fine-grained scheduling of work on that node.

Given the High Performance Computing resources available from a number of countries contributing to ATLAS, it is desirable to be able to run ATLAS data processing workflows on these resources. Doing so, and in particular doing so efficiently, presents a number of challenges:

- Many HPC centres discourage or prohibit jobs using a single node, or only a few nodes
- HPC centres are increasingly focused on accelerators (e.g. GPUs)

- HPC nodes may have limited or even no direct network connectivity to the outside world. The ATLAS computing infrastructure is built on CVMFS [4] to distribute software and configuration data, which assumes nodes will be able to access the internet.

Previously [5] we have built a solution (not yet fully deployed) using the Ray [6] framework. Since Ray is primarily a Python framework it could not be easily integrated into Athena. Integrating a Ray-based solution into the PanDA infrastructure therefore led to a convoluted system with multiple components running on the worker nodes.

It is therefore desirable to build a scheduler for Athena designed for use on HPCs, allowing Athena to schedule work over many nodes, not just one. Ultimately, it would also be desirable for this scheduler to natively handle accelerators, and be able to exploit "heterogeneous heterogeneous" resources with multiple node configurations. The concept envisaged is shown in Figure 1.

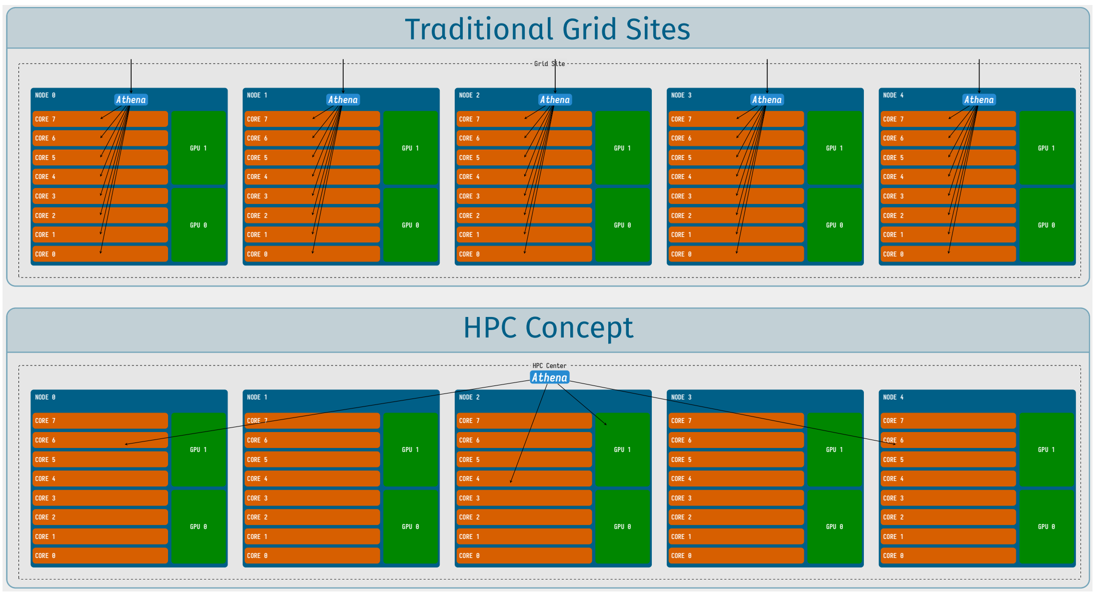


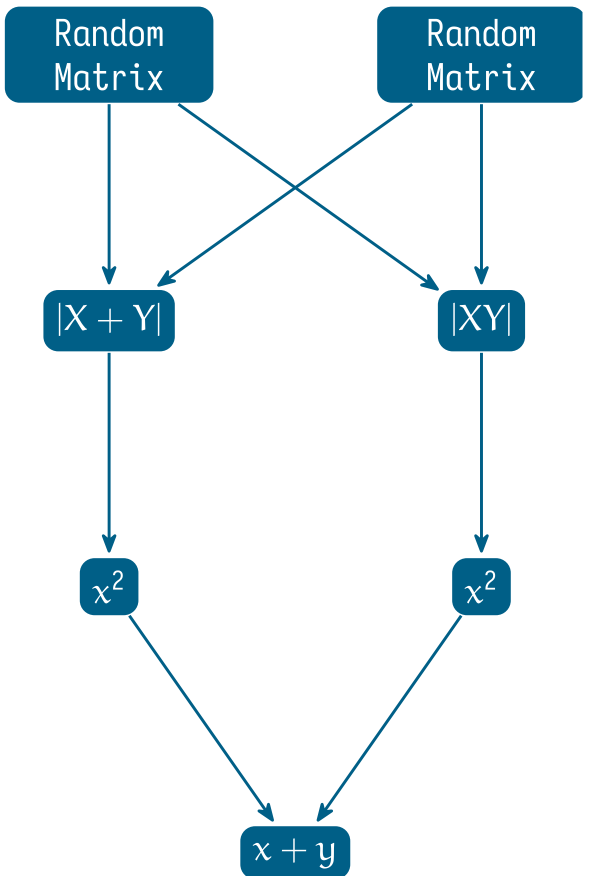
Figure 1: The concept envisaged for HPCs compared to the current scheduling model. In the current model tasks are scheduled onto all CPU cores on a given node equally, with a separate scheduler on each node. In the HPC concept they would be intelligently scheduled onto the best resource (across the cluster) for the specific task, given those available. There would be a single global scheduler across the cluster.

