Network Processing

A Multi-Threaded Multi-Processor Application

Mike O'Connor

oconnor@alumni.rice.edu

What is Network Processing?

I shall not today attempt further to define the kinds of material I understand to be embraced within that shorthand description; and perhaps I could never succeed in intelligibly doing so. But I know it when I see it...

- Potter Stewart, Associate Justice, US Supreme Court
JACOBELLIS v. OHIO, 378 U.S. 184 (1964)

Many people have different definitions…

Look at the applications…

Network Processing Applications

- WAN/LAN Switching and Routing,
- Multi-service/Multi-layer Switches/Routers
- Web/server Load balancing
- QoS solutions
- VoIP Gateways
- 2.5G and 3G wireless infrastructure Equipment
- Security - Firewall, VPN, Encryption, Access control
- Storage Area Networks

- Characteristic these applications share:

  Processing of packet-based digital networking data

Typical Router Per-Packet Processing

- **Layer 2 Processing**
  - Ethernet, Validation, Control Packet Extraction
- **RFC 1812 Validation Checks**
  - TTL, Version, Length (Header, Min, Max), Valid Src/Dst IP
- **VPN Identification**
  - Interface / Sub-interface, Ethernet VLAN, MPLS
- **Source & Destination IP Lookups**
- **Multi-Field Classification**
  - ACL, Filters, Billing, DiffServ BA
- **Policing & Statistics**
  - Interfaces Group of MIB II, DiffServ per color flows, MPLS flows
- **Load Balancing - ECMP**
- **Full Packet Editing & Header Insertion**
  - Fragment, Replicate, Mirror

"Plain" RISC with Hardware Assists

- ~1000 standard RISC instructions per packet
  - *Assuming off-loading to dedicated co-processors for address lookups, classifications, policing, statistics, plus packet reassembly, ordering, dispatch, and edit functions*
  - *Assuming no overhead for context switching of threads during long-latency co-processor operations*
- ~450 bytes of aggregate look-aside memory and co-processor I/O per packet
- ~64 bytes of packet memory I/O per packet

Wire-Rate Processing (Full Duplex)	Millions of MIPS Instructions per Second Required	Internal Bus I/O Bandwidth
100MbE / 300Kpps	*~300 MIPS*	*~160 MB/s*
1GbE / 3Mpps	*~3000 MIPS*	*~1.6 GB/s*
10GbE / 30Mpps	*~30000 MIPS*	*~16 GB/s*

Significant Parallelism Needed

- As packet rates increase, the packet arrival time diminishes to the point where multiple packets have to be processed by the NPU concurrently, in order to achieve wire-rate performance
- Multiple packet contexts are required to hide packet processing latency at progressively higher data rates

Wire-Rate Processing (FDX)	Packet Inter-arrival Time	Parallel Contexts Required
100MbE / 300Kpps	*3333 ns*	*1*
1GbE / 3Mpps	*333 ns*	*10*
10GbE / 30Mpps	*33 ns*	*100*

- Assumes constant 3.3usec latency to process a given packet
  – In practice, with more threads, latency per packet increases due queuing delays resulting from contention between threads

Silicon Access Networks iFlow Packet Processor

- True 20Gbps network processor
  - 20Gbps in + 20Gbps out
    - (*not* “Cisco Math” where 10 in + 10 out = 20)
- Targeted at core routers supporting wide range of protocols and functions
  - Cisco GSR12000 or Juniper T640-class boxes
- Multi-threaded/multi-processor architecture
- Custom, optimized network specific instruction set
- Handles 30-50M packets per second
  - Full Duplex 10GE or OC-192

Goals for iFlow Architecture

- Simple to program
- Scale architecture easily from 2.5Gbps to 40+Gbps without requiring software rewrite
- Serve a wide variety of application points

How to Organize the Chip?

- Several options for getting the necessary number of packets being processed in parallel
- Represent "ideological" points-of-view
  - Real chips tend to blend these

		Processor Type	
		Identical	Specialized
Organization	Parallel	Simple to program	na
	Pipelined	Like → But trying to make SW a little easier	Optimized for HW costs
	Ad-Hoc	na	Optimized for performance /flexibility

Parallel/Identical Organization

- Each processor/thread context pair “owns” a packet for it entire processing lifetime
- Programming model is as if writing for a single thread
  - Known as a “Run-to-Completion” programming model
- Pros:
  - Straightforward to analyze and debug
  - Scales across different implementations with minimal code changes
  - Graceful performance degradation with additional functionality
  - Performance not dependant on programmer skill to identify parallel activities
  - Reduces need for high-bandwidth inter-processor communication
- Cons:
  - All processors must be able to execute all code, reducing some implementation optimization opportunities
  - Without lots of high-bandwidth inter-processor communication, some things are hard

Pipelined/Specialized Org.

