

# Simulating Diverse HEP Workflows on Heterogeneous Architectures

Charles Leggett<sup>1</sup>, Illya Shapoval<sup>1</sup>, Marco Clemencic<sup>2</sup> and Christopher Jones<sup>3</sup>

<sup>1</sup> Lawrence Berkeley National Laboratory, Berkeley, 1 Cyclotron Rd, CA 94720, US

<sup>2</sup> European Organization for Nuclear Research, Esplanade des Particules 1, 1217 Meyrin, CH

<sup>3</sup> Fermi National Accelerator Laboratory, P.O. Box 500 Batavia, IL, US

E-mail: cleggett@lbl.gov, ishapoval@lbl.gov, marco.clemencic@cern.ch, cdj@fnal.gov

## Abstract.

We evaluate key patterns and estimate throughput bounds of simulated transformation of conventional high energy physics (HEP) data processing workflows to heterogeneous equivalents. The simulation parameter space includes the number of offloaded tasks, CPU/accelerator ratios of intra-task computations, offload latencies, and run time efficiency of offloaded computations. The simulation is performed for a diverse set of state-of-the-art event reconstruction scenarios from ATLAS, LHCb and CMS - the frontier HEP experiments of the Large Hadron Collider project.

## 1. Introduction

The next generation of HPC and HTC facilities, such as Summit at Oak Ridge and Perlmutter at NERSC, show an increasing use of GPUs and other accelerators in order to achieve their high FLOP counts. This trend will only grow with exascale facilities such as A21. In general, high energy physics (HEP) computing workflows have made little use of GPUs due to the relatively small fraction of kernels that run efficiently on GPUs, and the expense of rewriting code for rapidly evolving GPU hardware. However, the computing requirements for high-luminosity Large Hadron Collider (LHC) are enormous, and it will become essential to be able to make use of supercomputing facilities that rely heavily on GPUs and other accelerator technologies.

ATLAS [4] has already developed an extension to its multithreaded event processing framework, that enables the non-intrusive offloading of computations to external accelerator resources, and has begun investigating strategies to schedule the offloading efficiently. The same applies to LHCb [5], which, while sharing with ATLAS GAUDI [1, 2] as the underlying framework, has considerably different workflow. CMS [6]'s framework, CMSSW [7], also has the ability to efficiently offload tasks to external accelerators. But before investing heavily in writing many kernels for specific offloading architectures, we need to better understand the performance metrics and throughput bounds of the workflows with various accelerator configurations. This can be done by simulating a diverse set of workflows, using real metrics for task interdependencies and timing, as we vary fractions of offloaded tasks, latencies, data conversion speeds, memory bandwidths, and accelerator offloading parameters such as CPU/GPU ratios and speeds.

We present the results of these studies performed on multiple workflows from ATLAS, LHCb and CMS, which will be instrumental in directing effort to make HEP framework, kernels and workflows run efficiently on exascale facilities.

## 2. The GAUDI framework

GAUDI is a cross-experiment software framework providing a common infrastructure for building applications for HEP event data processing. The framework was designed around the principles of composability and reusability allowing flexible plug-and-play assemblies of general-purpose and specialized components. GAUDI-based applications span a broad range of HEP computing tasks from event and detector simulation, event triggering, reconstruction and analysis to detector alignment and calibration.

The GAUDI framework is used in several frontier HEP experiments among which are ATLAS and LHCb - the two of four major experiments of the LHC, as well as FGST, DayaBay, MINER $\nu$ A and LZ. This makes GAUDI a high-impact testbed for studying various scenarios of converting the state-of-the-art workflows of HEP data processing to workflows that can leverage the heterogeneous nature of the next-generation supercomputer facilities.

## 3. Methodology

The dominant factors affecting throughput of a data processing workflow are the task timing, the task precedence rules (i.e., control flow and data flow dependencies), as well as the policy and efficiency of task scheduling applied to the workflow. In this study, we assumed that sufficiently accurate evaluation of throughput bounds can be narrowed down to a simulation of execution flow that captures the above mentioned factors. This methodology facilitates a cross-experiment study that is abstract from experiment specific applications with all other framework-level aspects being equal. The GAUDI framework – our testbed of choice – provided most of the infrastructure for such a study.

