Automatic Multi-GPU Code Generation applied to Simulation of Electrical Machines

Antonio Wendell De Oliveira Rodrigues, Frédéric Guyomarc'H, Jean-Luc Dekeyser, Yvonnick Le Menach

Automatic Multi-GPU Code Generation applied to Simulation of Electrical Machines

A. Wendell O. Rodrigues, Frédéric Guyomarc'h and Jean-Luc Dekeyser
LIFL - USTL :: INRIA Lille Nord Europe - 59650
Villeneuve d'Ascq - France
{wendell.rodrigues,frederic.guyomarch,jean-luc.dekeyser}@inria.fr

Yvonnick Le Menach
L2EP - USTL
Cité Scientifique Bat.P2 - 59655
Villeneuve d'Ascq - France
yvonnick.le-menach@univ-lille1.fr

***Abstract*—The electrical and electronic engineering has used parallel programming to solve its large scale complex problems for performance reasons. However, as parallel programming requires a non-trivial distribution of tasks and data, developers find it hard to implement their applications effectively. Thus, in order to reduce design complexity, we propose an approach to generate code for hybrid architectures (e.g. CPU + GPU) using OpenCL, an open standard for parallel programming of heterogeneous systems. This approach is based on Model Driven Engineering (MDE) and the MARTE profile, standard proposed by Object Management Group (OMG). The aim is to provide resources to non-specialists in parallel programming to implement their applications. Moreover, thanks to model reuse capacity, we can add/change functionalities or the target architecture. Consequently, this approach helps industries to achieve their time-to-market constraints and confirms by experimental tests, performance improvements using multi-GPU environments.**

I. Introduction

Methods of numerical computing are essential in many scientific and industrial areas. Nevertheless, due to time constraints, communities of those areas are obliged to use parallel platforms to speed-up their results. There are many architectures suitable to parallelize scientific algorithms. Hybrid architectures based on CPU and other devices (e.g. GPU) are popular for economic reasons (i.e. price and energy consumption) and their good performance. However, creating applications on these architectures is an arduous task for non-specialists in parallel programming. This paper presents an approach that addresses:

1) **design methodology** based on MDE to generate automatically application code;
2) exploiting **higher performance multi-GPU** validated by a case study.

II. Background

A Graphics Processing Unit or GPU is the many-core processor attached to a graphics card. However, though it has diverse cores, its parallelism continues to scale with Moore's law. It is necessary to develop application software that transparently scales its parallelism. Proposals, such as OpenCL, have been designed to overcome this challenge. The Khronos Group released OpenCL [1] as a standard for parallel computing consisting of a language(which is an extension of C), API, libraries and a runtime system. OpenCL is based on a platform model that divides a system into one host and one or several compute devices. Compute devices act as co-processors(e.g. GPUs) to the host(e.g. CPU). An OpenCL application is executed on the host, which sends instructions, defined in special functions called kernels, to the device. Additionally, a single host can have multiple devices. OpenCL allows for creating contexts and queues in order to manage tasks being launched in all attached devices.

III. Application Design and Code Generation

A. MDE and MARTE

Model Driven Engineering (MDE) [2] aims to raise the level of abstraction in program specification and increase automation in program development. The UML profile for MARTE [3] extends the possibilities for modeling of application and architecture and their relations. MARTE consists in defining foundations for model-based description of real time and embedded systems.

B. Model Transformation Chain

In MDE, a model transformation is a compilation process which transforms a source model into a target model. This allows for adding, modifying, transforming model elements in order to achieve a final model closer to the real program application. For instance, the last model has explicit information about variables and task scheduling. In [2] there is an overview about the tools used in *model-to-model* and *model-to-text* process. Additionally, we have used the Gaspard2 [4] framework as the engine to chain and encapsulate these transformations.

IV. Case Study

The conjugate gradient (CG) method [5] is often used in modeling and simulation of electrical systems. It should only be applied to systems that are symmetric or Hermitian positive definite, and it is still the method of choice for this case. Input data are resulting from a FEM model of an electrical machine. The matrix is stored in *Compressed Sparse Row (CSR)* format having N=132651 and NNZ=3442951. The CG algorithm is modeled in MARTE as presented in the figure 2, where data reading and initial configurations are defined by stereotyped blocks. Highlighted gray blocks represent tasks, which are

