Raising the Bar for Using GPUs in Software Packet Processing

Anuj Kalia (CMU)
Dong Zhou (CMU)
Michael Kaminsky (Intel Labs)
David Andersen (CMU)
Software Packet Processing

• Is important

• But slow

eXpressive Internet Architecture
GPU acceleration

**IPv4/IPv6**: PacketShader[SIGCOMM 10], GALE[INFOCOM 11], GAMT[ANCS 13], NBA[EuroSys 15]

**NDN**: MATA[NSDI 13]

**NIDS**: Kargus[CCS 12], NBA[EuroSys 15], Snap[ANCS 14]

**Frameworks**: GASPP[ATC 14], Snap[ANCS 14], NBA[EuroSys 15]
Raising the bar: optimize CPU code

Factor speedup

- GPU acceleration
- CPU code optimization (G-Opt)

IPv4
- 1.1
- 1.3
L2 Switch
- 1.9
- 1.8
IPv6
- 4.5
- 4.0
CPU/GPU Packet Processing

> 10x computing power
~ 4x memory bandwidth
Rethink GPU advantages

- **Higher computation power**
  
  Most applications not compute intensive

- **Higher memory bandwidth**
  
  Most applications not memory intensive

- **Memory latency hiding**
Memory latency hiding

**GPUs**

- Fast, hardware-supported context switch

**CPUs**

- CuckooSwitch [CoNEXT 13]: manual group-prefetching
- Grappa [U. Washington]: lightweight context switching to hide RDMA latency
find(key *K, value *V) {
    int i
    for(i = 0; i < B; i++) {
        int idx = hash(K[i])
        value *ptr = table[idx].ptr
        V[i] = *ptr
    }
}

CPU memory latency hiding
Strawman: Group Prefetching

```c
find(key *K, value *V) {
    int i, idx[B]
    value *ptr[B]

    for(i = 0; i < B; i++) {
        idx[i] = hash(K[i])
        prefetch(&table[idx[i]])
    }

    for(i = 0; i < B; i++) {
        ptr[i] = table[idx[i]].ptr
        prefetch(ptr[i])
    }

    V[i] = *ptr[i]
}
```
int idx = hash(K[i])
value *ptr = table[idx].ptr
V[i] = *ptr

Toy hash table

Cuckoo hashing
Programmer writes batched independent code

```c
find(key *K, value *V) {
    int i
    for(i = 0; i < B; i++) {
        int idx = hash(K[i])
        value *ptr = table[idx].ptr
        V[i] = *ptr
    }
}
```

Switching on CPUs is fast with batched independent code.
G-Opt: Element switching!

Specialization to batched independent functions: Save state in local arrays. Switch using `goto`.

**Kernel threads**
Preemptive scheduling
Independent threads
~500 ns (2 M/s)

**Grappa’s user threads**
Cooperative scheduling
Same application
~25 ns (40 M/s)

**GPU threads (SIMD)**
From batched independent code
Hardware speed

**Generality**

**Speed**

**G-Opt elements**
From batched independent code
100-300 M/s
A G-Opt example

```c
find(key K, value V) {
    int idx
    value *ptr

    idx = hash(K)
    _expensive_(&table[idx])
    ptr = table[idx].ptr
    _expensive_(ptr)
    V = *ptr
}
```
Convert to batched function

```c
find(key *K, value *V) {
    int idx[B]
    value *ptr[B]

    idx[I] = hash(K[I])
    _expensive_(&table[idx[I]])
    ptr[I] = table[idx[I]].ptr
    _expensive_(ptr[I])
    V[I] = *ptr[I]
}
```
find(key *K, value *V) {
    int idx[B]
    value *ptr[B]

    idx[I] = hash(K[I])
    _expensive_(&table[idx[I]])
    ptr[I] = table[idx[I]].ptr
    _expensive_(ptr[I])
    V[I] = *ptr[I]
}

Prefetch, Save, Switch

PSS(addr, PP):
    // PREFETCH
    prefetch(addr)

    // SAVE
    PP[I] = PP

    // SWITCH
    I = (I + 1) % B
    goto *PP[I]
find(key *K, value *V) {
    int idx[B]
    value *ptr[B]

    // Setup code
    label_0:

    idx[I] = hash(K[I])
    PSS(&table[idx[I]], label_1)
    label_1:
    ptr[I] = table[idx[I]].ptr

    PSS(ptr[I], label_2)
    label_2:
    V[I] = *ptr[I]

    label_end:
    // Termination code
}

Why G-Opt works

More variables, code, branches = “Optimization”?
# G-Opt for Packet Processing

<table>
<thead>
<tr>
<th>Application</th>
<th>Code</th>
<th>Lines of code</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4 forwarding</td>
<td>DPDK library</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>IPv6 forwarding</td>
<td>DPDK library</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>Layer-2 switch</td>
<td>Our own</td>
<td>54</td>
<td>2</td>
</tr>
<tr>
<td>NDN forwarding</td>
<td>Our own</td>
<td>79</td>
<td>2</td>
</tr>
</tbody>
</table>
## Experiment Setup

<table>
<thead>
<tr>
<th></th>
<th>Intel Xeon E5-2680</th>
<th>NVIDIA GTX 980</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution units</strong></td>
<td>8 SandyBridge cores</td>
<td>2048 CUDA cores</td>
</tr>
<tr>
<td><strong>Sequential memory bandwidth</strong></td>
<td>51.2 GB/s</td>
<td>224 GB/s</td>
</tr>
</tbody>
</table>

40 Gbps network (2x dual port 10GbE)
IPv4 forwarding

1.6x throughput increase
Cost of IPv4 lookup ~9%
Layer-2 switch

Throughput (Mpps)

Total CPU cores

Baseline

G-Opt

GPU

1.8x throughput increase
IPv6 forwarding

- Throughput (Mpps)
- Total CPU cores

Baseline - GPU - G-Opt

- GPU (C=4) > G-Opt (C=4)
- G-Opt (C=5) ~ GPU (C=4)

3.8x throughput increase
GPU assumptions $\Rightarrow$ CPU opts

Optimizations for batched independent code

• **This talk:** G-Opt: General-purpose, automatic memory latency hiding

• **In paper:** Manual optimization of CPU pattern matching: 2.4x speedup

• **The future:** <your optimization here>
Summary

- Improve CPU packet processing under GPU assumptions
- Fast switching for memory latency hiding
- *Raising the bar with better baselines*
- Code is online: [https://github.com/efficient/gopt](https://github.com/efficient/gopt)
Thanks!