First, we relied on the PYTHON-based GAUDI scenario assembler [3] developed as part of the GAUDI test infrastructure and used in an earlier similar study done for LHCb [8]. We constructed configurable synthetic scenarios capturing the pertinent properties of the ATLAS, LHCb and CMS state-of-the-art workflows (see Section 4). In the scenarios, we substituted experiment-specific tasks with synthetic CPU-cruncher tasks that merely occupy the CPU with intensive mathematical operations for a certain period of time. The run time with which each CPU-cruncher task was allocated was determined by running the real workflows. CPU-cruncher tasks were set to obey the precedence rules which accurately replicated those of the real tasks they replaced, which included the control and data flow dependencies.

Second, we utilized the capacity of the GAUDI Avalanche Scheduler [8, 9] to schedule a synthetic scenario by resolving prescribed task timing and precedence rules thus replicating a regular data processing job. We then profiled the jobs in throughput scaling tests (see Section 5).

### 3.1. Simulating computational offloading

In the GAUDI framework, offloading all or part of computations of a task to an external accelerator blocks the software thread executing the task until the accelerator has finished and returned control to the task. The thread is idle while blocked, so we can simulate the offloading behaviour by making the thread `sleep` for a certain period of time. This permits the Linux kernel to suspend the idle software thread freeing the hardware resource for other tasks.

To explore the offloading phase space, we adjust the original task’s run-time  $t_{orig}$  with three parameters: the fraction  $f$  of offloaded computations, the efficiency  $eff$  of running on the accelerator (*ie* does it run slower or faster), and extra time  $t_{extra}$  to account for data movement overheads. Thus, the CPU and offload time of adjusted tasks are  $t_{cpu} = t_{orig}(1 - f)$  and  $t_{offload} = t_{orig}f(1 - eff) + t_{extra}$ , respectively.

### 3.2. Critical path analysis

The intra-event concurrency of the workflows is limited by the task precedence rules. As the Avalanche Scheduler resolves the rules, it can trace the task-to-task execution flow and determine the *critical path* (CP), which is a function of both the task precedence rules and individual task run times. We have extracted the CP for all the scenarios and used that information to configure the scenarios with selective on/off CP task offloading policies.

## 4. Scenarios

The ATLAS scenario (Figure 1, *left*) is based on a standard data processing workflow that transforms the raw data from the ATLAS detector through a series of steps into objects useful for physics analyses. This involves reconstructing the tracks of the particles that travelled through the sensitive elements of the detector, determining their energies, and applying discriminators to attempt to identify the types of particles. The source data that was used to extract the timings of the tasks was taken from a real data during a standard run at nominal energy and luminosity during the 2017 data taking period. A second data set was also used with the same chain of tasks which was taken during a special run with a much higher than normal number of interactions ( $\mu \approx 90$ ) that begins to approximate running conditions during the HL-LHC period.

The CMS scenario (Figure 1, *center*) is based on the workflow used to transform raw data from the CMS detector into quantities meaningful for physics studies. The transformations include applying calibrations to the raw information and then applying many pattern recognition tasks to the calibrated data in order to infer the presence of elementary particles, e.g. electrons or photons, within the detector as well as physics quantities of those particles. This workflow is referred to as *reconstruction*. This workflow is central to the computational work done by CMS and requires the greatest amount of computational resources of all workflows. The task timings used in this study were taken from an actual data processing job which was processing a fairly representative data sample taken during 2018.

Similar to the cases of ATLAS and CMS, the LHCb scenario (Figure 1, *right*) is based on the reconstruction workflow, where raw detector data is decoded and transformed into quantities suitable for physics analysis. The sequence of tasks and their timings were extracted from one of LHCb benchmarking jobs, referring to a typical sample of the 2017 data taking period.

