Active System Area Networks for Data Intensive Computations

Final Report

Submitted to

Department of Energy

Technical Contact

Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta GA, 30332
Email: sudha@ece.gatech.edu

DOE Patent Clearance Granted
Mark P. Dvorscak
(630) 252-2993
E-mail: mark.dvorscak@ch.doe.gov
Office of Intellectual Property Law
DOE Chicago Operations Office
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
1 Overall Project Goals: Review

The goal of the Active System Area Networks (ASAN) project is to develop hardware and software technologies for the implementation of active system area networks (ASANs). The use of the term "active" refers to the ability of the network interfaces to perform application-specific as well as system level computations in addition to their traditional role of data transfer.

This project adopts the view that the network infrastructure should be an active computational entity capable of supporting certain classes of computations that would otherwise be performed on the host CPUs. The result is a unique network-wide programming model where computations are dynamically placed within the host CPUs or the NIs depending upon the quality of service demands and network/CPU resource availability. The project seeks to demonstrate that such an approach is a better match for data intensive network-based applications and that the advent of low-cost powerful embedded processors and configurable hardware makes such an approach economically viable and desirable.

The tangible project goals are

- the demonstration of configurable network interfaces (NIs) comprised of embedded processors coupled with field programmable gate arrays (FPGAs) to implement data-intensive streaming computations during communication and the development of associated computational models.

- the use of the NIs embedded processors to effectively handle associated meta-information and perform computationally non-intensive operations best performed "close" to the network

- the movement of scheduling operations to the NIs and the use of the FPGAs to provide practical implementations of computationally demanding QoS scheduling disciplines for real-time communication

- the demonstration of an "extensible" software and quality management middleware layer that enables the dynamic placement of hardware (FPGA configurations) and software computations within the NIs and provides a uniform API to multiple heterogeneous network substrates.

Our approach is an experimental one driven by existing application implementations. The near term objective is the construction of an experimental test-bed with off-the-shelf FPGA and embedded processor cards. The funding from the Department of Energy is focused on the development of models and infrastructure that support NI-based computation in general and the support of QoS scheduling disciplines in particular.

2 Progress

In the last year a principal development milestone has been the selection of the Intel IXP1200 network processor as the software programmable component of the ASAN network interface, and the installation and evaluation of the IXP development environment and hardware.
Emerging Network Processors (NPs) such as the IXP 1200 offer the capability to operate on data at wire speeds when used as network "interface" (NI) processors in an Active System Area Network (ASAN). NPs are attractive in NIs for the same reasons they are attractive in network routers and switches: they are fast enough to keep up with wire speeds yet programmable in the style of processors. The ASAN project is using the Intel IXP1200 network processor in combination with a reconfigurable computing fabric as part of its approach to implementing application functions in the network interface of server machines.

The IXP1200 (and other NPs) offer multiple processors on a chip tailored to moving data at wire speeds. These chips have been developed in the networking industry to bridge the gap between ASIC designs and embedded processors. Unlike embedded processors to date, NPs offer the ability to handle every byte of "data" that flows through the system, not just packet "headers". The IXP1200, as a typical NP, includes a StrongARM core as a traditional embedded processor plus six four-way-multithreaded "microengines", each a small RISC core. The microengines are intended to move data continuously between the chip interfaces while performing minimal processing.

In our implementation, the NP will be coupled with a reconfigurable engine (a Xilinx Virtex FPGA) to gain the benefits of both technologies. In the simplest view, the NP is responsible for moving data between interfaces, memory and the FPGA while the FPGA performs custom computations on the data. The IXP1200 NP combines standard interfaces (PCI, fast and gigabit ethernet, SDRAM and SSRAM), high bandwidth data paths (6.4Gb/second) and, uniquely, a multiprocessor-on-a-chip architecture capable of moving data between all these interfaces as full speed. The FPGA then operates only on framed chunks of data orchestrated by the NP.

The combination of the NP and FPGA cases both the development of the platform itself and the development of applications that run on the platform. The NP's built-in standard interfaces leverage the development work done in the networking world. We need only develop the NP-to-FPGA interface rather than a series of soft cores or interfaces to external components. Further, the NP-to-FPGA interface is made easier by the fact that the NP is programmable rather than fixed and arbitrary. For applications, the combination of the NP and FPGA eases application development work by separating data motion from computation. For instance, the NP can buffer packets in SDRAM and pass only valid frames as a stream through the FPGA, minimizing overhead circuitry (and complexity) in the FPGA. Further, some computation is possible in the NP itself, offering an alternative implementation for some applications and a prototyping mechanism for algorithms that will eventually be embedded in the FPGA.