- Each processor/thread context pair "owns" a packet for a slice of its lifetime, before handing it to the next PE
  - Different processors can be adapted to tasks common in certain phased of packet processing – e.g. classification, editing, etc.
- Pros:
  - Processors can be optimized for given tasks, without "carrying extra baggage" – basically how most NP ASICs are architected
  - high-bandwidth inter-processor communication limited to neighbors in pipeline
  - More effective code space since processors are specialized to specific parts of the packet processing workload – all processors do not need to see all the code
- Cons:
  - Performance dependant on programmer skill to "load-balance" pipestages – throughput is limited by slowest stage
  - Different processors for each class of task require programmer to master several different target processor types
  - Ratio of different types of specialized processors may not reflect application workload

Pipelined/Identical Organization

- Each processor/thread context pair “owns” a packet for a slice of its lifetime, before handing it to the next PE
- Like Pipelined/Specialized but without the problems of guessing right ratio of each processor type and forcing programmers to learn multiple target architectures
- Pros:
  - high-bandwidth inter-processor communication limited to neighbors in pipeline
  - More effective code space since processors are dedicated to specific parts of the packet processing workload – all processors do not need to see all the code
- Cons:
  - All processors must be able to execute all code, reducing some implementation optimization opportunities
  - Performance dependant on programmer skill to “load-balance” pipestages – throughput is limited by slowest stage

Ad-Hoc/Specialized Org.

- Not as simple as the previous examples
- Each processor/thread context pair “owns” a packet for a variable amount of its lifetime, handing it to the other PE’s as need arises
  - Different processing elements can be adapted to tasks common in certain phases of packet processing – e.g. classification, editing, etc.
- Pros:
  - Maximum performance and flexibility
  - More effective code space since processors are dedicated to specific parts of the packet processing workload – all processors do not need to see all the code
- Cons:
  - High-bandwidth inter-processor communication required as any processor may pass handling a packet to any other processor
  - Performance dependant on programmer skill to “load-balance” and schedule different resources – a complex task
  - Different processors for each class of task require programmer to master several different target processor types
  - Ratio of different types of specialized processors may not reflect application workload

iFlow Packet Processor Approach

- Programmable elements take a Parallel/Identical organization
- Hardwired Coprocessors for different specialized processing elements for common tasks in certain phases of packet processing – e.g. classification, editing, etc.
- Interconnection between Coprocessors is “Ad-Hoc” though a large switch

iFlow Architectural Partitioning

PROCESSOR

- Significant Editing
  - Routing
  - Protocol Translations
  - Encapsulation Changes
- Complex Parsing
  - Layer 3 followed by 4, 5 etc.
- Complex Conditionals
- Multi-Pass Packet Ops

