

Task-to-Accelerator Mapping for Heterogeneous Systems Using Heuristics

Mohammad Samadi*, Tiago Carvalho, Luís Miguel Pinho

Polytechnic Institute of Porto, Portugal; email: {mmas,tdc,imp}@isep.ipp.pt

Sara Royuela

Barcelona Supercomputing Center, Barcelona, Spain; email: sara.royuela@bsc.es

Abstract

Heterogeneous hardware platforms can be used to improve the performance of computing systems in terms of application response time. They are in high demand, especially with the recent emergence of complex AI applications. Parallel applications can be used on these platforms to utilize most of the hardware system capacity and decrease the execution time. However, predictability is still a notable challenge on these platforms when used in time-critical systems due to the variability in the execution of parallel runtime systems, like OpenMP. Therefore, this paper proposes a new heterogeneous task-to-accelerator mapping approach using heuristics to improve the performance and predictability of OpenMP applications running on heterogeneous systems.

Keywords: parallel programming, OpenMP, heterogeneous systems, task-to-accelerator mapping, heuristics.

1 Introduction

Nowadays, graphics processing units (GPUs) are generally applied to accelerate diverse intensive computations, turning into general-purpose computing on GPUs (GPGPUs). In the Internet of Things (IoT) and Cyber-Physical Systems (CPS) domains, they can be applied to perform various tasks, such as pedestrian detection, eye tracking, and navigation [1, 2]. Besides, the performance of heterogeneous platforms can be improved using parallel programming models like OpenMP [3,4,5]. The timing and response time of programs running on the GPU should also be evaluated, similar to those running on the CPU. Therefore, an efficient task scheduling policy is necessary on these systems, especially for predictable OpenMP applications.

Contrary to traditional worst-case response time (WCRT) analyses for CPUs, analyzing heterogeneous systems is challenging because the number of processing elements (i.e., CPUs and GPUs) is high. Hence, mapping computation to processing elements becomes even more critical. Furthermore, replacement policies and exact cache architectures are not evident in modern GPUs as they offer a black-box model, as well as the effect of

cache and memory is not considered while using long-delay operations (e.g., hundreds of cycles for an L1 cache miss) [6]. Consequently, it is essential to perform an efficient mapping process on the GPU side to overcome the problems that have arisen.

Predictability is one of the main requirements in time-critical systems (e.g., CPS). It can be achieved by reducing the variability of application response times. Accordingly, mapping tasks to processing elements (e.g., accelerator) in parallel applications (e.g., OpenMP) should be performed efficiently to improve the load-balancing of queues (in multi-queue systems like LLVM) and the work-conserving nature of the mapping. In this way, response time variability can be reduced and, consequently, system predictability can be improved.

This paper proposes an efficient task-to-accelerator mapping for multi-GPU systems considering OpenMP applications. In OpenMP, device tasks entail a CPU segment, which is scheduled based on the readiness of the task and the availability of host resources, and a GPU segment, which uses GPU resources to schedule and run the offloaded kernel. The proposed method focuses on the GPU segment, once these tasks are available to start executing on the GPU. The method first places the GPU task into a GPU global queue. Then, one of the local queues, individually dedicated to each GPU device, is selected for the task. Finally, the task is dispatched to the GPU to be executed. The selection policy of tasks from the global queue, the selection of the local queues, and the selection of tasks from local queues to run with the GPUs are determined using various heuristic algorithms. The main objective is to improve the predictability of real-time parallel and heterogeneous applications by reducing the response time variability (and to improve schedulability by minimizing WCRT) of the parallel runtime, focusing on the scheduling and mapping processes of the parallel runtime.

2 Related work

This section describes relevant methods presented when this work was submitted in the scope of task scheduling in heterogeneous systems.

