# Raising the Bar for Using GPUs *in* Software Packet Processing

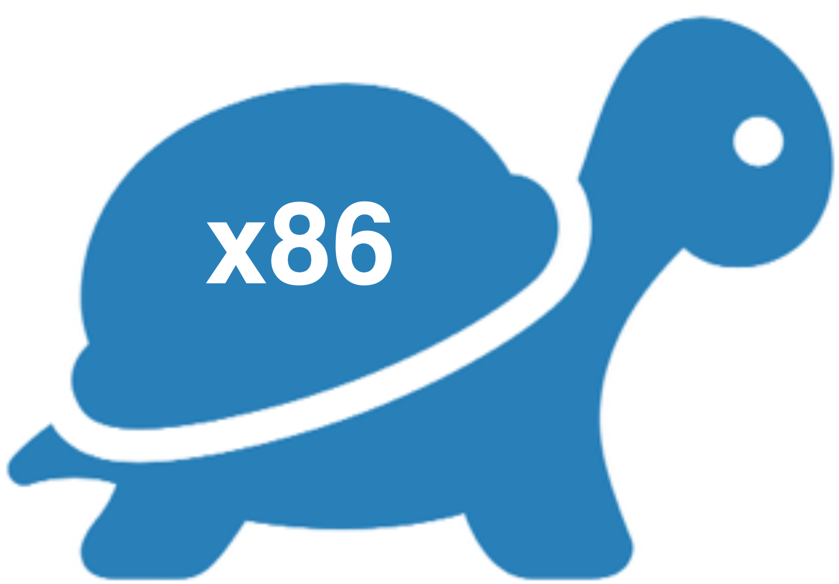
**Anuj Kalia (CMU)**
Dong Zhou (CMU)
Michael Kaminsky (Intel Labs)
David Andersen (CMU)

# Software Packet Processing

- Is important

eXpressive Internet Architecture

- But slow




# GPU acceleration

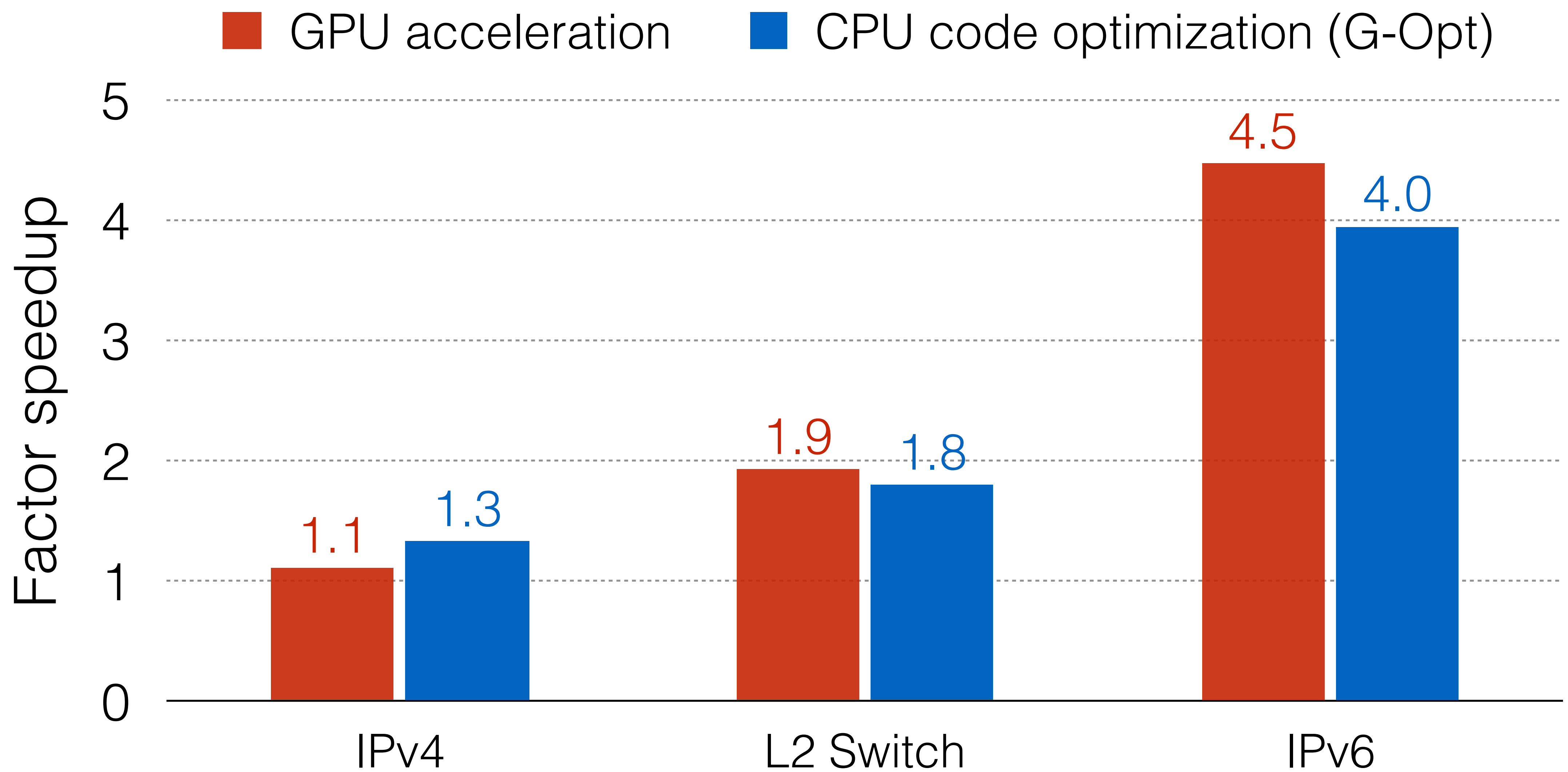
**IPv4/IPv6**: PacketShader[SIGCOMM 10], GALE[INFOCOM 11], GAMT[ANCS 13], NBA[EuroSys 15]