We have identified HPX [7] as a technology we could use to build this next-generation scheduler, and here we present a preliminary study of its suitability for the scheduling needs of ATLAS, which require executing a relatively small dependency graph of tasks repeatedly for a large number of collision events.

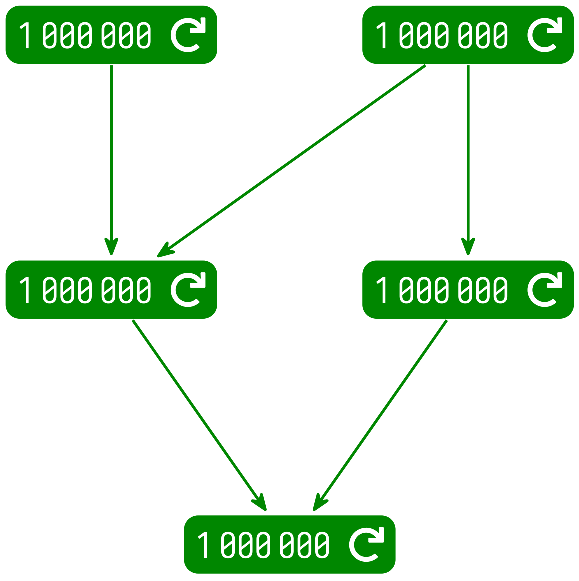
2. Method

To evaluate HPX's suitability for the scheduling needs of ATLAS, we used it to build two toy schedulers, evaluating its performance on a single node and across multiple nodes. All tests were performed using the CPU partition of NERSC's Perlmutter HPC (an additional GPU test was performed using one node of the GPU partition).

Single node performance was evaluated using a toy scheduler that evaluated a graph (Figure 2a) of matrix problems, once per event. The graph was traversed at compile-time and encoded as a series of HPX futures. Since the current (single-node) Athena scheduler is built on Intel's TBB library, performance was compared to an alternative implementation where the same graph was converted to a oneTBB [8] flowgraph. These examples are available on GitHub: `https://github.com/beojan/HPXDemo`.



(a) Single node



(b) Multi-node

Figure 2: The task graphs used for the two scheduling performance tests.

Multi-node performance was evaluated using a toy scheduler that evaluated a graph (Figure 2b) of busy loops, again once per event. Here, the graph was defined by dataflow dependencies at runtime. Each graph node declares its direct dependencies in a map which is then converted to a graph of a series of HPX futures. Performance was compared to an alternative implementation where the same graph was converted to a Ray task graph using the Ray DAG API. These examples are available on GitHub: `https://github.com/esseivaju/HPXDistributed` and `https://github.com/esseivaju/RayDistributed`.