Elliott et al. [7] presented GPUSync, a framework to improve parallel management of GPU-based computing resources. The

*PhD Candidate, Universitat Politècnica de Catalunya, Barcelona, Spain

framework manages the GPU budget and configures the system under different scheduling policies and prioritization schemes. Moreover, GPU resources are managed using job-level-static-priority (JLSP) schedulers, such as rate-monotonic (RM) and EDF. CPU and GPU allocations are performed under partitioned, clustered, and global basis models. This work does not enhance the load-balancing of FIFO queues, nor does it consider efficient algorithms to increase the degree of parallelism and minimize response time variability and WCRT.

Capodiceci et al. [8] presented a prototype of a deadline-based scheduler with preemption policy for GPU tasks in a virtualized environment. This work uses a constant bandwidth server to provide task isolation in applications using the NVIDIA software stack. It undergoes a pre-existing table-based method for event-driven GPU command submission, uses real-time extensions for CUDA and OpenGL, and prototypes a software scheduler running as a privileged guest on the hypervisor. This approach, however, does not exploit execution-related information (e.g., task execution time) that could reduce response times efficiently.

Kim et al. [9] introduced a server-based approach to solve two problems related to GPU management, including busy waiting and long priority inversion. This approach considers multi-core platforms equipped with a single GPGPU through synchronous GPU access. GPU resources are shared between multiple tasks based on a non-preemptive sequential process, using fixed-priority partitioned scheduling and a single lock for accessing the GPU. Despite its benefits, this method considers a single GPGPU device, so tasks cannot be executed simultaneously using multiple accelerators to reduce application response time.

Tsog et al. [10] studied static allocation approaches through a fork-join model to reduce the bottleneck condition imposed by the overuse of GPUs. Accordingly, it performs offline static allocation of parallel computing segments on the CPU or GPU to evaluate the effect of balanced CPU and GPU utilization in real-time systems. In addition, the schedulability of tasks is improved using different heuristics, such as non-greedy resource allocation. Similar to the previous work, this method considers single-GPU systems. Furthermore, although techniques like support vector machine (SVM) are used for task scheduling, they are not considered in the evaluations because they entail a high overhead.

Olmedo et al. [11] introduced a hierarchical arbitration technique for GPU scheduling on NVIDIA platforms and considered the application scheduler at the top level of the scheduling hierarchy based on CUDA applications. Streams are pushed into a First In, First Out (FIFO) queue, as these blocks are distributed between the streaming multiprocessor (SM) cores using a block scheduler. Although this work has inferred various heuristics applied in the hierarchical scheduling scheme adopted with the NVIDIA GPU subsystem, it does not apply stream reordering

techniques, limiting its benefits in GPU utilization and system predictability.

Amert et al. [12] presented the Time-Isolated Multicore Execution With Accelerator Locking (TimeWall) framework to provide time partitioning on multi-core + accelerator platforms. This framework consists of three elements: a table-driven scheduler for allocating time to partitions, in-partition global EDF schedulers, and a special locking protocol for arranging the accelerator accesses based on the time-slice boundaries. This method expects the CPUs to wait until GPU operations in the accelerator are completed (synchronous offloading in OpenMP), which can considerably increase task execution time and application response time.

Overall, the main limitations exposed with the existing methods are that most of them (i) consider a suspension-oblivious analysis, where the CPU is not free while the offloaded part is executing in the GPU, so utilization loss and response time of the application may increase, (ii) focus on single-GPU systems, limiting their applicability in modern clusters and advanced (domain or zonal) E/E architectures for autonomous vehicles, and (iii) do not employ efficient advanced algorithms for selecting appropriate queues and tasks to optimize task scheduling based on its characteristics (e.g., execution time) and to reduce application response time. In addition, some do not consider multiple queues for GPU devices, which can increase contention on the single queue and reduce the system's predictability. Accordingly, this paper addresses problems by proposing a heterogeneous mapping through the multi-queue system model and a suspension-aware analysis for predictable parallel applications using several heuristics with different characteristics.

