Master of Engineering Preliminary Thesis Proposal for 6.191

— Prototyping Research Results

Oskar Bruening

6th December 2002

Massachusetts Institute of Technology

Introduction

The speed at which computers run these days is amazing. According to Moore's law, computers double their speed every 18 month. This increase in processing speed is not due to one single factor, but many of them. Computers these days run at lower voltages and higher clock speeds. The size of the transistors has been decreasing with the increase in speed. Modern processors are stacked with speed-increasing tricks like multi-stage pipelines allowing the processor to run several instructions almost simultaneous. New technologies are explored in every possible direction. New architectures for future processors are developed everywhere. With Moore's law holding, people start asking more and more "how much longer?". As chips get smaller and more complex certain quantities still remain quasi-constant, like the wiring of a chip to its memory unit. In order to increase speed without necessarily decreasing the size of the chips, other approaches considered.

It appears like much research is focused on turning computers parallel. Of course, a computer with two chips can run twice as fast. The ideal of "the more processors, the better" still holds in modern super computers. However, by connecting thousands of individual processors to one giant computer, problems like efficient and equal distribution of processing jobs arise. An alternative to the supercomputers are the so called Beowulf clusters, usually a cheaper option. A Beowulf cluster consists of many individual computers connected together through a local area network (LAN) to one large identity. On Beowulf clusters the problem of efficient and equal job distribution is even more dramatic. Since the computers are connected through a network, which only has a limited bandwidth, communication between the different computers also becomes a major factor for these computations. In addition, for both, the super computer and the Beowulf cluster, the programmer has to have sufficient background knowledge on the working and the weaknesses of the network in order to be able to design the computational algorithm in a way that it can be executed in a parallel fashion. The development of efficient parallel algorithms, leaving no node idle is difficult and challenging work.

This is where the Raw Processor comes into play. The Raw chip is a completely new architecture. Unlike other architectures, like the x86, which are based on previous processors and carry many extra

functions with them, allowing for backwards compatibility, the Raw chip has been designed from scratch. The Raw chip uses the idea of parallel computation at a much lower level.

The Raw chip itself consists of a grid of 4-by-4 processors. Every processor consists of a processing and a local memory unit. In addition every processor in the Raw chip enjoys an additional feature unique to the Raw architecture. Every processor is equipped with a so called *switch*. The switch is a four directional bus connector, linking the processor with the four surrounding processors. This allows the processor to talk at the assembly level to its neighbors and therefore does not need to go through the different levels of networking, such as network protocols in order to speak with another neighbor. Communication and data sharing happens at instruction cycle speed. The switch is an integral part of the processor. It's instruction set, vaguely based on the clean RISC architecture instructions of the MIPS chips, is extended to allow for the given communication. In addition, the 16 tiles are easily extendable. Simply plug in another Raw chip and you have twice the computational power. When doing actual computation, those 16 tiles can then be split up to do several jobs. Four tiles could do this while an other 4 could do that to make the overall computation most efficient.

The Raw chip has been designed in corporation with the MIT Oxygen project. The chip is built for flexible signal processing. A Raw chip can be programmed to do almost any signal processing task, like a sound, video or TV card in a computer. This is where the fundamental difference of the Raw chip to other common processors lies. The Raw chip uses so called free-gates. The *hardwiring* of the chip can be programmed using human readable programming languages like Java and C. Consequently, for any signal processing job, the Raw chip is all you need.

Project Description

I will be designing and implementing the software end of a one-gigabit Ethernet router. Cemal Akcaba, also working for Prof. Agarwal, will be implementing the hardware related side on the evaluation board. The evaluation board consists of one Raw chip surrounded by FPGAs (Field Programmable Gate Arrays) to allow for hardware Raw-Pre-Processing. Cemal will in particular build an Ethernet packet processor, recognizing the individual packet frames.

A one-gigabit Ethernet packet processor is a very suitable application to test the performance of the Raw processor and will be primarily used as a demo, a proof of concept showing the capabilities of the new evaluation board and the Raw processor.

My project is to build the software running on Raw for the router. Cemal Akcaba will be building the Ethernet controller handing over identified packets from the Internet to the Raw chip. My software will take the packets, analyse the location to which they are going and send them out again through another Ethernet controller. It is important to understand the difference between a *hub* and a *switch* (the router is a specialized version of a switch). The hub simply connects all connections coming in with each other, like one big street intersection. When one end sends a package all other connectors receive. When two ends send packages into the connector the packages are lost and have to be resent. This is called a *collision*. It is clear that even though a hub might run well for small networks, the bigger the network, the more connection, the more collision, the slower the network.

Switches on the other side can be considered smart hubs. They do not simply connect all incoming lines, but rather accept all packages coming in, analyze where they are going using a look-up table and send them though the one and only link that leads to the target. Consequently, by not just connecting lines, but accepting, analyzing and sending packages, the router is like a small computer. The Raw chip is ideal for this kind of signal processing.

Assuming four connectors to the router Cemal Akcaba and I are building, the task can easily split up between the different tiles. One input tile will, if necessary, assist the FPGAs in recognizing and isolating the individual packages. A second output tile will then also be in charge of sending packages. The remaining eight tiles can then accept packages from any of the input tiles, analyze the packages in a *store-and-forward* fashion and redirect it to the corresponding output tile. Therefore, this Raw switch will act as type *shared-memory switch*, only now more than one package can be processed data time, avoiding the switch's own processing speed as bottleneck of the network.

