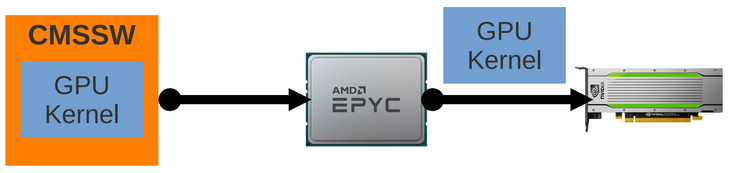
Optimising the Configuration of the CMS GPU Reconstruction

Abdulla Ebrahim[1] Andrea Bocci[2] Wael Elmedany[1] Hesham Al-Ammal[1]

[1] University of Bahrain [2] CERN

Objectives




```
myGpuKernel<<<numberOfBlocks, numberOfThreads>>>();
```

Figure 1. CMSSW GPU offloading.

- CMSSW offloads **40%** of the HLT processing to GPUs
- GPU Kernels require at least **2 parameters** to run on GPUs.
- The parameters have to be **tuned** to get the best performance.
- These parameters **affect the performance** depending on:
  - Properties of the algorithm
  - Size of the data
  - Type of GPU
- The **objectives** of this study are to:
  - Implement an autotuning framework
  - Find feasible strategies for autotuning
  - Autotune CMSSW on multiple GPUs

Autotuning Framework

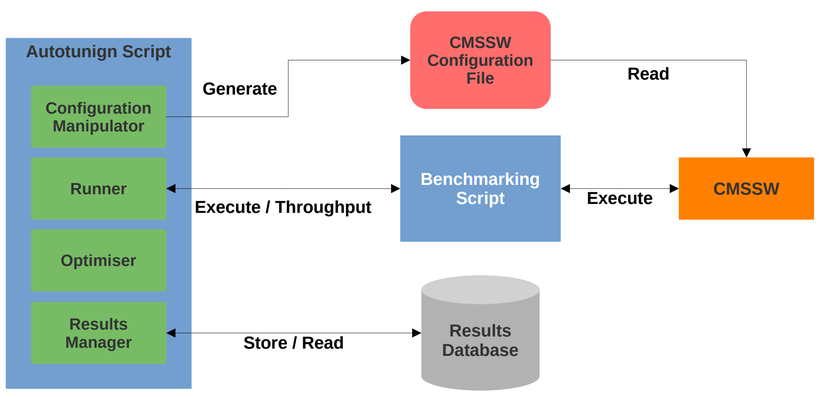


Figure 2. An illustration of the autotuning framework used in this work. It is based on the OpenTuner[1] framework.

- **Configuration Manipulator:** Generates CMSSW configuration files populated with different parameters from the search space.
- **Runner:** Executes CMSSW, and measures its performance using a benchmark script [2].
- **Optimiser:** Generates parameters from the search space to optimise the throughput.
- **Results Manager:** Stores and retrieve the results' data from the database.

Autotuning Environment

	Machine 1	Machine 2	Machine 3
CPUs	Intel Xeon Silver 4114U 2x 10 Core	AMD EPYC 7763 2x 64 Core	AMD EPYC 7763 2x 64 Core
RAM	96G	256G	256G
GPUs	NVIDIA A10 NVIDIA A30X	2x Tesla T4	2x NVIDIA L4
CUDA Version	12.1	12.0	12.1

Table 1. The specifications of the machines that were used to autotune and benchmark the CMSSW software.

	Machine 1	Machine 2	Machine 3
CMSSW Release	13_0_0	13_0_0	13_0_0
Number of Jobs	1	4	4
Number of Threads	12	32	32
Number of Streams	12	24	24
Number of Events	10000	10000	10000

Table 2. The configurations used during autotuning and benchmarking the CMSSW. The same configurations are used for the pixeltrack standalone program. The only difference is that the pixeltrack standalone program is always benchmarked with one job only.

Autotuning Methodology

Type of Parameters	Count
Blocks	6
Threads	23
Strides	3
Total	32

Table 3. The types of the tuned parameters and their count.