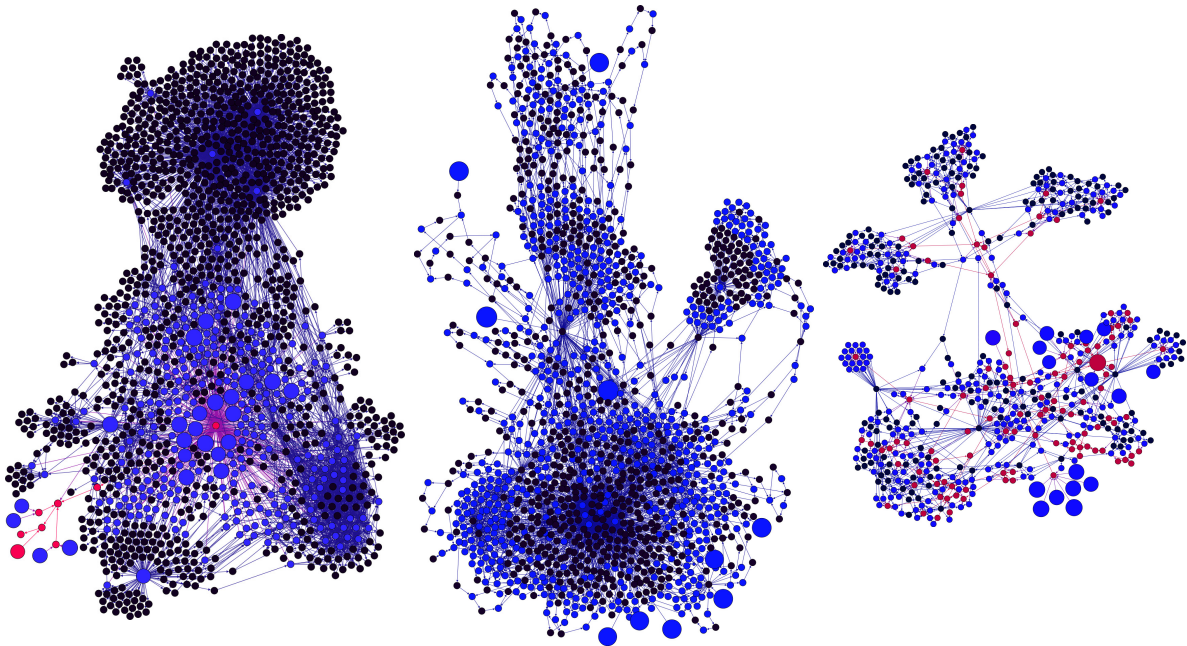
## 5. Results

### 5.1. CP analysis

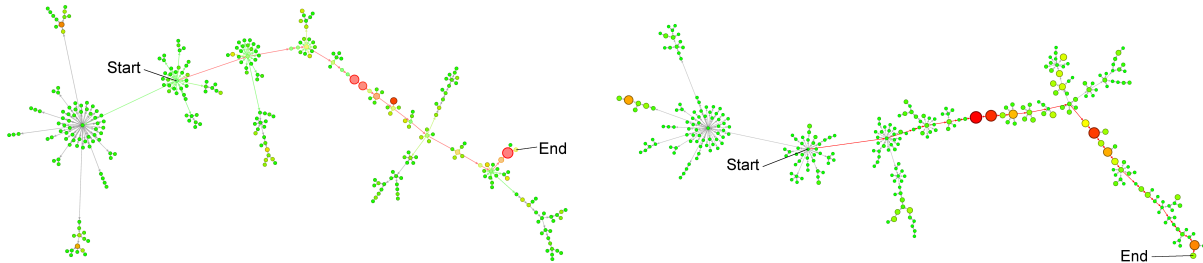
The total serial event processing times of the 4 different scenarios are shown in Table 1, as well as the number of tasks in each workflow. ATLAS reconstruction spends approximately 57% of its time on the CP (Figure 2) under normal Run 2 conditions. This number drops to 40% if we disregard the tasks that perform I/O, which cannot be offloaded to an accelerator. The CP comprises of only 10% of tasks. In the high- $\mu$  scenario, the event processing time increases by almost a factor of 10 due to the much higher multiplicity, and the CP lengthens to 75%. LHCb has much faster event processing times, due to the fact that the detector and associated event size is much smaller, but has a comparable number of tasks and CP (3) percentage to ATLAS. CMS reconstruction has almost twice as many tasks, but they tend to be smaller and due to the more serial nature of the execution flow, has a comparably longer CP than ATLAS or LHCb.

