ON THE POSSIBILITY OF DISTRIBUTED PARALLEL PROCESSING FOR ADVANCED GAMMA-RAY TRACKING

Adrian DUMITRESCU[1]

A fost analizată posibilitatea implementării unui sistem software distribuit pentru procesarea paralelă a datelor de poziție a interacțiunilor fotonilor gama. Studiul a constat în analiza avantajelor şi compararea lor cu inconvenientele asociate utilizării unei astfel de soluții software în cadrul proiectului AGATA. Cu toate că o implementare distribuită a procesării paralele de date aduce un spor de viteză considerabil, arhitectura curentă a sistemului de procesare induce limitări în privința posibilității dezvoltării si integrării unei astfel de soluții.

An analysis on the possibility of implementing a distributed parallel processing software system for advanced gamma-ray tracking data processing was performed. This study weighed the benefits and the disadvantages of using this approach to solve bottleneck problems in the data processing system of the AGATA project. Although the use of a distributed parallel processing software solution has been shown to bring considerable improvements in processing speed, the architecture of the current software processing module confines the development and integration of this solution.

Key words: parallel processing, AGATA, OpenCL, gamma-ray tracking.

1. Introduction

The use of hyper pure germanium detectors (HPGe) in array configurations in high resolution gamma-ray spectroscopy has led over the past twenty years to considerable progresses in the investigation of the nuclear structure. The first generations of tracking detector arrays were the EUROBALL and GAMMASHPERE [1]. In these projects, detection accuracy was improved by using Compton suppression shields or limiting Doppler broadening effects. Both of these problems lead to inefficiency in detecting gamma-ray interactions.

In the last decade, an innovative approach towards gamma-ray spectroscopy was developed. This method, in essence, implies the use of HPGe detectors to determine the energy and, most importantly, the position of each and every interaction. Digital spectrometry systems compare input signal shapes with stored reference signal shapes, which are actually system responses to specific types of interactions, in specific places, in the active detection medium. This method, known as pulse shape analysis (PSA), decomposes the signal from each

[1] PhD student, Faculty of Applied Sciences, University POLITEHNICA of Bucharest, Romania, e-mail: adrian_m_dumitrescu@yahoo.com

detector and compares it to reference signals, in order to produce the closest match and consequently locate the point of interaction. This allows for automatic Doppler or Compton corrections, which improve the quality of the output energy spectra. The largest and most ambitious project for gamma-ray tracking is the AGATA [2] project, an European HPGe detector array composed of 180 segmented crystals, which will offer 4π gamma-ray tracking in its full configuration.

One of the problems concerning gamma-ray tracking through pulse shape analysis is processing the huge data flow produced by the detectors. Data needs to be analyzed in real time, as storing large amounts of information would be very expensive. Thus, any factor in the system introducing speed limitations should be addressed. The PSA processes each individual event and its execution speed is proportional to the available computing power. The current paper investigates the possibility of solving the bottleneck problem associated with the current implementation of this algorithm, through a software solution relying on the computing power of multiple graphical processing units (GPUs).

2. Gamma-ray tracking in AGATA and the "bottleneck" problem

The AGATA (Advanced GAmma Tracking Array) experimental installation will consist of 180 highly segmented HPGe detectors in its final form. These detectors contain 36-fold electrically segmented germanium crystals, six fold azimuthally and six fold longitudinally. The way these detectors work and how they produce signals is beyond the scope of this paper and is described in reference [2]. Further details on the structure, operation and mechanics of the AGATA Demonstrator can also be found in reference [3].

Detector segmentation is not the only method of gamma-ray interaction positioning. Because charge carriers with different mobility (electrons and holes), formed at the point of interaction, drift towards collection electrodes at different speeds, the pulses outputted by detectors have characteristic shapes. These shapes consequently contain information about the radius at which the gamma photon interacted with the detection medium. This helps in creating a virtual "map" of possible interactions according to output pulse shape. Furthermore, in a segmented detector, adjacent segments also absorb part of the energy of the interacting photon, creating pulses with proportional amplitudes and shapes [4]. The shapes of the pulses, together with the segment in which the interaction took place, contain sufficient information to determine the type, energy and other properties regarding the event. Pulse shape analysis helps track gamma ray photons' interactions through the detector by effectively creating a virtual positioning grid. By digitizing the signals from each segment of the detector and comparing them with sets of reference signals in a database, it has been shown

that a tracking grid with a resolution of 1-2mm can be obtained [2]. As pulse shape analysis relies on comparing input samples to database entries in order to determine the position of the gamma photon, the tracking precision is given, in part, by the quality of these reference signals [5]. This comparison is done through the PSA algorithm that is named GridSearch and has the role of analyzing interaction events and determining their resemblance to stored reference signals.

