Raw to Pixel tracks on GPUs without intermediate copies

The aim of the project is to run a chain of GPU-accelerated modules, without copying back to the host the intermediate results.

CMSSW interface to heterogeneous algorithms

The aim of the project is to think and prototype the CMSSW interface towards heterogeneous algorithms further into the future. The “system” (which can be an edm::Service but not necessarily) makes the decision on which device (CPU, GPU, FPGA, you name it) the algorithm is run based on the available resources and the location of the input data (if any). It also tracks the location of the output data, and provides a mechanism to automatically transfer the data back to CPU when needed.

ECAL Multifit on GPU

The project consists in porting the ECAL Multifit algorithm to CUDA.

HCAL MAHI on GPU

The project consists in porting the HCAL MAHI algorithm to CUDA.

FPGA algorithms integration in CMSSW

This project consists in the study and integration for the first time of OpenCL code for FPGAs in CMSSW framework. Ideally, this will lead to the identification of the number/type of FPGA present of the system, along with the launch of a small OpenCL kernel.

Remote offload

This project consists in the exploration of techniques and technologies for the remote offload of specific functions to an accelerator located in another node.

Machine learning for HGCAL

Parallel Kalman Filter

Limits on power dissipation have pushed CPUs to grow in parallel processing capabilities rather than clock rate, leading to the rise of "manycore" or GPU-like processors. In order to achieve the best performance, applications must be able to take full advantage of vector units across multiple cores, or some analogous arrangement on an accelerator card. Such parallel performance is becoming a critical requirement for methods to reconstruct the tracks of charged particles at the Large Hadron Collider and, in the future, at the High Luminosity LHC. This is because the steady increase in luminosity is causing an exponential growth in the overall event reconstruction time, and tracking is by far the most demanding task for both online and offline processing. Many past and present collider experiments adopted Kalman filter-based algorithms for tracking because of their robustness and their excellent physics performance, especially for solid state detectors where material interactions play a significant role.

The aim of the hackathon is to improve the Kalman filter track reconstruction algorithm with optimal performance on manycore architectures. The combinatorial structure of the traditional versions of these algorithms is not immediately compatible with an efficient SIMD (or SIMT) implementation; the challenge for us is to recast the existing software so it can readily generate hundreds of shared-memory threads that exploit the underlying instruction set of modern processors.

Large scale training

In the recent years, several studies have demonstrated the benefit of using deep learning to solve typical tasks related to high energy physics data taking and analysis. Building on these proofs of principle, many HEP experiments are now working on integrating Deep Learning into their workflows. The computation need for inference of a model once trained is rather modest and does not usually need specific treatment. On the other hand, the training of neural net models requires a lot of data, especially for deep models with numerous parameters. The amount of data scales with the many parameters of the models which can be in billions or more. The more categories present in classification or the more wide a range of regression is performed the more data is required. Training of such models has been made tractable with the improvement of optimization methods and the advent of GP-GPU well adapted to tackle the highly-parallelizable task of training neural nets. Despite these advancement, training of large models over large dataset can take days to weeks. To take the best out of this new technology, it would be important to scale up the available network-training resources and, consequently, to provide tools for optimal large-scale trainings. Neural nets are typically trained using various stochastic methods based on gradient descent. One of the avenue to further accelerate the training is via data parallelism, where the computation of the gradients is computed on multiple subset of the data in parallel and used collectively to update the model toward the optimum parameters. Several frameworks exists for performing such distributed training, including framework already developed by the authors, all with their strengths and limitations. In this context, the further development of a new training workflow, which scales on multi-node/multi-GPU architectures with an eye to deployment on high performance computing machines will be the target of the hackathon.

Filtering doublets with DNN on ellipses

There is a sentiment in the community (see Andy's question on 9/4) that the only useful information the cluster shape can give is a prediction on the trajectory angles.
There are various way to extract an estimate on the trajectory angles from the cluster shape: for instance "fitting" an ellipse assuming uniform "illumination". See
https://github.com/cms-sw/cmssw/compare/master...VinInn:ClusEllipseTest#diff-4730b4cd3b8b01e713423bded243913c For a possible implementation based on
https://github.com/idl-coyote/coyote/blob/master/fit_ellipse.pro
http://www.idlcoyote.com/ip_tips/fit_ellipse.html
http://www.math.harvard.edu/archive/21b_fall_04/exhibits/2dmatrices/index.html
https://www.soest.hawaii.edu/martel/Courses/GG303/Eigenvectors.pdf
Or your preferred course in Geometry and rational-mechanics

The idea is to compare the performance of the current CNN with a simpler DNN that uses just the cluster shape parameters.
For the performance of a DNN to predict the trajectory angles (regression) using these parameters see for instance
https://github.com/VinInn/pyTools/blob/master/clusEllipseKeras.ipynb

Integrating fast inference in CMSSW

Inference on trained neural network model today is done through slow interfaces.
This can be accelerated by compiling the model for the accelerator architecture and pruning low-weight nodes. Furthermore, if data are already present on a GPU, this accelerator can be exploited to furtherly improve the performance of the inference engine, by parallelizing multiple queries at the same time.

PCA-based DNN filters/selection for em clusters in HGCAL

The aim of the project is to develop a DNN to filter/select em clusters in HGCAL using PCA-based (i.e. calo-only based) variables.

HGCal: Exploit shower shapes for better energy resolution

In addition to energy measurement, high granularity calorimeters also provide transversal and longitudinal shower-shape information. The aim of the project is to investigate whether the latter can be used to increase the energy resolution beyond the sum of the collected energies in all sensors belonging to that particular shower.
For this purpose, individual electrons are generated following a flat energy distribution and their showering is simulated in a simplified high granularity calorimeter.
The sensors of the calorimeter are calibrated such, that the sum of the sensor energies for each event corresponds already well to the electron energy.
The shower-shape information is provided as 3D images which can be used to train deep neural networks with the aim to increase the energy resolution.

HGCal parallel clustering

Development of the HGCal layer clustering using CUDA on GPUs.

Optimize Cuda, SIMD, OpenMP backends of SixTrackLib

SixTrackLib implements routines for tracking single particles in accelerator structures. It is written in C, with C/C++/Python API. It is optimized for speed and aims at achieving peak performance on CPU (SIMD, OpenMP) and GPU (OpenCL, CUDA). Present focus was on the OpenCL backend. The exercise aims at improving the other backends by measuring performance and exploring options. Present master is on https://github.com/SixTrack/sixtracklib.