Optimized Hardware Architecture and Communication Protocols for Static Routing

David Shoemaker
NE43-631
shoe@goldfish.lcs.mit.edu

1.0 NuMesh Background

NuMesh is a high-speed communication substrate optimized for static routing. Each NuMesh node contains a programmable finite state machine to mediate communications with neighboring nodes. Off-line analysis of internode communication provides separate communication instructions for each NuMesh node which are loaded during bootup. Each FSM provides control signals for its node during each cycle of a globally synchronous clock, allowing read and write operations to occur in a fixed cyclic pattern with a common period. A processor is attached to each NuMesh node for computation, while the NuMesh CFSM (Communication Finite State Machine) serves to provide a number of statically scheduled virtual communication channels between the processing elements. Each NuMesh node can be connected to up to six other nodes providing a base platform for a scalable parallel processing machine. Individual NuMesh modes can be connected like legos, with each NuMesh node allowed to have a different processing element. While not suited for general purpose applications, Numesh will provide exceptional performance for applications that take advantage of static communication. Because the requirement of off-line routing leads to no run-time routing decisions, the NuMesh architecture can be heavily pipelined resulting in very fast communication times. In addition, congestion delays can be minimized during compile time. This paper describes some of the hardware and software techniques that allow NuMesh to take advantage of static routing.

2.0 Hardware Overview

Previous NuMesh hardware prototypes exhibited a significant limitation. Keeping track of the state of the CFSM proved quite expensive. Since a number of different threads were being executed in a single CFSM, the state of the node required information proportional to the product of the state of all the individual threads. In addition, any allowed dynamic decision for a thread affected not only the thread in question but the state of the entire CFSM. The obvious solution to this is to break up the CFSM into a number of smaller FSMs with one controller assigned to each thread. This insight yields two architectural possibilities. First, the actual hardware could be replicated allowing true parallelism. Second, a single controller could be timesliced, operating on a different thread every cycle. The final design implements both of these ideas by having two physical controllers each operating on thirty-two virtual threads. The final design of one of the physical controllers is shown in figure 1.

Each stage of the pipeline is designed to maximize throughput. The delay is limited to either a small RAM read plus a register, or a node transfer plus a register. Any miscellaneous logic is completed either in parallel to these operations or in a separate stage. Because all the communication is choreographed at compile time, backwards compatibility for future hardware designs requiring more pipeline stages can be trivially handled in software by setting a compiler option. A brief description of the pipeline stages is discussed below:

1. **Schedule RAM** - The Schedule RAM consists of the virtual thread number for each pipeline, plus an index for the next schedule RAM instruction to be used. Its contents define an outer control loop of fixed size and determines what thread will be operated on for every clock cycle.
2. **Instruction RAM** - The instruction RAM contains a sixteen bit instruction that encodes the desired transfer for the thread. Every thread corresponds to exactly one instruction.
3. **Read Stage** - During this stage, one of the source ports is read, and the source data plus a valid bit is stored in a register. The source word may also come from one of several buffer registers stored on the node.
4. **Write Stage** - During this stage, the source word is written to the appropriate port destination or buffer register.

3.0 Communication Overview

By taking advantage of static routing, a fast and unique protocol can be utilized. Every cycle that a node is expecting to receive a word of data, it issues an *accept* bit to the node from which it expects to receive the data Similarly, every cycle a node expects to send data, it sends a *valid* bit to the receiving node. Therefore, every cycle, a communication can occur, and the success or failure of the transfer can be figured out at a later time. This allows a very fast cycle time, and a single cycle transfer for data. The only requirement is that both the sending and receiving nodes can act correctly based on the success or failure of a transfer. The protocol for each is described below:

1. Sending Node - The sending node looks at the *accept* bit to determine the success of its transfer. If the *accept* bit is set, the transfer completed successfully. If not, the transfer failed, and the sending node must try to send the data again later. The data word gets written into a buffer register for later use. There is one buffer register for every thread. The next cycle this thread operates, rather than reading the source word from a port, the thread takes its word from its buffer register.

