ECE 697J – Advanced Topics in Computer Networks

Network Processor Architectures

10/16/03
NP Architectures

• Last class:
  – Discussion of various functionalities that NP should implement
  – Question of general-purpose vs. hardware implementation

• This class:
  – More details on the internals of NPs
  – Main characteristics of different architectures
NP Architectures

• Large variety of possible NP architectures
  – Taxonomy difficult
• Instead: description by primary characteristics
  – Processor hierarchy
  – Memory hierarchy
  – Internal transfer mechanism
  – External interface and communication mechanism
  – Special-purpose hardware
  – Concurrent execution support
  – Programming model
• We will use these characteristics to understand and describe commercial architectures next week
• What is a processor hierarchy? Why would it be used?
• Processor hierarchy accommodates tasks of different complexity and different frequency
  – Low level of hierarchy: simple, frequent processing
  – High level of hierarchy: occasional, complex processing
• What kind of levels can we find on a router?
  – Several levels of data path processing
  – Several levels of control path processing
Hierarchical Functionality

- General purpose CPU:
  - Highest level functionality
  - Administrative interface
  - System control
  - Overall management functions
  - Routing protocols
  - Intermediate functionality
  - Higher-layer protocols
  - Control of I/O processors
  - Exception and error handling
  - High-level ingress (e.g., reassembly)
  - High-level egress (e.g., traffic shaping)

- Embedded processor:
  - Basic packet processing
  - Classification
  - Forwarding
  - Low-level ingress operations
  - Low-level egress operations

- I/O processor:
## Processor Hierarchy

- Possible levels on NP:

<table>
<thead>
<tr>
<th>Level</th>
<th>Processor Type</th>
<th>Programmable?</th>
<th>On Chip?</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>General purpose CPU</td>
<td>yes</td>
<td>possibly</td>
</tr>
<tr>
<td>7</td>
<td>Embedded processor</td>
<td>yes</td>
<td>typically</td>
</tr>
<tr>
<td>6</td>
<td>I/O processor</td>
<td>yes</td>
<td>typically</td>
</tr>
<tr>
<td>5</td>
<td>Coprocessor</td>
<td>no</td>
<td>typically</td>
</tr>
<tr>
<td>4</td>
<td>Fabric interface</td>
<td>no</td>
<td>typically</td>
</tr>
<tr>
<td>3</td>
<td>Data transfer unit</td>
<td>no</td>
<td>typically</td>
</tr>
<tr>
<td>2</td>
<td>Framer</td>
<td>no</td>
<td>possibly</td>
</tr>
<tr>
<td>1</td>
<td>Physical transmitter</td>
<td>no</td>
<td>possibly</td>
</tr>
</tbody>
</table>
Memory Hierarchy

• What is a memory hierarchy? Why is it used?
• Different memory technologies have different performance characteristics:
  – SRAM: access time 1.5-10 ns;
  – DRAM: access time 50-70 ns
  – Size on chip: SRAM size 10x that of DRAM
• Fast memory is expensive
  – Small on-chip memory
• Slow memory is cheap
  – Large off-chip memory
• Memory hierarchy gives good performance/cost trade-off
• Memory hierarchy is not “cached,” but used explicitly
  – Why?
Memory Hierarchy

Example:

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Rel. Speed</th>
<th>Approx. Size</th>
<th>On Chip?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control store</td>
<td>100</td>
<td>$10^3$</td>
<td>yes</td>
</tr>
<tr>
<td>G.P. Registers†</td>
<td>90</td>
<td>$10^2$</td>
<td>yes</td>
</tr>
<tr>
<td>Onboard Cache</td>
<td>40</td>
<td>$10^3$</td>
<td>yes</td>
</tr>
<tr>
<td>Onboard RAM</td>
<td>10</td>
<td>$10^3$</td>
<td>yes</td>
</tr>
<tr>
<td>Static RAM</td>
<td>5</td>
<td>$10^7$</td>
<td>no</td>
</tr>
<tr>
<td>Dynamic RAM</td>
<td>1</td>
<td>$10^8$</td>
<td>no</td>
</tr>
</tbody>
</table>

What data should be stored where?
- Packets?
- Queue structures?
- Routing table?
Memory Speed

• Two “speed” characteristics
  – Access speed
  – Memory bandwidth

• What is more important?
  – Access speed for random accesses
  – Bandwidth for large transfers

• Choice of memory depends on usage
  – DRAM for packets
  – SRAM for tables, etc.
Processing and Memory Hierarchies

- Processing scalability/parallelism:

- Memory size:
Internal Transfer Mechanisms

• NPs have several components that need to “talk” to each other:
  – Processors with co-processors
  – Processors with packet memory
  – Processors with control processor

• What kind of transfer mechanisms could be used?
  – Internal bus
  – Hardware FIFO
  – Transfer registers
  – Onboard shared memory
Transfer Mechanisms

