Simulation based Analysis of Memory Access Conflicts for Heterogeneous Multi-Core Platforms

Jens Brandenburg and Benno Stabernack
Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute
Video Coding & Analytics Department, Embedded Systems Group
Einsteinufer 37, 10587 Berlin, Germany
{jens.brandenburg,benno.stabernack}@hhi.fraunhofer.de

***Abstract*—Besides aspects of HW/SW partitioning, resource allocation and mapping, also the optimization of the memory subsystem plays a crucial role during the complex HW/SW co-design and co-optimization process. Especially for memory bound applications, like state of the art video codecs, the memory subsystem has become one of the bottlenecks limiting the performance gains from parallelization and HW accelerated approaches. Memory access conflicts, due to the concurrent access to a shared memory location, are a major source of this bottleneck. To develop counter strategies and to optimize the design, an in-depth analysis of all memory access conflicts is necessary and required. In order to provide this analysis, we propose a flexible tracing and profiling methodology, which provides a timing-accurate memory access conflict analysis for SystemC-based platform simulation models. In a case study this memory access conflict analysis is performed for a heterogeneous platform running a parallel high efficiency video coding (HEVC) intra encoder application. This analysis leads to an optimized design, which reduces the number of memory access conflicts and shows significant performance gains for the target video encoder application.**

I. Introduction

Nowadays the trend in video coding applications goes towards more complex algorithms and more data to process. For example, the high efficiency video coding (HEVC) standard supports at its highest level a video resolution of 8K (8192×4320) with 120 frames per second, which results in a raw data input rate of approx. 6TB [1] per second. A typical approach to speed up such an application is the parallelization of the algorithm and/or the acceleration of the most performance requiring parts of the algorithm by dedicated HW components. According to Amdahl's law, the theoretical improvement of such an approach can be calculated based on the relationship between improved and not improved portions of the algorithm but unfortunately most systems did not show the expected behaviour. One aspect limiting the expected parallelization improvements can be found in the shared memory, where the growing number of memory access conflicts increases the average memory access latency for all components. To avoid or reduce these memory access conflicts, an analysis of the developed system and its memory behaviour is an important requirement.

In order to aid the developer with this optimization task, we present a performance and memory access analysis tool based on the tracing and profiling of SystemC-based simulation models. This tool can be used in the early design phase of the HW/SW partitioning, when only a platform simulation model is available, as well as in later design phases, like the HW/SW co-optimization and refinement. The tool collects different low level trace data events and combines these data to analyse the memory access characteristics, in particular the memory access conflicts, of an observed platform simulation model. A demonstration of the tool's capabilities is given for a video coding use case where the analysis of the memory access conflicts leads to an optimized design, which improves the overall application performance.

[1] Assuming a 4:2:0 color coding scheme with a bit-depth of 8Bits per pixel.

II. Related Work

Memory access analysis approaches can be classified into intrusive and non-intrusive approaches. Intrusive approaches collect the trace data by executing a modified version of the analyzed software. This modification changes the execution of the software, which affects the timing accuracy of the collected trace data. Contrastingly, non-intrusive approaches collect the trace data without modifying the software, typically by the usage of modified hardware components. For this use, non-intrusive approaches mostly utilize simulation techniques, which leads to high execution times, especially in comparison to intrusive approaches, where the instrumented software often can be executed directly on the host processor.

However, also intrusive approaches may use simulation techniques to model some part of the target architecture as proposed for example in [1]. In this approach a cache simulator is added to the Valgrind framework [2] to model and analyze cache related metrics, e.g. number of cache accesses and cache misses for a generic cache implementation. In this context the Valgrind framework itself provides the infrastructure for the instrumentation of the observed software, based on a dynamic binary analysis (DBA) at runtime. A similar trace data collection approach is presented in [3], which implements a Valgrind based profiling tool to recognize memory access patterns in computational kernels. Therefore all memory accesses are traced, and then the tool checks if the access pattern of multiple accesses can be matched to a set of classifiers with predefined memory access characteristics.

