論文の内容の要旨

Compiler Optimizations for Coarse Grained Reconfigurable Architectures (CGRAs)

（粗粒度再構成可能アーキテクチャのためのコンパイラ最適化）

ラトナ　クリシュナムルティ

Reconfigurable Architectures are those which exploit a higher degree of spatial computation. Typical examples of these include Field Programmable Gate Arrays (FPGA). However, to run an application on an FPGA the number of resources on the FPGA must be sufficient to spatially accommodate them at the same time. Thus emerged a new architecture called Coarse Grained Reconfigurable Architecture (CGRA). CGRA was designed from scratch to have low reconfiguration overhead. This was achieved partially through the use of Coarse Grained building blocks instead of LUTs. Also CGRAs supported loading multiple contexts simultaneously to reduce the reconfiguration overhead. This meant that the fabric of the CGRA performed spatial execution of instructions within one configuration and was temporally multiplexed between several configurations. Further, unlike FPGAs, CGRA supports programming in high level languages viz. C. The spatio-temporal nature of execution in CGRAs makes the task of application compilation quite challenging, since they need to address both the spatial and temporal aspects of execution.

In this thesis, we develop a few compiler techniques which can be applied in the context of CGRAs, with an aim to reduce the total execution time. The total execution time on a CGRA includes the reconfiguration time i.e. the time spent in sending the configuration data before the execution starts and the instruction execution time i.e. the actual time it takes to execute a task on the fabric. We propose three different techniques to reduce the total execution time, which includes the reconfiguration time and the instruction execution time. Our work primarily focuses on those aspects of compilation which are unique to the spatio-temporal execution paradigm. We present the results for all these compiler optimizations in the context of CGRA called REDEFINE, which we use as our testbed. REDEFINE uses static dataflow on the execution fabric and dynamic dataflow to orchestrate various application sub-structures on to the execution fabric. The use of dynamic dataflow is different from previous CGRAs and makes orchestration logic simpler in hardware. REDEFINE also supports the use of custom function units to accelerate kernels. This is supported programmatically through the use of extern function calls. The performance of this CGRA is good and performs 10-15x times better than a General Purpose Processor.

Compilation on CGRAs involves two different aspects. One to determine the application sub-structures which are to be temporally multiplexed on to the fabric. The application substructures then need to be executed on the fabric through appropriate spatial placement, so as to reduce communication overhead. In REDEFINE, the application sub-structure too

is spatio-temporally executed on the fabric. This is done since over-allocating resources, more than the available instruction level parallelism is wasteful. So application substructures are partitioned, such that each partition containing several instructions are mapped to the same compute element. The compute element executes these instructions sequentially based on the availability of data. The task of partitioning is governed by the opposing forces of being able to expose as much parallelism as possible and reducing communication time. We extend Edge-Betweenness Centrality scheme, originally used for detecting community structures in social and biological networks, for partitioning instructions of a dataflow graph. We also implement several other partitioning algorithms from literature and compare the execution time obtained by each of these partitioning algorithms in the context of REDEFINE. Centrality based partitioning scheme outperforms several other schemes with 6-20% execution time speedup for various Cryptographic kernels. Centrality performs well on account of the appropriate choice of edges and use of a larger number of partitions. Further, we observe that these partitions tend to be unbalanced. Unbalanced partitions help reduce the perceived reconfiguration time by overlapping it with instruction execution. However, the use of unbalanced partitions is not beneficial in all cases and this observation was made in the context of two applications. We extend centrality based partitioning scheme to produce balanced partitions. We observe that the new scheme has the ability to improve performance up to 15% (over and above the Centrality scheme). REDEFINE using centrality based partitioning performs 9x times better than a General Purpose Processor, as opposed to 7.76x times better without using centrality based partitioning. Similarly, centrality improves the execution time comparison of AES-128 Decryption from 11x times to 13.2x times.

After the application sub-structures are partitioned, these partitions need to be appropriately placed on the fabric to minimize communication latency. This task is carried out by the mapping algorithm. We propose to demonstrate an interconnection independent mapping algorithm based on greedy and local search based heuristics. Unlike previous work, we experiment with several objective functions and evaluate the best possible mapping algorithm in the context of balanced centrality and centrality based partitioning algorithms.

It is observed that the orchestration of application substructures can take as much time as the execution of the instructions. These application substructures are sequenced based on data and control dependences. Substructure prefetching is used in CGRAs to hide the reconfiguration time while another substructure executes. In REDEFINE, these application substructures are referred to as HyperOps. Determining the successor for a HyperOp requires merging information from the control flow graph and the HyperOp dataflow graph. A new data structure called Control-Data Interaction Graph has been developed. The CDIG captures control and data interactions across HyperOps. Succession in many cases is data dependent. Since hardware branch predictors cannot be applied due to the non-binary branches, we employ a speculative prefetch unit together with a profile based prediction scheme. The results of profiling are annotated on the CDIG. Based on the results the successor HyperOp is selected. The prefetch unit was implemented in the simulator as a look up table. Every time a HyperOp is scheduled, a look up is performed to determine the

next most likely successor. The most likely successor is prefetched, but instruction execution is delayed until the point the decision is made. If the prediction is correct then the data is transferred. In case of a misprediction the HyperOp's allocation on the fabric is revoked. The misprediction penalty is typically four clock cycles. Simulation results show around 7-33% reduction in overall execution time, when compared to the overall execution time without prefetching.

In conclusion, we have proposed three different techniques to improve the performance of CGRAs. All of these techniques have been shown to give good results in the context of REDEFINE. However, these techniques are generic and can be applied to any CGRA.