STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIV**, Number 1, 2019
DOI: 10.24193/subbi.2019.1.06

EMBEDDED SYSTEMS WITH COMPONENT-BASED GPU SUPPORT: A STATE OF THE ART

ANTONIU MICLĂUŞ, ŞERBAN PETRESCU, AND ANDREEA VESCAN

Abstract. In order to deal with extremely large quantities of information, embedded systems need high capabilities in order to process the whole amount of data in real time. Two trends are present in the field: the usage of boards with Graphics Processing Units (GPUs) and the usage of component-based development (CBD). Components with GPU capabilities have the great advantage to be platform-independent. However, developing embedded systems with GPUs by using CBD was considered until very recently a problem with restricted availability and flexibility. By introducing specific GPU support for CBD in the form of flexible components and by improving their communication, a solution was identified and checked. Present paper aims to present a state-of-the-art and highlights the newest knowledge to date, articulating encountered confronted issues and describing existing solution approaches.

1. INTRODUCTION

Many modern embedded systems deal with huge amounts of data originating from the interaction with the environment. For example, the autonomous car developed by Google 1 processes up to 120 MB of data per second delivered through its sensors [1]. The data must be processed with a certain performance in order to handle, in real-time, the environment changes.

A solution to process these data with adequate performance is the usage of general-purpose Graphics Processing Units (GPUs), which, thanks to their architecture, excel for highly data-parallel applications. Today, embedded-board platforms contain GPUs and different platforms have different architectures. Depending on their characteristics, like size, energy consumption or computation power, different platforms are suitable in different contexts. For example,

Received by the editors: January 21, 2019.
2010 *Mathematics Subject Classification.* 68U01, 68M99.
1998 *CR Categories and Descriptors.* I.3.2 [**Computer Graphics**]: Graphics Systems – *GPU*; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems.
Key words and phrases. GPU-capability, component-based development, flexible component, allocation optimization.

there are platforms with high-computation GPU used in high-performance computing solutions, but also low-computation with low energy consumption such as GPU used in smart watches [3].

An alternative approach in the development of embedded systems is the usage of component-based development (CBD). CBD is a software engineering methodology that promotes the efficient system development through the composition of already existing software blocks called (software) components. CBD advertises the use and reuse of the same component in different contexts, which increases the development efficiency.

CBD is ineffective for embedded platforms that combine Central Processing Units (CPUs) and GPUs. This is due to the lack of specific support for GPUs. This overall challenge has several aspects. One of them refers to the development of components with GPU capabilities, which is complex, time-consuming and error-prone.

Another existing issue involves the reduced flexibility of the current way in which component-based applications with GPU capabilities are designed. The existing hardware-specific components have a reduced reusability between different hardware contexts.

The aim of this paper is twofold: firstly, a state of the art is provided, classifying the existing conducted research on CBD for GPU and augmented it with additional newest published approaches in the last year, and secondly to highlight the existing solutions for the encountered issues relating to usage of GPUs.

The reminder of the paper is organized as follows. A review of current contributions in the area of embedded systems with GPU is presented in Section II, followed by a focused overview in Section III, where only mechanisms to ease CBD for embedded systems with GPU capabilities are treated. The end of Section III also contains the related work. Section IV extracts the conclusions of the reviewing work.

2. EXISTING RESEARCH ON EMBEDDED SYSTEMS WITH COMPONENT-BASED GPU

Campeanu [4] conducted a Systematic Literature Review (SLR) and investigated existing studies related to CBD and GPU aspects. In his study 49 papers were considered and, from those, only 17 were devoted to the area of embedded systems. It was shown that the development of CBD for GPU-capability applications was first approached/published in 2009, and even today, this field is still poorly represented in the mainstream research for industry. The directions in which research on CBD for GPU was conducted were classified by Campeanu [4] to be: a) development improvement (33 papers);

b) performance improvement (10 papers); c) software-to-hardware allocation (5 papers); d) experience sharing knowledge (1 paper).

