Computational steering of GEM based detector simulations

Ali Sheharyar and Othmane Bouhali
Texas A&M University at Qatar, P. O. Box 23874, Doha, Qatar
E-mail: ali.sheharyar@qatar.tamu.edu

Abstract. Gas based detector R&D relies heavily on full simulation of detectors and their optimization before final prototypes can be built and tested. These simulations in particular those with complex scenarios such as those involving high detector voltages or gas with larger gains are computationally intensive may take several days or weeks to complete. These long-running simulations usually run on the high-performance computers in batch mode. If the results lead to unexpected behavior, then the simulation might be rerun with different parameters. However, the simulations (or jobs) may have to wait in a queue until they get a chance to run again because the supercomputer is a shared resource that maintains a queue of other user programs as well and executes them as time and priorities permit. It may result in inefficient resource utilization and increase in the turnaround time for the scientific experiment. To overcome this issue, the monitoring of the behavior of a simulation, while it is running (or live), is essential. In this work, we employ the computational steering technique by coupling the detector simulations with a visualization package named VisIt to enable the exploration of the live data as it is produced by the simulation.

1. Introduction
The field of Micro Pattern Gas Detectors (MPGD) has witnessed important development in the past three decades. New structures and complex geometries have been introduced to further enhance the performances such as the efficiency, the gain and the counting rate. The R&D of these detectors relies on a full simulation aiming at an optimization of the detectors parameters before a real prototype can be built and tested. The simulation can become cumbersome in particular in scenarios where complex structure and high detector voltages or gas with larger gains are simulated, leading to a wall time ranging from few days to few weeks.

These long-running simulations usually run on high-performance computers in batch mode where a queueing system is used. If the results lead to unexpected or unwanted output, then the simulation might be rerun with different parameters. It results in inefficient utilization of computing resource and increases the turnaround time for the scientific experiment.

To overcome this issue, the monitoring of the behavior of a simulation, while it is running (or live), is highly suited. One method of monitoring is to write the data, produced by the simulation, periodically to the disk. But, the disk being inherently slow can become a bottleneck and can affect the performance. Another approach is to use the computational steering technique, in which simulation is coupled with a visualization system to enable the exploration of "live" data as it is produced by the simulation.
In this work, we use the Gas Electron Multiplier (GEM) [1] as a case study. Since its introduction in 1997, the GEM has been used in many scientific applications [2]. It has been introduced as an amplification stage for gas based detectors. When a charged particle traverses a GEM based detector, it ionizes the gas molecules generating a freely moving electron that starts ionizing the gas molecules and produces the secondary electrons. These secondary electrons also ionize the gas and subsequently form an avalanche of electrons under the influence of the applied drift field. The avalanche drift, under the effect of the electric field, toward a readout board where the signals are amplified and recorded. In this work, we employ the computational steering method by coupling the GEM simulations with a visualization package named VisIt. A user can connect to the running simulation with the VisIt client over the network, and can visualize the "live" data to monitor the simulation behavior. Also, the simulation can be restarted immediately on the fly with different parameters without requiring resubmitting the job on the supercomputer.

In the GEM simulation, one of the commonly used simulation software is Garfield [3]. It has been successful in simulating a wide variety of gas-based detectors for many applications, including those beyond high-energy physics. In this work, the parallel version of the Garfield known as pGarfield developed by our team, has been used. The pGarfield is written in C++ and based on the MPI parallel framework. It provides a near linear speed up and excellent parallel efficiency.

2. Objectives
There are two primary objectives of this work. First is to provide an environment to perform the visual debugging and exploration of the simulations while they are running in real-time. It would help in monitoring the behavior of the simulation at any point in time on user's demand during the execution. It will help in detecting the errors at early stages that can save both the researcher's time and system resources. In addition, it should have zero or minimal impact on the performance of the simulation. The visual environment should not disrupt the simulation in any way and should take little amount of system resources. The second objective is to steer the execution of the simulation while it is running. That is, the users should be able to pause, resume or stop the running simulation. There should also be a way to change the simulation parameters and restart it without resubmitting the batch job to the supercomputer.

3. Computation steering with VisIt
The computational steering is the interactive control of the simulation during the execution while viewing the results of the calculations [4]. It enables the interactive control over both the simulation and the visualization (See Figure 8). This interactive visual inspection is necessary to allow researchers to examine the large quantities of data produced by the simulations.

![Figure 1: Computational steering](image)

The simulations are commonly run on the supercomputers in batch mode with no interaction possible through the command line interface. One way to interact with the simulation during the execution is via the file system. In this way, a simulation can write the results of the calculations periodically in the form of graphical images and/or numerical data to the file system.
accessible to the user. It can also monitor a predefined file containing the simulation parameters and take action appropriately if any change is detected. Another way of interaction with the simulation is over the network using TCP network ports. It is an efficient method as compared with the file system based method, especially in the large scale simulations involving millions of electrons, as the disk being inherently slow can become the bottleneck and can degrade the overall performance. Both of these methods, however, require major modifications in the simulation code.

