Introduction to Network Processors: Building Block for Programmable High-Speed Networks

Example: 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**

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
Dimensions, Power Consumption

Cisco GSR 12416
- Capacity: 160Gb/s
- Power: 4.2kW

Juniper M160
- Capacity: 80Gb/s
- Power: 2.6kW

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
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

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
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/fs are used for interconnection within the POP.
- Industry trend is towards large, single router per POP.
Modern router architectures

- Split into a fast path and a slow path
- Control plane
  - High-complexity functions
  - Route table management
  - Network control and configuration
  - Exception handling
- Data plane
  - Low complexity functions
  - Fast-path forwarding
Design choices for network products

- General purpose processors (GPP)
- Embedded RISC processors
- Network processors
- Field-programmable gate arrays (FPGAs)
- Application-specific integrated circuits (ASICs)

![Diagram showing speed and programming/development ease for different types of processors: ASIC, FPGA, Network processor, GPP, Embedded RISC Processor.]

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
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!
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

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
General purpose processors (GPP)

- Programmable
- Mature development environment
- Typically used to implement control plane
- Too slow to run data plane effectively
  - Sequential execution
- CPU/Network 50x increase over last decade
- Memory latencies 2x decrease over last decade
  - Gigabit ethernet: 333 nanosecond per packet budget
  - Cache miss: ~150-200 nanoseconds
Embedded RISC processors (ERP)

- Same as GPP, but
  - Slower
  - Cheaper
  - Smaller (require less board space)
  - Designed specifically for network applications
- Typically used for control plane functions
Application-specific integrated circuits (ASIC)

- Custom hardware
- Long time to market
- Expensive
- Difficult to develop and simulate
- Not programmable
- Not reusable
- But, the fastest of the bunch
- Suitable for data plane
Field Programmable Gate Arrays (FPGA)

- Flexible re-programmable hardware
- Less dense and slower than ASICs
- Cheaper than ASICs
- Good for providing fast custom functionality
- Suitable for data plane
Network processors

- The speed of ASICs/FPGAs
- The programmability and cost of GPPs/ERPs
- Flexible
- Re-usable components
- Lower cost
- Suitable for data plane
Network processors

- Common features
  - Small, fast, on-chip instruction stores (no caching)
  - Custom network-specific instruction set programmed at assembler level
    - What instructions are needed for NPs? Open question.
    - Minimality, Generality
  - Multiple processing elements
  - Multiple thread contexts per element
  - Multiple memory interfaces to mask latency
  - Fast on-chip memory (headers) and slow off-chip memory (payloads)
  - No OS, hardware-based scheduling and thread switching
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
Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
Network processor architectures

- Packet path
  - **Store and forward**
    - Packet payload completely stored in and forwarded from off-chip memory
    - Allows for large packet buffers
    - Re-ordering problems with multiple processing elements
    - Intel IXP, Motorola C5
  - **Cut-through**
    - Packet held in an on-chip FIFO and forwarded through directly
    - Small packet buffers
    - Built-in packet ordering
    - AMCC

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
Network processor: Processing

- Processing architecture
  - **Parallel**
    - Each element independently performs entire processing function
    - Packet re-ordering problems
    - Larger instruction store needed per element
  - **Pipelined**
    - Each element performs one part of larger processing function
    - Communicates result to next processing element in pipeline
    - Smaller code space
    - Packet ordering retained
    - Deterministic behavior (no memory thrashing)
- Hybrid

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
Various forms of Processors

Embedded Processor (run-to-completion)

Parallel architecture

Pipelined Architecture
Network processor: Memory

- Memory hierarchy
  - Small on-chip memory
    - Control/Instruction store
    - Registers
  - Cache
  - RAM
- Large off-chip memory
  - Cache
  - Static RAM
  - Dynamic RAM

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
Network processor: Interconnects

- Internal interconnect
  - Bus
  - Cross-bar
  - FIFO
  - Transfer registers
Network processor: Concurrency

- Concurrency
  - Hardware support for multiple thread contexts
  - Operating system support for multiple thread contexts
  - Pre-emptiveness
  - Migration support
Increasing network processor performance

- Processing hierarchy
  - Increase clock speed
  - Increase elements
- Memory hierarchy
  - Increase size
  - Decrease latency
  - Pipelining
  - Add hierarchies
  - Add memory bandwidth (parallel stores)
  - Add functional memory (CAMs)
Packet Flow Diagram: IXP 1200

- Optional Host CPU
- PCI Bus Devices
  - PCI Bus Interface
  - PCI Bus 32-bit @ 66 MHz

- SDRAM up to 256 Mb
- SRAM up to 8 Mb
- FlashROM up to 8 Mb
- Memory Mapped I/O Device (ex MAC Control)