There resulted a number of gaps/needs to be approached in the future:

- generally, no specific component models were used to build the solutions; in the studies that however approached the area in such a way, the most used were PEPPHER, UML, CCA and Rubus;
- more than 10 mechanisms were implemented to support GPU development, from which most applied were the programming and modeling mechanisms;
- the memory addressing was approached by an artifact, manual- or layer-based solutions;
- preferred programming languages were CUDA/OpenCL and C/Cpp (CUDA – developed specifically for NVIDIA, while OpenCL – more general and fits to platforms like AMD, Altera, IBM, Intel, NVIDIA, Samsung and Xilinx).

Since the Systematic Literature Review provided in [4], several other papers investigated the GPU in connection with other perspectives as: parallel applications, multiple streams and process variations.

In what follows, we present several other contributions not presented in [4].

With the aim of stream computing for real-time sensor correction, authors of [5] proposed a flexible and expandable on-board real-time data processing solution. The data coming from a high-resolution optical satellite was chosen while the proposed solution was based on multi-threading optimization and a CUDA collaborative strategy. The simulation prototype was implemented on an NVIDIA embedded GPU platform and it consisted in a double-module data parallel pipeline system. Programming, occupancy and data access improvements were used and checked. The on-board results were at the end compared against the same algorithms run on a Dell PowerEdge T630 Server, proving a feasible stream performance and low power consumption. Due to the good flexibility and expandability of the embedded GPU platform, the idea could be shifted to cover different applications in which optimization strategies to be adjusted accordingly while the number of pipeline modules should be redrafted in function of the computational requirements. To improve the on-board intelligent processing capability, authors of [5] also proposed the implementation of other algorithms, eg.: fusion, geo-rectification, region of interest (ROI) extraction, cloud-cover detection, target recognition and change detection real-time processing.

In order to simulate parallel applications running on GPUs, the authors of [6], proposed the adaptation of the performance Volkov model and its implementation on a MERPSYS simulator. CUDA and NVIDIA GPUs are already available in the model, while extensions are expected soon, to include AMD and the OpenCL frameworks. GPU modeling was very proficient with the Volkov model implementation, from the extendibility and feasibility perspective. The advantages offered by using MERPSYS with this model was proved in the directions: a) it provides possibility to assess the applications functioning for sizes of the data that exceed the hardware capabilities; b) hardware setups can be evaluated prior to computations/applications running; c) costs may be apriority predicted since it is possible to calculate the duration of the computation; d) shortening of the simulation times than the real runs is obvious. The model could be verified on different GPU hardware architectures and could be improved further on, by using double precision units, SFUs and shared memory. Also, an automation of the kernel analysis process is expected for this solution.

To solve the complex problem of correct dissemination of multiple streams coming from various sensors in a system, a recent solution based on an original architecture was proposed: the Parallel Data Distribution Service (PDDS) [7], published in 2018. It proposed solving the problem timely, reliable and scalable. PDDS centers its idea on parallelizing the model-related computation. The state estimation of sensor streams was made by involving general-purpose GPUs (GPGPUs) to obtain high efficiency in energy and good scalability. Practically, an original data distribution algorithm has been implemented on a modern embedded device using CUDA by extending the data distribution service of the object management group. Three GPGPU kernels were involved: prediction, compaction and update kernels. With PDDS, serial network stacks could be bypassed and subscribers could have access to fresh sensor data by using local sensor models and with no communication with its publishers. It was proved that this algorithm consumes just 5% energy if compared to similar algorithms in use.

An interesting and complementary approach was recently addressed by analyzing the embedded GPU aging problem as a result of processes variations [8], published in 2018. An aging-aware workload management technique was used, in which the main actors were the warp scheduler and instruction dispatcher. The technique functions like this: before the launching of the kernel function, the host configures the GPU, taking into account the results from a running algorithm. This one refers to the formation of warp and to the workload division and generates information to reconfigure the cluster and scale the heaviness of the embedded GPU. It was proved experimentally that by using such a technique, GPU may excellently be reduced in (72-95) % of cases. Compared to the complier-based-technique, the present aging-preventing technique is less susceptible to soft-errors.

Since CBD is ineffective for embedded platforms that combine central processing units (CPUs) and GPUs, one solution is the development of components with GPU capabilities/settings and GPU-specific environment information. Encapsulating inside the components all the information results in specific hardware components destined to particular GPU architectures. There are two possibilities: a) specialized components made to encapsulate GPU functionality – but they cannot function without GPU hardware; b) introduction of special adapters to facilitate automatic data transfer between CPU and GPU memory system. Practically solutions are encountered in case of Autosar, Rubus and IEC 61131 [4]. The disadvantage is connected to the limitations of the system developer due to the reduced reusability of hardware-specific components in different hardware contexts.