3. Results

Figure 3 shows throughput on a single node when using a variable number of threads. We see that HPX is slower to schedule than TBB at high thread counts for this test using 1000×1000 matrices, though the smaller test with 300×300 matrices in Figure 4 shows the opposite behaviour. We also see that both schedulers show an unexpected reduction in throughput at 128 threads, and we have been unable to narrow down the cause of this reduction.

Additionally we tested using HPX to schedule to GPUs, though these measurements should be interpreted with caution. As detailed in the First Impressions section, we found that HPX's

GPU capabilities were limited so this scheduling was done by invoking CUDA from tasks HPX scheduled onto CPU cores. It was also done with a single CUDA stream and therefore shows constant throughput and can only use a maximum of eight threads. Nevertheless the results are shown for completeness.

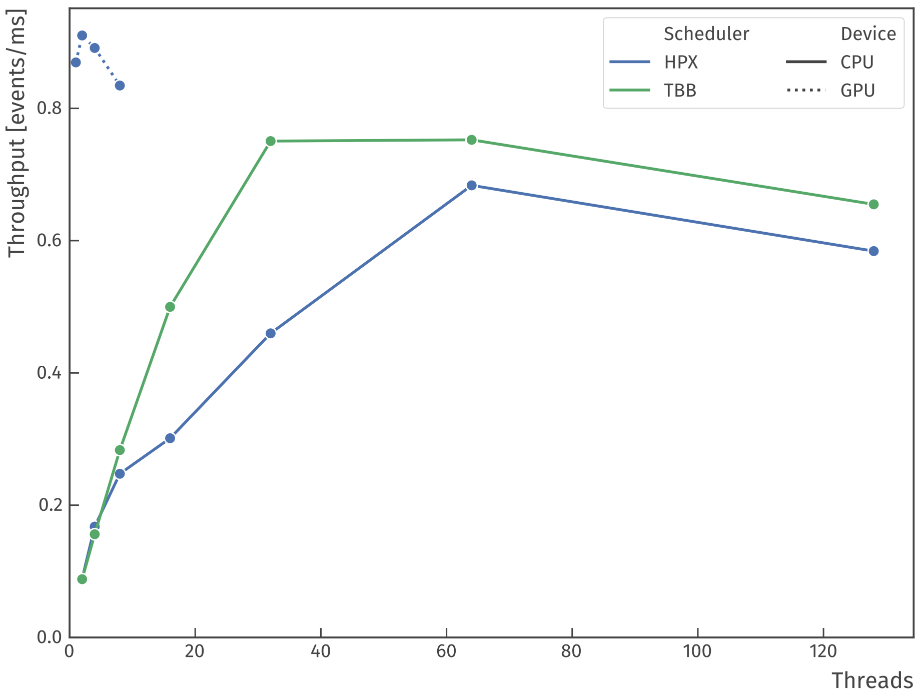


Figure 3: Single node performance comparison using 1000×1000 matrices.

Figure 5 shows throughput in a multi-node job with a varying number of nodes. Here we see that with a single scheduling thread throughput levels off around 23 nodes because the scheduling thread is unable to keep up with the pace at which jobs are completing. However this issue appeared to be resolved if scheduling was done using multiple nodes. On the other hand the Ray implementation exhibited significantly slower performance, struggled above 7 nodes, and parallel scheduling did not appear to improve the situation.

4. Conclusions and First Impressions of HPX

In certain scenarios it appears HPX may exhibit worse single-node performance than TBB. HPX also defaults to a one queue per hardware thread scheduling policy, which can lead to inefficiencies with tasks short enough that queues run empty. It is hoped that this will not be an issue in practice because production tasks should be long enough that scheduling should not be a bottleneck.

The programming model also poses some challenges that we may need to work around. For example, HPX requires the compute tasks to be expressed as functions that take and return futures. This presents a mismatch with the Athena model [9] where tasks are represented as classes with inputs and outputs encoded as members of special `Key` types, representing read and write handles into a central key-value store.