3 Project Results

3.1 Next Generation ASAN Testbed

The ASAN testbed consists of a cluster of eight Intel IA32 servers equipped with network processor PCI cards plus a number of standalone network processor evaluation systems, all donated by Intel. The servers have dual 1.7GHz Pentium IV Xeon processors and each support two Radisys ENP2505 cards with the IXP1200 network processor. We have been working since October 2000 with the evaluation systems and received the cluster in November, 2001.
So far we have:

1. Assembled an experimental environment both in simulation and in hardware. The simulation system uses Intel's microengine simulator combined with scripting to generate packet flows. The hardware consists of the cluster and evaluation boards above linked via gigabit and 100T networking. All components, including the embedded StrongARM in the IXP1200, run Linux for ease of development.

2. We have confirmed that the micro-engines can saturate the bandwidth of each interface with time left over for processing. For instance, there are 1.5 cycles/byte available for processing data copied from SDRAM to SDRAM while saturating the SDRAM bus or 3 cycles/byte available while saturating a full-duplex gigabit link.

3. We have trained ~15 students in the use of the hardware and simulation systems through special projects and a special topics class taught in the Spring semester 2002.

4. We have completed the architectural design of a custom board to host a combination of an embedded processor (the IXP) and FPGA (Virtex) which will exploit the maximum concurrent bandwidth afforded by the hardware devices.

A Stream Switch performs application-defined routing or filtering of stream data within the network. The core of a stream switch is a high-speed pattern classifier circuit which is instantiated on demand in reconfigurable logic. The dynamic nature of the application places stringent requirements on time to instantiate these circuits - on the order of seconds. We have developed tools for compiling pattern classification rule sets from several application domains into reconfigurable logic circuit. The challenge is in performing such compilation and chip reconfiguration in order of seconds. The stream switching environment is being placed on the IXP cluster. Towards this end we have:

1. Added application domains of internet firewall rules, unix filesystem protection rules and inverse colormap rules from image processing. The firewall rules are particularly promising: a ruleset of 112 100-bit rules from our College's Cisco firewall compiles to less than 1000 Virtex LUTs.

2. Extended the ruleset compiler to generate conventional multilevel lookup tables in RAM for comparison purposes. A 42000-route IP lookup table (82000 gates) requires about 1MB of lookup tables in RAM.

3. Procured evaluation hardware including Xilinx Virtex XCV800 and Altera 20K200 boards. The firewall ruleset has been demonstrated running at 33M lookups/second (10Gbit rates) on the 20K200 part.

Current efforts will use the IXP cluster to implement and validate the stream switching model.

List of Publications


### 3.2 Modular Hardware Schedulers for Wire Speed QoS Scheduling

This component of the project addresses the design and analysis of hardware support for packet scheduling at the network endpoints. The schedulers are targeted for implementation within the network interfaces. Our work has focused on implementations of stream schedulers for providing QoS in ASAN clusters. Building on our prior work with supporting Dynamic Window Constrained Scheduling (DWCS) algorithm on i960 RD 120 network interfaces (NIs) [1, 4], we have designed hardware implementations for support for up to 32 streams on Virtex FPGA cards. The completed scheduling system will provide hardware scheduling acceleration to the Network Interface (NI). While a commercial instantiation would be a single card or even a single chip System-on-a-Chip (SoC) device, in our research implementation the FPGA card will be present as a I/O bus peer to the Network Interface (NI) on the same bus segment. The objective is to use the NI memory to buffer frames and NI transceiver to drive the links while computing winning frames for scheduling from the FPGA card on the PCI bus. Packet times for Gigabit Ethernet frames are 12 microsecs (1500 byte frames) and preliminary results show that this can be achieved very easily (4 microseconds (write to FPGA card) + ( <1 microsecond for scheduling ) + 4 microseconds (read from FPGA card)).
The Architecture of the scheduler is shown in Figure 1. Base Register blocks store stream attribute priority values (deadlines and loss-tolerances) and feed these values to Decision blocks. The Base Register blocks and Decision blocks are arranged in a single-stage re-circulating shuffle exchange network. A winner is computed every log2n cycles and priority computation takes an extra cycle. The Decision block allows computation of multiple scheduler 'rules' in a single cycle and is shown in Figure 2. The current implementation is for a Virtex 1000 card.