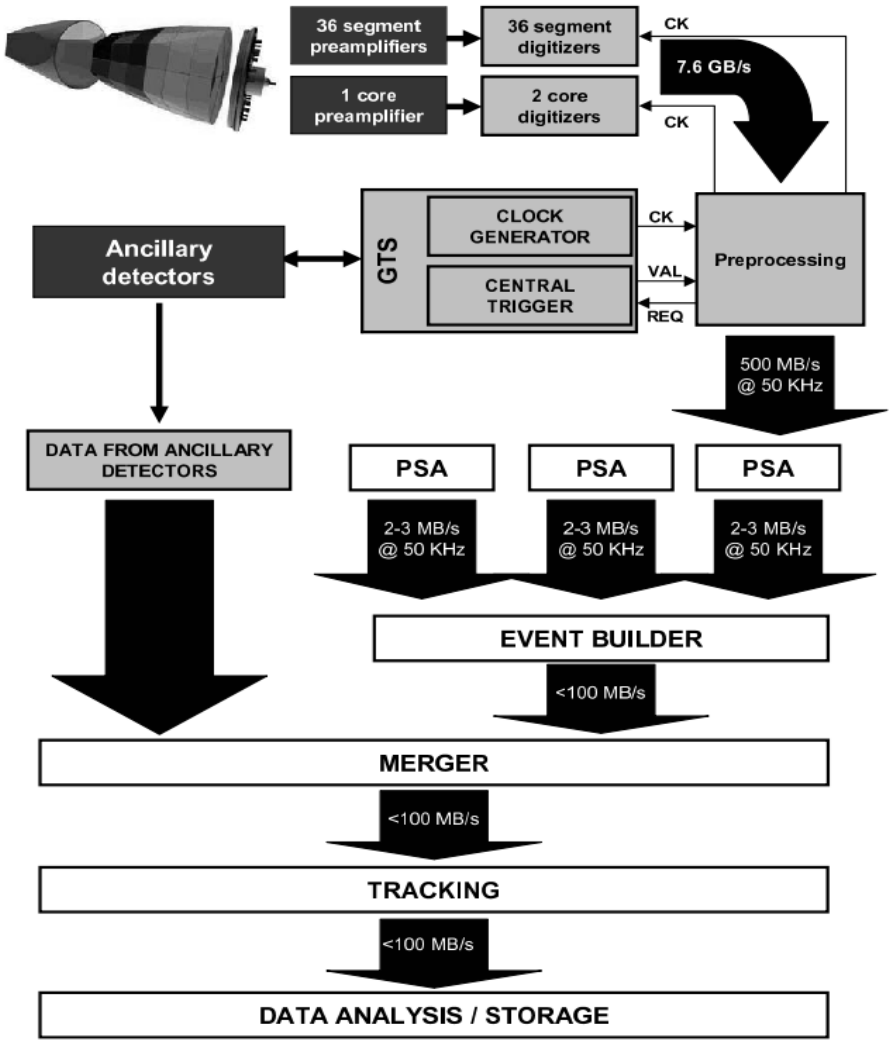

Fig. 1. AGATA DAQ. Analog electronics are represented in dark grey, digital electronics are represented in light gray and software processing in white boxes [7]. See text for discussion.