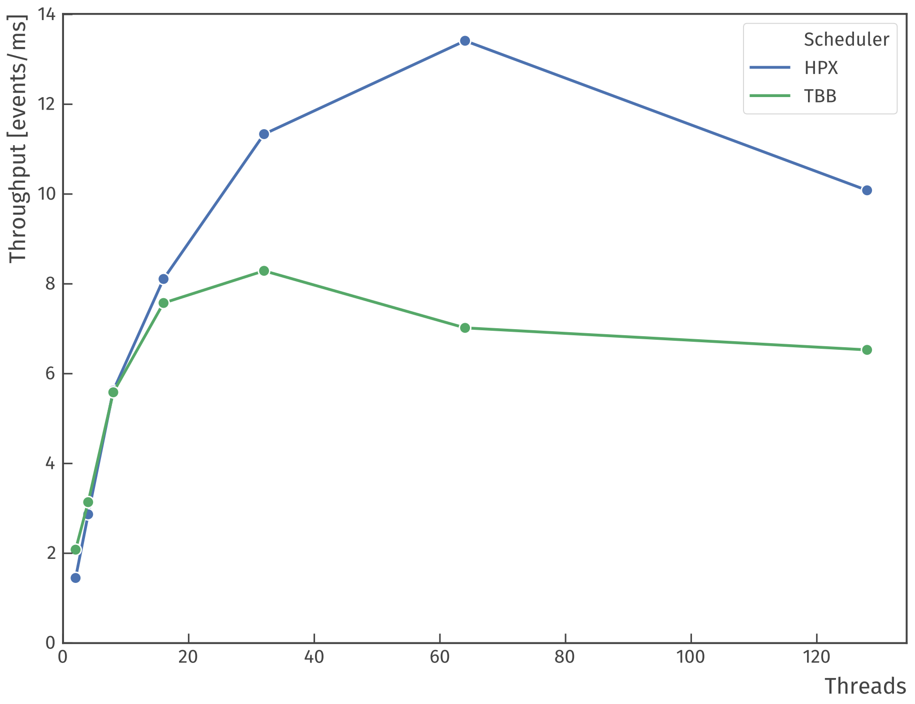


Figure 4: Single node performance comparison using 300×300 matrices.

Certain features such as the ability to control resource use are missing, leading to a requirement to manually throttle submission of tasks into HPX. It also appears the built-in CUDA support is very limited, which led to us implementing the GPU test by using CPU tasks calling CUDA directly.

In principle, however, it appears that HPX would be capable of handling the scheduling needs of ATLAS, and would enable us to use a single scheduler both across cores and across nodes. We are now working on a prototype implementation of a multi-node Gaudi scheduler using HPX. For this prototype we are using only a subset of HPX's features, utilizing void futures to avoid the need to express data dependencies as function parameters and return values. We have also chosen to retain much of the existing custom scheduler implementation in the new scheduler in order to retain the ability to control resource usage. We hope that with continued development, both of our code and on the part of HPX, more of the scheduling work can eventually be moved to HPX.

Acknowledgements
The authors would like to sincerely thank Hartmut Kaiser (Lousiana State University) and the HPX developers for their discussions guiding our development.

This work was supported by the DOE HEP LHC ATLAS Detector and Computing Support program at Lawrence Berkeley National Laboratory under B&R KA2102021. We also thank the ATLAS collaboration for its support and cooperation. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

