Abstract—Polymorphic processors attempt to merge the benefits of general purpose processors with performance gains from reconfigurable elements. In this paper, we present a novel polymorphic processor architecture. The integration of a network-on-a-chip (NoC) architecture as a replacement for the processor datapath creates unique requirements for the NoC design. We explore multiple NoC topologies as potential candidates for the creation of a polymorphic processor. A simulator based around the network simulator 2 (ns-2) software platform is created. Standard embedded processor benchmark programs are simulated to explore critical parameters and NoC design decisions impacting the performance of the polymorphic processor.

I. INTRODUCTION

It has been shown that integration of application specific processing elements with general purpose processors utilizing system on a chip (SoC) architecture will be a primary driver for future improvements in computational performance [1]. In this direction, polymorphic processors attempt to merge the advantages of general purpose processing with application specific design [2], [3]. Hence, the goal of a polymorphic processor is to integrate reconfigurable elements with a general purpose processor into a common design methodology and run-time control system. One of the major hurdles in the design of a polymorphic processor is flexible integration of heterogeneous reconfigurable elements at run time. In addition, performance goals require a tight coupling between reconfigurable elements and traditional general purpose processor components. To achieve both of these aims, it has been proposed to integrate network on a chip (NoC) design into the datapath of a general purpose processor [4].

The use of a NoC as a processor datapath imposes a unique set of requirements on the NoC. The traffic loads generated by the processor functional units are different than the traditional NoC loads. Furthermore the impact NoC parameters will have on the overall execution of the processor is not well studied. Therefore, this paper examines multiple NoC architecture topologies with the goal of finding a topology that will robustly meet the unique design requirements of a NoC datapath.

Section II gives an overview of the polymorphic processor design presented in prior work. A processor simulator design is presented in Section III. In Section IV the topologies and benchmarks used are presented and we analyze the results. Finally, Section V provides final conclusions.

II. NOC DATAPATH POLYMORPHIC PROCESSOR

To achieve the goals of a true polymorphic processor, reconfigurable elements must be incorporated into traditional microprocessor design. Many architectures have been proposed to try to achieve these aims, such as ARISE with its interface to custom compute units [5], MOLEN which uses a specialized back end compiler to optimize for coprocessors [6], and NAPA reconfigurable logic array communicating over the standard processor bus [7]. This paper focuses upon one of the platforms presented in detail in prior work [4], that of a network-on-a-chip based datapath. The proposed processor architecture is based upon the idea that as reconfigurable elements are incorporated into a microprocessor the complexity and scale of internal interconnects for the processor rapidly grow. Traditional fixed datapaths, with multiplexer control units, are insufficient in both flexibility and scalability to incorporate many reconfigurable elements. This processor architecture borrows from the system-on-a-chip paradigm and incorporates NoC based interconnects as communication channels between datapath functional units.

The overall architecture can be seen in Fig. 1. Each instruction is fetched and decoded as in a traditional microprocessor. After that, each instruction contents are packetized. This new data packet is then injected into the network with the destination of the first function unit required. When the function unit receives the packet, it executes the assigned operation. As an example, an ALU performs an address calculation, and then re-injects the packet to the network to travel to the next functional unit. Instruction execution progresses this way until computation is completed. This approach treats each instruction as a packet in a network. Packets travel the network visiting each functional unit needed to perform the execution.

The advantage of this system is primarily through scalability and flexibility. Any number of elements can be connected to the NoC. This allows for instantiation of multiple function units, for example multiple ALU or FPU units. Furthermore the units can be added or removed from the system in real time, using reconfigurable elements. It also supports the creation of heterogeneous specialized functional units; such as a customized butterfly unit to support faster FFT execution. This allows for the design of highly specialized application specific functional units. These highly specialized units can provide significant increase in performance for certain application

Architectural Topologies for NoC Datapath Polymorphic Processors

Joshua Weber, Erdal Oruklu, Jafar Saniie
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago, Illinois, 60616
workloads. The ability to optimize the design of the processor to the specific application is a primary benefit of a polymorphic processor which is made possible with the use of a NoC based datapath.

III. NOC SIMULATOR

A. Overview

A simulator was developed in order to evaluate the relative performance of various network topologies when applied to a NoC based polymorphic processor. The simulator is based around the Network Simulator 2 (ns2) software package. This software is used in the academic and research fields for exploration into the performance of many aspects of networking, including NoC [8], [9]. Since the software is open source, new code was written to extend the functionality. This new functionality allows for simulation of a experimental polymorphic processor executing a SPARC type instruction set.