Hardware component instrumentation based approaches can be found in [4], [5] and [6]. All three approaches are using

the SoCLib [7] system simulation component library to collect the trace data, either by adding special hardware monitoring components [6] or by modifying existing hardware components [4] and [5]. Hereby, the analysis in [6] is focussed on the latency of communication channels and their basic data structures. In comparison, the analysis in [4] is focussed on tracing memory accesses to detect and identify potential data races in a parallel algorithm. An extension of this data race detection tool is proposed in [5]. This tool additionally traces the memory access latencies to identify portions in the source code where a certain contention pattern arises.

A more generic instrumentation approach for SystemC simulation models is presented in [8]. This approach is based on the aspect oriented programming paradigm to separate the implementation of the trace data code from the functional code of the platform simulation model. In a process called aspect weaving the trace data code is inserted by a pre-processor into the platform simulation model to trace different events from the implementation of the hardware components. By using this instrumentation approach, [9] automatically traces all transaction level modeling (TLM) connections between different SystemC modules, to verify different pre-defined timing conditions for TLM based data exchanges.

In this paper we present an extension of our non-intrusive tracing and profiling tool [10]. This extension enables the collection of memory access characteristics and provides a memory access conflict analysis. Due to the simulation based approach, the tool can perform a timing-accurate analysis of all memory accesses. Moreover, the tool is capable of analyzing heterogeneous platforms and can also incorporate memory access data from HW only components into the conflict analysis. In comparison to the other discussed SystemC-based approaches, the tool does not require the instrumentation of the platform simulation model to collect the necessary trace data. Instead, an automatic platform design analysis is performed to configure the trace data collection module and to provide an easy adaption mechanism for new platform simulation models.

The remainder of this paper is organized as follows. In Section III the simulation based memory access conflict analysis methodology is described. Based on this analysis methodology, a case study for a heterogeneous and parallel HEVC intra video encoder is shown in Section IV. Results for the suggested optimization of the memory layout are discussed in Section V and finally a concluding summary is provided in Section VI.

III. Memory Access Conflict Analysis

A. Non-intrusive Trace Data Collection

Given a software based executable of the platform simulation model, a simple way to collect the trace data is to instrument this platform simulation model with a set of trace primitives, which have to be included at appropriate places. A downside of this approach is the lack of flexibility, because the instrumentation code has to be maintained, whenever the implementation is changed. Furthermore introducing new components or switching to a new platform simulation model requires from the developer the additional effort to add this instrumentation code. In case of SystemC, each platform simulation model is composed of basic standard components, e.g. SystemC modules, ports and signals, which enables a more flexible way to collect the required trace data by instrumenting these standard components directly.

In SystemC different platform components are either connected by a set of signals, in case of bit accurate designs, or by a TLM socket, in case of a transaction level model (TLM) design. Based on this observation it is sufficient to instrument the SystemC signal class and the TLM socket class to trace a memory access. For this instrumentation we choose the signal update function in the common SystemC signal base class and the transaction start/end functions in the common TLM socket base class. By adding callbacks from these functions to our trace data collection module, all signal changes and all TLM access within each SystemC-based simulation model can be traced.

Of course for complex designs with hundreds or more signals, it is not feasible and also not required to trace all changes. To reduce the set of traced events, a set of relevant events within the platform simulation model has to be identified. This identification is based on an automatic design analysis, which is performed at runtime, after the platform simulation model has been created. After the SystemC elaboration phase, all instantiated modules and signals are inspected by the trace data collection module and a map of design elements and their interconnections is created. Additionally the trace data collection module analyses static C++ design elements, to enable also the tracing of changes of SystemC module member variables. This information is relevant to observe also static design elements, like a program counter or a register set in an instruction set simulator, which is required to correlate software and hardware events in the observed system. Based on this automatically generated map of SystemC modules and their interconnections only the tracing of memory access signals is enabled inside the SystemC simulation kernel, without requiring to instrument or modify the used platform simulation model.

B. Memory Access Conflict Detection

To create memory access traces, it is necessary to collect information about the accessed memory address, the accessed size and the duration of this access. But in case the simulation model utilizes signals to carry out a memory access, this information has to be extracted from a set of multiple signals, each playing a specific role during the memory access process. Based on a meta-data specification of the signal roles for different data exchange protocols, the traced signal changes are combined to provide the required memory access information. An exemplary signal role configuration for the virtual component interface (VCI) standard is shown in Fig. 1.

