Towards a Model Transformation Tool on the Top of the OpenCL Framework

Tamás Fekete and Gergely Mezei
Budapest University of Technology and Economics, Budapest, Hungary

Keywords: Model Transformation, GPGPU, OpenCL, Pattern Matching, Graph Rewriting.

Abstract: Nowadays, applications must often handle a large amount of data and apply complex algorithms on it. It is a promising and popular way to apply the computation in parallel in order to meet the performance requirements. Since GPUs are designed to apply highly parallel computations efficiently, using CPU+GPU heterogeneous architecture have gained an increasing popularity in computation intensive applications. Model-driven development (MDE) is a widely used software development methodology in the software industry. MDE is heavily building on model transformations in converting and processing the models. Graph transformation-based model transformation is a popular technique in this field. It is based on isomorphic subgraphs matching, which often require serious computing power. Currently, model transformation tools are not capable of using the computation power of the GPUs. Our research goal is to create a general model matching and later a model transformation solution, which can take the advantages of the computation power of the GPUs. We are now focusing on pattern matching of the transformations. We would like to create a general solution which is independent of the hardware vendor; therefore, our method is based on the OpenCL framework. The novelty of this paper is a GPGPU-based pattern matching tool and some accelerating techniques to achieve faster computation. In this paper we present an overview of the solution and test results based on one of the biggest freely available movie database (IMDb). The main properties such as the performance and the scalability are discussed. The applied architecture and the steps towards the final solution are also included in the paper.

1 INTRODUCTION

Nowadays, software applications process and handle a huge amount of data. Therefore, the execution of complex algorithms on this data often becomes a heavily time consuming operation. Using parallelized algorithms is a promising way to improve the performance of the applications. We need a hardware device for this to supports massive parallelism. GPUs seem a perfect candidate for this, since nowadays they tend to have thousands of computations units for parallel evaluation. The CPU+GPU heterogeneous architecture have enough power source to develop extremely fast algorithms. The processing power of GPUs is already widely applied in several fields like image or audio processing. This trend can be extended to new domains as well.

On the market, numerous kinds of GPUs can be found. Using a vendor, or model specific language and framework would need a tremendous effort. To avoid this, the OpenCL framework has been created in 2009. OpenCL is a platform independent framework which can be used to handle the most widely used GPUs uniformly. OpenCL is an interface defined by Khronos Group (Khronos Group's website, 2015) and each product vendor has its own implementation. Although these implementations differ from each other, the interface grants the compatibility between the vendors.

Model-driven engineering (MDE) is a widely applied software development methodology in the software industry. Models are not only created for presentation purposes, but they are transformed, processed and often used directly or indirectly as the basis of the code generation. Therefore, to find efficient model transformation techniques is important and challenging part of the MDE. Several techniques exist, one of the most popular is the graph rewriting-based transformation which is also referred to as graph transformation. Graph transformation is based on an NP complete problem (subgraph isomorphism) and may need serious amount of time depending on the size of the input model. A solution

Fekete, T. and Mezei, G.
Towards a Model Transformation Tool on the Top of the OpenCL Framework.
DOI: 10.5220/0005792903550360
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 355-360
ISBN: 978-989-758-168-7

for the performance issue can be to use the aforementioned CPU+GPU architecture.

Our overall goal is to create a general model-transformation tool using the computing power of GPUs. The tool is referred to as the GPGPU-based Engine for Model Processing (GEMP). We have done several steps in creating GEMP (Fekete and Mezei, 2015). In that paper, we concluded that the usage of the GPUs in graph transformation tools is a promising direction. In this paper, we follow this path by introducing a general, GPU-based solution for pattern-matching.

The rest of the paper is organized as follows: In Section 2, the features of the OpenCL framework are collected and compared against other solutions. Moreover, a short overview of model transformation tools can also be found. In Section 3, the input domain model is described which is used for illustrating the non-functional properties of the GEMP. In Section 4, an architectural overview is given. The following two Sections pay attention to the data mapping to the OpenCL framework and introduce the core algorithms. In Section 7, the main properties of the tool are discussed, namely the performance and the scalability. Finally, in Section 8, we conclude and give an outlook for the possible researching directions.

2 RELATED WORK

There are several possibilities to realize heterogeneous computation tasks, especially for CPU+GPU based platforms. One of them is the usage of the popular OpenCL framework. Another widely applied way of accessing the GPUs is the usage of the CUDA (CUDA's website, 2015). It is necessary to question which of them is better considering several viewpoints such as performance, scalability, or the difficulty of the integration. In a paper, (Veerasamy et al., 2014), they introduced the usage of both the CUDA and the OpenCL in deeper details, but they do not give suggestion which is the better to integrate. Both of them have their own advantages which heavily depend on the actual problem that must be solved. The main reason of using the OpenCL framework over CUDA is that OpenCL can be used by many hardware manufacturers realizing the interfaces.