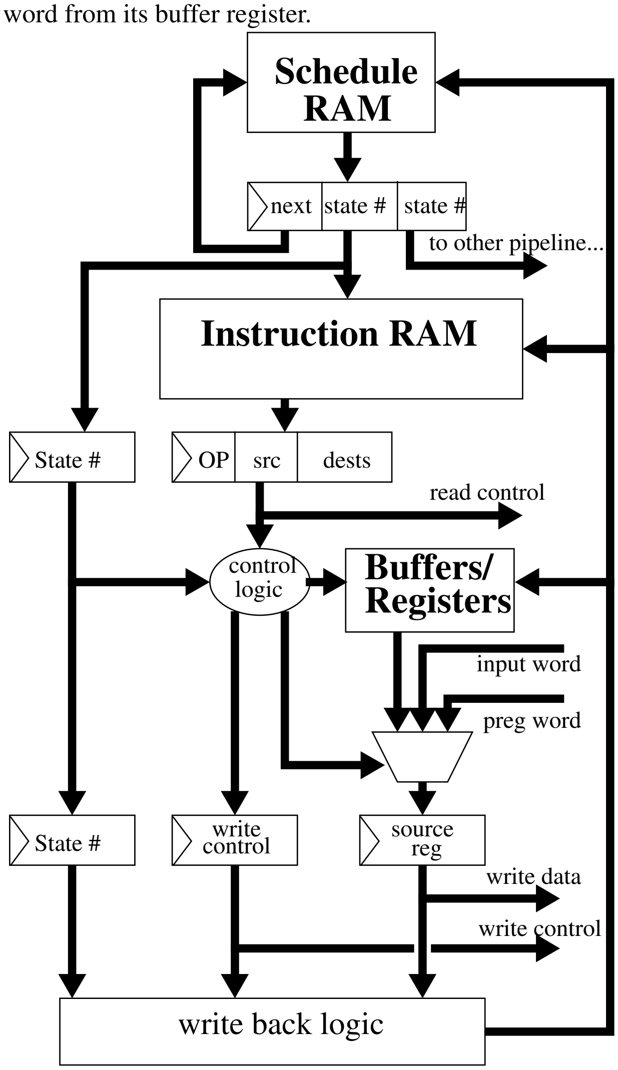

FIGURE 1. Hardware Architecture

2. Receiving Node - The receiving node determines if its data is valid by looking at the *valid* bit it receives from the sending node. The only requirement of the receiving node is that if it receives an invalid word, it ignores the word and does not try to pass it to another node on the following cycle.

Three types of communication are supported in the NuMesh system:

1. Flow Control - Flow control transfers utilize the protocol described above. They are expected to be the primary means of communication exercised.

2. Conditional Flow Control - The conditional flow control protocol will be used primarily for the transfer of packets. When reserving bandwidth for the transfer of a packet, the desired effect is to have either all or none of the packet sent. Similarly, if part of the packet gets buffered, then the entire packet should be put in buffer registers. To accomplish this, a conditional flow control instruction is executed. The effect of this instruction is to mimic the transfer success of the instruction from the previous cycle. For instance, if the previous word was buffered, the current word gets buffered regardless of the *accept* bit the node receives. Likewise, if the previous cycle failed to yield a valid source word, the current instruction will assume that no valid word is received without looking at the desired port. When sending a packet, the header transfer thread would be a normal flow control instruction, while the body words would be transferred with conditional flow control instructions.

3. No Flow Control - Flow control can also be turned off resulting in data being blindly transferred from node to node. The validity of such data can be examined by the processor or completely ignored in the case of purely static models such as simple video streams.

4.0 Status

Currently, the NuMesh architecture is fully designed, and an integrated gate array chip is being implemented. A fully functional simulator has been written to test models of communication. The first complete application we have working on the system is the Travelling Salesman Problem. Although, this program uses diffusion scheduling - primarily a dynamic scheduling algorithm - it was shown to work effectively when forced into a static model. Ideas for a video board and for significant software support have also been initiated.

References:

Ward, Steve, et al, "The NuMesh: A Modular, Scalable, Communication Substrate", *International Conference on Supercomputing,* July 1993.