Transport Triggered Interconnection Network for a Scalable Application-Specific Processor

Stefan Hauser*, Nico Moser†, Carsten Gremzow†, Ben Juurlink*

* *AES, TU Berlin, Franklinstrasse 28/29, 10587 Berlin, Germany*

† *RT, TU Berlin, Franklinstrasse 28/29, 10587 Berlin, Germany*

KEYWORDS: Transport Triggered Architecture, Application Specific Processor, Instruction Level Parallelism, Interconnection Network

ABSTRACT

Transport triggered architectures ([Cor98]) are a known concept to exploit instruction level parallelism. Our approach is to increase the average utilization of the included processing units (PU) by adapting the configuration to a given application. Supplementary we intend a sparse and customized interconnection network (ICN) to reduce the latency and the area of the ICN.

1 Introduction

The ongoing development in the field of processor architectures changed dramatically during the last few years. Increasing the clock frequency is no longer possible, because of physically aspects, so the focus is moving to multi- and many-core architectures. These trends indicate more and more parallelism in different levels. Additionally the impact of application specific architectures rises. Instruction and data level parallelism are realized by concepts such as transport triggered architectures (TTA) or more commonly known techniques like SIMD units. Our primarily goal was to connect processing units in a flexible and dynamical way. To achieve this goal we utilized TTA methods to control the interconnection network. The flexibility of the basic architecture gives us a skeletal structure to design derived version of this architecture for special purposes. By using even more complex PUs than in other TTA projects, another important objective is to achieve a better utilization of all PUs. Therefore

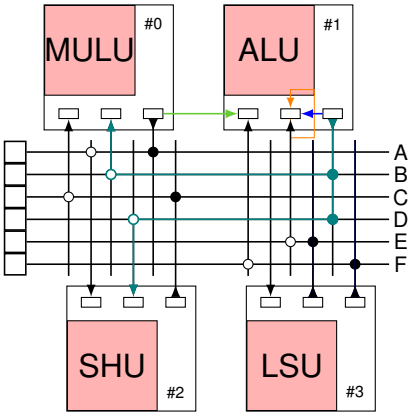
[1]E-mail: {hausers,moser,gosper,juurlink}@cs.tu-berlin.de

we have to make sure that sufficient data is available.

Our approach is more than a concrete architecture, because every part can be adapted to an own application. Such an adaption could be done with the help of a runtime analysis tool, such as the one in [Gre08].

This abstract gives a brief introduction to the architecture with a focus on the deployed ICN.

2 Architecture



net: (default) The data is received from the ICN.

hold: The current content of an input register is held.

link: Direct connection to an input port of another unit.

acc: The unit behaves as an one-address-machine.

multicast: Date can be used for different units and ports.

Figure 1: Overview of the main architecture concept on the bases of an exemplary instance, wherein the different modes of transporting data are emphasized.

An example of the architecture is depicted in Figure 1. As our design can be adapted to an application this is only an introduction example to visualize the main functionality. Thus we present a system that consists of three PUs and one load store unit (LSU). The horizontal lines represent busses: the number of busses limits the maximal number of concurrent possible data movements. Practically every operation of a PU consumes two operands, so that a reasonable number of busses corresponds to twice the number of PUs. However, adapting the number of busses to the number of PUs does not suffice to supply every PU during every clock cycle with new operands, because practically every instruction of the PU consumes two operands, but generates only one result. If we want to ensure a better utilization of all PUs and if we are interested in a reduction of network traffic, we have to investigate different solutions. We propose different modes to solve these problems. Normally the PUs achieve data via the ICN, but that is not necessary in all cases. Using analysis of different benchmarks ([MHG09]), such as *mpeg2* or *jpeg2000* we discovered that groups of special operations, for example *multiply accumulate* (MAC), appear very often. Our architecture supports this with direct links between several PUs. From the MAC operation we also conclude that for several times the output of one PU can directly be consumed by this unit. This one-address-mode is also supported by our design. Additionally an input register of a PU can be put into a hold mode so that data can be reused for multiple operations. These enhancements reduce also the communication via the ICN. To increase the number of accessible data items, we allow a

multicast in the ICN. Furthermore we connected a special LSU, that provides more than one data item in each clock cycle.