- **Blocks:** Number of blocks in the grid. It can be dependent on the size of the data and the number of threads. The selected six parameters are independent and can be changed freely.
- **Threads:** Number of threads in each block. Most common parameter. It is not dependent on any other parameter.
- **Strides:** Controls the number of data points that will be processed by each thread. Some of the algorithms allow this parameter to be changed freely.

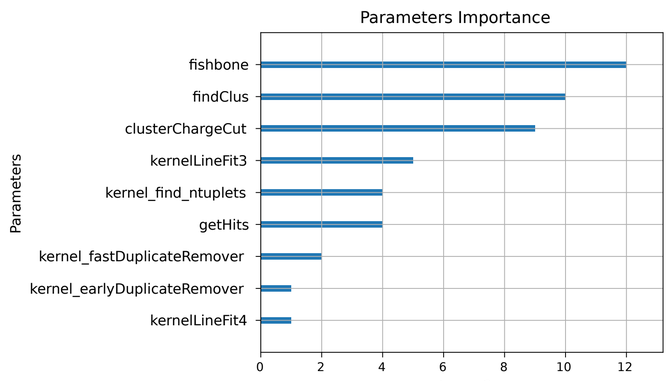


Figure 3. A dataset of ~250000 entries was created by tuning the 32 parameters randomly using the low fidelity benchmark. That dataset was used to train a boosted decision trees model by XGBoost. A plot of the **importance score** was generated from the model, which shows the number of times a parameter appears in the trained ensemble of decision trees. From this analysis, the size of the search space was limited to **7 parameters only instead of 32.**

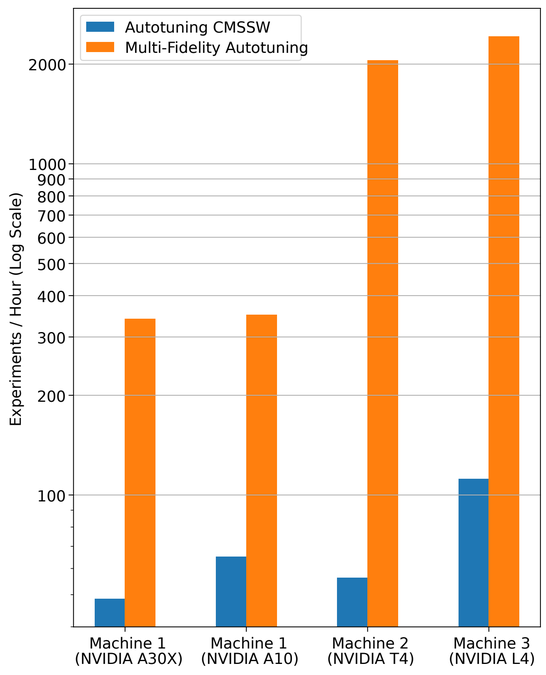


Figure 4. The plot shows the difference in time between directly autotuning CMSSW and Multi-Fidelity Autotuning [4]. Instead of autotuning CMSSW directly (**High fidelity** benchmark data), a minimal implementation [3] (**Low fidelity** benchmark data) known as the **pixeltrack standalone** is used. The top performing configurations are plugged into CMSSW and benchmarked.

Autotuning Runs	8 runs, 2 for each GPU, one for direct autotuning, the other for multi-fidelity.
Autotuning Run Duration	6 Hours / Run, The tuning time is fixed before the run starts.
Optimisation Algorithm	AUC Bandit Sliding window meta technique [1] Uses multiple search techniques Rank techniques based on their performance
Final Benchmarks	Top configurations are benchmarked 10 times First three results are thrown away The seven left are averaged

Table 4. Details of the autotuning and benchmarking methodology.