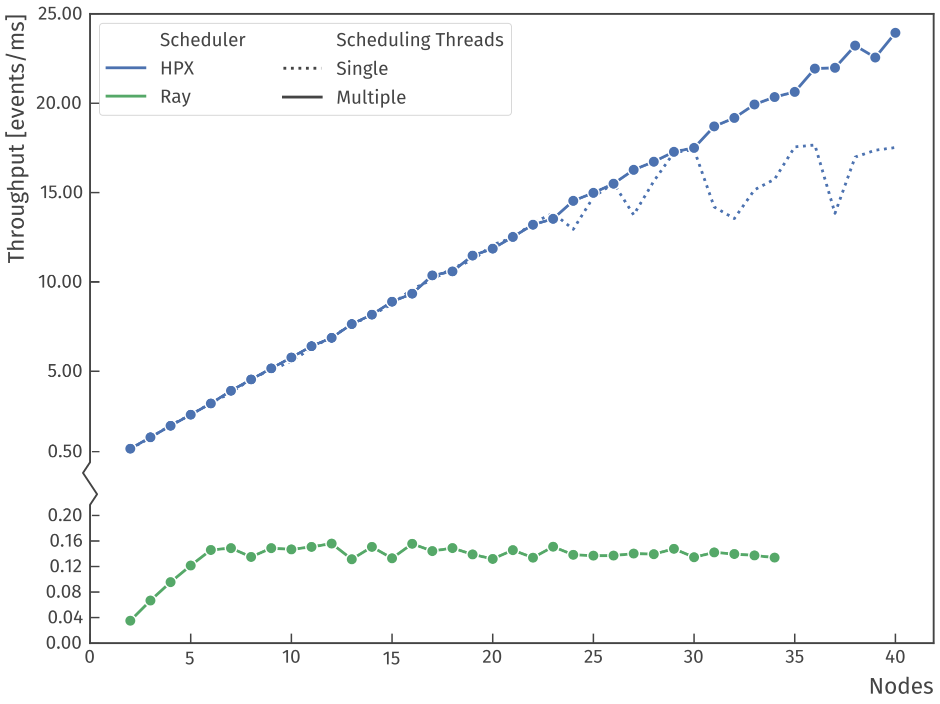


Figure 5: Multi-node performance comparison.

References

[1] ATLAS Collaboration 2021 Athena URL `https://zenodo.org/record/4772550`

[2] LHCb Collaboration and ATLAS Collaboration 2019 Gaudi v33r0 URL `https://zenodo.org/record/3660964`

[3] Nilsson P 2009 The PanDA System in the ATLAS Experiment *Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research — PoS(ACAT08)* (Erice, Italy: Sissa Medialab) p 027 URL `https://pos.sissa.it/070/027`

[4] Blomer J, Bockelman B, Buncic P, Couturier B, Dosaru D F, Dykstra D, Ganis G, Giffels M, Nikola H, Hazekamp N, Heikkila S, Isgandarli S, Meusel R, Mosciatti S, Popescu R, Randall J, Shaffer T, Teuber S, Thain D, Tovar B, Traylen S and Weitzel D 2020 The CernVM File System: v2.7.5 URL `https://zenodo.org/record/4114078`

[5] Muškinja M, Calafiura P, Leggett C, Shapoval I and Tsulaia V 2020 *EPJ Web of Conferences* **245** 05042 ISSN 2100-014X URL `https://www.epj-conferences.org/10.1051/epjconf/202024505042`

[6] Moritz P, Nishihara R, Wang S, Tumanov A, Liaw R, Liang E, Elibol M, Yang Z, Paul W, Jordan M I and Stoica I 2018 Ray: a distributed framework for emerging AI applications *Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation* OSDI'18 (USA: USENIX Association) pp 561–577 ISBN 978-1-931971-47-8

[7] Kaiser H, Diehl P, Lemoine A, Lelbach B, Amini P, Berge A, Biddiscombe J, Brandt S, Gupta N, Heller T, Huck K, Khatami Z, Kheirkhahan A, Reverdell A, Shirzad S, Simberg M, Wagle B, Wei W and Zhang T 2020 *Journal of Open Source Software* **5** 2352 ISSN 2475-9066 URL `https://joss.theoj.org/papers/10.21105/joss.02352`

[8] Intel® oneAPI Threading Building Blocks URL `https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html`

[9] Kama S, Leggett C, Snyder S and Tsulaia V 2019 *Proceedings, 23rd International Conference on Computing in High Energy and Nuclear Physics (CHEP 2018): Sofia, Bulgaria, July 9-13, 2018* **214** 05018 URL `https://doi.org/10.1051/epjconf/201921405018`