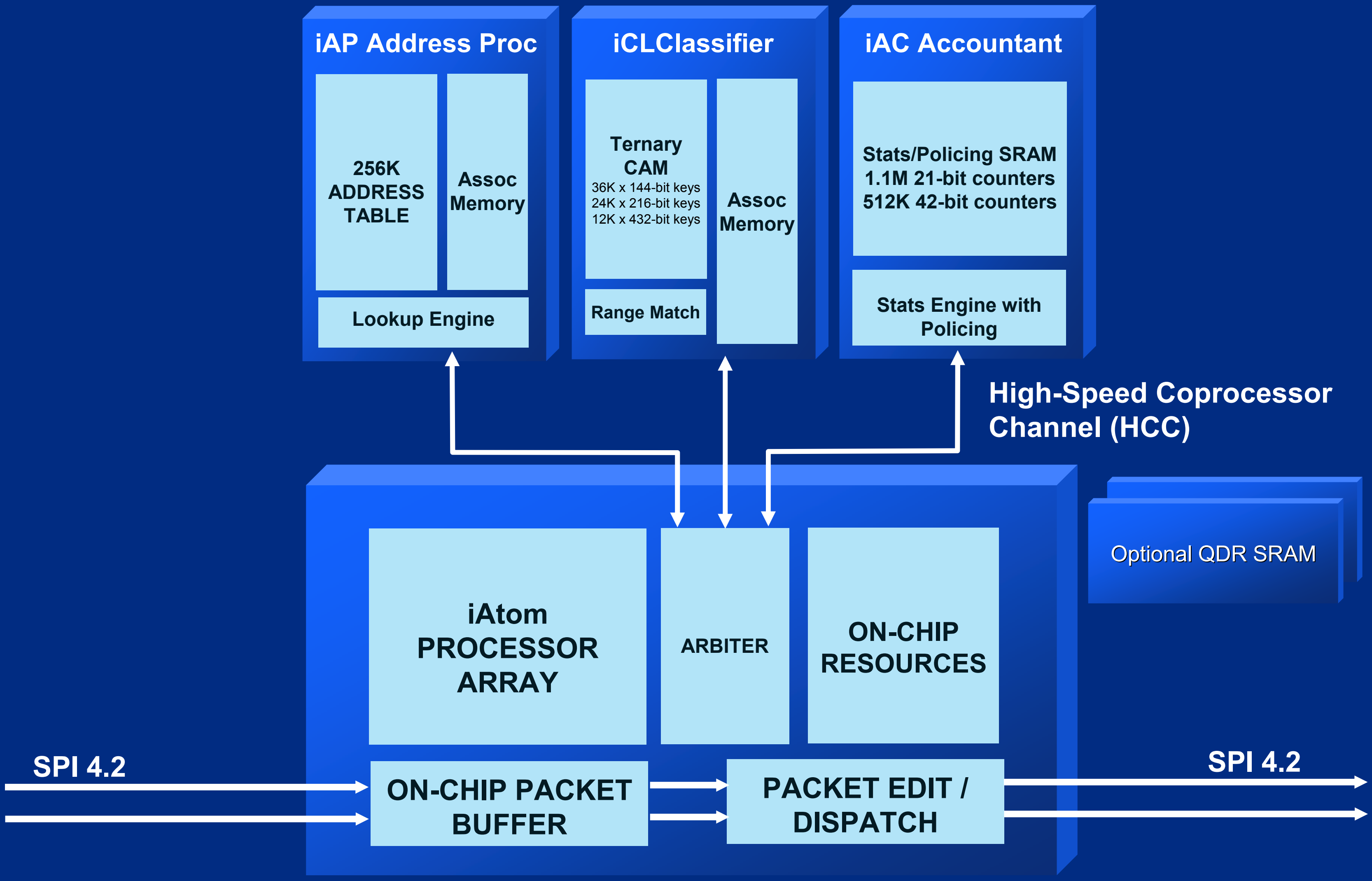
COPROCESSOR

- Bounded Lookups
  - 36-, 48-, 288-bits etc.
- Accounting/Policing
  - Simple arithmetic ops based on a lookup result
- Simple request/response interaction with NPU

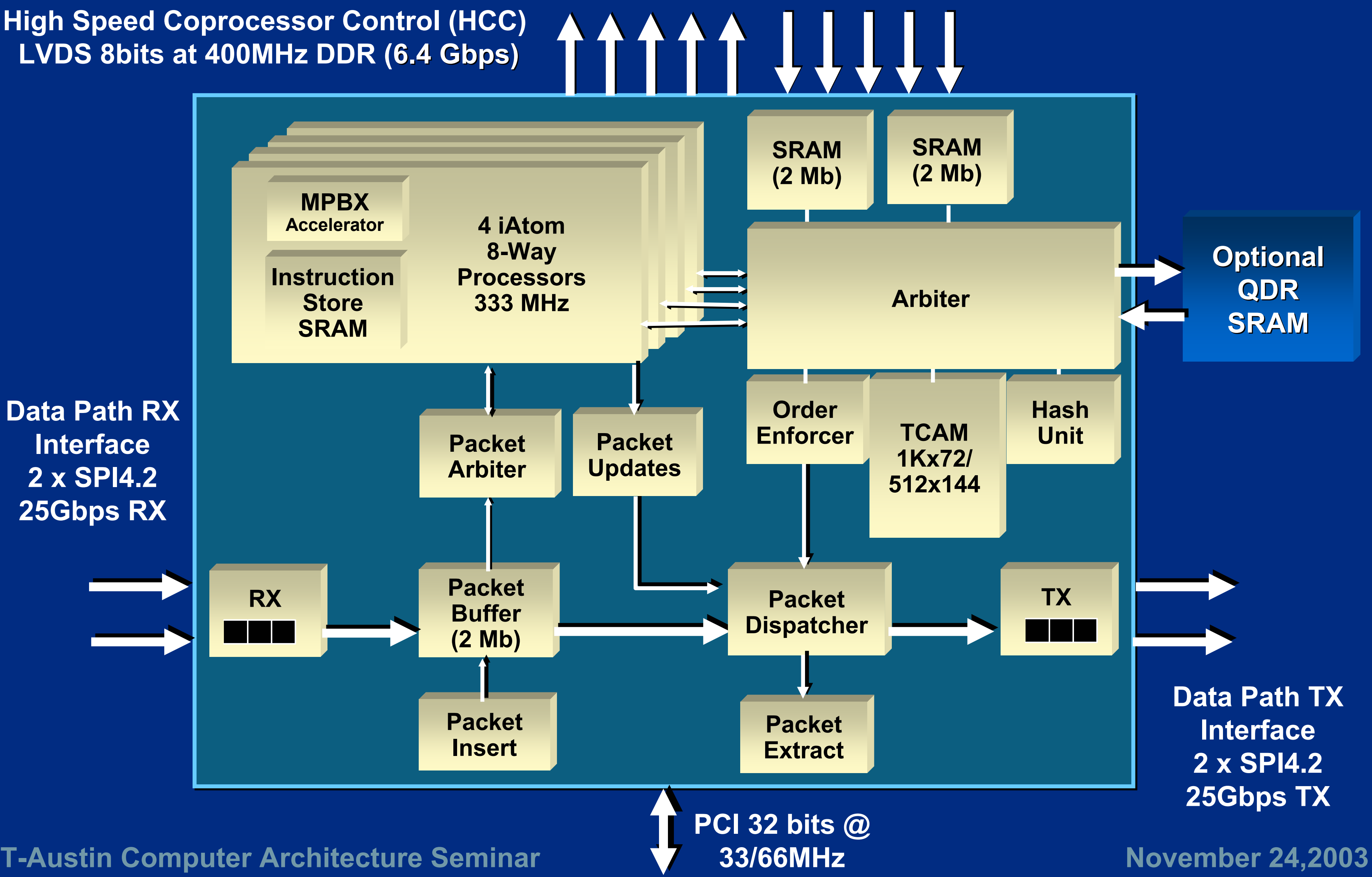
iFlow Coprocessor Functions:

