PLUG: Flexible Architecture for High-speed Lookups

Lorenzo De Carli, Amit Kumar, Yi Pan, Cristian Estan, Karu Sankaralingam
University of Wisconsin-Madison
High-speed Lookups Today
High-speed Lookups Today

Many lookup modules
Specialized unique chips
High-speed Lookups Today

Long design cycle
Expensive to design ASICs
Slow deployment of new protocols
High-speed Lookups Today

Limited upgradability
High-speed Lookups Today

Problem: Functionality defined in hardware
Can we be build a single flexible and efficient lookup module?

One lookup module
Rapid deployment of new protocols
Upgrading is easy
Executive Summary

Current line-card

Future line-card

PLUGs replace lookup modules at layer 2 and above
### Why flexible lookup?

#### Ethernet

- **Entry0**
- **Entry1**
- **Entry2**
- **Entry3**

- **Bucket 0**
- **Bucket 1**
- **Bucket 2**
- **Bucket n**

#### IP Lookup

- **Memory 0**
- **Memory 1**
- **Memory 2**
- **Memory 3**

#### Similarities
- Simple processing elements working w/ local memories
- Move computation to data points
- Proximity of data to processing elements
- Dynamic access to local memories

#### Differences
- Number and size of memories
- The actual processing structures
- Fixed data patterns

---

**PLUG captures similarities in hardware, differences in software**
Outline

Programming model

Applications: *dataplane lookup for protocols*

PLUG Hardware

Results
Programming model

- Data structure implemented as an object
- External interface
  - Routines and Messages
- Lookup transformed into physical pipeline on chip
Data-structure to Dataflow graphs

Level 0
Level 1
Level 2

Input
Level 0 summaries → Level 0 children → Level 0 results

Level 1 summaries → Level 1 children → Level 1 results

Level 2 summaries → Level 2 children → Level 2 results

Output
Lookup Dataflow Graphs

- **Data blocks**
  - Primitive building blocks for tables

- **Logical page**
  - Collection of similar data blocks

- **Edges**
  - Connect logical pages

- **Codeblocks**
  - Read data-blocks
  - Specify next step
C++ Framework

class ip_l0_page:
public plug_page {
private:
  vector<plugdata_blk> l0_s;
public:
  void read_mask(msg*);
  void read_summary(msg*);
  void read_final(msg*);
}

class ipfwd:
public plug_object {
private:
  vector<plug_page> pages;
public:
  lookup(plug_msg *);
  update(plug_msg *);

  ipfwd::create_dfg(void) {
    plug_connect(p0,p1);
    plug_connect(p1,p2);
  }

  void read_summary(msg*);
  void read_final(msg*);
}
Hash Table Variants

- **Basic hash table**
  - Entry 0
  - Entry 1
  - Entry 2
  - Entry 3

- **Hash table with fingerprints**
  - Fingerprints
  - Table 0 entries
  - Table 1 entries

- **Table with split entries**
  - Entry 0 key high
  - Entry 1 key high
  - Entry 2 key high
  - Entry 3 key high

- **D-left hash table**
  - Table 0 entry 0
  - Table 0 entry 1
  - Table 1 entry 0
  - Table 1 entry 1

- **Table with combined entries**
  - Entries 0 and 1
  - Entries 2 and 3

- **Values for all entries**
Irregular data-structures can be expressed as dataflow graphs
Outline

Programming model

Applications: \textit{dataplane lookup for protocols}

PLUG Hardware

Results
Applications-Ethernet

- **Implementation**
  - Hash table lookup(2-left), 4 entries per bucket
  - Each entry 48-bits addr, 12-bits port, 3-bits timestamp
Applications-Ethane

- Misbehaving hosts
  Key: MAC src addr + input port 
  Value: timestamp

- Permitted flows
  Key: src+dst addr for MAC, IP, L4+
  IP protocol + input port 
  Value: timestamp + output port 

- Implementation
  - Hash table with blacklisted misbehaving hosts
  - Hash table with per-flow forwarding entries
  - One split entry per bucket for flow tables
Model can represent many lookups
Outline

Programming model

Applications: *dataplane lookup for protocols*

PLUG Hardware

Results
Architecture

- Scalable tiled architecture
- Simple processing elements
- Simple on-chip network
Architecture

Fixed delays
Static instruction ordering

PE Array
SRAM Bank
Core0 Core1 Core2 Core3
Core12 Core13 Core14 Core15
Core16 Core17 Core18 Core19