Workflow	All tasks	CP tasks	CP w/o I/O
ATLAS	309: 14.6s	20: 8.33s	17: 5.78s
ATLAS high $\mu$	309: 127s	32: 95.7s	29: 85.7s
CMS	707: 13.4s	147: 8.38s	145: 7.32s
LHCb	282: 491ms	13: 259ms	11: 234ms

**Table 1.** Task counts and timings for the different workflows.



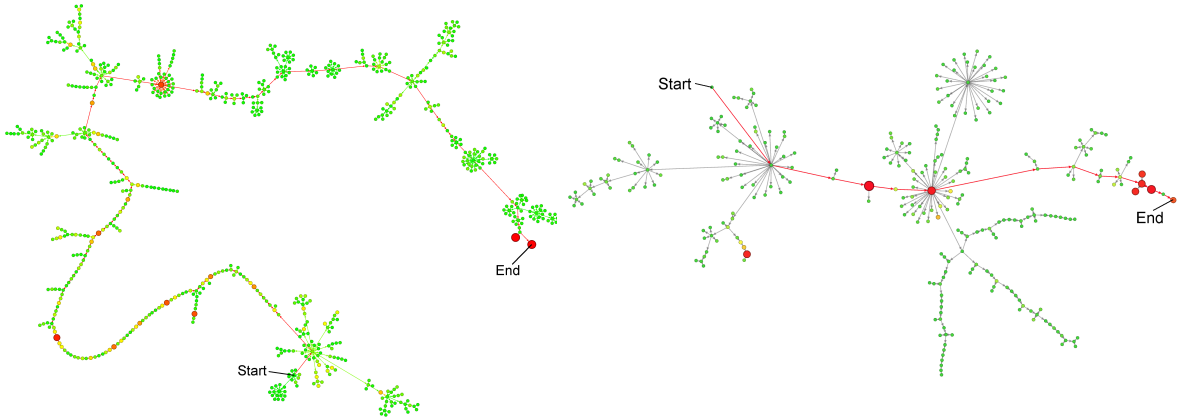
**Figure 1.** Unified graph of control flow (CF) and data flow rules used in the event reconstruction workflow of (left) ATLAS, (center) CMS and (right) LHCb experiments. Blue nodes represent tasks, black ones denote data entities, while red nodes - CF decision nodes serving for CF decisions aggregation. Larger blue nodes represent tasks that either don't have any data inputs, or that load initial data from disk. These nodes are the entry points for scheduling the graph. The larger red node is the root CF decision hub, which is resolved the last in the process of scheduling the graph at which point an event is considered processed.



**Figure 2.** CP of the ATLAS scenario for regular (left) and high  $\mu$  (right) event data. Each node represents a task, and is sized and colored proportionally to its run time (small and green are fast tasks, while large and red are slow). The start- and end-tasks of each scenario are marked. Despite identical task precedence rules, the CPs are different in each case due to the different task timings. Visualizing the CP aids both in understanding the available concurrency in a workflow, and in helping to identify where optimization or offloading could benefit.