Figure 1 Architecture of the DWCS Scheduler
3.2.1 FPGA Hardware Instantiation of a QoS Packet Scheduler

Building hardware packet schedulers for reconfigurable logic and FPGAs involves a functional simulation phase, a timing simulation phase and a hardware instantiation verification and validation phase. Our design has focused on building hardware packet schedulers in reconfigurable logic to schedule and stream up to 32 streams. We developed VHDL design code for the hardware packet scheduler to schedule and stream 4, 8, 16 and 32 streams and verified functionality and timing during functional and timing simulation phases. We have completed construction of a hardware packet scheduler in Virtex V1000 FPGAs to schedule up to four streams and are validating and verifying functionality of the packet scheduler for all possible inputs (a fully functional hardware instantiation).

The hardware instantiation (netlist in xilinx bitstream form) is downloaded to a Celoxica PCI card (has four 2MB Sram banks with 32b/33MHz PCI). A memory interface was developed that interfaces with the “control and steering block” shown in Figure 1. Arrival times of packets in input streams are available in SRAM banks. Winners are computed by the scheduler across the input streams and winner candidates are deposited in the SRAM banks. Details of the QoS packet scheduler are available in [1, 4] and hardware implementation details are provided in [2, 3].
3.2.2 Implementation Results

This Section presents implementation results from placed-and-routed netlists and actual hardware instantiations of our designs. Figure 3 shows the area and timing characteristics of hardware packet schedulers capable of scheduling a varying number of streams.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Implementation (Xilinx Virtex Chip, rev/Toolchain)</th>
<th>CLB Count (Area)</th>
<th>Achievable Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) 2R - D (two streams)</td>
<td>Xilinx Virtex / Synopsys FPGA Express / Fndm 2.1i V300BG432-6</td>
<td>7%</td>
<td>64.65 MHz</td>
</tr>
<tr>
<td>2) CSL - 4R - 2DB (four streams)</td>
<td>V1000BG560-6 (Synopsys FPGA Express/ fndm 3.3i)</td>
<td>10% (four stream)</td>
<td>16.5 MHz</td>
</tr>
<tr>
<td>3) CSL - 32R - 16DB (32 streams)</td>
<td></td>
<td>74% (map, 32 stream)</td>
<td>no placed timing available</td>
</tr>
<tr>
<td>4) M-CSL-4R-2D (four streams w/ memory)</td>
<td>V1000-BG560-6 (Simplify 7.1SE 4.1)</td>
<td>16%</td>
<td>15.9 MHz (placed-routed functional netlist)</td>
</tr>
</tbody>
</table>

Figure 3 Xilinx Virtex V1000 implementation results

Architecture 1 is the basic canonical structure of a hardware QoS packet scheduler with two register blocks (with stream attributes) feeding a decision block. Architecture 2 is an implementation capable of handling four streams with a control & steering block, four register blocks (hold stream attributes) and two decision blocks. This implementation takes only 10% of the Xilinx Virtex chip area and can be clocked at 16.5 MHz (a winner can be computed every 0.2 microseconds easily meeting the link requirements of a gigabit network link (MTU Ethernet frame time of 12 microseconds)). Results from scheduling four streams is shown in Figure 4. The Figure shows the cumulative packet counts from scheduling four streams in the ratio (1/8 : 2/16 : 2/8 : 4/8) with loss-tolerances (7/8, 14/16, 6/8, 4/8). Loss-tolerance specifications allow the bandwidth to be divided among the four streams and the aggregate packet count shows the bandwidth allocated to be the same for stream 1 and stream 2 and then double for stream 3 (over stream 1 and stream 2) and finally double for stream 4 (over stream 3). The results were obtained on a final placed and routed netlist using the Aldec EDIF/VHDL simulator with timing used from an SDF file generated using the Xilinx synthesis tools. 32 streams map to about 74% of the Virtex chip area and experience with Xilinx place and route tools suggest that such large CLB utilization will not allow complete placement and routing. We expect 16 streams to be mapped, placed and routed in a Virtex 1000 chip with ease with Virtex II FPGA and future chips allowing placement and routing of 32 stream designs and higher.
3.2.3 Experimental QoS End-System Architecture

Our End-system architecture for providing QoS in Active System Area networks consists of CPU boards like Σ0 (see [1, 4]) and IXP 1200 interacting with Celoxica reconfigurable logic boards on a PCI backplane. CPU boards buffer streams in their large DRAMs and stream packets to the network as they are equipped with gigabit Ethernet transceivers. PCI reconfigurable logic boards provide hardware acceleration for scheduling computations. The hardware QoS scheduler generates winners over the PCI bus to CPU boards and receives arrival times from the CPU board to determine FIFO order of streams (when required for scheduling). The ASAN QoS end-system architecture used in our work is shown in Figure 5.