In a paper, (Yan et al., 2014), there are benchmarks showing how the OpenCL framework can be effectively used on different hardware components. They compared several hardware components like multi-core CPUs, AMD and NVIDIA GPUs. They also considered the differences between their and other existing results. They measured the GFLOPS both on GPUs and CPUs and collected their experiments in case of different kinds of tasks.

The usage of the OpenCL framework is challenging to those, who do not have experience in hardware close programming. Probably this is the main reason why there are so many libraries for OpenCL and for other multi-platform environments. In a paper, (Viñas et al., 2015), there is a discussion about the extension of a Heterogeneous Library with OpenCL; this modification provides easier access to OpenCL framework. Significantly decreasing the number of lines in the source code is achieved and introduced in the paper.

There are many OpenCL-based graph libraries and wrappers which can be used in realizing an OpenCL-based model-transformation tool. Using GPUs in a graph library is introduced by (Che et al., 2014). In this paper, solutions of several popular graph problems are modified to achieve the GPU-based version of them. They also realized that programming of the GPU can be difficult for a regular programmer and the implementation of the graph application can take a big effort. Therefore, they created a library which is called *BelRed*. The software building blocks are implemented on the top of the OpenCL framework. The performance of the library is represented on a case study. The main advantages of the *BelRed* library are its portability and the fact that the programmer does not need care of the kernel code writing and hardware close working.

In a paper, (Xu et al., 2014), they introduce how important and critic the graph processing components nowadays are. The paper also focuses on the algorithm mapping between the host and the GPU which is the biggest challenge in the effective usage of GPUs. They compiled 12 graph applications into the GPU device, studied the performance and suggested several approaches to accelerate the performance of the GPU-based algorithms.

There are several papers and studies which collect and classify the model transformation tools. In a paper, (Jakumeit et al., 2014), there are tools which are described, for example GREAT, IncQuery, Fujaba, Groove, Henshin, MOLA and Viatra2. None of these tools can use the power of the GPUs in model processing. Our current research focuses on the challenges to create a general purpose pattern matching tool, namely the first step towards creating a GPU-based model transformation engine. The verification and validation of GEMP is applied on two levels: (1) Low level functional properties are

tested using unit tests. (2) For testing and illustrating the high level non-functional properties, a domain model with a huge size of source is introduced.

3 INPUT DOMAIN MODEL

In this section, we introduce an input domain model which is used to illustrate the non-functional properties of GEMP, especially the performance and the scalability. The internet movie database (IMDb's website, 2015) (IMDb) is used as the input data. IMDb is the largest film and TV show related database. It has approximately 3.3 million titles, 6.5 million personalities (actors, directors, etc.). From the IMDb, a huge input model can be created. Several interesting and complex patterns can be searched in this database for testing purposes. In the current research, a text file-based database is used which contains information on several domain concepts, like movies (subtitle, creation time, and rate), actors (with played film), and producers (with film). Figure 1 shows an example for a pattern to be searched.

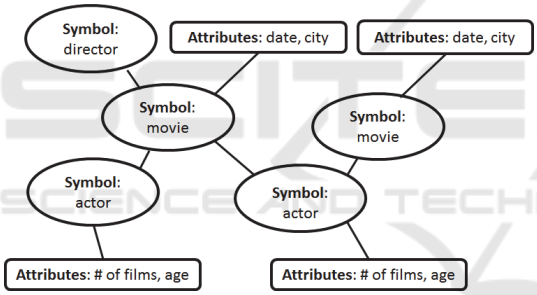

Figure 1: Example for a pattern to be searched.

4 ARCHITECTURE

The OpenCL kernel source code must be written in the C99 language (with some restrictions). The language of GEMP is C++11. OpenCL versions are backwards compatible. Since NVIDIA GPUs support only OpenCL 1.2 version, we decided to use this version in order to maximize hardware independency.

4.1 Component Model

Each component has a well separated functionality. We used the interface oriented programming paradigm to easily access the services of GEMP. Most frequently used design patterns are the *adapter* and the *façade*. For example, between the OpenCL API and the base libraries of the tool, the *CGPUAccess* component provides the connection and it is implemented as an adapter, while *CTaskRunner* is a façade. The domain model is created in the *CModelManagement* component which can be used for other domains. The most important components of the framework are shown in Figure 2 and in the list below of it (with their key roles).