• Internal Bus
  – Normal bus with arbiter to control access
• Hardware FIFO
  – First-In-First-Out storage in hardware
  – Each slot has fixed size (e.g., 64 bytes)
  – Used to forward packet pointers
• Transfer Registers
  – Similar to hardware FIFO, but without sequential access
  – Read and write from/to any register possible
• Shared Memory
  – Large shared buffer with random access by everybody
• What are the pros and cons of each approach?
External Interfaces

• Several external interfaces on a NP chips
• Memory interfaces
  – Access to large off-chip storage
  – Used for packets and large tables
• Direct I/O interfaces
  – Access to link interfaces
  – Also: low-speed control over serial interface
• Bus interfaces
  – Access to “other” devices: control CPU, etc.
  – PCI bus or similar
• Switching fabric interface
  – Access to switching fabric
  – Several standards available (e.g., CSIX by NP Forum)
Specialized Hardware

- Discussed last class
- Also, a few hardware components that control and coordinate internal mechanisms on NP
Concurrent Execution

- Support for concurrent execution on processors
  - Thread support in hardware
- Threads require state (=register file)
  - Registers
  - Control information (program counter, etc.)
- Threads switching
  - When stall occurs
  - Due to preemption
- Thread switching overhead
  - Most NPs can switch in “zero cycles”
- Different from software threads(!)
Programming Model

• Interrupt/event-driven vs. thread-based
  – Discussed before
• Dispatch mechanisms
  – Defines how threads are initiated
• Implicit vs. explicit parallelism
  – Different abstraction for programmer
Parallelism

• Different ways of exploiting parallelism on architectural level
  – How can processors be arranged to exploit parallel processing?
• Simple parallel architecture
  – Processors are replicated and do the same
• Pipelined architecture
  – Processors do partial work
  – Packets are being sent through
• Hybrid
  – Combination of parallel and pipelined
Parallel Architectures

- **Workload:**

- **Parallel:**

- **Pipelined:**
Limitations on Scale

- NPs cannot be built arbitrarily large
- Physical space limits
  - Chip no larger than 400mm² (typically much smaller)
- Clock skew and propagation limits
  - Smaller feature size and larger chips make problem worse
- Pin limits
  - Packaging technology (pins need some room)
  - Connections from chip to package
- Internal and external communication limits
  - Pin limits and frequency limits
- Power consumption and heat limits
  - How much power does an NP consume?
Power Density

by Fred Pollack
Lab Project 1

- TCP/IP Flow Identification
- Your program should read a trace of packets and tell the following statistics at the end of the run:
  - Number of flows observed. That is the number of unique 5-tuples consisting of IP source and destination, layer 4 protocol number, and source and destination ports.
  - Number of properly terminated TCP flows.
  - Maximum number of active flows at any given point.
  - 5-tuple of flow with most data transferred.
- We provide you with support functions to read and write packet trace
  - You just need to write “packet processing code”
Lab Project 1

• How do you identify flows?
• How do you keep track of the flow “state”?
Lab Project 1

- You need to use a UNIX/Linux/Cygwin environment
- Your program should be written in C
- Files from course homepage:

$ ls -l
 total 10776
-rw-r--r-- 1 wolf None 87164 Oct 16 11:11 AIX-1058228090-1.tsh
-rw-r--r-- 1 wolf None 451 Oct 16 10:33 Makefile
-rw-r--r-- 1 wolf None 10930612 Oct 16 11:11 OSU-1058584197.tsh
-rw-r--r-- 1 wolf None 7556 Oct 16 11:13 bench.c
-rw-r--r-- 1 wolf None 1589 Oct 16 11:10 bench.h
-rw-r--r-- 1 wolf None 1993 Oct 16 11:16 flowid.c
-rw-r--r-- 1 wolf None 252 Oct 16 11:14 flowid.h
-rw-r--r-- 1 wolf None 991 Oct 14 17:32 instructions.txt
Lab Project 1

• Modify file “flowid.c”:
• Three key functions for initialization, processing, and end
  • init_flowid()
    – Initialize any data structures that you need
  • int flowid(packet *pass_packet)
    – Your packet processing code
    – Packet is in memory that pass_packet points to
  • finish_flowid()
    – Called at the end of the trace, so you can output results
• Compilation and running is explained on handout
Lab Project 1

• You should turn in a brief (around 2-4 pages) report that contains the following:
  – A diagram showing the fields of and IP and a TCP header.
  – A diagram of a TCP state machine that you use to determine the state of a connections.
  – The results from your measurement (see above) for both sample traces.
  – A printout from your flowid.c and flowid.h file (does not count towards page count).

• Lab 1 is due 10/30/03 in class
  – Help/discussion session 10/28/03 if needed
Next Class

• Next Class: commercial NPArchitecture
  – Read Chapter 15

• Everybody Volunteers get to present one:
  – Multi-Chip Pipeline by Agere
  – Augmented RISC Processor by Alchemy
  – Embedded Processor Plus Coprocessor by AMCC
  – Pipeline of Homogeneous Processors by Cisco
  – Configurable Instruction Set Processor by Cognigine
  – Pipeline of Heterogeneous Processors by EZchip
  – Extensive and Diverse Processors by IBM
  – Flexible RISC Plus Coprocessor by Motorola