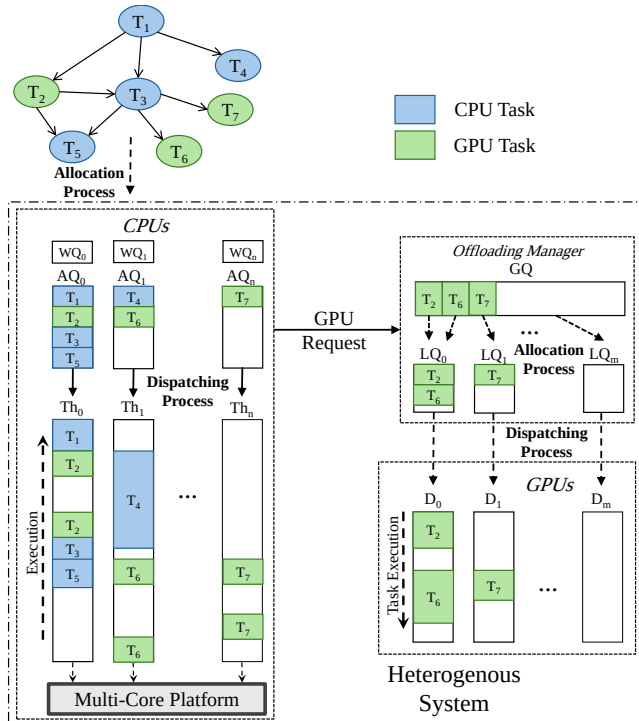
3 System model

This work assumes a multi-core platform with one or multiple GPU devices suitable for modern time-critical systems (e.g., CPSs). The system is supposed to have two types of tasks: CPU tasks and GPU tasks. CPU tasks are executed using only CPU cores, while GPU tasks are executed using both CPU and GPU (called host and device, respectively) resources. GPU tasks are dispatched to GPU(s) sequentially and non-preemptively. Furthermore, memory copy time values are task-specific.

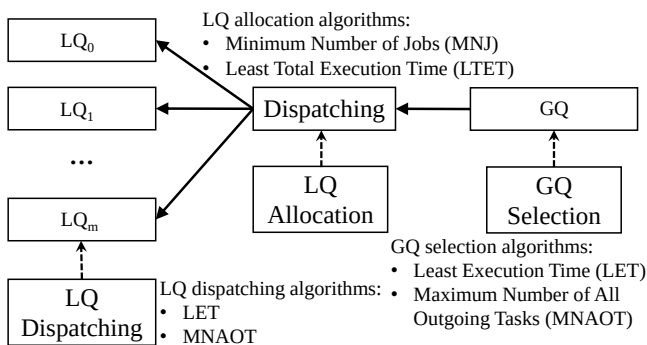
A task τ_i is characterized as $\tau_i = (T_i, D_i, C_i, G_i, \xi_i, \sigma_i)$, where T_i is the period of each job of the task, D_i is the relative deadline, C_i is the sum of the execution time of CPU execution segments, G_i is the execution time of GPU execution segment, ξ_i is the memory copy time from host to device, and σ_i is the memory copy time from device to host. Tasks are managed using a directed acyclic graph (DAG), where they may involve data dependencies on each other. In addition, the deadline is used in the system model to evaluate the schedulability of the system based on the obtained response time and the relative deadline.

4 Proposed approach

Figure 1 shows the proposed mapping methodology, including an overview of the system and the algorithms applied during the offloading process. As illustrated in Figure 1a, each CPU has an allocation queue AQ to execute CPU tasks (in blue) and the CPU execution segments of GPU tasks (in green). In addition, it has a waiting queue WQ to execute the second CPU segment of GPU tasks after the GPU segment is completed on the GPU. The task execution order in each GPU device is from top to bottom. For example, T_2 is executed before T_6 in the first device (i.e., D_0).



(a) System overview



(b) Offloading algorithms

Figure 1: Methodology of the proposed task-to-accelerator mapping.