Results

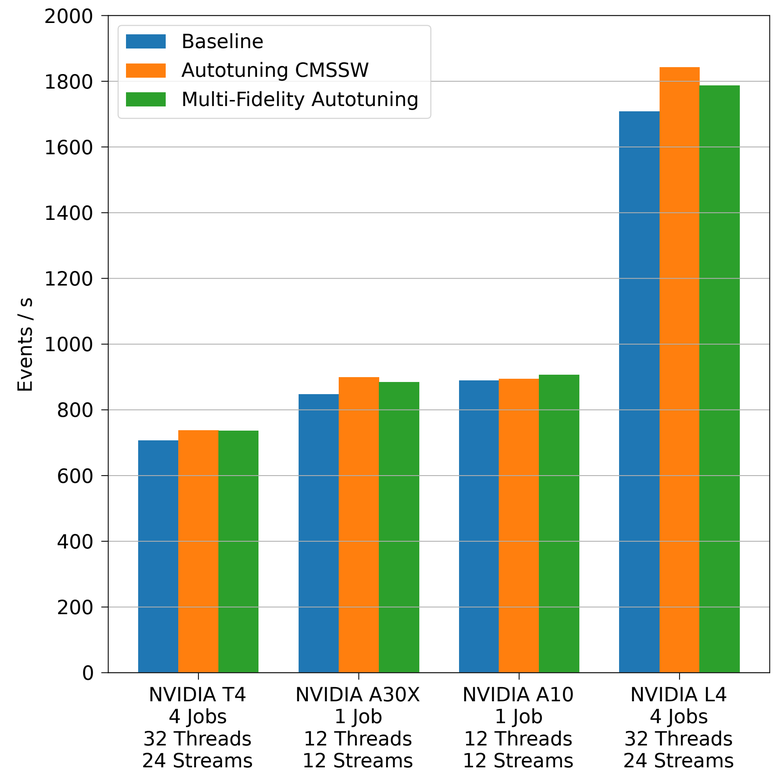


Figure 5. A comparison between the throughput of the baseline parameters, the parameters found after autotuning CMSSW directly, and the parameters found from autotuning the pixeltrack standalone program. The results show that the autotuning successfully increased the throughput on all the GPUs used in the study. The Multi-Fidelity approach was able to produce the best parameters on the A10 GPU and the T4 GPU. .

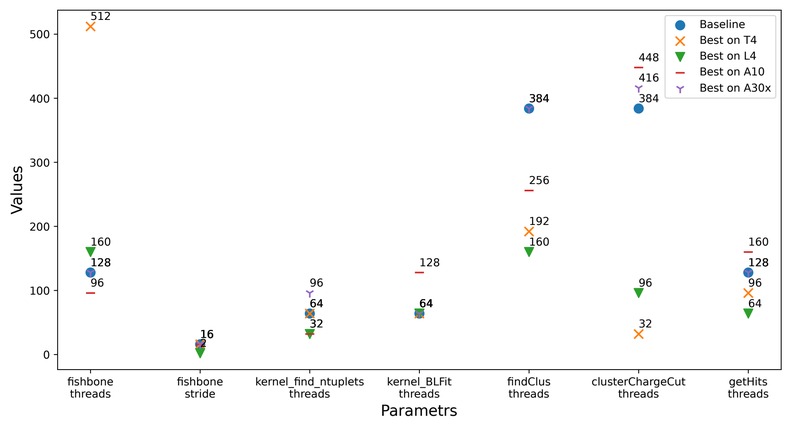


Figure 6. Shows the best parameters found for each GPU. The best parameters found for each GPU varies which shows the importance of the autotuning exercise. This study used GPUs from the same vendor, It is expected that the parameters change even more on GPUs from multiple vendors.

Conclusion & Future Work

In this work, we showed the importance of autotuning in achieving the best performance for CMSSW. We created an autotuning framework and presented its effectiveness in finding better configurations than the current hand tuned parameters. In addition, we presented autotuning strategies to make the process as feasible as possible. In the next step, we will work on using **advanced parameters search techniques** to reduce the time of finding the best parameters.

References

[1] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning." Proceedings of the 23rd international conference on Parallel architectures and compilation. 2014.

[2] https://github.com/cms-patatrack/patatrack-scripts

[3] https://github.com/asubah/pixeltrack-standalone/tree/autotuning

[4] Lindauer, Marius, et al. "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization." J. Mach. Learn. Res. 23.54 (2022): 1-9.

Acknowledgments

The experiments presented in this paper were carried out using the facilities of the Benefit Advanced AI and Computing Lab at the University of Bahrain (see https://ailab.uob.edu.bh) with support from Benefit Bahrain Company (see https://benefit.bh)