- ✓ **Address Lookup**
- ✓ **Flow Lookup**
- ✓ **Accounting**

Basic Data Flow Example



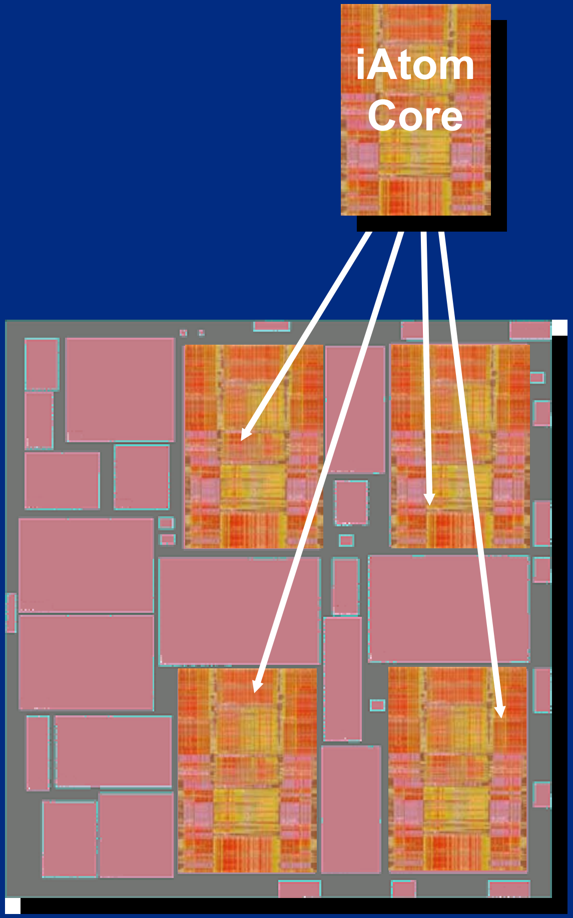
Inside the iPP Chip



On-Chip Resources

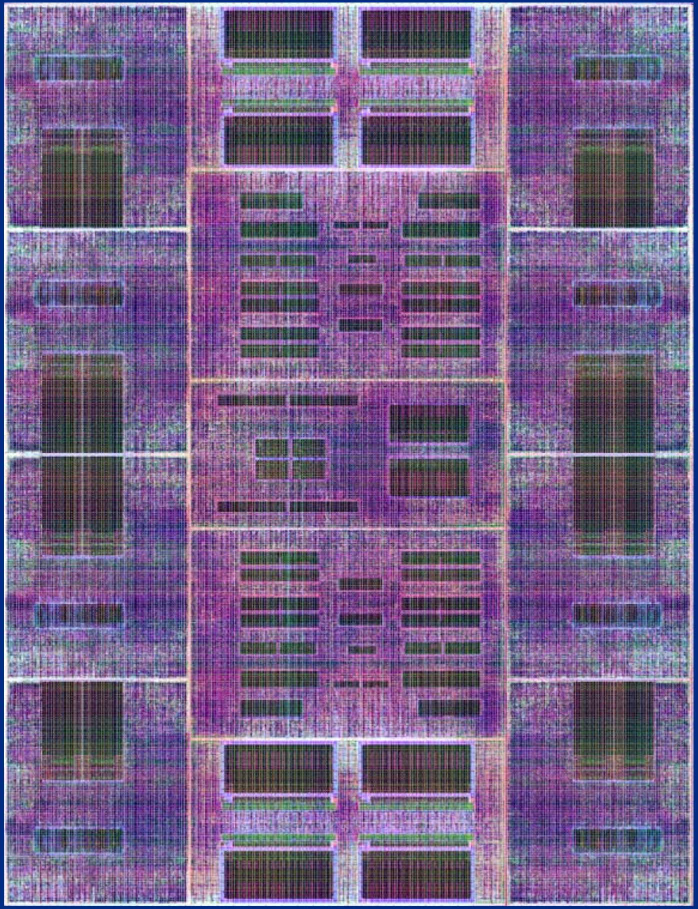
- Packet Buffer
  - 240K Bytes
  - Performs Re-assembly for up-to 21 logical connections
  - No off-chip packet buffering needed
- 2 general purpose SRAM arrays
  - 256K bytes each
  - 128-bit internal bus width each
  - Instruction store is separate (within iAtom)
- General purpose Ternary CAM
  - 72K (ternary) bits with 32K bits of associated data
  - Keys up to 144 bits supported
- Hash Unit
  - Hashes 128-bit input to 2 32-bit keys using 2 different CRC polynomials
  - Also includes a Modulo Engine for computing remainder of 8-bit divide
- Resource Arbiter
  - Full output-buffered switch maximizes useful bandwidth to on-chip and external resources

