ECE 697J – Advanced Topics in Computer Networks

ACE Programming Model and SDK
11/13/03
Overview

• Programming Model
  – Active Computing Element (ACE) Abstraction
  – Allocation of ACEs to microengines
  – Packet Queues

• Software Development Kit
  – Simulator
  – Example: IP forwarding

• Lab 2: IP forwarding and classification on IXP1200
Last Class

- Active Computing Element (ACE) abstraction:
Microengine Assignment

- Packet processing involves several microblocks
- How should microblocks be allocated to microengines?
  - One microblock per microengine
  - Multiple microblocks per microengine (in pipeline)
  - Multiple pipelines on multiple microengines
- What are pros and cons?
  - Passing packets between microengines incurs overhead
  - Pipelining causes inefficiencies if blocks are not equal in size
  - Multiple blocks per microengine causes contention and requires more instruction storage
- Intel terminology: “microblock group”
  - Set of microblock running on one microengine
Microblock Groups

- Microblock groups can be replicated to increase parallelism
Microblock Group Replication

- Performance critical groups can be replicated:

  - Single core component (not replicated) communicates with multiple groups
  - Multiple inputs, multiple output

- Additional complexity:
Control of Packet Flow

• Packets require different processing blocks
  – IP requires different microblocks than ARP
  – Special packets get handed off to core

• “Dispatch Look” control packet flow among microblocks
  – Each thread runs its own dispatch loop
  – Infinite loop that grabs packets and hands them to microblocks
  – Return value from microblock determines the next step

• Invocation of microblock is similar to function call
Dispatch Loop

• Example:
  - Two microblocks (ingress + IP)

```c
while (1) {
    Get next packet from input device(s);
    Invoke ingress microblock;
    if ( return code == 0 ) {
        Drop the packet;
    } else if ( return code == 1 ) {
        Send packet to ingress core component;
    } else {
        /* IP packet */
        Invoke IP microblock;
        if ( return code == 0 ) {
            Drop packet;
        } else if ( return code == 1 ) {
            Send packet to IP core component;
        } else {
            Send packet to egress microblock;
        }
    }
}
```
Dispatch Loop Conventions

• Parameters passed to microblock:
  – Buffer handle for frame that contains a packet
  – Set of state registers that contain information about the frame
  – A variable called dl_next_block in which return value gets stored

• State registers:
  – Information about packet: length
  – Information generated by software: classification result
  – Registers can be changed by microblock

• Return values:
  – Meaning assigned by programmer
  – Conventions: zero = “drop packet”, other values for “pass on” and “send to core” etc.
Packet Queues

- Packet flow depends on packet data
- Processing time depends on packet data
- Packet movement can’t be predicted
  - Microblocks need to continue processing without waiting
- Packets need to be buffered
  - “Communication Queues”
  - Unidirectional FIFO (yes, really FIFO)
  - Bidirectional communication requires two queues
- Also between microblocks and core
  - Single queue for all microblock group instances
  - Uses exception mechanism “IX_EXCEPTION”
  - Exception handler in core determines further steps
Packet Queue Example
Crosscalls

- Mechanism for non-packet communication between ACEs
  - Similar to remote procedure calls and remote method invocations
- Caller and callee need to agree on parameters
  - Interface Definition Language (IDL) specifies details
  - IDL compiler creates “stubs” to handle marshaling
- Types of crosscalls
  - Deferred: caller does not block, asynchronous notification
  - Oneway: caller does not block, no return value
  - Twoway: caller blocks, callee returns value
- ACEs are prohibited from twoway calls
  - No blocking allowed
- Other control software (non-ACE) may use all types
## SDK

- **Software Development Kit:**

<table>
<thead>
<tr>
<th>Software</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>C compiler</td>
<td>Compile C programs for the StrongARM</td>
</tr>
<tr>
<td>NCL compiler</td>
<td>Compile NCL programs for the StrongARM</td>
</tr>
<tr>
<td>MicroC compiler</td>
<td>Compile C programs for the microengines</td>
</tr>
<tr>
<td>Assembler</td>
<td>Assemble programs for the microengines</td>
</tr>
<tr>
<td>Simulator</td>
<td>Simulate an IXP1200 to debug code</td>
</tr>
<tr>
<td>Downloader</td>
<td>Load software into the network processor</td>
</tr>
<tr>
<td>Monitor</td>
<td>Communicate with the network processor and interact with running software</td>
</tr>
<tr>
<td>Bootstrap Reference Code</td>
<td>Start the network processor running</td>
</tr>
<tr>
<td>Example programs for the IXP1200 that show how to implement basic functions</td>
<td></td>
</tr>
</tbody>
</table>
Software Setup

Figure 3: Linux software configuration for the Intel IXDP1200 Advanced Development Platform using only a Windows NT development environment

I XM1200 Network Processor Base Card

Windows NT platform

Microengine Toolchain: I XP1200 Microengine Development Environment

StrongARM Toolchain:
GNU C/C++ cross-compiler under Cygwin Linux IDE (optional)

Figure 4: Linux software configuration for the Intel IXDP1200 Advanced Development Platform using both Windows NT and Linux development environments