(1) CGPUAccess: It creates the main OpenCL context and the command queue. Some objects are delegated to other components which need access to the GPU device. Inside of the component (2) initializes the connection between the host and the GPU device using the OpenCL API. At the same time, this context stores each run-time object provided by the OpenCL API.

(2) Business Logic (CPrephase1, CPrephase2, CPhase1, CPhase2): This component realizes the business logic (BL) of the tool. Each contained inner components are easily exchangeable and extendable. The BL uses the kernel source code and responsible for the compiling process using other components.

(3) CTaskRunner: The tasks are scheduled here. All kinds of time scheduling tasks must happen in this component.

(4) CLogging: It is responsible, not only for logging, but provides the output models as well.

(5) IMDbAccess: It manages reading of the domain model and provides the graph data to (6).

(6) CModelManagement: It is responsible for the domain model creation. It can also create the input graph and process the results.

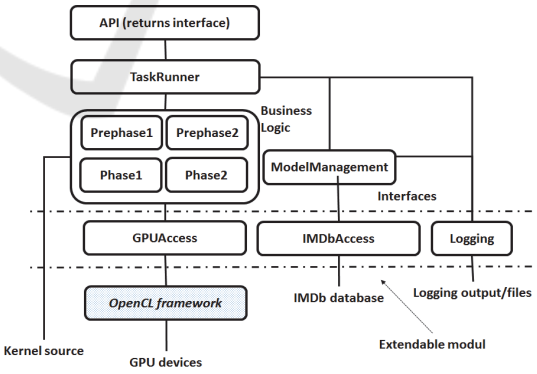

Figure 2: The base component model with the most important blocks.

4.2 Dynamic Behaviour and Testing

In the first place, GEMP provides interfaces to the task evaluation and to the configuration. As soon as the task is chosen, it is queued for execution. The *CModelManagement* prepares the domain model for

the tasks. In the current case, the IMDb source is the only one that can be processed. The *CTaskRunner* manages the configuration according to the user settings. Pattern matching is executed in two main steps. In the first step, we search for topological matches with the pattern. Nodes are represented by their ID. At the second step, we use the topological matching parts of the host model (the results of the first step) and evaluate attribute constraints on them. The reason for dividing the matching algorithm into two steps is that copying all nodes of the host model with all of their attributes to the GPU memory would be inefficient. To use two steps, we can reduce the amount of data to copy. Namely, we need to copy attributes only for those nodes, which are part of a topological matching structure.

During the implementation, according to the test-driven development methodology (TDD) (Canfora et al., 2006), a unit test is created right at the beginning of the implementation. Test cases are created and the expected results are calculated by hand. At every run of the unit test, the framework compares the received and the expected results in an automated way. A few examples on the unit tests: (1) There are cases when there is no result. Handling an empty result buffer must not cause failure. (2) There are possible scenarios when the results overlap. We must find each of them at this time. (3) There are test cases for error handling. We must be able to log information about all unwanted events from inside and outside of the kernel source.

5 DATA MAPPING

The model graph is represented as a hash table, where the ID of the vertex and the list of their neighbours are stored. The advantage of using a hash table is that finding an element based on its ID requires O(1) time. Although, by using hash table, a little bit more memory is needed, the time is more important in the current case. The result is stored in a different kind of structure. There are two kinds of data which must be mapped: At the first step of matching, (topological check), graphs consist of numerical elements (IDs). At the second step (attribute check), attributes are represented by strings (they are serialized to strings).

In case of graph mapping, the graph is converted as illustrated in Figure 3, in order to achieve the required format (the passing of 2D arrays is not allowed in case of OpenCL 1.2). The original, two dimensional structure of the graph is mapped into two one dimensional structures: (1) The first structure contains the list of the neighbours one by one from the first to the last vertex. (2) The second structure contains the starting positions of the neighbour lists. The second part is a helper structure to process the first structure. Using this two arrays and the size of the second array, all kinds of graph can be passed. Advantages of this structure are the degree of compression and the time of accessing elements in the graph.

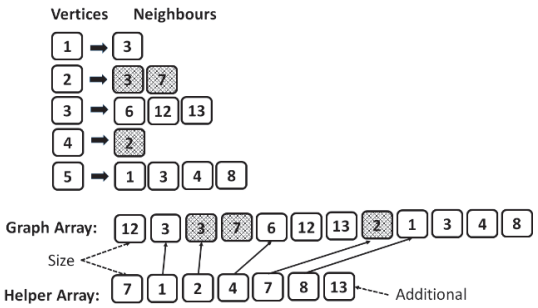

Figure 3: The input of the OpenCL kernel code must be a one dimensional array.