The data acquisition system (DAQ) built around AGATA is composed of analog electronics, digital electronics and software processing modules as shown in figure 1. Signals digitized from all the 36 segments of the detector plus the detector core add up to 7.6 GB/s of data that needs to be analyzed for a single detector. This means that the whole AGATA detector array of 180 detectors would produce an impressive 1.3 TB/s of digitized data that needs processing. This signal is inputted to the preprocessing part of the DAQ, which discards useless information contained in the digitized signal, and produces a stream of samples consisting of detected events. The data stream outputted by the preprocessor still has a bandwidth as high as 500 MB/s per detector. This data is further processed through software complied inside a computer farm. It was estimated [7] that 100 CPUs (Intel Xeon E5420 running at 2.5GHz) would be needed to run the current implementation of the PSA algorithm at a 50 kHz event rate to analyze the data from one detector crystal in real time. For now, in order to analyze events in real time, the event rate has been reduced to about 1 kHz. This means that the PSA farm needs only a few CPUs per detector crystal for real time analysis. As can be seen in figure 1, data analysis could encounter a bottleneck, as the processing hardware needed in the PSA farm increases both with the number of crystals and with the rate of input events' signals. The key to solving the bottleneck problem in the PSA is either finding a faster and more efficient way to process data with the given hardware infrastructure or using parallel processing techniques together with hardware that can provide parallel computing. Both approaches have been researched and improved versions of the grid search algorithm, through which peak shape analysis is done, have been developed [6, 7].

3. NARVAL, GPGPU and the GridSearch implementations

The software environment that facilitates data processing and dataflow management is named NARVAL (Nouvelle Acquisition temps-Réel Version 1.6 Avec Linux) and is a distributed data acquisition system that manages dataflow from and towards different processing modules. Its architecture is based on the concept of actors (processing or data management modules), which communicate data with each other. Details regarding this modular distributed system are described in reference [8]. Of interest to this paper is the fact that at execution time, every actor (unaware of the topology of the whole system) has assigned a single process that can run on one core of the machines in the computing system. Also individual actors can be developed separately as long as they interface with NARVAL for loading, control and communications. PSA is implemented through C++ based classes as processing code inside a NARVAL actor. For development and testing purposes, a NARVAL emulator has been developed since debugging inside the real system is very hard and, in some cases, impossible. The difference

with respect to NARVAL is that data processors run on the same machine and are executed sequentially, until the last processing module in the dataflow path is reached. Using this emulator, several PSA actors were developed by the AGATA team, including the ones presented in references [6] and [7]. Of interest to this paper is *E. Calore's* implementation [7] that proposes the development of the grid search algorithm, which lies at the heart of the PSA, using GPGPU (General-Purpose computing on Graphics Processing Units) techniques. This means using the GPU to perform computations that have traditionally been done using CPUs. The main reason for using these methods is that GPUs offer a lot more power for highly data parallel computations [7]. In the aforementioned implementation of the GridSearch algorithm for AGATA, the GPGPU OpenCL standard specification was used. A full description of the OpenCL specifications is available in reference [9]. Of interest to this paper is the fact that this framework makes available multiple platforms through which data processing can be done.

Calore's work showed that OpenCL implementations of the GridSearch algorithm are possible, as the problem which they solve is parallelizable to a certain degree [7]. The problem addressed by the GridSearch algorithm, as mentioned above, is comparing the acquired signal to a basis and evaluating the figure of merit expressed through equation 1. The algorithm needs to find the basis record which minimizes the figure of merit for each segment j iterated through available segments, each sample i iterated trough the signal, and, of course, from the available base points. The figure of merit is actually the metric used for the search, being the difference, to a certain power p (for example p=2 corresponds to Euclidean metric), between acquired points and basis points.

$$FoM = \sum_{j \in Segm} \sum_{i=T_0}^{T_{end}} | S_{ij}^{m} - S_{ij}^{b} |^{p} \tag{1}$$

As the problem was parallelizable under the SIMD (single instruction multiple data [10]) paradigm, several implementations in OpenCL for the GridSearch algorithm were developed. The CPU based algorithm had three loops: one over base points, one over the segments and one over the samples. All these three loops had to be iterated, so that a local figure of merit, named chi2, could be calculated. This figure of merit was then evaluated and the signal basis with the smallest metric was chosen as the closest approximation for the real signal. The OpenCL implementations were aimed at reducing the number of iterations required to compute chi2, by parallelizing calculations through multiple running kernels. The first implementation parallelized calculations by running just two loops, one over segments and one over acquired samples, for each thread representing base points. An optimized version of this implementation used

explicit calculation and other memory related optimization to improve execution speed. The third and final implementation attempt further parallelized calculation by looping only over segments and running different threads for base points and samples. Test results for these implementations showed that massive speed increases could be obtained compared to the CPU implementation. The increase in event processing speed is even more prominent when using float input data [7]. This allows the PSA to obtain a better position resolution. The results obtained in the tests ran by *Calore* are reproduced in this paper in order to showcase the evident speed increases obtained through these OpenCL implementations.