Given the combined memory access traces from multiple platform components, e.g. processing elements and/or HW accelerators, the detection of memory access conflicts can be realized. In this case a conflict is detected, whenever two different memory accesses from different sources temporally

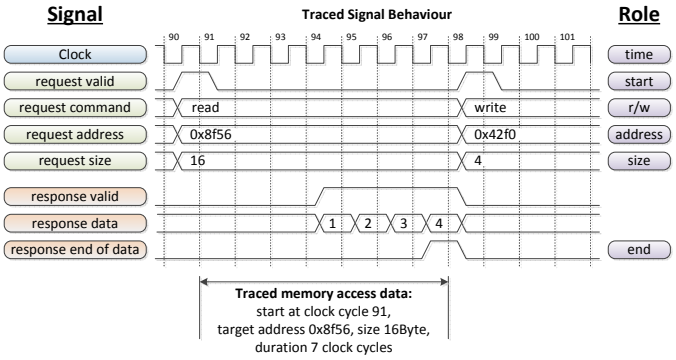

Fig. 1. Combining the traced signal changes to collect different memory access information.

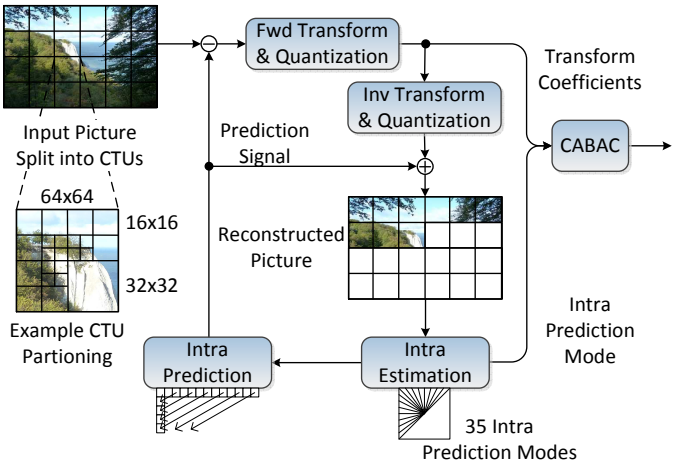

Fig. 2. HEVC intra encoder model.

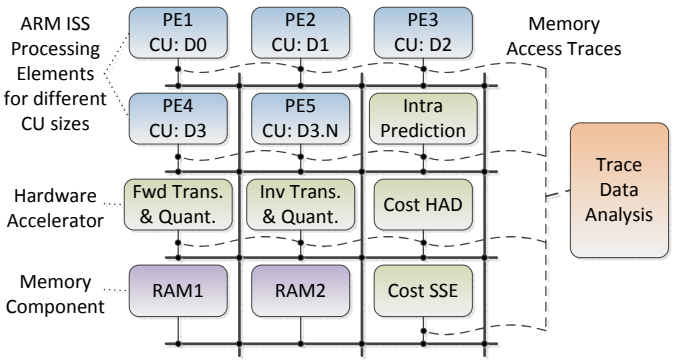

Fig. 3. Instrumented platform simulation model.

overlap and the execution of at least one of these accesses is delayed. The implemented conflict detection process is based on a windowed approach, where each memory access is stored in a FIFO buffer and removed, when the end time of the access is below the start time of a newly detected access. Therefore only the small set of memory accesses in the FIFO has to be compared with each other to detect all memory access conflicts during a simulation model run. For the evaluation of these conflicts each conflict is either attributed to different data structures by its traced memory addresses or to different executed program instructions by tracing the current program counter.

IV. Case Study

A. High Efficiency Video Coding

The HEVC standard is an example of a block-based hybrid video coding scheme. In this scheme the input frame is split into a set of smaller blocks called coding units (CUs), and each CU is encoded by a combination of prediction, transform and entropy coding algorithms, as shown in Fig. 2. In case of the HEVC main intra profile the prediction uses only reconstructed samples from the current frame facilitating the spatial correlation between adjacent samples.

For a high video compression efficiency it is beneficial to adapt the CU size (block size) to the input video data content. Therefore the encoder supports different CU sizes ranging from 64×64 to 8×8 samples. To adapt the CU size to the input video data content, the encoder can decide to use either the whole CU for the compression algorithm or to split the CU recursively into four smaller CUs, which leads to a tree like coding structure called coding tree unit (CTU).