iAtom
Network Instruction Set Core



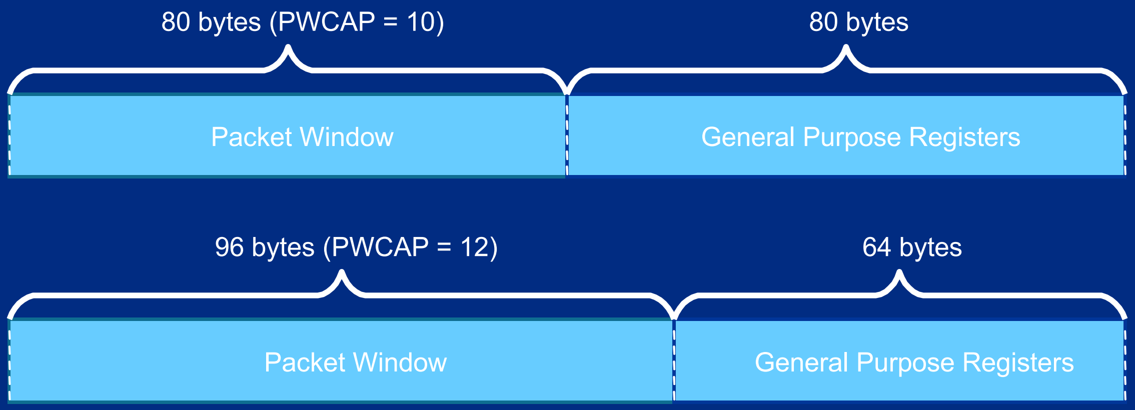
- 32 processors per iPP organized as
  - 4 iAtom cores
    - 8 processors per iAtom
- 8 threads per processor
  - Total of 256 threads per iPP
  - Can operate on 256 packets simultaneously
- Highly optimized network instruction set

iAtom Architecture



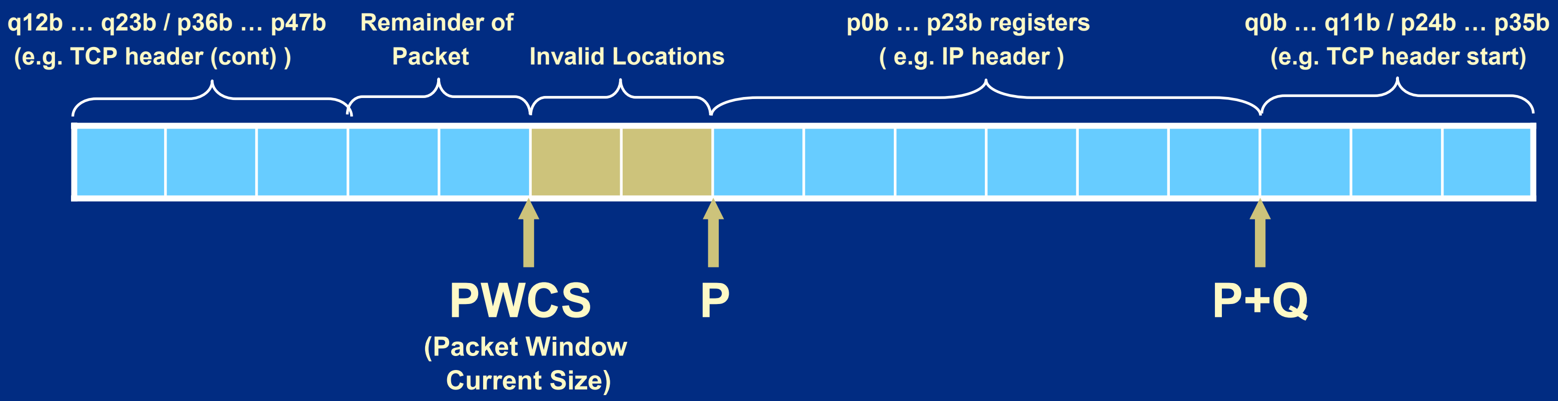
- Each iAtom core contains:
  - 8 network processor units
  - 8 register files
    - 8 thread contexts each
  - Instruction store
  - Arbiters for accessing off-iAtom resources
  - MPBX: Massively Parallel Branch Accelerator
- All processor elements are identical
- Network processor units are simple, 6-stage pipelined, single issue processors

Register File



- Register file is 160 bytes in size (per context)
  - Not including various special registers (e.g. Condition Codes)
- Registers are byte addressable
  - Can be 1, 2, or 4 bytes
  - No alignment restrictions
- Register file holds window of packet data
  - This window is configurable to be between 64 and 128 bytes
  - Remainder is used as general-purpose registers

Packet Window



- Packet window logically appears as a sliding window over the packet data
  - Physically organized as a circular buffer
  - Indexed indirectly via two offsets: P & Q
- Multiple offsets into packet window enables same code to be used in multiple situations
  - Example: TCP processing can be identical code regardless of length of IP header

Instruction Set Overview

- 47 instructions
  - Most common RISC operations supported
  - Several networking-specific operations
  - Many unique to iAtom
- Each instruction can handle any register width
  - 'add r6w = p11b, r4h'

    Adds an 8-bit value to a 16-bit value and stores the result in a 32-bit value
- Most instructions can also specify one immediate
  - 'sub r2b = p8b, 1'

    Subtracts 1 from the 8-bit value in the packet and stores the result in a general-purpose register
- Move instructions can be conditionally executed

