Network Processors: Building Block for Programmable High-Speed Networks

Introduction to the Intel IXA

- Shiv Kalyanaraman
- Yong Xia (TA)
- shivkuma@ecse.rpi.edu
- http://www.ecse.rpi.edu/Homepages/shivkuma
What do switches/routers look like?

Access routers
e.g. ISDN, ADSL

Core router
e.g. OC48c POS

Core ATM switch

Shivkumar Kalyanaraman
Cisco GSR 12416

Capacity: 160Gb/s
Power: 4.2kW

Juniper M160

Capacity: 80Gb/s
Power: 2.6kW
Where high performance packet switches are used

- Carrier Class Core Router
- ATM Switch
- Frame Relay Switch

The Internet Core

Edge Router

Enterprise WAN access & Enterprise Campus Switch
Where are routers?
Ans: Points of Presence (POPs)
Why the Need for Big/Fast/Large Routers?

- **Interfaces**: Price >$200k, Power > 400W
- Space, power, interface cost economics!
- About 50-60% of i/ifs are used for interconnection within the POP.
- Industry trend is towards large, single router per POP.
What’s a Network Processor

- Router vendors have built speed into their devices by pushing functionality down into hardware (ASICs).
  - **ASIC:** Application Specific Integrated Circuits
- Fast but custom-made => expensive
- Long time-to-market

*Network processors look to avoid these pitfalls by introducing specialized, software controlled devices that can be customized quickly. But they also process packets at near-wire speeds!*
How does the IXA simplify the ASIC based design?

- A Typical ASIC Based Design
  - A processor to handle routing information and higher level processing
  - ASICs to handle each packet
- An IXP 1200 Design
  - StrongArm Core to handle routing algorithms and higher level processing
  - Microengines to handle packet processing
Applications of Network Processors

- Fully programmable architecture
  - Implement any packet processing applications
  - Examples from customers
    - Routing/switching, VPN, DSLAM, Multi-service switch, storage, content processing
    - Intrusion Detection (IDS) and RMON
- Use as a research platform
  - Experiment with new algorithms, protocols
- Use as a teaching tool
  - Understand architectural issues
  - Gain hands-on experience with networking systems
Intel IXP Network Processors

- **Microengines**
  - RISC processors optimized for packet processing
  - Hardware support for multi-threading
- **Fast path**
- **Embedded StrongARM/Xscale**
  - Runs embedded OS and handles exception tasks
  - Slow path, Control plane

**ME 1**
**ME 2**
**…**
**ME n**

**StrongARM**

**Control Processor**

**Media/Fabric Interface**

**SRAM**

**DRAM**

**Shivkumar Kalyanaraman**
Packet Flow Diagram: IXP 1200

- Optional Host CPU
- PCI Bus Devices
  - PCI Bus: 32-bit @ 66 MHz
  - PCI Bus Interface
- SDRAM
  - up to 256 Mb
- SRAM
  - up to 8 Mb
- FlashROM
  - up to 8 Mb
- Memory Mapped I/O Device (ex MAC Control)
- IX Bus Interface
  - IX Bus: 64-bit @ 66 MHz (up to 85 MHz, point-to-point)
  - 10/100/1000 Mb Ethernet MACs
  - ATM, T1/E1, SONET, xDSL, Etc.
  - Other IXP1200 Network Processor
- StrongARM* Core
  - @ 200 MHz

Source: Level One 1999

Shivkumar Kalyanaraman
The IXP 1200 product line represents Intel’s first attempt in the area (it was actually inherited when they purchased Digital).

The IXP 1200 is a single-board chip, designed with abstractions in mind.

Since this is a new area, and it’s designed to be used with many different types of hardware and software, the documentation is sketchy.

To achieve wire-fast speeds with software, the goal is to hide latency with parallelism. Processing packets is inherently parallel, and necessary for fast applications.
**Intel’s Gear (2)**

- **IXP2850**
  - Designed for use in virtual private networks, secure web services, and storage area networks.

- **IXP2800**
  - Able to handle line rates ranging from OC-48 to OC-192.