B. The Network Simulator

The Network Simulator (ns-2) is a discrete event simulator targeting networking research. It has been used in a wide variety of networking research including use in simulation of NoC architecture and protocols. It provides support for a wide range of networking techniques, in addition to robust access to the underlying framework to allow for custom modification and extension of the simulator.

The ns-2 simulator is implemented in C++ with an interface layer based upon OTcl. All major simulation objects are created as discrete objects, which then have a OTcl representation. This allows for easy Tcl based scripting while still maintaining the execution speed advantages of C++.

C. Polymorphic ns-2 Extensions

Ns-2 was modified and extended to support simulation of a processor executing with a NoC datapath. To facilitate this, a new custom set of applications, agents, nodes, and packets were added to the ns-2 simulator. All functional units within the processor are represented in the ns-2 simulator as an application. These applications then communicate with each other using the ns-2 node network simulation.

The primary application of interest is that of the instruction decode / packetizer functional unit. This unit reads the next instruction to be executed and transforms it into a ns-2 compatible network packet. This packet is broadcast and addressed to all functional units needed for processing and injected into the NoC for delivery. Each functional unit takes delivery of the packet, waits a programmable amount of time to simulate execution processing delay, and then delivers the packet back to the NoC for transmission to the next functional unit.

The packetizer allows for out of order execution, in order to maximize the amount of parallelism that can be extracted from the executing program. It has a window size parameter and keeps track of all data dependencies. This window function works by allowing a selectable number of data dependency free instruction following the currently executing instruction to be issued to the functional units. Hence, the packetizer monitors data dependencies and tracks instructions in-flight. The window size has a large impact on the overall performance of the processor. A polymorphic processor achieves performance increase with additional functional units. In order to take advantage of it, the processor must extract instruction level parallelism from the instruction stream. The use of a flexible window size allows for a straight forward extraction of instruction level parallelism through out-of-order execution. There are many other techniques that could be used to extract this parallelism that are outside the scope of this work.

All functional units have a parameterized processing time that can be varied to more accurately model the processor execution. This allows for the model to be tailored to the actual execution time of functional units. In addition, since all function units are parameterized independently, it allows for modeling of more complex interactions, such as when the delays for functional units are different. For a traditional pipelined datapath, all stages work with the same delay budget. The use of a NoC datapath allows for a divergence from that requirement which allows mixing of both simple and complex functional units with varying computational times.
Fig. 2. Network Topologies

### TABLE I

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>1,830,916</td>
</tr>
<tr>
<td>IFFT</td>
<td>1,892,382</td>
</tr>
<tr>
<td>CRC32</td>
<td>15,692,471</td>
</tr>
<tr>
<td>BasicMath</td>
<td>15,604,602</td>
</tr>
<tr>
<td>ADPCM.decode</td>
<td>34,833,465</td>
</tr>
<tr>
<td>ADPCM.encode</td>
<td>38,894,390</td>
</tr>
</tbody>
</table>

**IV. NETWORK TOPOLOGY SIMULATIONS**

**A. Benchmark Programs**

A suitable set of benchmark programs has been compiled for execution on this processor simulator. The majority of the benchmarks come from the well-known MiBench benchmark suite [10]. MiBench provides a set of commercially representative embedded systems programs. Extracted from this set and used are the CRC32, FFT, IFFT, ADPCM.encode, ADPCM.decode, and BasicMath benchmarks. This set of benchmarks provides a varied application load to get an initial impression on the performance of the proposed processor design.

Since the simulator models only relative computational performance and does not implement the full execution of the processor, it cannot execute a raw program. In order to overcome this, the benchmark programs are first run on the TSIM LEON3 SPARC simulator. As the benchmarks are executed, a trace of all executed instructions is captured. This provides a cycle-accurate, in-order instruction trace of the benchmark when it is executed on a commercially available SPARC processor. The instruction trace is then used as input to the ns-2 based simulator. This allows for very basic modeling of the instruction execution time, without being concerned with the full simulation of SPARC processor, significantly easing the implementation and execution time of the simulator.
When evaluating the performance of a processor using a benchmark, it is important to look at the distribution of instructions. The instruction distribution for the benchmark programs is shown in Fig. 3. It is clear that all benchmarks are heavily computationally driven, shown by the relatively high percentage of integer and floating point instructions compared to controller oriented branch and I/O load/store instructions. This computational focus is representative for many of the embedded processing tasks performed by application specific processors. The benchmarks also represent a distribution of weighting between integer and floating point instructions. This distribution allows us to examine the performance of the processor with the various workloads present in real world applications. A final statistic of interest is the total instruction count, shown in Table I. This shows that the benchmarks cover a range of scaling sizes for various workloads.