Computed results, that cannot be consumed immediately, should be buffered in registers. Usual architectures include a global register file, which can be accessed be any PU. Such a solution, however increments the network traffic, so that we decided to buffer the data within the dedicated PU, which is similar to a distributed register file, so that the number of registers scales with the number of PUs. With this modification the complexity of register allocation for compilers increases, but we assume that higher performance can be achieved by using locality advantages.

3 Interconnection Network

The ICN is, as Figure 1 implies, bus based[2], which is the common way of connections within a processor to achieve low latency. Another possibility to reduce the delay is to decrease the number of subscribers to a bus, which also limits the occupied area. Since the architecture should be adapted to the given application, we can specialize the ICN, also, if we scheduled the operations to the different PUs with the aid of runtime analysis. A net configuration can be described in XML and, with our net generator, transformed to a semantic equivalent and synthesizable VHDL description. The generator provides additionally the exact binary decoded addresses for the different bus sources and targets, which can be used to adapt the provided assembler.

However, as already mentioned, we use PUs that can perform more than one operation and therefore they need special opcode, which can be delivered by the ICN with virtual addresses as it was proposed in the MOVE-Project ([Cor94]). As our PUs support a couple of different instructions and the ICN increases accordingly, we decided to separate control and transport instructions. To achieve this every PU gets its own instructions out of a dedicated memory and the network gets a couple of operations to set up the right connections.

The interface between a PU and the network is independent of the actual functional unit (FU), so that further FUs can be integrated if they achieve the specification. An interface is needed to decode and set up the different modes for the input ports of a PU. Besides, it is responsible for the local register file.

4 Programming Model

The creation of a concrete instance provides additionally to the VHDL description an assembler. Thus, the code becomes more readable with the use of textual labels for targets and sources. Input and output registers for every unit provoke a two stage pipeline, which is actually not applicable for such an architecture. Nevertheless these registers are necessary if the complexity of the PUs reached the target value and a hold mode should be available. A hardware solution for pipeline hazards such as interlocking or bypassing are not lpractically

[2]Since the architecture is targeted of Xilinx FPGAs, the network was actually constructed out of multiplexers.

in this concept. Hence, solving conflicts has to be done within the program code with e.g. delay slots or an interleaved implementation.

5 Results and Conclusions

Separation of control and transport information reduces the program size by about 27%, because of that normally more busses than PUs are included. If there is no boundary, every bus instruction contains control information for a PU, even if is not needed. Furthermore this version requires less logic, so that for a configuration with 12 PUs and 24 busses about 5% less slices are occupied with a separated configuration of control and transport instructions on the Xilinx Spartan 3A DSP 3400 FPGA. According to the programming model this modification is not very intuitive, but we can hide this from the programmer by using a simple script.

As presented in [MHG10], reducing the connections per bus tends to reduce the critical path length and the occupied area. In general a configuration with fewer connections per bus but with more PUs consumes less area and exhibits a shorter critical path than a comparable design with more sources and targets for each bus.

The programming of such an architecture is not very common, so that we developed a couple of tools to support the development. With this framework we are able to develop an instance for small applications such as the Fibonacci numbers. For more complex benchmarks we have to automate the scheduling and allocation with the assistance of the results from the runtime analysis tool.

References

[Cor94] H. Corporaal. Design of transport triggered architectures. In *VLSI, 1994. Design Automation of High Performance VLSI Systems. GLSV '94, Proceedings., Fourth Great Lakes Symposium on*, pages 130–135, 1994.

[Cor98] Henk Corporaal. *Microprocessor architectures from VLIW to TTA*. Wiley, Chichester [et al.], 1998.

[Gre08] C. Gremzow. Quantitative global dataflow analysis on virtual instruction set simulators for hardware/software co-design. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pages 377–383, 2008.

[MHG09] Nico Moser, Stefan Hauser, and Carsten Gremzow. Reduzierung der Kommunikation in TTA-Verbindungsnetzen mittels Laufzeitanalyse. In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 107–116, 2009.

[MHG10] Nico Moser, Stefan Hauser, and Carsten Gremzow. A hybrid transport/control operation triggered architecture. In *Workshop on Parallel Systems and Algorithms (PARS 2010) at ARCS 2010 - Architecture of Computing Systems*, pages 121–125, feb 2010.