North
addr, datain
dataout

Router 0
Router 1

West
East
South
Program Execution

PLUG Compiler
PLUG Compiler

Partition pages to fit on tiles
PLUG Compiler
PLUG Execution
PLUG Execution

Lookup 0

1

40
PLUG Execution

Lookup transformed into physical pipeline on chip
Architecture Highlights

- Globally statically scheduled
  - Conflict free execution
  - Fixed delays

- Memory dominated design
  - 74% chip area SRAM

- Throughput of one lookup every cycle
  - 1 billion decisions per second at 1GHz
Hardware exploits simplicity to achieve efficiency
Outline

Programming model

Applications: *dataplane lookup for protocols*

PLUG Hardware

Results

27.63.2.3 64.65.1.3 78.1.3.24

Ethernet IP Lkup Seattle Ethane
Results

- **PLUG parameters**
  - 256KB SRAM, 32 PEs, 8 routers
    - Design tradeoffs in paper
  - 16 tiles, 4MB storage, 32 \( mm^2 \) area

- **Specialized modules**
  - Netlogic NLA 9000 (IPv4 Lookup)

- **Metrics**
  - Throughput and power
All Lookups on Single Hardware
Power under 2 Watts for all applications
Comparison to NLA9000

<table>
<thead>
<tr>
<th>Metric</th>
<th>NLA9000</th>
<th>PLUG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>6.5 (W)</td>
<td>1.4 (W)</td>
</tr>
<tr>
<td>Latency</td>
<td>160 (ns)</td>
<td>148 (ns)</td>
</tr>
<tr>
<td>Throughput</td>
<td>300 (MDPS)</td>
<td>633 (MDPS)</td>
</tr>
</tbody>
</table>

Higher throughput and lower power
Simple hardware provides efficiency
Conclusions

- **Flexibility**
  - Rapid development of new functionality
  - Update functionality in deployed equipment

- **Core ideas**
  - Intuitive dataflow programming model
  - Simple tiled hardware

- **Performance**
  - Throughput, latency, power and area are competitive
Future work

- Improve tools
  - Optimized scheduler and scheduling heuristics
  - Optimized code-generator and compiler

- Applications
  - Packet classification
  - Signature matching (transition table compression and running the loop on the PLUG)

- Architecture
  - Microarchitecture details
  - Prototype chip implementation
http://www.cs.wisc.edu/vertical/plug

Questions?
Backup slides
Flexible Better Than Specialized Paradox

- Specialized hardware
  - May use associative hardware broadcast

- PLUGs off-load complexity to software
  - May require larger dies

- Limitations:
  - Multiple accesses require multiple routine invocations
  - Specialized hardware may be richer within its sub-domain
Multiple Applications on one PLUG

IPv6   Seattle   Ethane
IPv6 Lookup and Seattle
# Review of Applications

<table>
<thead>
<tr>
<th>Task</th>
<th>Core lookup</th>
<th>Current solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP lookup</td>
<td>longest matching prefix</td>
<td>associative matching (TCAM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pipelined tries (SRAM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pipelined tries (SRAM)</td>
</tr>
<tr>
<td>Packet classification</td>
<td>multi-dimensional prefix and range match</td>
<td>associative matching (TCAM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>decision trees (SRAM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>forests of tries (SRAM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>linear traversal, etc. (SRAM)</td>
</tr>
<tr>
<td>Ethernet forwarding</td>
<td>keyed lookup</td>
<td>associative matching (CAM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hash table (SRAM)</td>
</tr>
<tr>
<td>Signature matching</td>
<td>look up next DFA state</td>
<td>transition table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compressed (SRAM)</td>
</tr>
<tr>
<td>NAT</td>
<td>keyed lookup</td>
<td>hash table (SRAM)</td>
</tr>
<tr>
<td>Seattle</td>
<td>keyed lookup range matching</td>
<td>hash table (SRAM)</td>
</tr>
<tr>
<td>Ethane</td>
<td>multiple keyed lookup</td>
<td>hash table (SRAM)</td>
</tr>
</tbody>
</table>
void l_work(plug_logical_message *msg) {
    int ip_part = msg->get_data(0);
    int addr = msg->get_data(3);
    p_word<15> block = get_l_data_word(addr);
    r_ret = block[0];  // base index
    // count bitmap
    for(int i = 0; i<=(ip_part&0x1F); i++) {
        if (block[i+1]) ++r_ret;
    if (block(ip_part & 0x1F)+34) {
        c_ret = block[33];
        .... // more code here
    } else c_ret = -1;
    omsg.dest_code_blk_num = 0;
    omsg.dest_page(4);
    omsg.data.set_word(0, r_ret-1);
    omsg.data.set_word(1, msg->get_data(1));
    omsg.data.set_word(2, msg->get_data(2));
    omsg.data.set_word(3, c_ret);
    send(o_msg);
}