B. Network Topologies

To limit the scope of the work, four main topologies were studied and compared. The topologies presented are shown in Fig 2 and consist of a Ring, a Modified Ring, a 2D Mesh, and a Crossbar architecture. The ring architecture provides a very simple base line implementation. Due to the fact that a processor datapath is highly sequential, it is very similar to the flow of data through a traditional datapath. The modified ring structure attempts to provide a more robust scalability than the standard ring. In this architecture, similar functional units are placed in parallel on the ring structure. This incurs a minor increase in link cost, but provides the same total delay over the ring regardless of number of function units instantiated. The next architecture studied is the standard 2D mesh architecture. The architecture is a rectangular layout with bi-directional links between neighboring nodes. The final architecture explored is a crossbar. The crossbar architecture is an extreme architecture in which every node has a link to every other node. This shrinks all communications to only a single hop away, but comes with a very sharp increase in link cost as the network scales in nodes.

In addition to exploration of the various topologies, the impact of increasing the number of available functional units was also explored. The increase in the number of functional units, and the ease of integration to the processor design is crucial for a polymorphic processor. Since all of the benchmark programs are computation heavy, increasing the number of computational functional units (both ALU and FPU) directly impacts the overall performance.

C. Results

The total execution time for all benchmarks can be seen in Table II. This table shows the number of cycles executed in each benchmark. In addition, normalized performance values are presented in Fig 4. These values show the performance of each topology relative to the results from the base line Ring 1 ALU 1 FPU topology. All the values presented were obtained using an instruction window size of 16, with an assumed NoC clock rate 8 times the datapath clock. It is reasonable to diverge the design into separate clock domains. The NoC would function independently of the functional units. With the NoC only needing to perform data transfers, the delay time can be optimized much easier for high clock frequencies. This higher clock frequency helps to prevent the processor from simply becoming entirely NoC delay limited.

All architectures exhibit scalability, and increase performance as the number of function units increase. Furthermore, they perform very similar with only a single functional unit. The advantages only become significant as the number of functional units increase. As the number of function units is increased modestly from one to two ALUs/FPUs, significant performance improvements are achieved. It is important to note that a processor of this type would allow performance boost to be selectively applied during run time as reconfigurable elements are configured to act as additional functional units.

The Ring architecture is very simple and has very low resource cost; however, its performance is very lackluster. The performance increase from additional function units is completely hindered by increased packet delay through the network. The Modified Ring architecture helps to mitigate this to some extent by eliminating the growth of delay when adding additional functional units. It still exhibits a similar weakness in overall structure, the ring based architecture delay is too high and prevents optimal processor execution. As seen from the results the 2D Mesh architecture looks very promising as a candidate for NoC architecture. The max number of nodes compared to delay grows at a polynomial rate. In addition, for small mesh sizes, a large portion of communication occur between neighbor nodes. The Crossbar architecture represents a reference point for comparison. As it provides a connection from every node to every other node it allows for a constant delay time of 1 for all packet transmissions. This can be seen as an upper limit to performance gains independent of architectural choices.
Fig. 4. Normalized Performance

Fig. 5. Link Cost vs Average Normalized Performance
The hardware resource consumption of the architectures is very important and can be seen in Fig. 5. This figure estimates the cost of each architecture as the total number of individual point to point unidirectional links. It demonstrates the superior performance of the 2D Mesh architecture. With much lower total cost, it is able to achieve a performance that is only slightly reduced from the upper limit Crossbar implementation. In addition, it is able to gain significant improvements in performance with the increased cost of additional functional units.

V. CONCLUSION

The use of a NoC as a datapath for a polymorphic processor introduces unique requirements for NoC design parameters. This paper has explored a set of potential candidate topologies with the goal of evaluating the impact each will have on the performance of a polymorphic processor. In order to accomplish this, a network simulator tool, ns2, was utilized to perform network simulation, with a set of extensions developed to enable simulation of a theoretical polymorphic processor. Industry standard benchmark programs were used to implement real-world application specific workloads. From the study of the results, it can be shown that the ring architecture while having low total cost, are not well suited for NoC topology choices. The 2D mesh architecture is able to perform robustly, within close performance of the theoretical upper limit provided by the crossbar architecture, and has good support for providing an easily scalable design.

REFERENCES


