The handle http://hdl.handle.net/1887/21017 holds various files of this Leiden University dissertation

**Author:** Balevic, Ana  
**Title:** Exploiting multi-level parallelism in streaming applications for heterogeneous platforms with GPUs  
**Issue Date:** 2013-06-26
Conclusion

Modern heterogeneous platforms support not only the traditional types of parallelism, such as ILP, vector, data, task, and pipeline parallelism, but also provide platform-level and component-level parallelism, i.e. it is possible to exploit not only a number of different architectural components on the platform, but also different types of parallelism within each component. To exploit the rich parallelism opportunities offered by heterogeneous platforms, we believe that having a multi-level program model is a prerequisite. This thesis makes a first but important step in this direction by introducing a hierarchical internal representation into the polyhedral framework, and a novel method to derive this representation from the standard polyhedral program model and then transform it into a multi-level program (MLP) featuring different forms of parallelism. As such, this thesis opens doors for future research on highly efficient tailor-made parallel program generation and auto-tuning for the next generations of multi-level heterogeneous platforms with diverse accelerators.

7.1 Summary of Work and Contributions

In this thesis, we presented a novel methodology for transformation and more efficient mapping of streaming applications onto heterogeneous platforms with GPU accelerators. Compared to homogeneous multicore platforms, heterogeneous platforms pose not only additional, but also more complex programming challenges to application designers. On the one hand, heterogeneous platforms offer multiple levels and multiple types of parallelism. On the other hand, the sheer range of parallelization opportunities makes it harder to implement and map parallel applications onto such platforms. The concepts and techniques presented in this thesis are designed to exploit task, data, and pipeline parallelism on heterogeneous platforms with massively
data-parallel accelerators (e.g., GPUs).

On the conceptual level, our work revolves around HiPRDG, the hierarchical intermediate program representation that we have introduced into the polyhedral model (Chapter 4). The intermediate representation of an application in the form of the HiPRDG facilitates automatic derivation of structured multi-level programs featuring different types of parallelism at each level. For example, by first transforming the application into HiPRDG and then generating the parallel code, it is possible to obtain a top-level PPN with coarse-grain tasks which can be executed in a pipeline fashion. Further, it is possible to discover fine-grain data parallelism at the next level of the HiPRDG, and automatically generate task and data parallel kernels using the KPN2GPU tool which implements the parallelization approach described in Chapter 3. Finally, using the offloading principles described in Chapter 5, task and data parallel kernels obtained by the KPN2GPU can be offloaded for computation on a GPU. Moreover, the stream buffer design enables streaming of data to the GPU, and overlapping of computation and communication. The multi-level parallel program obtained in this manner can execute in asynchronous data-driven manner on a heterogeneous computing platform.

Contributions  The main contributions of this thesis are the following:

(I) A method for discovery and exploitation of data and task parallelism in PPN representation for mapping onto massively parallel accelerators, such as GPUs.

(II) A novel hierarchical program representation in the polyhedral model and a method for hybrid generation of multi-level (parallel) programs with token granularities adjustable at each level.

(III) A novel solution for efficient stream buffer design for model-based overlapping of communication and computation on heterogeneous platforms with discrete accelerators.

Contribution I  We presented a systematic approach for identification, extraction and modeling of fine grain data parallelism in the polyhedral process network (PPN) specification in Chapter 3. We leverage a state of the art method to identify data parallelism and introduce a data parallel view (DPV) onto PPNs. The DPV is an extension of the PPN model with concepts required for capturing data parallelism within PPN nodes and channels. As an extension to the Compaaan compiler framework, we implemented the KPN2GPU tool. The KPN2GPU tool detects data parallelism in the program specification, builds a DPV model which captures data and task parallelism, and provides a CUDA compiler back end for automatic kernel and host code generation. In addition, we provide a model-based method for exploiting the
7.1. SUMMARY OF WORK AND CONTRIBUTIONS

recently introduced hardware/software support for concurrent kernel execution [105] on modern compute-capable GPUs. This makes it possible not only to exploit data parallelism, but also to exploit task-parallel execution for better GPU utilization.

**Contribution II** In Chapter 4, we introduced a novel hierarchical program representation in the polyhedral model called *Hierarchical Polyhedral Reduced Graph* (HiPRDG). HiPRDG is an intermediate program model based on polyhedral representation that spans multiple levels. At the top-level, there is a standard polyhedral reduced dependence graph (PRDG) which can be used as a starting point for model traversal. Zooming in into each node of the top-level PRDG reveals the definition of the node’s statement. The definition is captured in form of a PRDG and stored within a node at the lower level of hierarchy. By providing a modular representation that is structured into multiple levels of hierarchy, the HiPRDG model is well suited to be used as a basis for hybrid generation of multi-level parallel programs. To split a standard program model into a multi-level program model we introduced a technique called *slicing*. We first decide on the number of levels and desired token granularity for each level, and then slice the program model at the loop levels (depths) corresponding to the desired token granularity at each level. Finally, we showed how to generate a coarse-grain PPN of a streaming application for exploiting task and pipeline parallelism at the platform-level. Further, we used the notion of hierarchy in the model to generate data-parallel bodies for selected tasks for leveraging massive parallelism within the accelerator. The result of the derivation is a hybrid (task, data, and pipeline parallel) multi-level parallel program which can be mapped to a heterogeneous platform with a data parallel accelerator in a more efficient manner than the initial PPN. Case studies in Chapter 6 showed that by leveraging hybrid multi-level parallelization and accelerator offloading a 4× performance improvement can be achieved compared to the performance of the initial PPN. In addition, the techniques presented in Chapter 4 implicitly enable adjustment of token granularity, thus also addressing one of the key challenges for mapping streaming applications onto heterogeneous platform. Adjustment of the token granularity alone has been demonstrated to improve performance of the task-parallel execution by 27% on a multi-core CPU, with even more profound impact on the GPU execution.