Table 1

Grid search implementations comparison as presented by E. Calore in [7]. See text for discussion.

Device	CPU	GPU		Comparison
Type of data input	Original	1D opt	2D opt	Speed increase CPU vs GPU
short	275 ev/s	650 ev/s	1250 ev/s	X 4.54
float	65 ev/s	650 ev/s	1100 ev/s	X 16.92

4. The possibility of distributed parallel processing for AGATA

Tests showed considerable improvements were obtained in event processing speed for OpenCL implementations of the GridSearch algorithm. Furthermore, major speed improvements were obtained while using the above described implementations to process float data [7].

Further speed increases are theoretically possible through the usage of multiple OpenCL environments to process data in a distributed manner. Aside from the benefits of the above mentioned parallel processing implementations, a multiple environment calculation system would make possible distributed parallel computing. In other words, this system would be able to run OpenCL parallel processing kernels on any compatible OpenCL device. As the OpenCL specification is widely adopted by hardware manufacturers, virtually all processing components inside a modern computer are supported as OpenCL devices, either natively or via a software compatibility layer. The AMD APP SDK [12], enables CPUs and compatible GPUs to be used as OpenCL devices. For example, this means that a processing machine with one CPU and 2 GPUs, a common configuration amongst high end personal computers, can process data using three separate OpenCL devices. In this example, one could run the same OpenCL code (kernel) on all three devices, as compatibility would be assured by the software platform. Other studies have shown the possibility of utilization of OpenCL devices, by connecting them over the network through Hybrid OpenCL

[13]. The level of abstraction brought by the usage of OpenCL devices allows the possibility of writing OpenCL applications capable of exploiting any compatible hardware found on the system.

The first step in making the GridSearch algorithm run on multiple OpenCL devices was a software routine that would analyze the configuration of the available hardware. For this purpose, a routine that scans the available hardware for OpenCL devices was developed based on a template from the AMD APP SDK. An important fact to point out is that this function did not rely on or load the aforementioned SDK (software development kit). In other words, it was developed with the objective to be as lightweight as possible in order to avoid adding code overhead. The routine consisted of five components which ran sequentially. First, the initializeHost function was run in order to initialize host computer memory. Then, the initializeCL function queried if there was an available OpenCL platform and if so, it created an OpenCL context through which it could detect available OpenCL devices. At this stage, OpenCL devices were discovered and listed to the user. This function was followed by the runCLKernels function that loaded kernels to devices according to their capabilities. The last two functions handled the cleanup of OpenCL environment and the host memory. CleanupCL freed the GPU memory and released the kernel, the program, the command queue and the OpenCL context. CleanupHost freed the host memory, the host buffers and ended the method. The workflow of this method is presented in figure 2.

Fig. 2. Algorithm of the multiple OpenCL device scanning routine. See text for discussion.

This routine was developed outside the NARVAL emulator, in order to simplify testing and debugging. Still, the OpenCL kernel computed the figure of merit presented in equation 1 with random sets of numbers, in order to test compatibility and speed of execution. The multiple device routine was run on CPU and on GPU and found to be compatible with both, as they were recognized as OpenCL devices. Restraints in code manipulation and development had to be considered before moving to the integration with the NARVAL emulator. This came as a consequence of the need to minimize the changes brought by this development to the NARVAL software environment. Because stability is a key requirement for the PSA library, only the inner loop of the GridSearch algorithm was available to be modified. Aside from reliability, other developers should be able to add functionality and develop new software applications without having to deal with the complexity of OpenCL development.

At this point, it was found that the inner loop of the GridSearch algorithm cannot be separated to a degree that would make possible running this function on multiple OpenCL devices. This is because, without substantial modifications, the current architecture of the GridSearch algorithm does not allow the calculation of the figure of merit in a separated code module. A key concern for future development was that adjacent software libraries changes should be minimal and should not affect other components. Because the routines necessary to recognize, map, load and use OpenCL devices need to be inserted into the startup routines of NARVAL actors using distributed parallel processing, major changes to the architecture of the system had to be made. Keeping in perspective that this type of approach would solve only a certain bottleneck problem and would most certainly impede future development in other areas, it was decided that the disadvantages of added overhead and complexity far outweigh the advantages brought by the event processing speed. Furthermore, the lack of flexibility brought by choosing this approach would require future developments to take into account the lack of support for non-parallelizable algorithms. Other limitations regarding the OpenCL implementation of the GridSearch algorithm are noted in [7].