The offloading process is carried out with an offloading manager that contains a global memory queue GQ and multiple local memory queues LQs dedicated to the GPUs. Since LQ

includes a limited capacity, it is assumed that the GQ is shared between all LQs . GPU execution segments are first allocated to the GQ with CPUs (through the offloading manager) and then propagated (and allocated) to LQs using the offloading manager. Note that multiple tasks can be executed on different GPUs concurrently. The process is performed asynchronously, so a suspension-aware analysis is performed. Therefore, the CPU assigned to a GPU task is not suspended while the GPU segment runs, so that it can run other tasks. Furthermore, each GPU includes an independent lock controlled by the offloading manager, in which access to the GPU is locked while a kernel is running. Figure 1b shows three main processes performed using the offloading manager: GQ selection, allocation, and dispatching.

GQ selection: As mentioned above, each *LQ* has a limited capacity, where the capacity is the size of the queue in number of tasks, so GPU execution segments first are allocated to the *GQ* and then allocated to one of the *LQs*. When the *GQ* is not empty and at least one of the *LQs* has blank space to get a new task, a GPU segment is allocated to an *LQ*. Multiple tasks can exist in the *GQ* simultaneously, so one of the tasks is selected for each allocation using the GQ selection process. Accordingly, two algorithms are suggested for this process: Least Execution Time (LET) and Maximum Number of All Outgoing Tasks (MNAOT). LET selects the task with the shortest (GPU segment) execution time, and MNAOT selects the task with the most significant number of outgoing nodes in the Task Dependency Graph. They can improve the work-conserving nature of the mapping process by executing thin tasks and tasks with more successors first, respectively, and increasing the number of ready tasks in the system.

LQ allocation: If the number of LQ s that include a blank space is more than one when allocating a task from the GQ to the LQ , one of the LQ s is selected. LQ allocation selects the GPU to map the task given by the GQ selection process. For this purpose, two LQ allocation algorithms are suggested: Minimum Number of Jobs (MNJ) and Least Total Execution Time (LTET). MNJ selects the LQ with the fewest tasks, and LTET selects the LQ with the smallest total execution time ¹. These algorithms can improve the load-balancing of queues based on the number of tasks and execution time, respectively, to speed up the task execution process and reduce application response time and variability.

LQ dispatching: If a GPU device is idle and its LQ is not empty, one of the tasks in the LQ , if more than one, should be selected. Two LQ dispatching algorithms are suggested to select the most appropriate task among the existing ones. As the process of selecting a task is the same for the LQ s and the GQ , the LQ dispatching algorithms are the same as those used in the GQ selection process. The mapping process on the CPU side is performed using heuristic-based task-to-thread mapping algorithms presented in a previous work [13].

¹The total execution time of each *LQ* should be updated before this process.

Since two heuristics are used in each process, eight combinations of the algorithms can be considered for the mapping process. Therefore, the goal is to identify the best heuristics using evaluation results.

Task execution time can be determined using a randomized process in random DAGs and profiling tools (e.g., Extrae [14]) in benchmarks (i.e., synthetic and real-world applications). As the proposed heterogeneous mapping approach is presented for time-critical systems implemented with OpenMP, the worst-case execution time (WCET) is used to calculate task execution time based on multiple executions.

Furthermore, the application response time can be determined using two timestamps, one before and one after the main code block. After running the application several times, response times can be analyzed using a boxplot to determine response time variability and WCRT. In this way, the system's predictability and schedulability can be well studied.

5 Conclusions and future work

This paper introduces the preliminary design of a heterogeneous task-to-accelerator mapping method to reduce application response time and variability, the main requirements in real-time parallel applications. Using the OpenMP accelerator model, GPU tasks are divided into a CPU segment, which schedules the task into an offloading manager once it is ready (i.e., all its input dependencies, if any, are resolved), and a GPU segment, which includes the kernel offloaded and executed in the GPU. The device-side methodology consists of a global queue and several local queues, each dedicated to a GPU. An offloading manager controls the whole process, where GPU segments are allocated to the global queue, then allocated to local queues, and finally executed using GPUs. The methods of selecting the most appropriate tasks from the global and local queues and choosing the most suitable local queues are performed using heuristic algorithms.

