Abstract

A Coarse-Grained Reconfigurable Architecture (CGRA) is a processing platform which constitutes an interconnection of coarse-grained computation units (viz. Function Units (FUs), Arithmetic Logic Units (ALUs)). These units communicate directly, viz. *send-receive* like primitives, as opposed to the shared memory based communication used in multi-core processors. CGRAs are a well-researched topic and the design space of a CGRA is quite large. The design space can be represented as a 7-tuple (C, N, T, P, O, M, H) where each of the terms have the following meaning: C - choice of computation unit, N - choice of interconnection network, T - Choice of number of context frame (single or multiple), P - presence of partial reconfiguration, O - choice of orchestration mechanism, M - design of memory hierarchy and H - host-CGRA coupling. In this thesis, we develop an architectural framework for a Macro-Dataflow based CGRA where we make the following choice for each of these parameters: C - ALU, N - Network-on-Chip (NoC), T - Multiple contexts, P - support for partial reconfiguration, O - Macro Dataflow based orchestration, M - data memory banks placed at the periphery of the reconfigurable fabric (reconfigurable fabric is the name given to the interconnection of computation units), H - loose coupling between host processor and CGRA, enabling our CGRA to execute an application independent of the host-processor's intervention. The motivations for developing such a CGRA are:

- To execute applications efficiently through reduction in reconfiguration time (i.e. the time needed to transfer instructions and data to the reconfigurable fabric) and reduction in execution time through better exploitation of all forms of parallelism: Instruction Level Parallelism (ILP), Data Level Parallelism (DLP) and Thread/Task Level Parallelism (TLP). We choose a macro-dataflow based orchestration framework in combination with partial reconfiguration so as to ease exploitation of TLP and DLP. Macro-dataflow serves as a light weight synchronization mechanism. We experiment with two variants of the macro-dataflow orchestration units, namely: hardware controlled orchestration unit and the compiler controlled orchestration unit. We employ a NoC as it helps reduce the reconfiguration overhead.

- To permit customization of the CGRA for a particular domain through the use of domain-specific custom-Intellectual Property (IP) blocks. This aids in improving both application performance and makes it energy efficient.
- To develop a CGRA which is completely programmable and accepts any program written using the C89 standard. The compiler and the architecture were co-developed to ensure that every feature of the architecture could be automatically programmed through an application by a compiler.

In this CGRA framework, the orchestration mechanism (O) and the host-CGRA coupling (H) are kept fixed and we permit design space exploration of the other terms in the 7-tuple design space. The mode of compilation and execution remains invariant of these changes, hence referred to as a framework.

We now elucidate the compilation and execution flow for this CGRA framework. An application written in C language is compiled and is transformed into a set of temporal partitions, referred to as HyperOps in this thesis. The macro-dataflow orchestration unit selects a HyperOp for execution when all its inputs are available. The instructions and operands for a ready HyperOp are transferred to the reconfigurable fabric for execution. Each ALU (in the computation unit) is capable of waiting for the availability of the input data, prior to issuing instructions. We permit the launch and execution of a temporal partition to progress in parallel, which reduces the reconfiguration overhead. We further cut launch delays by keeping loops persistent on fabric and thus eliminating the need to launch the instructions. The CGRA framework has been implemented using Bluespec System Verilog. We evaluate the performance of two of these CGRA instances: one for cryptographic applications and another instance for linear algebra kernels. We also run other general purpose integer and floating point applications to demonstrate the generic nature of these optimizations. We explore various microarchitectural optimizations viz. pipeline optimizations (i.e. changing value of T), different forms of macro dataflow orchestration such as hardware controlled orchestration unit and compiler-controlled orchestration unit, different execution modes including resident loops, pipeline parallelism, changes to the router etc. As a result of these optimizations we observe $2.5\times$ improvement in performance as compared to the base version. The reconfiguration overhead was hidden through overlapping launching of instructions with execution making. The perceived reconfiguration overhead is reduced drastically to about 9-11 cycles for each HyperOp, invariant of the size of the HyperOp. This can be mainly attributed to the data dependent instruction execution and use of the NoC. The overhead of the macro-dataflow execution unit was reduced to a minimum with the compiler controlled orchestration unit.

To benchmark the performance of these CGRA instances, we compare the performance of these with an Intel Core 2 Quad running at 2.66GHz. On the cryptographic CGRA instance, running at 700MHz, we observe one to two orders of improvement in performance for cryptographic applications and up to one order of magnitude performance degradation for linear algebra CGRA instance. This relatively poor performance of linear algebra kernels can be attributed to the inability in exploiting ILP across computation units interconnected by the NoC, long latency in accessing data memory placed at the periphery of the reconfigurable fabric and unavailability of pipelined floating point units (which is critical to the performance of linear algebra kernels). The superior performance of the cryptographic kernels can be attributed to higher computation to load instruction ratio, careful choice of custom IP block, ability to construct large HyperOps which allows greater portion of the communication to be performed directly (as against communication through a register file in a general purpose processor) and the use of resident loops execution mode. The power consumption of a computation unit employed on the cryptography CGRA instance, along with its router is about 76mW, as estimated by Synopsys Design Vision using the Faraday 90nm technology library for an activity factor of 0.5. The power of other instances would be dependent on specific instantiation of the domain specific units. This implies that for a reconfigurable fabric of size 5×6 the total power consumption is about 2.3W. The area and power (84mW) dissipated by the macro dataflow orchestration unit, which is common to both instances, is comparable to a single computation unit, making it an effective and low overhead technique to exploit TLP.