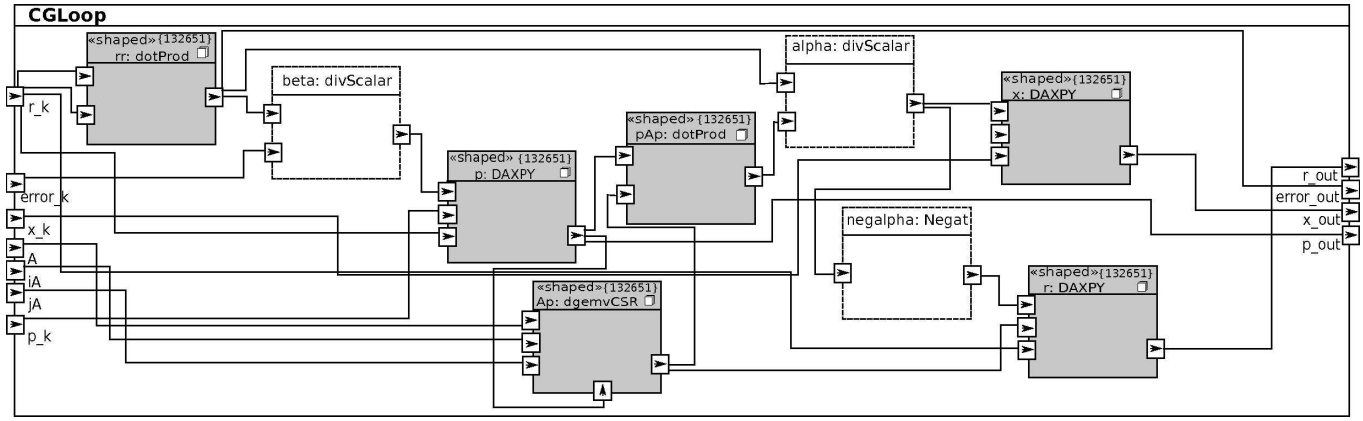

Fig. 1. Conjugate Gradient UML/MARTE Model

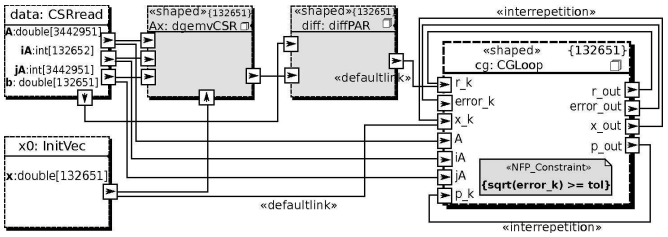

Fig. 2. UML/MARTE Model for Setup and CG Overview

mapped onto as many devices as we want to distribute the task job. Tasks, such as DGEMV(sparse), are repetitive and, thus, potentially parallel. The CGLoop is a 132651 loop which some of its input data are recovered between continuous iterations. A *continue-condition* is specified by a constraint attached to the CG block, so the loop can stop before running all iterations. The figure 1 is an internal view of the CGLoop modeled in the figure 2. Here scalar operations run on CPU processor, and repetitive operations run on GPU processors. Details about deployment of elementary tasks (operations), data and task allocation to architecture, scheduling, grid definition, and so on, can be found in [4] and they are not discussed in this paper due to scope and space limitation.

V. Results

We used four double-precision implementation versions of CG. The first one (the reference) is sequential and uses the Matlab's *pcg* function. The other ones are automatically generated OpenCL implementations whose kernels are launched onto 1, 2 and 4 devices, respectively. The number of used devices depends of the task allocation process. The hardware used is composed by a *2.26GHz Intel Core 2 Duo* processor and S1070 unit (4 Tesla T10 Nvidia GPU). Usually, manually written codes have better performance than automatic ones. However, these automatically generated CG implementations have an expressive performance(table I) compared to sequential code (time results include just computing and data transfer times in CG loop). The multi-GPU aspect is verified in the two latest versions. The code generation compiler decides equally the task partitioning to the multiple devices. The gain is not linear(though significant) due to extra data transfers among cpu and devices. A detailed analysis about solvers and Multi-GPU can be found in [6].

conjugate gradient	#iter	time(s)	speed-up	gflops
Matlab PCG	117	3.17	1	.303
OpenCL (1 GPU)	116	0.659	4.81	1.45
OpenCL (2 GPU)	116	0.461	6.87	2.07
OpenCL (4 GPU)	116	0.380	8.34	2.50

TABLE I
Performance Results; N=132651, NNZ=3442951, tol=1e-10

VI. Conclusion

In this paper, we purposed an approach that allows to decrease the application development time for parallel algorithms used in scientific areas. Additionally, the produced code can exploit multi-GPU platforms. Therefore, non-specialists in parallel programming can create applications using the potential power processing of their hybrid architecture. Experimental results show us that this aim is achieved properly for the conjugate gradient method.

Acknowledgment

This work is funded by the *Conseil Régional Nord-Pas-de-Calais*, Valeo and GPUTech and it's part of the Gaspard2 project, managed by DaRT team of LIFL/INRIA Lille.

References

[1] Khronos Group, "OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems," http://www.khronos.org/opencl.
[2] D. Lugato, J.-M. Bruel, and I. Ober, *Modeling, Simulation and Optimization - Focus on Applications*. In-Tech, March 2010, ch. 2, pp. 19–30.
[3] OMG, "Modeling and Analysis of Real-time and Embedded systems (MARTE)," http://www.omgmarte.org.
[4] DaRT Team - INRIA, "Graphical Array Specification for Parallel and Distributed Computing (Gaspard2)," https://gforge.inria.fr/projects/gaspard2.
[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, October 1996.
[6] A. Cevahir, A. Nukada, and S. Matsuoka, "High Performance Conjugate Gradient Solver on multi-GPU Clusters using Hypergraph Partitioning," *Computer Science - Research and Development*, vol. 25, pp. 83–91, 2010.