- **IXP2400**
  - Designed for OC-12 to OC-48 network access and edge applications.
Various forms of Processors

Embedded Processor (run-to-completion)

Parallel architecture

Pipelined Architecture
Intel Internet Exchange Architecture

- **Micro-engine technology** — a subsystem of programmable, multi-threaded RISC micro-engines that enable high-performance packet processing in the data plane through Intel® Hyper Task Chaining. This multi-processing technology features software pipelining and low-latency sequence management hardware.

- **The Intel IXA Portability Framework** — an easy-to-use modular programming framework providing the advantages of software investment protection and faster time-to-market through code portability and reuse between network processor-based projects, in addition to future generations of Intel IXA network processors.

- **Intel® XScale™ technology** — providing the highest performance-to-power ratio in the industry.
IXP: A Building Block for Network Systems

- Example: IXP2800
  - 16 micro-engines + XScale core
  - Up to 1.4 Ghz ME speed
  - 8 HW threads/ME
  - 4K control store per ME
  - Multi-level memory hierarchy
  - Multiple inter-processor communication channels

- NPU vs. GPU tradeoffs
  - Reduce core complexity
    - No hardware caching
    - Simpler instructions → shallow pipelines
  - Multiple cores with HW multithreading per chip
XScale Core processor

- Compliant with the ARM V5TE architecture
  - support for ARM’s thumb instructions
  - support for Digital Signal Processing (DSP) enhancements to the instruction set
  - Intel’s improvements to the internal pipeline to improve the memory-latency hiding abilities of the core
  - does not implement the floating-point instructions of the ARM V5 instruction set
Microengines – RISC processors

- IXP 2800 has 16 microengines, organized into 4 clusters (4 MEs per cluster)
- ME instruction set specifically tuned for processing network data
- 40-bit x 4K control store
- Six-stage pipeline in an instruction
  - On an average takes one cycle to execute
- Each ME has eight hardware-assisted threads of execution
  - Can be configured to use either all eight threads or only four threads
- The non-preemptive hardware thread arbiter swaps between threads in round-robin order
Why Multi-threading?

- Executing code
- Waiting for signal
- Ready to execute

Microengine

Microengine thread

Time

t1 t2 t3
Packet processing using multi-threading within a MicroEngine

Execution Time = $8 \times T_a$

Packet $n$  Thread 0
Packet $n+1$  Thread 1
Packet $n+2$  Thread 2
Packet $n+3$  Thread 3
Packet $n+4$  Thread 4
Packet $n+5$  Thread 5
Packet $n+6$  Thread 6
Packet $n+7$  Thread 7
Registers available to each ME

- Four different types of registers
  - general purpose, SRAM transfer, DRAM transfer, next-neighbor (NN)
- 256, 32-bit GPRs
  - can be accessed in thread-local or absolute mode
- 256, 32-bit SRAM transfer registers.
  - used to read/write to all functional units on the IXP2xxx except the DRAM
- 256, 32-bit DRAM transfer registers
  - divided equally into read-only and write-only
  - used exclusively for communication between the MEs and the DRAM
- Benefit of having separate transfer and GPRs
  - ME can continue processing with GPRs while other functional units read and write the transfer registers
Hardware Features to ease packet processing

- **Ring Buffers**
  - For inter-block communication/synchronization
  - Producer-consumer paradigm

- **Next Neighbor Registers and Signaling**
  - Allows for single cycle transfer of context to the next logical micro-engine to dramatically improve performance
  - Simple, easy transfer of state

- **Distributed data caching within each micro-engine**
  - Allows for all threads to keep processing even when multiple threads are accessing the same data
# Different Types of Memory

<table>
<thead>
<tr>
<th>Type of Memory</th>
<th>Logical width (bytes)</th>
<th>Size in bytes</th>
<th>Approx unloaded latency (cycles)</th>
<th>Special Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local to ME</td>
<td>4</td>
<td>2560</td>
<td>3</td>
<td>Indexed addressing post incr/decr</td>
</tr>
<tr>
<td>On-chip scratch</td>
<td>4</td>
<td>16K</td>
<td>60</td>
<td>Atomic ops 16 rings w/at. get/put</td>
</tr>
<tr>
<td>SRAM</td>
<td>4</td>
<td>256M</td>
<td>150</td>
<td>Atomic ops 64-elem q-array</td>
</tr>
<tr>
<td>DRAM</td>
<td>8</td>
<td>2G</td>
<td>300</td>
<td>Direct path to/from MSF</td>
</tr>
</tbody>
</table>
IXA Software Framework