In CBD, the components interact through interfaces: port-based and operation-based. Eg.: the port-based interfaces comprise of access points for sending/received data of different types between components. The white-box components are readable source code changeable by the programmers. The components' functionality is accessed through the interface, and their internals are visible from outside. The developer has access to their interface and internals. A component is constructed by following the specifications of a component model; it is well established how components interact with each other when they are assembled into a system.

3. PROPOSED SOLUTIONS FOR GPU SPECIfiC SUPPORT: THE PIPE-AND-fiLTER COMPONENT MODELS

This section emphasizes and presents the proposals to overcome the shortcomings identified in SLR [4].

The GPU, being the processing unit equipped with a parallel architecture, cannot function without a CPU. CPU coordinates all the GPU-specific activities (data transfer/execution of GPU functionality). Embedded-board platforms with different GPU architectures exist (2 types):

- discrete (dGPU) - has its own private memory (Condor GR23);
- integrated (iGPU) - on the same chip as the CPU, sharing the same memory (AMD Kabini4).

Embedded-boards with iGPU architectures are the predominant platforms in industry, low - priced, - sized and - energy usage. dGPUs have large physical sizes, incorporate more (GPU) resources and used by systems requiring high performance.

To develop an application, several hierarchical steps are taken: 1) a platform is set with installed driver (contains one or several execution devices, eg. 1-CPU and 2- GPU (iGPU and dGPU) devices); 2) the devices are selected so as to execute the functionality; the commands given by the host (CPU) to the device (iGPU)/kernel are sent using a command queue mechanism; the functionality should be defined before setting the platform; 3) allocation of device memory (buffers), either as input or output for the kernel function; 4) a program to hold the defined kernel is created and compiled (kernel arguments are assigned by using the allocated In/Out buffers) + specify the number of threads for the kernel execution; 5) the kernel is executed and its results are transferred back to the host; 6) the resources (memory buffers, program, context, command queue, kernel) are released.

The pipe-and-filter component models are based on [4]: a) flexible components; b) optimization of the groups of flexible components; c) a support for component communication is designed.

A flexible component, being a white-box with readable and modifiable source code, its functionality is expressed in parallel using the OpenCL syntax. It can be executed either on CPU or GPU. The component does not contain any environment-specific information (it is not binded to a particular processing unit). During system design, the system developer decides on which hardware (CPU or GPU) the flexible components should be allocated onto. In order to be executed on the specified hardware, the required environment information is generated automatically.

The accomplishment of the solution is implemented on two levels: the component- and the system-level. Using the core functionality and the information on the number and data types of the (In and Out) data ports provided by the flexible component, a full component was generated, ready to be executed on the hardware. The resulting generated component contains constructor + behavior function + destructor. At system level, based on the component connections and component-to-hardware allocation, artifacts/adapters were generated where needed. The adapters take data from one component and provide it to the connected component in the appropriate memory location

The use of flexible components having functionalities that may be executed either on CPU or GPU has the next advantages: 1) component-level mechanisms automatically generate environment-specific information that allows the component to be executed on different hardware; 2) system developer has a larger design-space to choose from; 3) the adapters automatically transfer data between components.

3.1. **Flexible Component-based Applications with GPU Capabilities.**

The approach presented in [9] aims at enhancing the flexibility in designing component-based applications with GPU capabilities by introducing flexible components that owe functionalities that may be executed either on CPU or GPU. In this way the developer may focus only on implementing the functionality while having a larger design-space to choose from. Component-level mechanisms automatically generate environment-specific information so that the component may be executed on different hardware. The adapters automatically transfer data between components, taking in consideration the platform specifications. The benefits of employing flexible components refer to: canceling the developer's responsibility of handling the component environment-specific information; providing a higher system feasibility due to a larger design-space; increasing the component communication efficiency by the generated adapters.