Figure 5 End-system QoS Architecture
Entity 2 is a J2O board (see [1, 4]) with a 66MHz i960 RD processor and Entity 1 is a Celoxica reconfigurable logic board. Note that interactions 3 and 4 occur over the PCI backplane. We have built queue structures and packet dispatch logic for streams in the J2O board with PCI backplane transfer functionality and are currently integrating the packet buffering and dispatch logic on the processor board with hardware acceleration for scheduling computations using the Celoxica Virtex board (Entity 1). Experimentation is also under way to use IXP 1200 processor boards for providing packet buffering and packet dispatch and FPGA interaction. Towards this end we are developing Linux system software with queue structures and packet dispatch logic on the host processor that can be ported to a Linux system on the IXP 1200 StrongArm with ease. Results from peer-peer PCI transfers between J2O boards on different PCI backplanes and description of queue structures implemented is provided in [1, 4]. Results attained with novel operating system facilities that have the express purpose of facilitating the dynamic quality management of end-user applications in OS kernels and next, in communication co-processors, are described in [8].

3.2.4 Status and Future Work

We are currently experimenting with the four stream hardware QoS scheduler (functional in FPGA hardware) for different usage scenarios and different input traffic patterns. Integration with the peer processor board on the PCI backplane will complete the end-system QoS architecture. Future implementations will have larger number of streams. Another aspect of this work is also to improve the timing of functional hardware schedulers to meet the packet time requirements of Infiniband and emerging 10G Ethernet networks. Critical path analysis, area reduction, retiming and pipelining of critical path circuits will be used to yield improved timing characteristics. We also hope to retarget our design to the Virtex II architecture [6] (with actual hardware multipliers) to allow more logic utilization and allow scheduling of a larger number of streams from a single chip package; PowerPC core (in the Virtex-II pro series) for instance, integrated in the same die can provide the control-flow logic needed by ASAN QoS schedulers.

List of Publications


4. Raj Krishnamurthy, Karsten Schwan, Richard West and Marcel Rosu, "A Network Co-


7. Raj Krishnamurthy, Sudhakar Yalamanchili, Karsten Schwan and R. West, "Scalable Architecture and Hardware for Scheduling Gigabit Packet Streams", Full paper is being prepared for submission.


3.3 Network Interface Based Computation

This component of our work explores mechanisms for the dynamic placement of computations on the NI. An issue with NI-based computation is the placement of functionality onto NIs, such that sufficient information about the data being sent and received is available so that data may be successfully and meaningfully operated on. We have addressed this issue in two ways: (1) by experimenting with the execution of higher levels of a protocol stack on the NI, such as fragmentation and reassembly, and (2) by placing application-specific information about the data being shipped onto NIs. For (2), we have developed runtime interfaces using which applications (not just the operating system) can extend OS kernels. These interfaces have been implemented for the Linux operating system. We are extending these interfaces "further", onto NIs, so that application-specified functionality can be moved, at runtime, onto NIs, and applied to the applications' data moving through the NIs. Finally and perhaps most importantly, we have developed novel programming models using which end user applications can define for and then apply to their application-level communication streams certain NI-based computations [5,6,7].

Early results with the IXP demonstrated the potential of this component for an Active System Area Network:

- We have ported an experimental quality-of-service packet scheduling algorithm, Window-based Rate Adaptive Scheduling (WRAP) to the micro-engines on the IXP processors.
Porting of a lightweight protocol layer (R-UDP) to the hardware micro-engines. This layer will provide low latency reliable message handling for ASAN streams, and provide an easy connection to our high-level application environment. The idea is to demonstrate our ability to apply application-specific operations on application-level packets in IXP-based NIs, at full link speeds.

Development of specific co-processor-based packet manipulations, such as packet mirroring (a paper on initial results attained on a cluster computer - not yet using the IXP boards - appeared in the 2001 HPDC conference[2]), packet splitting and striping across multiple remote machines (addressing large packets), and efficiently dealing with ubiquitous message formats like XML (a paper on this topic - not yet using the IXP boards - also appeared in the 2001 HPDC conference[1]).

Extension of the ASAN testbed environment to integrate the IXP1200 hardware and Virtex FPGA boards from Celoxica.

Development of a reconfigurable hardware "loader-linker" module, which supports multiple concurrent hardware modules. This software is being developed on the Strongarm core and will orchestrate the invocation of hardware processing in the ASAN system.

Development of 'safety support' for dynamic co-processor extension. This software will permit end users to extend R1 functionality such that new extensions cannot threaten the performance and/or correct functioning of currently running extensions.

We are currently working on several ASAN applications in areas including network intrusion detection, firewall filtering, remote visualization, and QoS scheduling.

The IXP testbed will serve as the platform for the implementation, testing, and validation of ASAN concepts.

List of Publications