Networking Optimized Instructions

- Some examples of instructions particularly useful in networking:
  - Addchk – Checksum Addition
    - One’s complement addition
    - Used in checksum generation
  - Subchk - Checksum Subtraction
    - One’s complement subtraction
    - Allows checksum delta’s to be computed for incremental edits
  - Sllmrg – Logical Shift Left and Merge with Bit Mask
    - Shift value from one register left, selects a range of bits, and merges these with another register value
    - For example, allows bits 11:3 of A to overwrite bits 28:20 of B.
  - Kgen – Generate Key
    - Generates a “key” to by used by subsequent lookup or MPBX operations
    - Extracts 2 nibble-aligned ranges from input and appends them to key buffer
    - Can be done repeatedly to build large keys

iAtomC Example

iAtomC implementation of a packet “5-tuple” lookup:

```
lookup  {rFlowId, rPolicingContext, rStatsId} =
        {reqdesc, ip.da, ip.sa, ip.protocol, tcp.sp, tcp.dp, ip.tos}, HCC_1;

        kgen r8h, p16w;
        kgen p12w, p9b;
        kgen q0h, q2h;
        kgen p1b, null;
        lookup %rd[2] = /*key buffer,*/ 0x48;
```

- Compiles to 5 instructions
- 5 iAtom clock cycles to execute
- Thread suspends waiting for coprocessor results
- Results parsed in background according to result descriptor 2 and assigned to **rFlowId**, **rPolicingContext**, and **rStatsId** variables

MPBX "if-then-else" Accelerator

- Massively parallel branch accelerator
  - Up to 128, 88-bit wide compare and branch instructions simultaneously
  - Think "giant parallel if-then-else"
- Significantly accelerates the execution of complex, bit-oriented conditional branching statements
- Implemented with local TCAM tightly coupled with processor

```
switch  {
     // IEEE 802.3ac tagged Ethertype
     case  dix.ethertype==ETH_TYPE_TAGGED && vlanEnabled.z: ReceiveError(IF_ERR_ETH_VLAN_DISABLED);
     case  dix.ethertype==ETH_TYPE_TAGGED && maxVlan.gt:    ReceiveError(IF_ERR_ETH_INVALID_VLAN);
     case  dix.ethertype==ETH_TYPE_TAGGED:                  EthernetTaggedType;
     // Ethertype
     case  dix.ethertype==ETH_TYPE_IP:                            EthernetRemoveEnetHdr(14,ETH_IP);
     case  dix.ethertype==ETH_TYPE_IPV6:                          EthernetRemoveEnetHdr(14,ETH_IPV6);
     case  dix.ethertype==ETH_TYPE_MPLS:                          EthernetRemoveEnetHdr(14,ETH_MPLS);
     case  dix.ethertype==ETH_TYPE_ARP:                           ToCp(TOCP_ETH_ARP);
     // IEEE 802.1 LLC/SNAP  (Ethertype < 0x800 indicating length)
     case  dix.ethertype==ETH_TYPE_IEEE && ieee.ethertype==ETH_TYPE_IP &&
           ieee.xaaaa==0xAAAA && x30.eq:                    EthernetRemoveIeeeHdr(14,ETH_IEEE | ETH_IP);
     case  dix.ethertype==ETH_TYPE_IEEE && ieee.ethertype==ETH_TYPE_IPV6 &&
           ieee.xaaaa==0xAAAA && x30.eq:                          EthernetRemoveIeeeHdr(14,ETH_IEEE | ETH_IPV6);
     case  dix.ethertype==ETH_TYPE_IEEE && ieee.ethertype==ETH_TYPE_ARP &&
           ieee.xaaaa==0xAAAA && x30.eq:                          ToCp(TOCP_ETH_ARP);
     case  dix.ethertype==ETH_TYPE_IEEE && ieee.ethertype==ETH_TYPE_MPLS &&
           ieee.xaaaa==0xAAAA && x30.eq:                          EthernetRemoveIeeeHdr(14,ETH_IEEE | ETH_MPLS);
     default:                                                     ReceiveError(IF_ERR_ETH_UNKNOWN_PROT);
}
```

iAtomC implementation of Ethernet header parsing: 2 clocks

Coprocessor Operations

- iAtom has extensive support for utilizing coprocessors
- ‘lookup’ instruction issues requests to coprocessors
  - Keys built using ‘kgen’ instructions
  - Specifies one of 64 result descriptors which specify where different fields reside in result data
  - Coprocessor can be either on-chip or external
  - Coprocessor operations which do not produce results like statistics increments are issued with ‘store’ instructions
- Results are parsed *in the background* according to the specified result descriptors
  - Each extracted field is placed in a specific register in the thread’s context
  - No code or NPU cycles are wasted extracting result fields from coprocessor requests