Because these different CU size compression stages could be executed independently from each other, the encoder algorithm has been parallelized internally by using one task for each CU size compression stage. This results in five parallel CU compression kernels, which will be denoted by their depth in the CTU tree by D0 for size 64×64, D1 for size 32×32, D2 for size 16×16, D3 for size 8×8 and D3.N for size 8×8 with an additional split of the prediction unit. Additionally different time consuming parts of the algorithm, like transform, quantization, intra prediction or cost calculation, have been implemented as HW-accelerated components, to further speedup the platform simulation model. Each of these HW accelerator components has a set of control registers for programming and a DMA controller for data exchange. A more detailed description of the used HEVC intra encoder is given in [11].

The implemented platform simulation model for this use case analysis contains five ARM instruction set simulators (ISS) for each of the parallel CU size compression stages, five HW accelerator components, a worm-hole switched network on chip (NoC) and two memory components, as shown in Fig. 3. Most commonly used synchronization data structures are located in the second memory component *RAM2*, as described in [11]. The components for this platform simulation model are cycle accurate and bit accurate (CABA) components provided by the SoCLib project [7]. In order to detect memory access conflicts, all outgoing data connections from processing elements and HW accelerators will be analyzed and evaluated, as shown in Fig. 3.

B. HEVC Encoder Analysis

To visualize the distribution of the memory access conflicts, the collected data is sampled at different time intervals and for different memory regions forming a spatial temporal grid. In

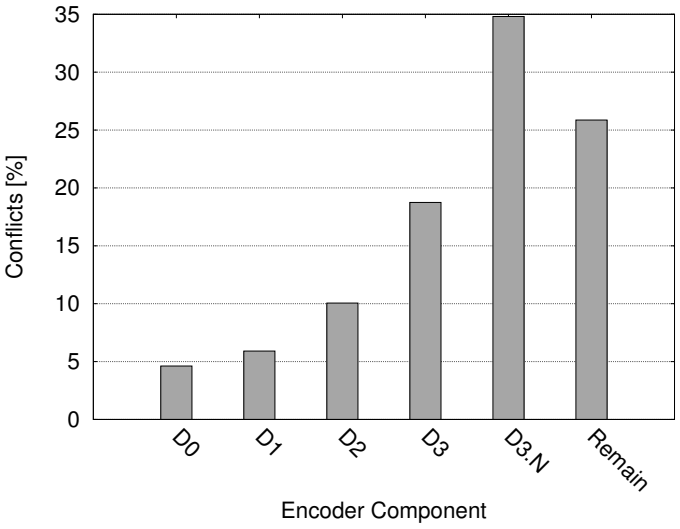

Fig. 5. Distribution of memory access conflicts per parallel CU compression stage.

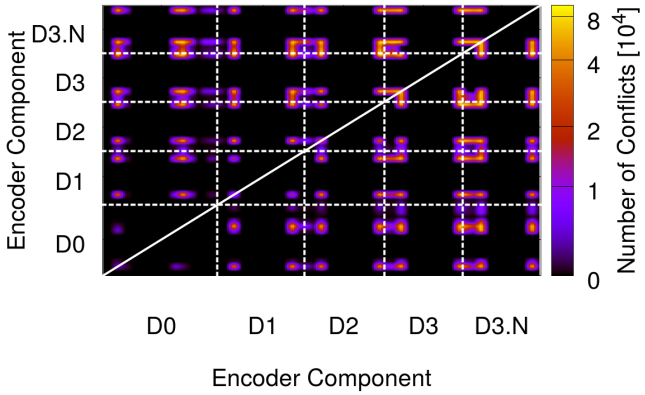

Fig. 6. Memory access conflict matrix for CU compression stages.

Fig. 4 this sampled data is shown for an exemplary encoder run, where the first three CTUs have been encoded. As can be seen in this figure, most conflicts are found in the memory regions marked by the horizontal lines, which contain the data structures for each CU compression stage. Furthermore no conflicts can be found in the initialization phase, because in this phase only one processing element is allocating and initializing all data structures. Afterwards, the memory access conflict pattern is quite similar for each encoded CTU, because the parallel algorithm has frequent synchronization points inbetween, which results in a similar conflict behaviour.