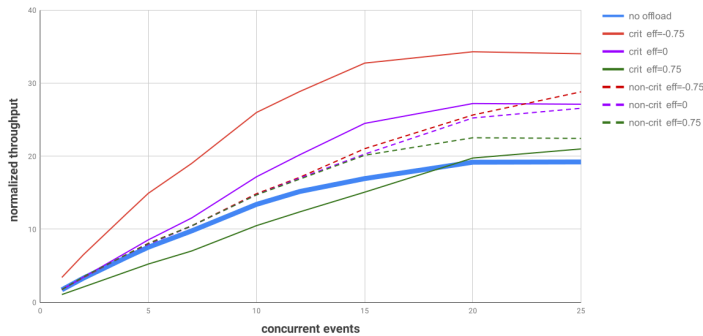
### 5.2. Throughput analysis

In order to study the effects of offloading different tasks, we ran the simulation varying the offloading efficiency and offloading fraction with different selections of offloaded tasks. We first chose to investigate the difference between offloading tasks on the CP vs. those not on it. In order to maximize the difference, we fixed the offloading fraction at 0.9, varied the offloading efficiency between -0.75 and 0.75, and offloaded either all the tasks on the CP, or all the tasks not on the CP. The results of this study, which was performed with 35 threads devoted to task scheduling on a machine with 40 hyperthreaded cores, can be seen in Figure 4, where we show the event processing throughput, normalized to the serial event processing time, as a function of the number of concurrent events. The dotted lines show the throughput when all CP tasks are



**Figure 3.** CP of the CMS (left) and LHCb (right) scenarios. Notations repeat those of Figure 2.

offloaded, and the solid lines are for offloading the non-CP tasks. The thick line is a reference with no task offloading. This plot shows that it will be much more beneficial to offload tasks that are on the CP as opposed to those not on it. Furthermore, improving the accelerator performance *cf* the CPU makes a larger difference with CP tasks. This is because if the accelerator takes much longer to execute the task than the CPU, it has the effect of lengthening the CP. This can be overcome by increasing the number of concurrent event, though this may be limited by other system resource constraints.

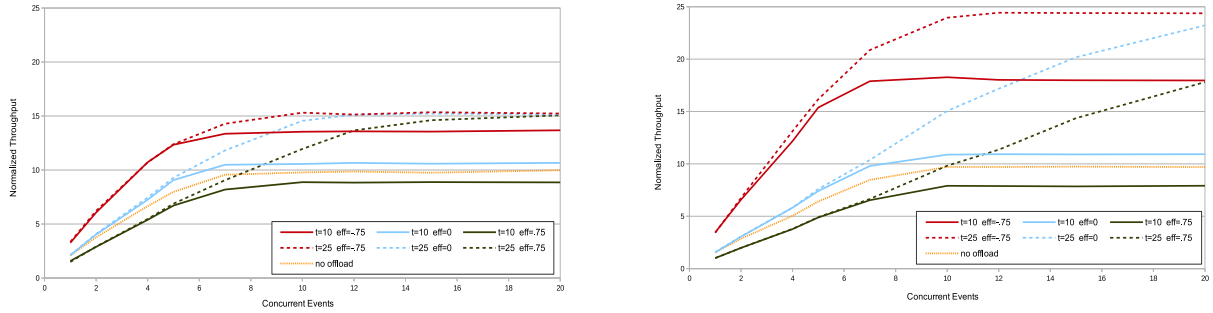


**Figure 4.** Throughput scaling for ATLAS scenario, varying offloading efficiency.

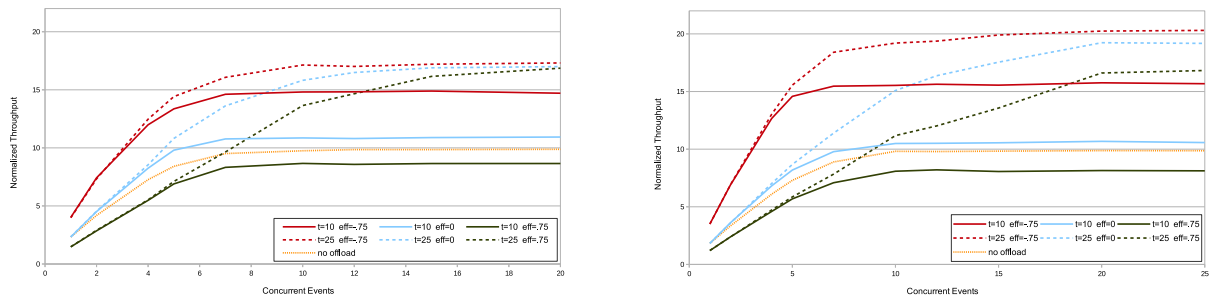
When the simulation is run with as many software threads as hardware threads, we see less than full occupancy of the CPU, as the hardware threads associated with the offloaded tasks are often idle, waiting for the accelerator to return (*ie sleep* to finish). In order to maximize CPU occupancy, we need to oversubscribe the CPU with more software threads than hardware threads. We simulated this by using `taskset` to limit the number of hardware threads to 10, and increasing the number of software threads between 10 and 25. In Figures 5 and 6 we see the dramatic benefit of oversubscribing the CPU for ranges of offloading efficiencies. We can see that the combination of oversubscribing the CPU and increasing the number of concurrent events allows us to overcome any accelerator inefficiencies, as the latency of the offloaded tasks is hidden when the CPU is kept fully occupied with other work.

