Estimation and Optimization of the Performance of Polyhedral Process Networks

Haastregt, S. van

CHAPTER 7

CONCLUSIONS

In this dissertation, we presented techniques that allow a designer to implement MP-SoCs using the Daedalus system-level design methodology, while taking into account design constraints on system performance. The techniques presented in this dissertation leverage the Daedalus methodology to provide a forward synthesis flow that bridges the specification and implementation gaps. However, the Daedalus methodology did not yet provide a satisfactory solution to satisfy the performance constraints of a designer. In the conventional forward synthesis flow, the designer knows only after a time-consuming forward synthesis step if performance constraints are met. Instead, the designer should obtain feedback faster, possibly at the expense of reduced accuracy, allowing him to avoid a time-consuming forward synthesis step if he knows a design will not satisfy his constraints. We identified three central research problems in Section 1.2. We presented techniques to address these three central problems in Chapters 3, 4, and 5.

The first central problem we addressed was the synthesis problem. We found that the current forward synthesis flow lacked support for RTL implementations for particular classes of input programs and application characteristics that the PNGEN compiler could already process. Our solution to this problem in Chapter 3 consists of four contributions. The first contribution is a characterization of function implementations, which allows us to reason about performance of systems. The second contribution incorporates novel optimizations that were performed by the PNGEN tool, but which were not yet incorporated in the generated RTL architecture, into ESPAM. This allows us to handle a broader class of input programs. The third contribution comprises optimizations for the LAURA processor model's evaluation logic blocks. These optimizations involve pipelining of expression data paths and storage of compile-time

evaluated expressions in ROMs. Pipelining of the evaluation logic blocks enables a LAURA processor to run at a higher clock frequency, which may be required to meet application design constraints. The use of ROMs enables a LAURA processor to handle more complex domains that may result from transformations. The fourth contribution consists of a novel reordering buffer design. This allows Daedalus to generate RTL implementations for applications that exhibit out-of-order communication. The reordering buffer was designed such that replacing a regular FIFO with a reordering buffer does not increase the latency in cycles of read and write operations to the buffer. As a result, transformations that introduce out-of-order communication no longer cause an increased communication latency. The reordering buffer thus enables performance gains of such transformations.

The second central problem we addressed was the performance estimation problem. We found that no applicable performance estimation methods existed that could handle polyhedral process networks implemented using LAURA processors. Estimating the performance of pipelined execution of process iterations was lacking. Such performance estimations are essential to reason about design constraints on system performance. We have investigated and presented performance estimation techniques at four different levels in the Daedalus design flow in Chapter 4. The first performance estimation technique is RTL simulation, which works on the RTL implementation that is the final output of Daedalus. Instead of prototyping this RTL implementation on an FPGA, we simulate the RTL that implements the system. We found that RTL simulation is not feasible for systems containing programmable processors, because of long simulation times. The second performance estimation technique is SystemC simulation, which works at the mapped model of the system. SystemC simulation is faster than RTL simulation, but less accurate. The third performance estimation technique is MCM analysis, which works on the parallel model of the application. The MCM analysis technique is analytical, which has the advantage that estimation time does not depend on the application workload. This leads to performance estimation times that are shorter than SystemC or RTL simulation times. However, we cannot define tight bounds on the inaccuracy of the MCM method, nor whether the method overestimates or underestimates the actual throughput. This model is theoretically attractive and gives insight in the behavior of a PPN, but is impractical because of the lack of accuracy bounds. The fourth performance estimation technique is a novel profiling-based approach for PPNs, named cprof, which works directly on the sequential code. This allows one to obtain accurate results, often in less than one second, without deriving a PPN.

The third central problem we addressed was the transformation problem. We found that it is not trivial for a designer to select a set of transformations and transformation parameters such that a design constraint on performance is met. We have pre-

sented four PPN transformations (i.e., splitting, merging, stream multiplexing, and scheduling) in Chapter 5. For each transformation, we analyzed factors that affect the efficacy of the transformation. This aids the designer to select the appropriate transformations needed to satisfy performance constraints. The first transformation is splitting, which duplicates a process such that throughput may be increased at the expense of increased resource cost. We have proposed analytical and profiling-based strategies to select the splitting factor. The second transformation is merging, which combines multiple processes such that resource cost is reduced, potentially at the expense of decreased throughput. We have identified a special case in which merging of LAURA processors can reduce resource cost while not affecting throughput. The third transformation is stream multiplexing, which increases throughput of multiple PPN executions. We have identified criteria to assess when a stream multiplexing transformation is beneficial, and have presented how to select the stream multiplexing factor such that the latency of a single PPN execution is not affected. The fourth transformation is scheduling, which reorders iterations of a process to increase pipeline utilization. We have identified criteria to assess when a scheduling transformation can be applied to achieve improved pipeline utilization and, consequently, higher throughput.

To validate our solutions to the three central problems, we have conducted a case study using an industrially relevant application used in wireless communication receivers. We compare the extended Daedalus tool flow with the commercial AutoESL high-level synthesis tool in Chapter 6. Specifically, we have focused on a channel matrix preprocessor subblock of a sphere decoder. A manually crafted RTL reference design was available to us. Using a continuous refactoring-based design flow, we were able to replicate the architecture of the reference design using both AutoESL and Daedalus. Refactorings in the AutoESL flow consist primarily of pragma annotations. Refactorings in the Daedalus flow consist primarily of source code restructurings and transformations discussed in Chapter 5. We were able to meet the tight performance design constraint using AutoESL, but not using Daedalus as low-level clock frequency aspects have not been engineered out in the Daedalus tools. Nonetheless, we were able to replicate the architecture of the reference design using Daedalus, which means Daedalus can handle industrially relevant applications. The cprof technique and transformations presented in this dissertation proved essential to obtain the desired architecture of the application in the Daedalus design flow in a short amount of time. Moreover, the cprof technique allowed evaluating alternative design points at the sequential code level.

By addressing the three research problems, we have established a powerful system-level design flow capable of solving industrially relevant design problems, as the designer knows if his design will satisfy his performance constraints. This makes

it worthwhile to explore various transformations of the system, still at the sequential code level. When finding a satisfactory design point, the designer commits to the time-consuming forward synthesis flow, knowing that the design will satisfy his performance constraints.