ld112 R12-R18 [R11]; read memory
mvp 5, R19,0,R2,12; last 5 bits
sub R20,R19,16; of ip-address
cut R15,R19,0; count bitmap
cut R11,R19,12
add R15, R15, R11
je R20, loop2; count 2nd bitmap?
    cut R18,R19,0
cut R14,R19,12
    add R15, R15, R14
    j loop3
loop2:
    Nops; pad with nops
loop3:
    add R1,R15,R13; add base index
    add R4,R14,R12;
    snd R0-R4 N1; send message

mvp macro is
expanded into more instructions
Area

Plug area: 1.5X to 1.9X of ideal
Plug vs. Ideal power estimates

Plug Power: 3.1X to 3.7X of ideal
Lookups in network equipment

- Performance-critical part of packet processing
- Typically involves accessing large data structure with poor temporal locality
  - Using separate lookup module justified
- Example uses for lookups
  - Forwarding tables for layer 2 and 3
  - Packet classification, address blacklists
  - Per flow state lookup
  - Compressed transition tables for DFAs used in deep packet inspection
Cost of flexibility

- Memories overhead
  - Scheduling losses (some banks/tiles “unreachable”)
    - Map multiple application to the Plug
  - Fragmentation losses (some banks not fully utilized)
    - Many small 64KB banks instead of large memory

- Processing elements overhead
  - Programmable µcores versus custom hardware
    - Microcode repeated in each tile page mapped to
  - Scheduling losses (some µcores are not used)
  - Fragmentation losses (some banks not fully utilized)
  - Clock gating for unused cores.

- Communication network
  - Responsible for <5% of area and <4% of power in PLUG
Schedule & execute - Ethernet

- Break the data str. and map to grid
- Hash Dst MAC address
- Compute the index of the bucket holding the value
- Send a message to the PLUG
- Message is forwarded to the row holding the bucket
- Parallel lookup
- Return result
Conflict-free execution: in tile

- **Rule 1**: Provide enough cores for “longest” code block
  - Execution model gives each lookup a new core
- **Rule 2**: Resource constrained instruction.
  - One of each type.
  - Must be aligned to the start of code blocks.

Need N cores of N instruction code-block
Conflict-free execution: global

- Must ensure no contention in routers
  - External
    - Plug receive a single message in a given cycle.
  - Internal
    - Code-block rules guarantee only one message emitted by tile every cycle.

**Rule 3: Propagation discipline**
- Deterministic dimension order routing.
  - Total propagation delay same for all paths

- All this is done by Scheduler
  - Partition pages into small physical pages
  - Map physical pages to grid
Applications-Seattle

- **Implementation**
  - A hash table mapping host addresses to their locations
  - A DHT ring for looking up the resolver switch
  - A next hop array with the next hop port for each switch
Latency for all tiles: 10 clk

Ethernet
Left table

Right table

Ether
IPv4 -
IPv4 -

Latencies for tiles on previous page:

A: $10 + 16 + 9 = 25$ ck
Latencies:
A-F: 14 ck
G: 20 ck
H: 20 ck
I: 20 + 9 = 29 ck
J,K: 9 ck

SEATT
Latencies:
A, E: 21 ck
B, C, F, G: 10 ck
D, H: ??

Ethan
Applications-Ethernet

- **Implementation**
  - Hash table lookup (D-left)
  - 4 entries per bucket
  - Each entry 48-bits addr, 12-bits port, 3-bits timestamp
Applications-Ethane

- **Implementation**
  - Hash table with blacklisted misbehaving hosts
  - Hash table with per-flow forwarding entries
  - One split entry per bucket for flow tables.
Applications-Seattle

- Implementation
  - A hash table mapping host addresses to their locations
  - A DHT ring for looking up the resolver switch
  - A next hop array with the next hop port for each switch
High speed lookups today

- Hardware customized to data plane protocols
  - Even network processors use custom lookup modules
- Advantages of custom hardware
  - High performance and low power
- Disadvantages of custom hardware
  - Expensive to design (many) custom ASICs
  - Long design cycle for new equipment
  - Limited upgrades to data plane in deployed equipment

Functionality defined in hardware
High-speed Lookups Today

- Hardware customized to data plane protocols

Problem: Functionality defined in hardware

Fast...but specialized chips, long design-cycle, limited upgrades
Main result: All lookups on **single** hardware