Supplementary to the solution proposed in [9], the authors of [10] underline the fact that the flexibility offered by component-based applications complicates the allocation process; it adds additional complexity (due to undecided CPU or GPU execution) and constraints to be considered (CPUs and GPUs properties). Therefore an optimization of the flexibility offered by the component-based embedded systems is necessary. Practically in [10] it is proposed a model to optimize the memory usage, the energy usage and the execution time. The novelty is provided in the formal description of the optimization model, which supports the usage of mixed integer nonlinear programming to compute optimal allocation schemes.

3.2. **Boosting the Resource Utilization.**

In order to mitigate the ever-increasing computational demands of modern embedded systems platforms equipped with GPU processors, an alternative solution is proposed in paper [11] by the boosting of the resource utilization of embedded systems with GPUs. Practically the idea is that the non-critical functions can benefit from the resources of the critical functions during the intervals when they are not used. The method provided in [10] allows the automatic computation of the unused resources in the critical part of the system followed by the distribution of the computed resources to the non-critical parts. The method makes use of a run-time monitoring engine that monitors the critical part of the system to detect any changes in its resource requirements. The considered run-time resources are the system memory and GPU computation threads. By calculating the unused memory based on the actual resource usage by the critical part of the system and by having the information regarding the amount of available memory, the non-critical part of the system can benefit from the available extra resource.

3.3. **Practical Demonstrations of Flexible Components Versatility.**

It was demonstrated that the pipe-and-filter style implemented by Rubus component model is suitable for streaming of events-type of applications and allows an easy mapping between the interaction model and the control specifications required by embedded and real-time systems [4]. The Rubus component consisted of 3 parts: constructor (executed once, before the system execution and allocates the component resource requirements), behavior function (functionality of the component, executed each time when the component is triggered) and destructor (execution when the system is switched off and releases the allocated resources).

In case of the vision system of an underwater robot [2], the hardware platform was an electronic board with a GPU, connected to various sensors (two cameras) and actuators (thrusters) [4]. The continuous flow of data produced by the cameras is processed by the robot's vision system using the GPU. Two camera components were connected to the physical camera sensors. The received data was converted into readable frames and forwarded to the Merge and Enhance component that merged and reduced the noise of the two received frames. The resulting frame was converted to grayscale. Due to the nature of computations (image processing), a set of flexible components were used: Merge And Enhance, Convert Grayscale, Edge Detection, Compress RGB and Compress Grayscale. The frames were of m-elem type, where the maximum size (RGB and grayscale) varied from component to component, depending on the functionality. To evaluate the approach, 4 allocation scenarios were used: 1) all flexible components are allocated to the GPU; 2) all flexible components are allocated to the CPU; 3) and 4) alternate in allocating the flexible components to CPU and GPU. For each scenario, there were used 3 different hardware platforms that contain GPUs. As an output, three produced frames were compared (the input to Object Detection and Logger) from all 12 combinations of scenarios and platforms; all combinations generated identical output frames. For scenario 1 (all flexible components allocated on GPU), for platforms with dGPU architecture, there were generated two CPU-to-GPU adapters and three GPU-to-CPU adapters. When all flexible components were allocated to CPU, there was no need for adapters. For shared virtual memory architectures, there were generated only CPU-to-GPU adapters; there was no need for GPU-to-CPU adapters because all components (regular and flexible) had direct access to the same shared virtual memory system.

Even if up until very recently CBD presented a very reduced attractiveness and solvability in the area of embedded systems with GPU, yet notably progress has been made in the last couple of years. For a compact overview of novelties in the field, Table 1 synthesizes the advancement in the field.

Issue / GPU mechanism	**New solution**	**References**
Flexible and expandable on-board GPU real-time data processing	Multi-threading optimization/ CUDA collaborative strategy	[5]
Prediction of application performance on various GPUs	Theoretical models embedded in MERPSYS	[6]
Parallel Data Distribution Service architecture	Parallelizing the model-related computation/ general-purpose kernels of GPUs/ extending the data distribution service of the object management group	[7]
Improving GPU aging process	Aging-aware workload management technique/ reconfigure the cluster and scale the heaviness of theembedded GPU	[8]
CBD: Platform independent components	Flexible component/executed on GPU or CPU /grouping/communication via adapters/ adapters generated automatically	[4], [9]
CBD: Rubus component model - extended	Implemented with flexible components/ groups/adapters	[4], [10]
CBD: GPU platforms with components application optimization	Method providing different allocation schemes for flexible components/ in function of optimization criteria	[4], [11]

Table 1. Development of embedded systems with GPU

4. CONCLUSIONS

Facilitation of component-based development of embedded systems with GPUs is a need in alternative finding of solutions for high-demand processing of huge data streams resulted from real-time environment sensor-systems. Even if considered until very recently as a limited/abandoned track, CBD contribution proves its high potential in specific contexts.