The main objectives of this work are to improve queue load-balancing by selecting suitable queues for tasks and to improve the work-conserving nature of the mapping process by selecting appropriate tasks from the queues. Different heuristic algorithms are applied at each stage of the mapping process to investigate which ones can achieve these objectives. Subsequently, response time variability can be reduced to improve the predictability of parallel applications, which is a fundamental requirement in time-critical systems.

A future work will use the evaluation results of the proposed method to refine and extend the work depicted in this paper by suggesting enhancements and new algorithms for GPU scheduling. Furthermore, the performance of the work will be evaluated under multiple applications with different configurations on supercomputers like the MareNostrum 5 accelerated partition (ACC)².

²<https://www.bsc.es/supportkc/docs/MareNostrum5/overview/>

6 Acknowledgment

The authors thank the anonymous reviewers for their helpful comments and feedback. This work was supported by the KDT Joint Undertaking (JU) program under grant agreement No 101140216 (project MATISSE), co-funded by the European Union's Horizon Europe research and innovation programme and the Portuguese Fundação para a Ciência e a Tecnologia (FCT), and the HiPART project, with reference PID2023-148117NA-I00, financed by MICIU/AEI/10.13039/501100011033 and FEDER, UE.

References

- [1] Z. He, Y. Sun, B. Wang, S. Li, and B. Zhang, "Cpu-gpu heterogeneous computation offloading and resource allocation scheme for industrial internet of things," *IEEE Internet of Things Journal*, vol. 11, no. 6, pp. 11152–11164, 2023.
- [2] N. R. Srivastava and V. Mittal, "Applications of gpus for signal processing algorithms: A case study on design choices for cyber-physical systems," in *Emergence of Cyber Physical System and IoT in Smart Automation and Robotics: Computer Engineering in Automation*, pp. 147–159, Springer, 2021.
- [3] D. Sevim, B. Bilgin, and I. Akturk, "Evaluating performance and energy efficiency of parallel programming models in heterogeneous computing systems," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 309–319, IEEE, 2024.
- [4] C. Yu, S. Royuela Alcázar, and E. Quiñones Moreno, "Enhancing heterogeneous computing through openmp and gpu graph," in *The 53rd International Conference on Parallel Processing (ICPP'24): August 12th-15th, 2024, Gotland, Sweden: main conference proceedings*, pp. 534–543, Association for Computing Machinery (ACM), 2024.
- [5] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," in *Euro-Par 2009 Parallel Processing*, pp. 863–874, Springer, 2009.
- [6] D. Ernst, G. Hager, M. Knorr, G. Wellein, and M. Holzer, "Opening the black box: Performance estimation during code generation for gpu," in *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 22–32, IEEE, 2021.
- [7] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *2013 IEEE 34th Real-Time Systems Symposium*, pp. 33–44, IEEE, 2013.
- [8] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for gpu with preemption support," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, pp. 119–130, IEEE, 2018.

- [9] H. Kim, P. Patel, S. Wang, and R. R. Rajkumar, "A server-based approach for predictable gpu access with improved analysis," *Journal of Systems Architecture*, vol. 88, pp. 97–109, 2018.
- [10] N. Tsog, M. Becker, F. Bruhn, M. Behnam, and M. Sjödin, "Static allocation of parallel tasks to improve schedulability in cpu-gpu heterogeneous real-time systems," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, pp. 4516–4522, IEEE, 2019.
- [11] I. S. Olmedo, N. Capodiecici, J. L. Martinez, A. Marongiu, and M. Bertogna, "Dissecting the cuda scheduling hierarchy: a performance and predictability perspective," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 213–225, IEEE, 2020.
- [12] T. Amert, Z. Tong, S. Voronov, J. Bakita, F. D. Smith, and J. H. Anderson, "Timewall: Enabling time partitioning for real-time multicore+ accelerator platforms," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, pp. 455–468, IEEE, 2021.
- [13] M. Samadi, S. Royuela, L. M. Pinho, T. Carvalho, and E. Quiñones, "Time-predictable task-to-thread mapping in multi-core processors," *Journal of Systems Architecture*, vol. 148, p. 103068, 2024.
- [14] "Extrae, bsc performance tools," 2024.