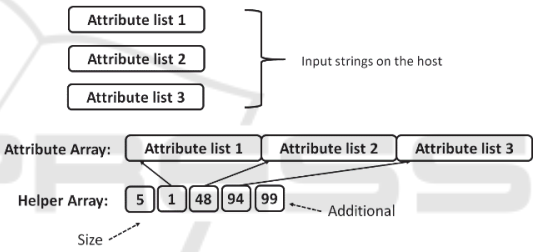

Figure 4: Concatenate string to OpenCL kernel code.

In the second step, a different data structure is needed. At this time, the OpenCL buffer must be big enough for each attribute. To store the beginning of the attributes, a helper array is used as well. Figure 4 shows this kind of mapping.

Considering the result buffer (*output data from the GPU*), there is a common reserved space in the GPU device global memory. Each thread can reserve a part from it and fill it any time. As soon as the thread cannot manage the reserving of memory for a new result, buffer overflow is occurred and handled.

6 KERNEL SOURCE

There are several important viewpoints for creating an effective kernel codes, one of them is the memory management. There are four kinds of memory areas in the OpenCL programming model: private, local, global and the host side memory. Another viewpoint is to choose the number of the working threads (*referred to as the local group number*). Choosing the

right number depends on the hardware device and therefore GEMP must recognize the GPU and manage the configuration according to it. Similarly, data types are important in case of different GPUs, using unsigned positive numbers are not worse than using other types and provide lots of values to store bigger graphs. We measured several cases.

In the first kernel code, the size of the result (*one searched pattern*) is known and must be filled. The algorithm seeks for candidates for the actual place and if it has been found, it tries to find the candidate for the next place (Fig. 5). As soon as there is no candidate left, the algorithm steps back and tries to find another matching vertex for the last place. If the algorithm reaches the first element or the element after the last one, it ends (*with failure, or success respectively*). The algorithm can be illustrated by a state machine, which is studying the current state, and makes a decision to accept it and step to the next state. Searching does not go deeper in the graph than the size of the pattern.

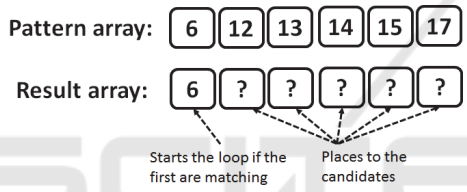

Figure 5: The task is to find matching result to the pattern.

In the second phase, the attributes (*strings*) of the vertices must be passed to the GPU device. Only the attributes of the result of the first phase are passed and processed one-by-one. Each thread processes an attribute (one vertex has one attribute). Processing means to find the requested condition in the given attribute. For instance we want to find each pattern where 4 actors play in the same movie and the actor has the name Jack, the pattern graph has four node in the current case and each result find using the first kernel. In the second step, only the attributes of the results are copied and processed to find pattern where the name attribute contains the Jack name.

7 PERFORMANCE AND SCALABILITY

Both of the time and the memory consumption must be monitored for studying the performance and the scalability during the execution. The following time measurement points are identified: (1) Reading the input data. (2) Converting the data to OpenCL input format. (3) Preparing the kernel code and copying the data to the GPU device, managing the computation and reading the result. (4) Processing and reading the result. (5) Creating the string and other buffers to the second phase. (6) Running the kernel of the second phase. (7) Giving the result of the second phase. Considering the memory usage, there are also predefined points, when the size of the memory is limited for testing purposes: (1) Allocating memory to the input model. (2) Creating result buffer on the GPU device. (3) Creating string input buffer on the GPU device in the second phase.

7.1 Input Model

Both the first (topological match) and the second (attribute checking) phase have to be able to deal with large input graphs. GEMP divides the graphs (generated from the IMDb database in the current case) randomly and processes them in several rounds. The missing results are processed on the host side parallel to the GPU threads. According to our measurements, the heavily divided graph results less findings on the GPU device. Dividing of the input graph is studied and measurements evaluated in a earlier paper (Fekete and Mezei, 2015).

In this paper, we introduce one additional step in the first kernel, namely using a pivot point in pattern matching. We select the first vertex to match in the pattern and find all candidates in the host model, thus creating starting points for matching. Then, possible starting points are counted and for each vertex, a worker thread is started. This means that we can significantly reduce the number of threads (all host nodes vs. nodes matching the pivot point of the pattern). We tested with several examples using the IMDb database. We achieved more than 10% performance increase in our test cases.