By starting with a review of state-of-the-art of embedded systems with GPUs, we initially classified the papers in the field - based on categories, mostly using the collected data presented in a doctoral thesis from 2018 [4] which extracted information from internationally recognized databases. Then we emphasized the newest ones, which have not been reviewed to date.

The main focus was however devoted to reviewing solutions in the area of CBD, where, when preparing the component with GPU capability one needs to take into account: on one hand, the component functionality, the required environment information and GPU settings, and on the other hand the separation between component and the system development.

By analyzing the concepts recently introduced of flexible components, then flexible groups and then optimized groups, a feasible solution was showed to emerge. The component communication was facilitated by using specialized artifacts/adapters that automatically transfer data between CPU- and GPU-allocated flexible components.

In view of recent solutions which were already implemented and tested, the component-based GPU support proves its power and advantages, as compared to other solutions analyzed above.

References

[1] K. Bimraw, *Autonomous cars: Past, present and future, a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology*, The 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Colmar, France, July, 2015, pp. 191–198.

[2] C.Ahlberg, L. Asplund, G. Campeanu, F. Ciccozzi, F. Ekstrand, M.Ekstrom, J. Feljan, A.Gustavsson, S. Sentilles, I. Svogor, and E. Segerblad, *The Black Pearl: An Autonomous Underwater Vehicle*, Technical report, Mälardalen University, Sweden, 2013.

[3] G. Keramidas, *Ultra Low Power GPUs for Wearables*, Think Silicon, http://lpgpu.org/wp/wp-content/uploads/2014/09/HiPEAC_wearables.pdf, Jan. 2015.

[4] G. Campeanu, *GPU Support for Component-based Development of Embedded Systems*, Ph. D. Thesis, School of Innovation, Design and Engineering, Mälardalen University Doctoral Dissertation 264, Sweden, 2018.

[5] M. Wang, Z.Q. Zhang, Y. Zhu, Z. P. Dong, Y.Y. Li, *Embedded GPU Implementation of Sensor Correction for On-Board Real-Time Stream Computing of High-Resolution Optical Satellite Imagery*, Journal of Real-Time Image Processing, vol. 15, no. 3, (2018), pp. 565–581.

[6] T. Gajger, P. Czarnul, *Modelling and Simulation of GPU Processing in the Merpsys Environment Scalable Computing-Practice and Experience*, Scalable Computing-Practice and Experience, vol. 19, no. 4, Special Issue: IS, (2018), pp.: 401–422.

[7] W. Kang, J. Kim, *PDDS: Scalable Sensor Data Distribution for Cyber-Physical Systems Using GPGPUs*, IEEE Internet of Things Journal, vol. 5, no. 3, (2018), pp. 2025–2036, 2018.

[8] H. Lee, M. Shafique, M.A. Al Faruque, *Aging-Aware Workload Management on Embedded GPU Under Process Variation*, IEEE Transactions on Computers, vol. 67, no. 7 (2018), pp. 920–933.

[9] G.Campeanu, J. Carlson, and S. Sentille, *Flexible Components for Development of Embedded Systems with GPUs*, 24th Asia-Pacific Software Engineering Conference (2017), p. 219–228.

[10] G. Campeanu, and S. Mubeen, *Scavenging Run-time Resources to Boost Utilization in Component-based Embedded Systems with GPUs*, Intl. J. Adv. Software, vol. 11, no 1 and 2 (2018), p. 159–169.

[11] G. Campeanu, J. Carlson, and S. Sentilles, *Allocation Optimization for Component-based Embedded Systems with GPUs*, The 44th Euromicro Conference on Software Engineering and Advanced Applications, Prague (2018), pp. 101–110.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 Kogălniceanu, Cluj-Napoca, 400084, Romania

Email address: `{mais1577, psis1589}@scs.ubbcluj.ro, avescan@cs.ubbcluj.ro`