## 6. Conclusions

While the computational nature of any task will ultimately determine whether it can be offloaded, knowing that offloading tasks on the CP has a greater effect on overall performance will help choose which tasks to scrutinize. This filtering will be very useful as there are a large number of tasks in each workflow. Furthermore, these studies have shown that even if the task runs more slowly on the accelerator than the CPU, it is still beneficial to offload it. If tasks that are not on the CP are chosen to be offloaded, our studies have shown that it may be necessary



**Figure 5.** Throughput scaling for ATLAS scenario for regular (left) and high  $\mu$  (right) event data, varying offloading efficiency from  $-0.75$  to  $0.75$ , with 10 and 25 software threads.



**Figure 6.** Throughput scaling for LHCb (left) and CMS (right), varying offloading efficiency from  $-0.75$  to  $0.75$ , with 10 and 25 software threads.

to increase the number of concurrently scheduled events to maximize throughput. We have also shown that when using the GAUDI framework, it is essential to oversubscribe the CPU with more software threads than hardware threads in order to keep the CPU fully occupied with useful work while the threads that manage the offloaded tasks wait. Ultimately, offloading any task will improve performance, though careful tuning of the number of concurrent events and threads is required to maximize performance without negatively impacting other system resources.

## Acknowledgments

Support for CMS is given by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

Lawrence Berkeley National Lab work was supported by the Office of Science, Office of High Energy Physics, of the U.S. Department of Energy under contract DE-AC02-05CH11231.

## References

- [1] Barrand G et al 2001 GAUDI - a software architecture and framework for building HEP data processing applications *Comput. Phys. Commun.* **140** 45–55
- [2] The GAUDI framework [software] Available from <http://cern.ch/gaudi/> [accessed on 2019-05-05]
- [3] Shapoval I GAUDI synthetic scenario assembler [software] Available from <https://gitlab.cern.ch/gaudi/Gaudi/blob/master/GaudiHive/python/GaudiHive/precedence.py> [accessed on 2019-05-05]
- [4] Aad G et al 2008 The ATLAS experiment at the CERN Large Hadron Collider *J. of Instrumentation* **3** S08003
- [5] Augusto Alves A Jr et al 2008 The LHCb detector at the LHC *J. of Instrumentation* **3** S08005
- [6] Chatrchyan S et al 2008 The CMS Experiment at the CERN LHC *J. of Instrumentation* **3** S08004
- [7] Jones C D et al 2006 *Proc. CHEP 2006* **1** (India: Macmillan) 248–251
- [8] Shapoval I 2016 Adaptive scheduling applied to non-deterministic networks of heterogeneous tasks for peak throughput in concurrent Gaudi CERN-THESIS-2016-028 Available from <http://cds.cern.ch/record/2149420> (Geneva: CERN)
- [9] Shapoval I et al 2015 Graph-based decision making for task scheduling in concurrent Gaudi *Proc. of IEEE Nuclear Science Symp. and Medical Imaging Conf.* DOI:10.1109/NSSMIC.2015.7581843 INSPEC:16356796 (San Diego: IEEE)