Figure 2: VisIt architecture diagram for in-situ visualization of the parallel simulation [5]

Another approach of achieving the steering in the running simulation is to use an existing steering and visualization frameworks. It does not require major modifications in the code as compared with former methods. One of the most commonly used framework that does not require much code modifications is VisIt [6]. It provides a client/server architecture in which the tasks of visualization and data analysis are separated into different components. The client program runs on the local computer and leverages the hardware acceleration. The server programs runs on the large supercomputers remotely and perform computations in parallel. Moreover, all compute servers maintain identical data flow networks and execute using different pieces of the larger data set. In addition, VisIt’s compute servers are optimized for distributed memory parallelism using MPI. Figure 2 shows the software architecture of the VisIt, illustrating the interaction among the clients, parallel simulation code and its data flow network (VisIt server). One of the major benefits of using VisIt is its In-Situ Visualization capability that enables the development of the visual representation of the simulation results on the fly using just the system memory without requiring writing to the disk. It is very important, as the disk is inherently very slow as compared with the system memory and it can create a bottleneck and slow down the whole simulation.

A typical in-situ analysis of a parallel simulation in VisIt proceed as follows:

(i) User submits a job to the supercomputer and wait until it starts execution.
(ii) When the job is scheduled, the simulation code launches and starts execution.
(iii) The simulation code periodically checks for connections from the VisIt clients.
(iv) When a connection request is detected, the simulation loads the VisIt server library and allows it to complete the connection. If there is no connection request, the simulation continues.
(v) After the VisIt server library is loaded and connection is completed, VisIt server asks the simulation for a description of its meshes and data types (metadata).

(vi) VisIt server passes the simulation metadata to the VisIt client. With this information, user can build a data flow network in VisIt client and apply several visualization operators/filters.

(vii) When the user applies a filter, each VisIt server executes the identical data flow network on different pieces of larger data set.

(viii) When the user draws a plot (a visualization operator that produces a visual output). The results from all VisIt servers are combined and transmitted to VisIt client over the network if the size of output data is small. In case of large output, the images can be rendered on the VisIt servers in parallel with the help of VisIt’s scalable rendering feature.

4. Instrumentation of the GEM simulation
VisIt provides its in-situ visualization and steering features via a programming library known as libsim library [7]. The pGarfield simulation code is instrumented with the libsim library due to which it acts in many ways like a VisIt compute engine. The simulation runs on large supercomputer remotely performing the electron avalanche calculations while the VisIt engine listens for any request by the VisIt client. The instrumentation of the simulation is an incremental process and usually requires minimal additions to the source code. This process is mentioned below:

4.1. Phase 1: Adapting the Main Loop
The instrumentation of the simulation code starts with the re-organization of the execution control flow – usually, a simulation code have a main loop where it performs the calculations in iterative manner. It requires adding few lines of code that, first of all, allows establishing the connection to the VisIt GUI if any interaction by the user is detected. Next, after the connection is established, it receives and serves requests for data queries. Finally, it disconnects and let the simulation continue after the client completes the visualization session. This execution control flow has been depicted in Figure 3. The solid line shows the control flow of the main loop in absence of user interaction whereas the dashed line shows the control flow when user interaction is detected and in-situ visualization session begins.

![Figure 3: Simulation control flow loop instrumented for in-situ visualization (single-threaded)](image-url)
Note that the aforementioned instrumentation process applies only to a single-threaded application that runs in an iterative manner. VisIt assumes that the execution of single simulation step does not take longer. When a step is solved, the VisIt library function is used to check for any user interaction. But, what if a simulation step takes long time to complete (as in case of the pGarfield simulations)? VisIt documentation does not provide any information about such cases. It can be imagined that the VisIt client would not behave in a responsive manner. For the pGarfield simulation, each avalanche event may take significant amount of time to complete. Due to this long execution time, user might have to wait until the simulation of current event finishes. It may reduce the system response and forfeits the benefits of in-situ visualization as system would not response in a predictable manner.

To keep the system responsive while the event is being simulated, a multi-threaded execution model has been implemented in the pGarfield simulation client. One thread (main thread) simulates the electron avalanche events whereas another thread (VisIt thread) listens for the connection requests by the VisIt client. This approach keeps the two pieces of code (simulation and visualization codes) separate and also keeps the application design simple, easy to understand and coherent. With a separate thread always listening, hence incurring minimal performance overhead, the queries by the VisIt client can be answered in a responsive manner.

4.2. Phase 2: Metadata and Data Access Functions

After the first phase, the VisIt client can connect to the live simulation but it won’t be able to run any visualization query because the simulation has not advertised any of its data structures. This phase instruments the simulation code to define the data interface with the VisIt compute engines. It adds the metadata and data access code so the VisIt compute engine can access the simulation’s data structures [8]. The metadata access code is a function that provides metadata to the VisIt so it knows the names of the meshes (lines in case of electron traces) and scalar/vector variables (electron id, energy, etc.) that are available for plotting. Whereas, the data access code is a function that passes the actual data to the VisIt so it can be used in plots. The data access code may need to transform the internal data structures in a format compatible with the VisIt’s compute engine.