**Contribution III** In Chapter 5, we extended the baseline PPN mapping model with concepts required for offloading computation of PPN nodes on a GPU. We also introduced the communication protocol which significantly reduces the amount of data movement involved with PPN implementation on shared-memory architectures (such as multicore CPUs), thus reducing the pressure on the memory subsystem. As the main part of Chapter 5, we introduced an efficient *stream buffer* design [17] for communication between a pair of producer-consumer processes executing on heteroge-
neous devices. Using the stream buffer design for host-accelerator communication, data transfers to/from GPU and computation can be pipelined on high-end GPUs equipped with two DMA engines. For data-intensive streaming applications, the time spent in copying the data from host memory to GPU memory, and back, can outweigh the benefits of GPU parallelization. However, the experiments in Chapter 5 indicate that a significant improvement of the overall GPU execution time is possible using the stream buffer design. The mechanisms described in this chapter provide a model-based method to achieve data-driven asynchronous execution with overlapping of communication and computations.

7.2 Prerequisites for Further Progress

The research presented in this thesis creates a bridge between distributed memory parallelization with PPNs and shared memory parallelization techniques for exploiting data parallelism. As such, it opens numerous possibilities for further research, but it also poses a set of novel requirements that an experimentation framework would need to satisfy. Combining transformations implemented in different, even closely related polyhedral frameworks, such as [1, 2, 81] to name a few, is nowadays a technically challenging and time consuming task standing in the way of research progress. First and foremost, further research on hybrid, multi-level parallelization for heterogeneous platforms requires a standardized polyhedral intermediate representation for data exchange between polyhedral compiler frameworks. The polyhedral compiler research community would also greatly benefit from a unified, modular and extensible open source framework based on standardized interfaces, which could be used as a basis for further research on transformations, experimentation and reporting scientific progress.

7.3 Directions for Future Work

The research work presented in previous chapters creates a basis for multi-level hierarchical representation in the polyhedral model and opens the doors for further research and experimentation in numerous directions. Some of the ideas for further work are the following:

Parallelism affinity An attractive topic for further research is guided derivation of parallel, multi-level programs. For this purpose, we propose to use the notion of parallelism affinity to describe different platform components. With the concept of parallelism affinity, we can specify what is the preferred type of the parallelism for the given architectural component, e.g., task parallelism for a multicore CPU and
7.3. DIRECTIONS FOR FUTURE WORK

data parallelism for a GPU. Moreover, it would be easy to extend the parallelism affinity concept to include information on the number, types and characteristics of parallelism levels for each component. Similarly, we can use the notion of communication affinity to describe the preferred type and characteristics of the communication on the platform, e.g. whether it exposes shared memory or distributed memory architecture at some level. The combination of multi-level parallelism affinity and communication affinity specifications of heterogeneous platforms can be used to guide automatic derivation, mapping, and performance tuning of architecture-specific program modules from our hierarchical intermediate representation leading to efficient, custom-tailored generation of parallel code.

Balancing pipeline and data parallelism Data parallelism and pipeline parallelism are opposing concepts in the sense that opting for maximal data parallelism in a streaming application leaves no pipeline parallelism and vice versa. We can generate applications that exploit both data and pipeline parallelism by slicing the program model into multiple levels, and then transforming the components in one level for data or for pipeline parallelism. The size of the tokens passed between the levels determines the granularity of tasks at each level, and implicitly the balance between top-level pipeline parallelism and data parallelism. It is possible to perform empirical search on program variants to select a good combination of parameters for the program execution. Another interesting topic would be the design of an analytical model for determining the optimal balance of data and pipeline parallelism for the given platform and application combination.

Data space transformations Another promising topic for further research are data space transformations in the polyhedral model for enabling the construction and exchange of arbitrary composite tokens between program modules. The extension and further refinement of support for composite tokens could possibly increase the general applicability of PPNs for processing not only streaming applications, but also for acceleration of iterative numerical applications that process large data sets (Big Data), e.g., by partitioning the data set processed at each simulation step into tokens and streaming these tokens to and from the GPU.

Overall, our approach is currently targeted towards parallelization of computationally intensive streaming applications and their mapping onto heterogeneous platforms. However, the proposed future work directions will improve both the application coverage and the performance of automatic parallelization and mapping, making our model-based approach a promising alternative to manual parallelization.
7.3. DIRECTIONS FOR FUTURE WORK