The difference between a *hub*, a *switch* and a *router* is the level of abstraction it does it's analyses on. A hub works at the *physical level*, simply connecting wires. A switch works at the so called

Datalink level, looking at the packages and where they are going according to the MAC address. The router the package analyses runs at the *network level.* Instead of simply analyzing the MAC address and basing the redirect on this, the router looks at the actual IP address to do the redirecting, but does not differ otherwise from the operation of a switch. Running at the network level only adds slight complication, as the router needs to understand the network protocol (IP), while the switch only needs to deal with the package header embedded information.

Initially I intend to build a switch first and on successful completion extend it to a router. When converting the switch to a router, potentially the eight tiles used for analyses can be divided up in two or three for the package analysis and five or six tiles for the network protocol related analysis.

Project Break Up (and Milestones)

The project can be broken up into several parts. These parts will most likely be implemented in the suggested order, since it represents a logical proceeding. All sections marked as *'optional'* are not necessary to make a simple switch work, but present interesting and challenging expansions allowing the switch / router to compete (and be compared to) modern industrial devices. In addition, easy implementation of additional feature would proof the obvious flexible advantages of a Raw chip to VLSI designed chips.

In- and Output Tiles

These are the tiles that interface with Cemal Akcaba's Ethernet controller. Once we get started on the project a protocol between the Raw chip and the FPGAs has to be developed. The in and output tiles will implement this protocol allowing the Raw chip to receive, manipulate and send packages. This is a potential error 42 part of the project and therefore might be delayed to a later point in time, when the Ethernet controller is working. Until then, small simulations can be written faking the receiving and sending of packages to the remaining of the Raw chip to allow for development on those parts.

Package Analysis

This accounts for most of the work the Raw chip has to do. The analysis can be structured into individual subsections, each accounting for separate milestones in the project.

Header Recognition

The first thing to do with an incoming package is to extract its header and from the body. Further the target MAC address has to be identified. Since the format of a package is standardized this should not be a hard thing to do.

Target Identification

The Raw chip now needs to do a lookup to find the out where the package needs to go. This, two, the lookup of port to target MAC address is fairly simple. It could be made more complex by doing more regular expression like pattern matching instead of one-on-one matching.

Lookup Table Maintenance

Initially the switch does not know where to send the package since it does not know, which MAC address corresponds to which port. The lookup table by a method called *transparent bridging*. If the MAC address of an incoming packages does not yet exist in the the address is added and marked as being in the *learning state*. The package is then *flooded*, meaning it is sent to all available ports, except the one it came from. When then a package returns from that very same address, the port is marked and the link is established. From now on all incoming packages for that address will be *forwarded* through the right port. Using this set up also package *filtering* can also be implemented, by knowing which address go which way and if a package comes of which we know it came the wrong way, it can be canceled.

Spanning Trees (optional)

Switches work well in a *star-network*, but in *ring-networks* more than one way leads to a computer and the switch needs to take many possible routes into account that lead to the same target. Therefore the previously given set up will not work effectively. The Digital Equipment Corporation, therefore developed the *spanning tree protocol* (STP) standardized in the IEEE 802.1d. The spanning tree allows for multiple possible ports directing a package to the same address. Using the spanning tree algorithm (STA) the ports are weighted so the shorter connection to a certain address is favored over a longer one. In order to make the switch / router more useful for general applications, this part would be an interesting extension.

Router (optional)

There are when turning the switch into a router a few extra algorithms have to be implemented.

Internet Protocol

The *Internet Protocol* (IP) would have to be implemented. An algorithm, potentially running on its own tile, has to parse the incoming packages and recognize the IP addresses other other IP header related information. Since the protocol runs on top of the package layer, this information is not embedded in the package header, but parsed in its body. A look up tree (and spanning tree, if implemented) would also have to be added particular for the IP protocol. Additionally the router needs to be equipped with it's own IP address requiring a small IP engine, parsing the own IP address into the IP protocol in the packages.

Using this IP protocol in combination with the package redirecting would turn the switch then effectively into a router. However there are more options that make a router more useful than the switch.

Firewall Filtering (optional)

For network security reasons, modern routers are equipped with filters to reject unwanted packages. Implementing dynamic filters could potentially be an interesting part of the project.

Configuration Dialup (optional)

Since the router is part of the network with its own IP address, modern routers allow the user to dial up to them and configure them through the dialup. This would be another challenging extension.

VLAN Support (optional)

To organize large networks and large network grows, Virtual Local Area Networks (VLAN) have been developed allowing a number of computer connected through the router to be considered as own domain.

Resources Required

The single most important resource is a working, programmable Raw chip, ideally on an evaluation board. In addition, for software development a simulator of the raw chip is desirable. To my knowledge the simulator and the Raw compiler (rawcc) already exist and are therefore not a bottleneck.

Technical Risks

Not many technical risks exist. Most of the tools have already been developed. Only the evaluation board and the interface with the Ethernet controller developed by Cemal Akcaba could produce error #42 problems.

What's Next

During the last semester of my final year in spring 2003, I plan on working as a UROP for Prof. Agarwal. I will study the Raw chip and its possibilities and limitations in more detail and experiment with it to get a feeling for the chip. In addition, I will make more concrete outlines for the individual parts of the programs and start implementing smaller sections for testing purposes. By the end of the term I should have a good feeling about what and how it will actually be implemented for my M.Eng. thesis. During summer and Fall 2003 and spring 2004 I plan on staying here at MIT working on this thesis in order to graduate in June 2004.