A summary of the memory access conflicts distribution in percent for data structures used only by the different CU compression stages is given in Fig. 5. As can be seen, in summary most of the memory access conflicts (approx. 74%) can be contributed to these data structures, with an increase of conflicts towards the lower depth CU compression stages. The remaining part *Remain* contains conflicts in data structures shared between the CU compression stages, like video data input, bit-stream data output or the reconstructed picture buffers.

To visualize the conflicts between the CU compression stages, Fig. 6 shows a memory access conflict matrix, where a conflict is registered for each pair of involved memory addresses. As can be seen, each stage interferes with each other stage in a similar way, but most conflicts are located at the beginning and at the end of each CU compression stage data structure.

In summary most conflicts can be located in the data structures of each CU compression stage with an increase towards the lower CU compression stages. These memory structures are quite small (approx. 512KB) in comparison to the overall memory footprint of the encoder. Therefore a simple counter measure to reduce the number of memory access conflicts is, to add further memory components to the platform and to map the CU compression data structures to each of these new separate memory components. Due to the NoC interconnect, multiple concurrent accesses become possible as long as these accesses are routed to different memory components.

V. Experimental Results

In a first experiment each of the five different CU compression data structures is mapped to a separate memory component added to the platform simulation model. For each memory component the access latency is configured in a range from 5 to 50 clock cycles to model different main memory access delays. As can be seen in Fig. 7, the speedup between the basic platform simulation model and the model with the additional memory components increases for higher memory access delays. In case the memory access delay is small, which in terms means a fast main memory, the improvement of the execution time is small, when adding additional memory blocks. On the other side, systems with slow main memory will benefit from adding additional memory blocks, which means for a final design a trade-off is possible between the memory access delay and the number of memory components.

As shown in the previous section Sec. IV-B, the number of memory access conflicts increases for lower CU compression stages. Therefore in a second experiment only a subset of the CU compression data structures is mapped to the additional memory blocks, whereas the memory access latency is fixed at 20 clock cycles. In one configuration, shown on the left side of Fig. 8, only one single CU compression component is mapped to a new memory component. Additionally, in another configuration, shown on the right side of Fig. 8, the number of mapped CU compression data structure is successively increased, starting from the lowest CU compression stage, until all data structures have been mapped to different memory blocks.

As expected, mapping the D3.N data structures to a separate memory component, shows the highest speedup in case only one separate memory component is available. In case multiple