4.3. Phase 3: Control and Steering Functions

In this phase, the user-defined commands are advertised to the VisIt Compute Engine. VisIt exposes these commands as buttons in its client GUI and allow the user to interact with the simulation code. These commands give an opportunity to perform the steering of the simulation from within VisIt. In addition to these commands, VisIt also supports control functions, that let user control the execution of the simulation. For instance, user may pause the running simulation if an interesting feature is discovered and want to investigate it in details; once the investigation is done, the simulation can be resumed.

4.4. Results

Figure 4 shows the tracks of two electron avalanches (events) from different views. The avalanches have been colored differently to distinguish one from another. Each avalanche keeps record of the position, and several other scalar attributes (such as electron energy, event ID or electron ID, etc.) of all electrons being generated in an event. These variables help in developing various visualizations of the avalanches that can assist in understanding the simulation behavior. For instance, Figure 5a shows two avalanches and maps the individual electrons to different colors. It can be clearly seen that how the primary electrons (blue color trace) drift through the detector from the top and create the secondary electrons. The primary electron has ID equal to zero whereas all secondary electrons have IDs greater than zero. Figure 5b also shows two avalanches color-coded with respect to their event IDs. This representation can be used to
Figure 4: Different views of two electron avalanches

Figure 5: Color mapping

compare multiple avalanches in terms of their sizes and gains. For instance, it is clear that the avalanche with red color is bigger than the one in blue color. Similarly, Figure 5c shows another representation in which traces are colored with respect to the simulation time.

In addition to the color mapping operations, VisIt also supports several visualization operators and filters that can be applied to the avalanche data. One of such operators is the Threshold operator. The Threshold operator extracts the portions of the input dataset whose scalar values lie within the specified range. Figure 6 demonstrates the result of applying this operator on the avalanche ID scalar attribute. Figure 6a shows two avalanches in a combined visualization and Figures 6b and 6c show the two avalanches separately. This operator helps in analyzing the individual avalanches and electrons in depth. VisIt supports a long list of operators. The complete list can be found at [6].

To study the simulation’s behavior over time, the visualizations can be captured periodically. Figure 7 shows the screenshots of the VisIt’s graphical output window captured after every two minutes. The avalanches have been colored with respect to their avalanche IDs. In this case, the simulation client simulates the detector based on the triple-GEM (three layers of GEM) in parallel fashion over eight CPUs. For the remote control and on-the-fly modification of the simulation parameters, a customized user interface has been implemented using VisIt as shown in Figure 8. It shows a dialog box having information about the simulation such as its name, host at
which it is running, start date and time, and number of worker processes. In the same dialog, it also shows simulation status (currently stopped) and the buttons for each user-defined command (halt, step, run and update). The dialog window on the right is a custom user interface designed in Qt GUI API [9]. This dialog is opened when user clicks the 'Custom' button (highlighted with an arrow in the figure). It shows the gas composition of the currently running simulation and also the number of completed and total events to be simulated. The same interface can be used to specify new compositions of gases (Argon, Carbon-dioxide and Methane), and restart the simulation by pressing the 'Restart' button.

5. Conclusion
The GEM simulations especially those with complex scenarios such as those involving high detector voltages or gas with larger gains are time consuming and computationally intensive. The monitoring of the behavior of a simulation while it is running plays an important role as it allows the researcher to make decisions earlier rather than waiting until it finishes. A researcher may want to terminate or restart a simulation with different parameters if an abnormal behavior is detected. Doing it early would not only result in efficient utilization of resources but would also expedite the scientific discovery process. In this work, we have instrumented a GEM simulation with a computational steering API provided by a visualization software named VisIt. We have demonstrated how an active GEM simulation can be visually inspected, and also how the user-defined commands and parameters can be modified on the fly to control the simulation running remotely on the supercomputer.

References
[1] Sauli F 1997 Nuclear Instruments and Methods in Physics Research Section A 386 531–534
[2] Sauli F 2016 Nuclear Instruments and Methods in Physics Research Section A 805 2–24
[3] Veenhof R 1984 Cern Program Library W 5050 2001
[4] Marshall R, Kempf J, Dyer S, Yen C C and Dye S 1990 ACM SIGGRAPH Computer Graphics 24 89–97 ISSN 00978930 URL http://dl.acm.org/citation.cfm?id=91394.91424
[5] Whitlock B, Favre M J and Meredith S J Eurographics Symposium on Parallel Graphics and Visualization
[6] Visit visualization tool https://wci.llnl.gov/codes/visit/
[7] Kuhlen T, Pajarola R and Zhou K 2011
[8] Getting data into visit 1.5.4 https://wci.llnl.gov/codes/visit/1.5.4/GettingDataIntoVisIt1.5.4.pdf
[9] Qt project http://qt-project.org/
Figure 7: Screenshots of the VisIt graphical window at two minute interval.

(a) 2 min  (b) 4 min  (c) 6 min  (d) 8 min

(e) 10 min  (f) 12 min  (g) 14 min

Figure 8: Steering interface in VisIt