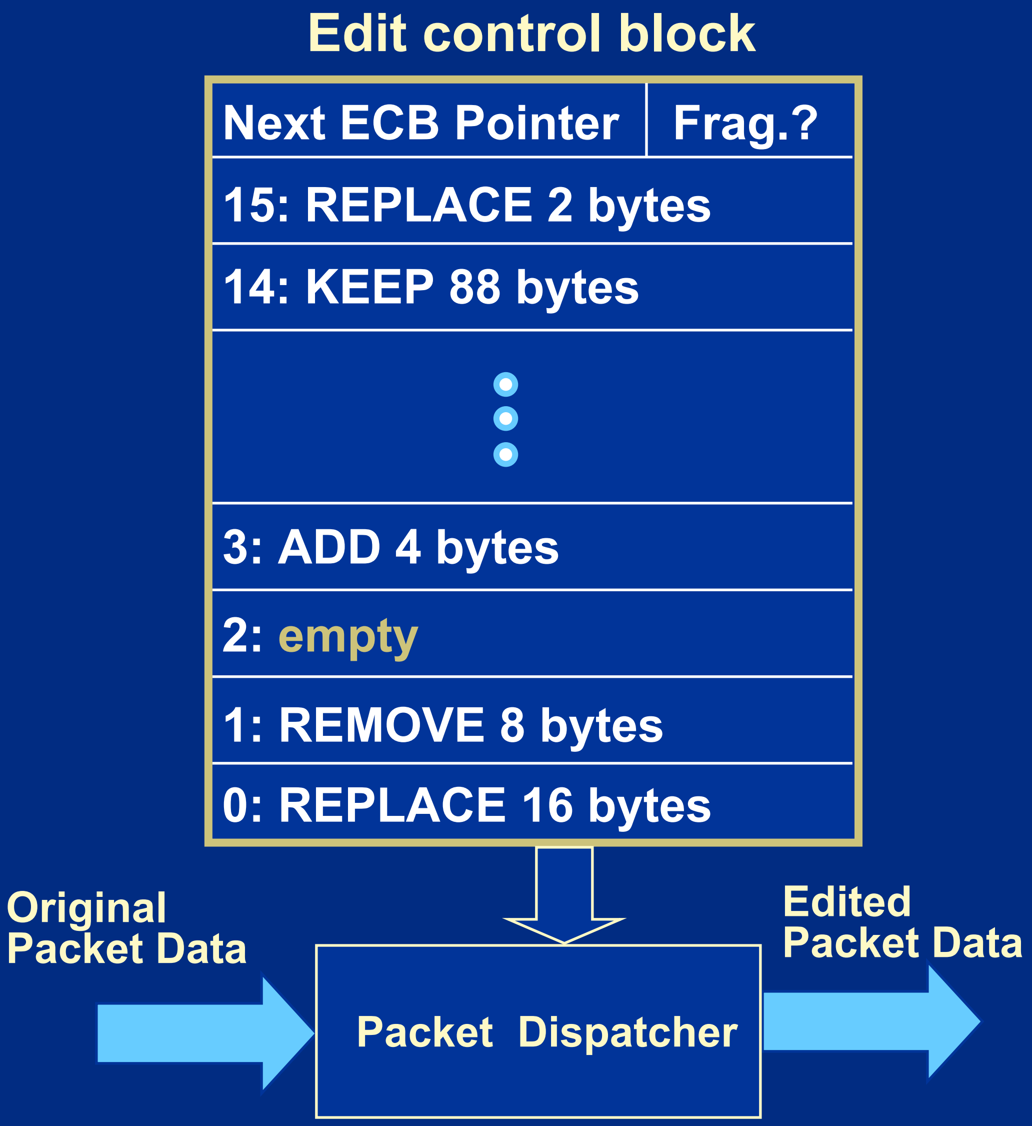
Thread Switching

- Zero-cycle context switch
  - No penalty to change threads
- Network processing units context switch on:
  - Loads/Stores/Lookups
    - Hides latency to access off-chip resources
  - Branches/Jumps/MPBX Switch
    - NPUs do not have branch-prediction hardware
    - Hides latency to access instruction store
  - Awpp/Pullreset/Pullnew
    - Hides latency to access packet buffer
  - Halt/Endtask/Sleep
    - Thread control operations
- When lookup result, instruction data, or packet data has been received the thread becomes eligible to continue execution
  - From point of view of the programmer, all of the instructions above appear to execute in a single cycle

Order Enforcer

- Maintains a number of "ordered flows"
- Every packet is bound to a given Flow Identifier
  - Can be simple (like ingress interface)
  - Can be result of complex CAM lookup
- Whenever packets must be serviced in order:
  - A thread informs the order enforcer it has reached a given ordering point
  - The order enforcer prevents this thread from executing until all packets with an earlier timestamp *bound to the same flow* have passed the ordering point
- Very simple programming model
  - "Fire and Forget"
- "Unordered" semaphores for traditional critical sections are also supported

Packet Editing and Dispatch



- All packet edit commands written into Edit Control Blocks (ECBs)
- Each ECB holds 16 edit commands, and ECBs can be chained together
- Commands may be posted to an ECB in any order, and empty command slots are allowed
  - Useful for MPLS label pushing
- Each ADD or REPLACE command can specify up to 16 bytes of data
- An ECB can also specify whether to fragment a packet
- Packet Dispatcher reads packet data and applies edit commands prior to transmit

"Plain" RISC with Hardware Assists

- ~1000 standard RISC instructions per packet
  - *Assuming off-loading to dedicated co-processors for address lookups, classifications, policing, statistics, plus packet reassembly, ordering, dispatch, and edit functions*
  - *Assuming no overhead for context switching of threads during long-latency co-processor operations*
- ~450 bytes of aggregate look-aside memory and co-processor I/O per packet
- ~64 bytes of packet memory I/O per packet