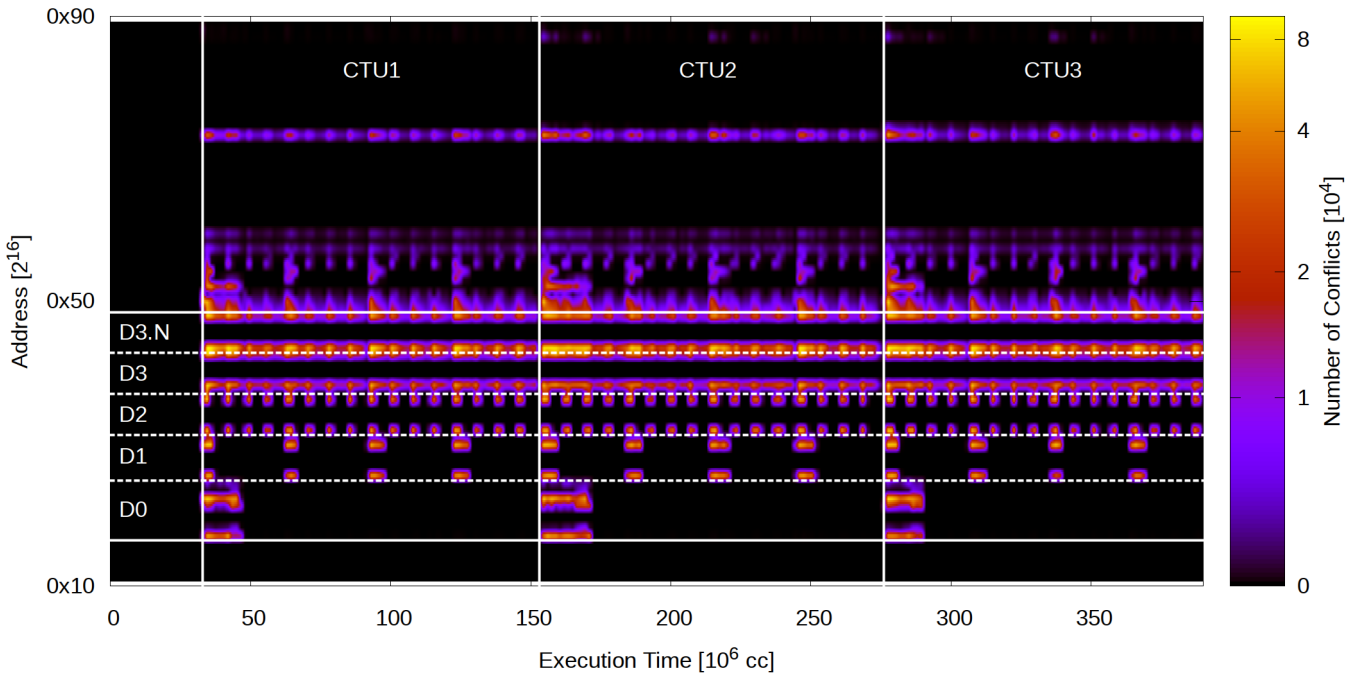

Fig. 4. Memory access conflict trace over execution time for encoding three CTUs. Vertical lines depict the beginning of a new encoded CTU. Horizontal lines mark the memory interval containing CU compression data for each stage.

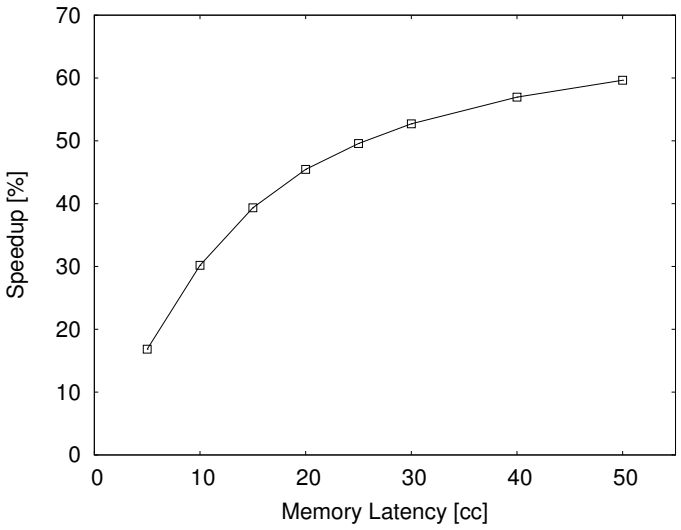

Fig. 7. Encoder execution time speedup of the optimized platform simulation model for different memory access latency parameters.

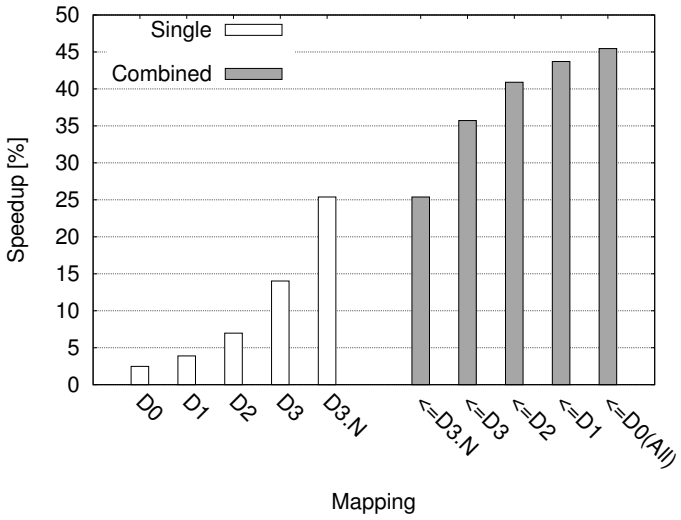

Fig. 8. Encoder execution time speedup of the optimized platform simulation model for different memory mappings.

separate memory blocks are available, the distribution of memory access conflicts can be used to prioritise the mapping of the different data structures and to explore a trade-off between performance and HW costs for additional memory components.