- SDRAM Interface
- SRAM Interface
- 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

- Microengine 1
- Microengine 2
- Microengine 3
- Microengine 4
- Microengine 5
- Microengine 6

Source: Level One 1999

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
IXP 2800

- **IXP2800 features:**
  - 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 multi-threading per chip

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
μ-engine functions

- Packet ingress from physical layer interface
- Checksum verification
- Header processing and classification
- Packet buffering in memory
- Table lookup and forwarding
- Header modification
- Checksum computation
- Packet egress to physical layer interface
μ-engine characteristics

- Programmable microcontroller
  - Custom RISC instruction set
  - Private 2048 instruction store per μ-engine (loaded by StrongARM)
  - 5-stage execution pipeline
- Hardware support for 4 threads and context switching
  - Each μ-engine has 4 hardware contexts (mask memory latency)
1. Packet received on physical interface (MAC)
2. Ready-bus sequencer polls MAC for mpacket
   Updates receive-ready upon a full mpacket
3. μ-engine polls for receive-ready
4. μ-engine instructs FBI to move mpacket from MAC to RFIFO
5. μ-engine moves mpacket directly from RFIFO to SDRAM
6. Repeat 1-5 until full packet received
7. μ-engine or StrongARM processing
8. Packet header read from SDRAM or RFIFO
   into m-engine and classified (via SRAM tables)
9. Packet headers modified
10. mpackets sent to interface
11. Poll for space on MAC
    Update transmit-ready if room for mpacket
12. mpackets transferred to interface
EXTRA SLIDES (optional)
Intel’s Gear (1)

- 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.
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.
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
MicroEngine v2

- Local Memory: 640 words
- 128 GPR
- 128 GPR
- 128 Next Neighbor
- 128 D Xfer In
- 128 S Xfer In

From Next Neighbor:
- LM Addr 1
- LM Addr 0
- P-Random #
- CRC Unit
- CRC remain
- Local CSRs
- 2 per CTX

D-Push Bus
- B_op
- Prev.A

S-Push Bus
- A_op
- Prev.B

Control Store:
- 4K Instructions

32-bit Execution Data Path
- Multiply
- Find first bit
- Add, shift, logical

D-Pull Bus
- D-Push Bus
- S-Pull Bus

To Next Neighbor
- A_Operand
- B_Operand

128 D Xfer Out
128 S Xfer Out

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI

D-Push Bus
- Lock 0-15
- Status and LRU Logic (6-bit)

S-Push Bus
- TAGs 0-15
- CAM
Why Multi-threading?
Packet processing using multi-threading within a MicroEngine

Execution Time = 8 \times T_a

Packet n
Packet n+1
Packet n+2
Packet n+3
Packet n+4
Packet n+5
Packet n+6
Packet n+7
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7

Shivkumar Kalyanaraman

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
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

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
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>
Micro-engine C Compiler

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

---

Microengine C

```c
{ 
    unsigned int j, k, crc = 0;
    k = i << 8;
    for( j = 0; j < 8 ; j++)
    { 
        if( (crc ^ k) & 0x8000)
            crc = (crc << 1) ^ 0x1021;
        else crc <<= 1;
        k >>= 1;
    }
    return crc;
}
```

Example shows:

50% fewer lines of code required than with assembly

For simplified, portable, microengine programming!

---

Microcode

```
Optimized code generated by the Microengine C compiler

crc_table#: 
  alu_shf[B1, --, B, A0, <<8]  
  immed[A0, 0, 0]  
  immed[B0, 0, 0]  

L_3#: 
  alu[A2, B1, XOR, A0]  
  br_bset[A2, 15, L_4#]  
  alu_shf[A0, --, B, A0, <<1]  
  br[L_5#]  

L_4#: 
  alu_shf[B2, --, B, A0, <<1]  
  immed[A0, 4129, 0]  
  alu[A0, B2, XOR, A0]  

L_6#: 
  alu_shf[B1, --, B, B1, <<1]  
  alu[B0, B0, +, 1]  
  alu[---, B0, -, 8]  
  brl=cout[L_3#]  
  alu[B0, --, B, A0]  
  rtn[A1]  
```
Core Components and Microblocks

XScale™ Core

- Core Component
- Core Component
- Core Component

Core Component Library
Resource Manager Library

Micro-engines

- Microblock
- Microblock
- Microblock

- Microblock Library
- Intel/3rd party blocks
- User-written code
- Core Libraries

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
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

Based upon presentations from Raj Yavatkar, Intel and Wu-Chang Feng, OGI
For more info....

- OGI/Portland State IXA course:
- Prof. Wu Chang Feng