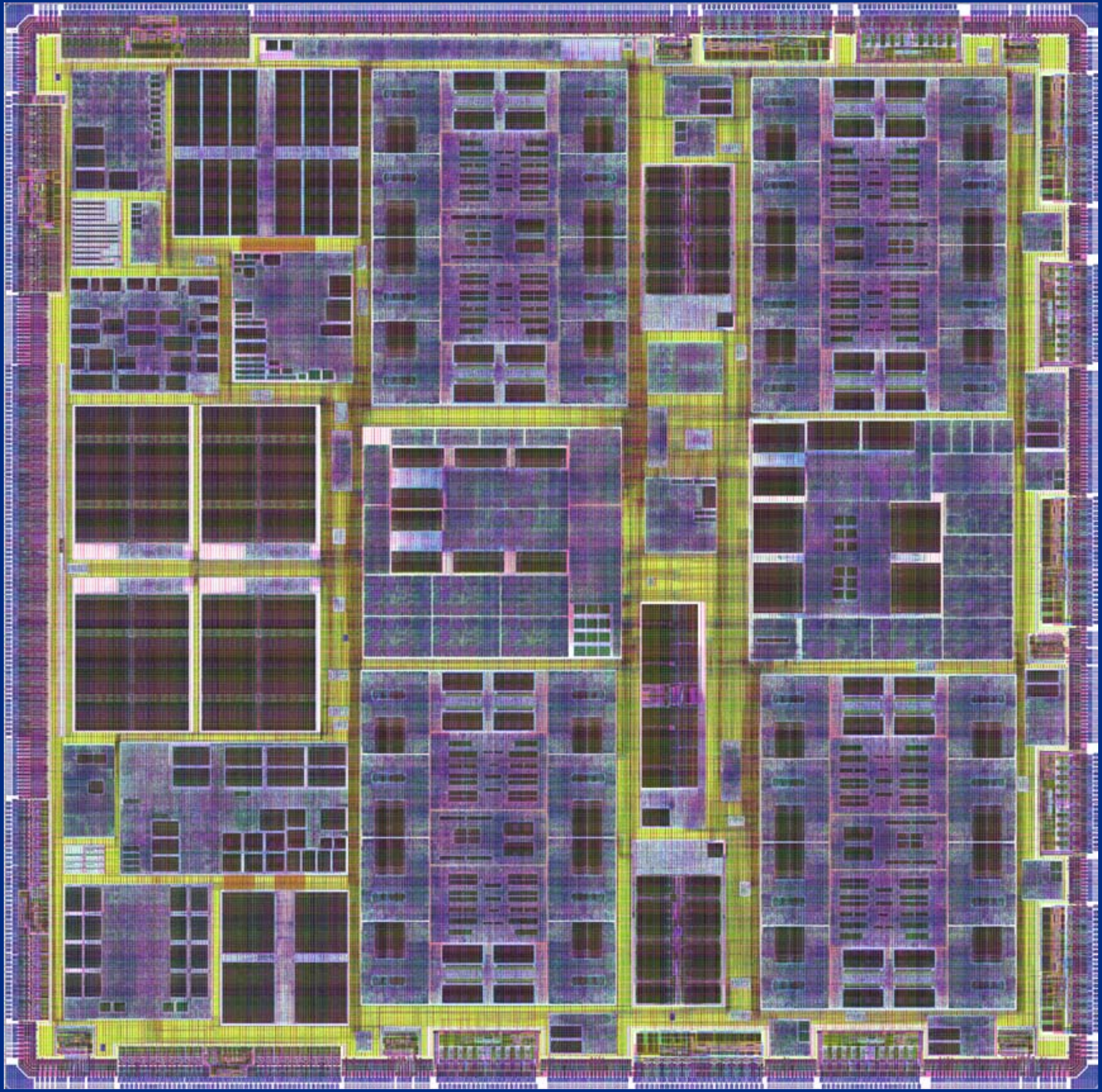
Wire-Rate Processing (Full Duplex)	Millions of MIPS Instructions per Second Required	Internal Bus I/O Bandwidth
100MbE / 300Kpps	*~300 MIPS*	*~160 MB/s*
1GbE / 3Mpps	*~3000 MIPS*	*~1.6 GB/s*
10GbE / 30Mpps	*~30000 MIPS*	*~16 GB/s*

iAtom with Hardware Assist

- ~240 iAtom instructions per packet
  - *Assuming off-loading to dedicated co-processors for address lookups, classifications, policing, statistics, plus packet reassembly, ordering, dispatch, and edit functions*
  - *Assuming no overhead for context switching of threads during long-latency co-processor operations*
- ~450 bytes of aggregate look-aside memory and co-processor I/O per packet
- ~64 bytes of packet memory I/O per packet

Wire-Rate Processing	Millions of iAtom Instructions per Second Required	Internal Bus I/O Bandwidth
100MbE / 300Kpps	*~72 MIPS*	*~160 MB/s*
1GbE / 3Mpps	*~720 MIPS*	*~1.6 GB/s*
10GbE / 30Mpps	*~7200 MIPS*	*~16 GB/s*

Silicon Access iFlow Packet Processor



- 32 processors
  - **256 Threads**
- 128.1 Gbps Aggregate I/O Bandwidth
  - **64 Gbps HCC**
  - **51.2 Gbps SPI 4.2**
  - **10.8 Gbps QDR**
  - **2.1 Gbps PCI**
- 333 MHz
- 7M Gates
- 18 Mbits of SRAM
- 175M transistors
- 1036 pin BGA
- 0.13u “G” TSMC
- First pass Si success

Q & A