5. Conclusions

Research on the possibility of improving the OpenCL implementation of the grid search algorithm by making it scalable on multiple machines/GPUs was done. The first step was creating a software method that recognizes available OpenCL devices. This was done by building code on top of an OpenCL code template (code derivation). The second thing to do was integrating the OpenCL device scan method in NARVAL. At this step, it was discovered that because of the current architecture of the GridSearch algorithm, substantial changes would have to be made in order process data using multiple OpenCL devices. At this point, it was decided that this implementation would lack the flexibility for further development and would not be suitable to the ever changing computational needs of the project. As minimizing code modifications was one of the most important objectives, the development of this implementation was abandoned because the benefits would be outweighed by the overhead and inflexibility to future development. These disadvantages also overcome the advantages brought by the event processing speed, even if this speed would allow real time data processing. Furthermore, reliability would also be an issue. The result of this research uncovered that without substantial modifications, the current architecture of the GridSearch algorithm does not allow data processing in a separated code module, as adjacent software libraries would also have to be rewritten. This would add more overhead for future developments and complicate the environment. At this

point, it was concluded that the proposed implementation would not be overall beneficial to the project.

It was also noted that NARVAL is an ever changing software environment and the code in this implementation cannot be used in all use cases. The code is very particular to a certain bottleneck problem and it has certain limitations related to up scaling. These limitations arise from data transit and memory management issues related to using multiple machines to process the data flow.

However, in the future there may come a time when parallel processing of data for the AGATA software platform will become a viable option. This opportunity will arise provided the AGATA project would create a department or group to handle development, implementation, testing, support and track changes for the OpenCL processing environment. This group would have the role of providing stability and support regarding the OpenCL multiplatform environment based on the specific needs of AGATA.

Acknowledgements

The author acknowledges the financial support within the project POSDRU, PRODOC, financed through the contract POSDRU/88/1.5/S/61178. The author would like to thank the people who took part in the AGATA project, especially Dino Bazzaco, Calin Ur and Enrico Calore for their support and guidance through this analysis project.

R E F E R E N C E S

[1] *J. Eberth and J. Simpson,* Prog. Part. Nucl. Phys. **60**, 283, 2008.
[2] *S. Akkoyun et al.,* Nucl. Instr. and Meth. **A668**, 26-58, 2012.
[3] *A. Gadea et al.*, Nucl. Instr. and Meth. **A654**, 88, 2011.
[4] *N. Warr, J. Eberth, G. Pascovici, H. G. Thomas, and D. WeiShaar*, European Physical Journal A, **20**(1):65-66, 2003.
[5] *M. C. Schlarb,* PhD thesis, Fakultät für Physik der Technischen Universität München Physik- Department E12, 2009
[6] *R.Venturelli, D.Bazzacco*, LNL Ann. Rep. 2004, p.220, INFN-LNL, 2005.
[7] *E. Calore,* Tesi di laurea specialistica, Università degli Studi di Padova, Facoltà di Ingegneria,Dipartimento di Ingegneria, dell'Informazione, 2010.
[8] X. *Grave, R. Canedo, J.-F. Clavelin, S. Du, and E. Legay*, IEEE-NPSS Real Time Conference, 0:65, 2005.
[9] *Kronos Group*, OpenCL Specification, **1.0, rev. 48**, 2009.

[10] *W. D. Hillis and G. L. Steele Jr.*, Data Parallel Algorithms, Communications of the ACM, Vol. 29, No. 12, pp. 1170-1183, Dec. 1986.
[11] *Advanced Micro Devices, Inc*, AMD APP SDK **v2.6**, 2011.
[12] *R. Aoki, S. Oikawa, T. Nakamura, S. Miki*, 2011 IEEE 9th Int. Symp., Parallel and Distributed Processing with Applications (ISPA), Japan, 2011.