7.2 Result Buffer and Processing

If the optimal number of the worker threads is not configured for one round (*can be needed if the graph cannot be processed in a single round*), the computation time can slow down. Considering this kind of importance of the number of the worker threads, there are three formulas developed in earlier (Formula 1-3). If the kernel must be started too frequently (*buffer overflow or barely used buffer*), the performance is decreased.

$$newThreadNum = initThreadNum * \frac{\max BuffSize}{currentBuffSize} * C \quad (1)$$

$$newThreadNum = initThreadNum * C \quad (2)$$

$$\frac{\max BuffSize}{currBuffSize} > C \quad (3)$$

The *newThreadNum* refers to the number of the global worker threads that must be used in the subsequent runs of the kernel code. The number of the global worker threads used in the test measurement is denoted by the *initThreadNum*. *C* can be any kind of positive number, which increases or decreases the speed of the buffer size changing. The *maxBufferSize* and the *currentBufferSize* denote how many numbers can be stored and the size of the buffer used in the current round.

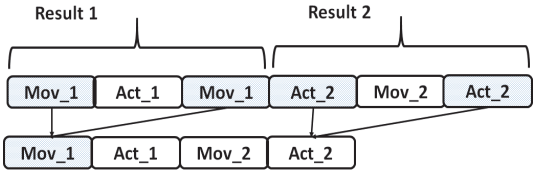

Figure 6: One attribute list can be appeared in more results.

One attribute can appear in several results and copying all of them would be only the waste of the power source. Instead, an attribute is copied only ones which is illustrated in Figure 5. In this case an additional helper structure is applied. The result of the test measurement heavily depends on the input model and the pattern to be searched. But in this case, we also could achieve at least 10% performance gain.

8 CONCLUSIONS

In this paper, a general pattern matching tool (GEMP) is presented based on our earlier studies and solutions. The tool contains two main steps with optional pre-processing steps. This two steps ensures that only the mandatory attributes are copied to the GPU device thus reducing time and memory. According to the new architecture, the applied domain model is much easier to be exchanged and all kinds of domain models can be used. Users can access and configure the tool using interface oriented techniques which makes GEMP a user friendly and easily testable.

As the pattern matching part of the tool is now complete, we are going to focus on graph-rewriting in the future and we also need study how we can integrate our tool into existing tools. Obviously, managing graph-rewriting effectively on the GPU device is not an easy task (consistency, performance, memory management issues) and we will face several challenges during the work. Some of them are already known, e.g. which part of the graph is necessary to copy to the GPU device and how is the update applied? Studying and solving these points we can achieve GPU-based model transformation tool with full functionality. This is our new step besides creating other case studies and apply more tests.

ACKNOWLEDGEMENTS

This work was partially supported by the TÁMOP-4.2.1.D-15/1/KONV-2015-0008 project.

REFERENCES

CUDA website, viewed 30 October 2015. www.nvidia.com.

E. Jakumeit, S. Buchwald, D. Wagelaar, L. Dan, A. Hegedus, M. Herrmannsdorfer, T. Hornf, E. Kalnina, C. Krause, K. Lano, M. Lepper, A. Rensink, L. Rose, S. Watzoldt & S. Mazanek, 2014. A survey and comparison of transformation tools based on the transformation tool contest. *Special issue on Experimental Software Engineering in the Cloud*.

G. Canfora, A. Cimitile, F. Garcia, M. Piattini & C. Aaron Visaggio, 2006. Evaluating advantages of test driven development: a controlled experiment with professionals. *ISESE '06 Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, USA.

IMDb website, viewed 30 October 2015, www.imdb.com/interfaces.

Khronos Group, viewed 30 October 2015, www.khronos.org/opencl.

Q. Xu, H. Jeon & A. M., 2014. Graph processing on gpus: Where are the bottlenecks?. *Workload Characterization (IISWC)*.

S. Che, B. M. Beckmann, S. K. Reinhardt, 2014. Belred: Constructing gpgpu graph applications with software building blocks. *High Performance and Embedded Computing (HPEC)*.

T. Fekete & G. Mezei, 2015. Creating a GPGPU-accelerated framework for pattern matching using a case study. *Eurocon2015*, Salamanca, Spain.

Veerasamy, Bala Dhandayuthapani & Nasira, G. M., 2014. Exploring the contrast on GPGPU computing through CUDA and OPENCL. *Journal on Software Engineering*.

Viñas, Moiśes; Fraguela, Basilio B.; Bozkus, Zeki & Andrade, Diego, 2015. Improving OpenCL Programmability with the Heterogeneous Programming Library. *Procedia Computer Science*, *International Conference On Computational Science, ICCS*.

Yan, Xin; Shi, Xiaohua; Wang, Lina & Yang, Haiyan, 2014. An OpenCL micro-benchmark suite for GPUs and CPUs. *Journal of Supercomputing*.