I XM1200 Network Processor Base Card

Windows NT platform

Microengine Toolchain: I XP1200 Microengine Development Environment

StrongARM Toolchain:
GNU C/C++ cross-compiler Linux IDE (optional)

Linux platform
Simulator

- Cycle-accurate simulation of IXP1200
- Allows for easy experimentation
  - Packet generator
  - Visualization for thread behavior, memory accesses
  - Runs under Windows
- We will use simulator for Lab 2
  - Part I: run existing IP forwarding example, collect statistics
  - Part II: make a minor modification for classification
- We have lab machines set up for you
  - You can also install simulator on your own machine (big!)
IP Forwarding Example

• Full-blown RFC1812-compliant IP forwarding
  – Lots of special cases
  – Look for main program structure
  – 4 uE for IP processing (0-3)
  – 3 uE for output queuing (4-5)

• Run program and collect workload statistics
  – Thread behavior
  – Memory accesses
  – Instruction coverage
  – Etc.
Tilman Wolf 20
```c
// main loop
while (1)
{
    xbuf_alloc($pop_xfer, 1):
    if (packet_buf_addr == UNALLOCATED)
        buf_pop($pop_xfer[0], FREELIST_HANDLE, sig_done); // if
    endif

    port_rxrdy_chk(@rdready_inflight, rec_req);
    critsect_enter[@req_inflight] ; block other contexts from sending
    port_rx_request(rec_req); // get mpacket

#ifdef RFC1812
    port_rx_receive(exception, recv_port, rec_state, ETHER_100M); // get mpacket status
#else
    port_rx_receive(exception, rec_state, ETHER_100M); // get mpacket status
#endif //RFC1812

    mpacket_received#:
    if (packet_buf_addr == UNALLOCATED)
        buf_wait();
        #if (FREELIST_ID == 0)
            #define BASE_ADDR SRAM_BUFF
        #else
            #define BASE_ADDR RAM_BUFF
        #endif
    while ($pop_xfer[0] == BASE_ADDR)
        buf_pop($pop_xfer[0], FREELIST_HANDLE, ctx_swap); // if no
    endwhile
    buf_dram_addr_from_sram_addr(packet_buf_addr, $pop_xfer[0], FREELIST_HANDLE);
    move(descriptor_addr, $pop_xfer[0]);
    endif
    xbuf_free($pop_xfer[0]);
}```
Performance Statistics

Statistics were gathered starting at cycle 1.

Show complete list   Show customized list

Select statistic:
- Latency distribution for SDRAM ref's returning data to f4
- Latency distribution for SDRAM ref's returning data to f5
- Latency distribution for SDRAM ref's returning data to f6
- Latency distribution for SDRAM non-read lock ref's returning data to f7
- Latency distribution for SDRAM non-read lock ref's returning data to f8
- Latency distribution for SDRAM non-read lock ref's returning data to f9
- Latency distribution for SDRAM non-read lock ref's returning data to f10

Total samples: 122. Min = 0, Max = 299, Ave = 36

<table>
<thead>
<tr>
<th>Cycles</th>
<th># of Samples</th>
<th>Percent</th>
<th>Cumulative percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>16</td>
<td>1.39</td>
<td>1.39</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>8.32</td>
<td>21.87</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>1.39</td>
<td>21.87</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>0.75</td>
<td>32.62</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>0.75</td>
<td>32.62</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>0.75</td>
<td>32.62</td>
</tr>
<tr>
<td>22</td>
<td>9</td>
<td>7.44</td>
<td>35.15</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>0.75</td>
<td>35.15</td>
</tr>
</tbody>
</table>
if (packet_buf_addr == UNALLOCATED)
buf_wait();
#if (FREELIST_ID == 0)
#define BASE_ADDR SRAM_BUFF_DESCRIPTOR_BASE
#else
#define BASE_ADDR (SRAM_BUFF_DESCRIPTOR_BASE + (HALF_BUFFER_COUNT * 4))
#endif
while ($pop_xfer[0] == BASE_ADDR)
buf_pop($pop_xfer[0], FREELIST_HANDLE, ctx_swap);  // if no
.endw
buf_dram_addr_from_sram_addr(packet_buf_addr, $pop_xfer[0], FREELIST_HANDLE);
move(descriptor_addr, $pop_xfer[0]);
Lab 2

• Part I: Collect statistics
  – Microengine utilization for all microengines
  – Detailed statistics of one thread from uE 0 and one from uE 5
  – Processing power of microengines (in MIPS).
  – Memory utilization and bandwidth.
  – Latency distribution for SDRAM refs for microengine 0 and SRAM non-read_lock refs for microengine 0. Show a graph.
  – Show a screenshot for the thread history that shows overlapping SRAM and SDRAM requests by the same microengine.
  – Identify the overall delay for either request (in cycles). What factors contributed how much to the overall delay?

• DUE NEXT TUESDAY.
Lab 1 Results

• Grading: 20 points total
  – Results: 10 points
  – Code: 3 points
  – TCP state machine + explanation: 2+1 points
  – IP and TCP headers: 1+1 points
  – Report (written content): 2 points

• Average: 16.6
• Max: 20
• Min: 14
Next Class

- Microengine programming
  - Assembler
  - Instructions
  - Register access
  - Assembler directives
  - Etc.

- Read Chapter 24
- Turn in Part I of Lab 2