External Processors

Control Plane Protocol Stacks

Control Plane PDK

XScale™ Core

Core Components

Core Component Library

Resource Manager Library

Microengine Pipeline

Microblock Library

Microblock

Microblock

Microblock

Protocol Library

Utility Library

Hardware Abstraction Library
Micro-engine C Compiler

- C language constructs
  - Basic types,
  - pointers, bit fields
- In-line assembly code support
- Aggregates
  - Structs, unions, arrays

Microengine C

- This code is used to populate CRC16 table for CRC calculation
  unsigned
  crc_table(unsigned int l) int

```c
{
  unsigned int j, k, crc = 0;
  k = i << 8;
  for(j = 0; j < 8; j++)
    if((crc ^ k) & 0x0000)
      crc = (crc << 1) ^ 0x1021;
    else crc <<= 1;
  k <<= 1;
}
return crc;
```

Optimized code generated by the Microengine C compiler

Example shows:
50% fewer lines of code required than with assembly

For simplified, portable, microengine programming!

Microcode

- This code is Microcode equivalent of the CRC calculation to the left

<table>
<thead>
<tr>
<th>crc_table#:</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu_shf[B1, ←, B, A0, &lt;&lt;8]</td>
</tr>
<tr>
<td>immed[A0, 0, 0]</td>
</tr>
<tr>
<td>immed[B0, 0, 0]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I_3#:</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu[A2, B1, XOR, A0]</td>
</tr>
<tr>
<td>br searchText[A2, 15, I_4#]</td>
</tr>
<tr>
<td>alu_shf[A0, ←, B, A0, &lt;&lt;1]</td>
</tr>
<tr>
<td>br[↑_5#]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I_4#:</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu_shf[B2, ←, B, A0, &lt;&lt;1]</td>
</tr>
<tr>
<td>immed[A0, 4129, 0]</td>
</tr>
<tr>
<td>alu[A0, B2, XOR, A0]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I_6#:</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu_shf[B1, ←, B, B1, &lt;&lt;1]</td>
</tr>
<tr>
<td>alu[B0, B0, +, 1]</td>
</tr>
<tr>
<td>alu[←, B0, - , 8]</td>
</tr>
<tr>
<td>brl=cout[I_3#]</td>
</tr>
<tr>
<td>alu[B0, ←, B, A0]</td>
</tr>
<tr>
<td>rtn[A1]</td>
</tr>
</tbody>
</table>
Core Components and Microblocks

XScale™ Core Components:
- Core Component
- Core Component
- Core Component

Core Component Library

Resource Manager Library

Microblock Library

Microblock

Microblock

Microblock

Legend:
- Microblock Library
- Intel/3rd party blocks
- User-written code
- Core Libraries
What is a Microblock

- Data plane packet processing on the microengines is divided into *logical functions called microblocks*.
- Coarse Grained and stateful.
- Example:
  - 5-Tuple Classification, IPv4 Forwarding, NAT.
- Several microblocks running on a microengine thread can be combined into a *microblock group*.
  - A microblock group has a *dispatch loop* that defines the dataflow for packets between microblocks.
  - A microblock group runs on each thread of one or more microengines.
- Microblocks can send and receive packets to/from an associated Xscale Core Component.
Technical and Business Challenges

- **Technical Challenges**
  - Shift from ASIC-based paradigm to software-based apps
  - Challenges in programming an NPU
  - Trade-off between power, board cost, and no. of NPUs
  - How to add co-processors for additional functions?

- **Business challenges**
  - Reliance on an outside supplier for the key component
  - Preserving intellectual property advantages
  - Add value and differentiation through software algorithms in data plane, control plane, services plane functionality
  - Must decrease time-to-market (TTM) to be competitive
For more info....

- Jonathan Gunner (gunnej@rpi.edu)
- Slide Contributions from Kerry Wood and Shruti Gorappa
- OGI IXA course: http://www.thefengs.com/wuchang/work/cse58x_spring2003/