VI. Conclusion

This paper presents a simulation based analysis of memory access conflicts for a heterogeneous multi-core platform. The memory access trace data is collected non-intrusively from a SystemC-based simulation model. Therefore the SystemC simulation kernel has been instrumented, which facilitates a flexible and generic trace data collection approach. In order to control the trace data collection module, an automatic design analysis of the SystemC-based simulation model is performed at runtime. The analysis of memory access conflicts is based on the collection of all memory access data from each platform component, e.g. processing elements and HW accelerators.

By this approach all sources for memory access conflicts are observed and also heterogeneous designs can be analyzed.

To show the capabilities of the implemented memory access conflict analysis, a heterogeneous and parallel HEVC intra video encoder has been examined in a case study. In this case study the main sources for memory access conflicts have been identified, which leads to an optimized platform design. The speedup of this optimized design depends on the initial memory access latency but even for small latencies (10cc) a performance gain of 30% and more could be observed. In future work we plan to extend the analysis tool to perform an automatic hotspot analysis and to provide automatically a segmentation of the involved data structures to derive possible platform optimizations.

REFERENCES

[1] J. Weidendorfer, M. Kowarschik, and C. Trinitis, "A tool suite for simulation based analysis of memory access behavior," in *Computational Science - ICCS 2004*, ser. Lecture Notes in Computer Science, M. Bubak, G. van Albada, P. Sloot, and J. Dongarra, Eds. Springer Berlin Heidelberg, 2004, vol. 3038, pp. 440–447. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24688-6_58

[2] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol. 42, no. 6, pp. 89–100, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1273442.1250746

[3] E. Park, C. Kartsaklis, T. Janjusic, and J. Cavazos, "Trace-driven memory access pattern recognition in computational kernels," in *Proceedings of the Second Workshop on Optimizing Stencil Computations*, ser. WOSC '14. New York, NY, USA: ACM, 2014, pp. 25–32. [Online]. Available: http://doi.acm.org/10.1145/2686745.2686748

[4] D. Hedde and F. Petrot, "A non intrusive simulation-based trace system to analyse multiprocessor systems-on-chip software," in *Proc. 22nd IEEE Int Rapid System Prototyping (RSP) Symp*, 2011, pp. 106–112.

[5] S. Lagraa, A. Termier, and F. Pétrot, "Data mining MPSoC simulation traces to identify concurrent memory access patterns," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 755–760. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485471

[6] D. Genius, "Measuring memory access latency for software objects in a NUMA system-on-chip architecture," in *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on*, Jul. 2013, pp. 1–8.

[7] N. Pouillon, A. Becoulet, A. de Mello, F. Pecheux, and A. Greiner, "A generic instruction set simulator API for timed and untimed simulation and debug of MP2-SoCs," in *Rapid System Prototyping, 2009. RSP '09. IEEE/IFIP International Symposium on*, Jun. 2009, pp. 116–122.

[8] D. Déharbe and S. Medeiros, "Aspect-oriented design in SystemC: Implementation and applications," in *Proceedings of the 19th Annual Symposium on Integrated Circuits and Systems Design*, ser. SBCCI '06. New York, NY, USA: ACM, 2006, pp. 119–124. [Online]. Available: http://doi.acm.org/10.1145/1150343.1150378

[9] M. Kallel, Y. Lahbib, R. Tourki, and A. Baganne, "Verification of SystemC transaction level models using an aspect-oriented and generic approach," in *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2010 5th International Conference on*, Mar. 2010, pp. 1–6.

[10] J. Brandenburg and B. Stabernack, "A generic and non-intrusive profiling methodology for SystemC multi-core platform simulation models," in *Proceedings of the 25th international conference on Architecture of Computing Systems*, ser. ARCS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 135–146.

[11] ——, "Exploring the concurrent execution of HEVC intra encoding algorithms for heterogeneous multi core architectures," in *Design and Architectures for Signal and Image Processing (DASIP), 2015 Conference on*, Sep. 2015, pp. 1–8.