**NDN**: MATA[NSDI 13]

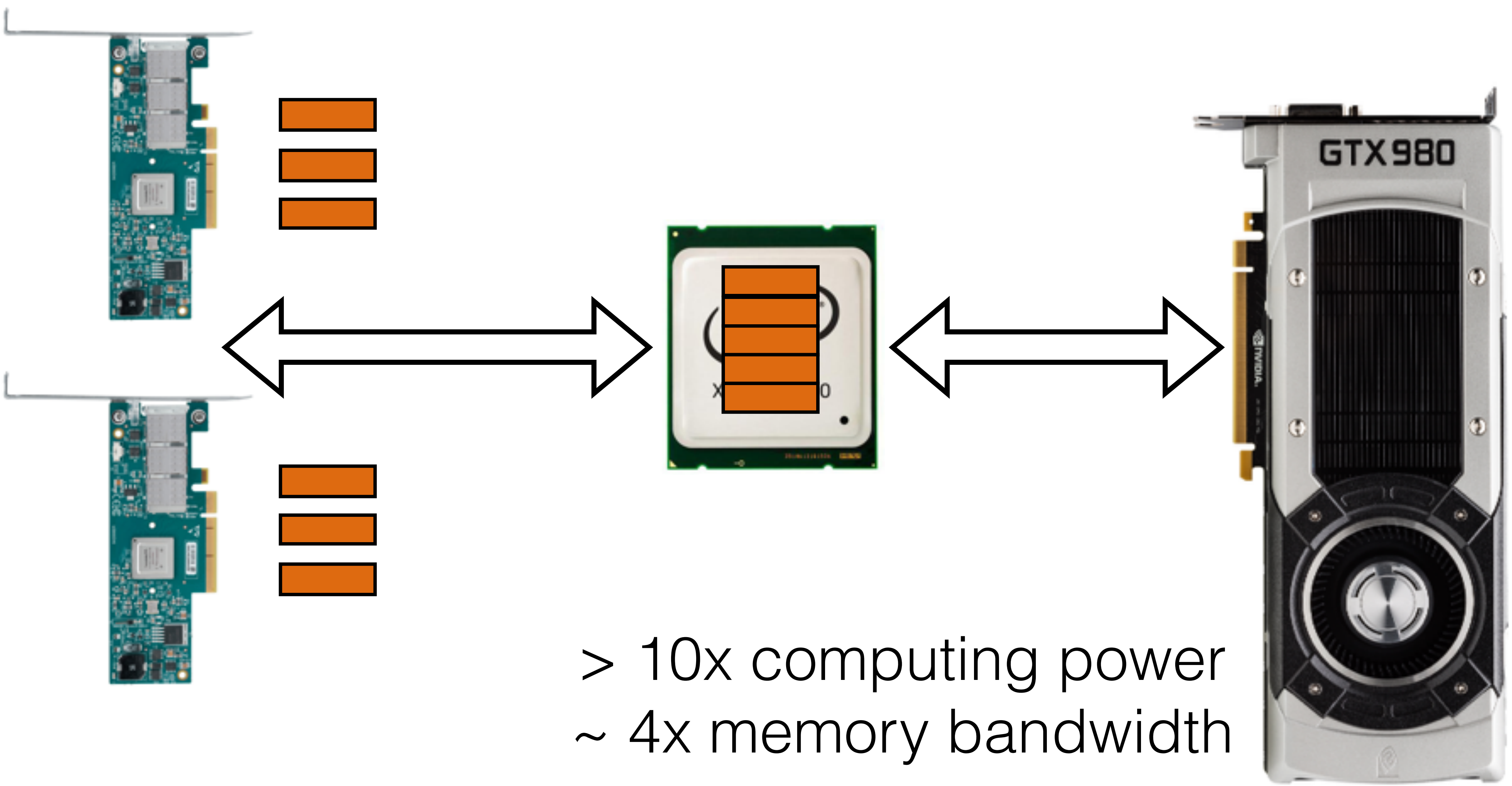
**NIDS**: Kargus[CCS 12], NBA[EuroSys 15], Snap[ANCS 14]

**Frameworks**: GASPP[ATC 14], Snap[ANCS 14], NBA[EuroSys 15]

# Raising the bar: optimize CPU code



# CPU/GPU Packet Processing



# Rethink GPU advantages

~~Higher computation power~~

Most applications not compute intensive

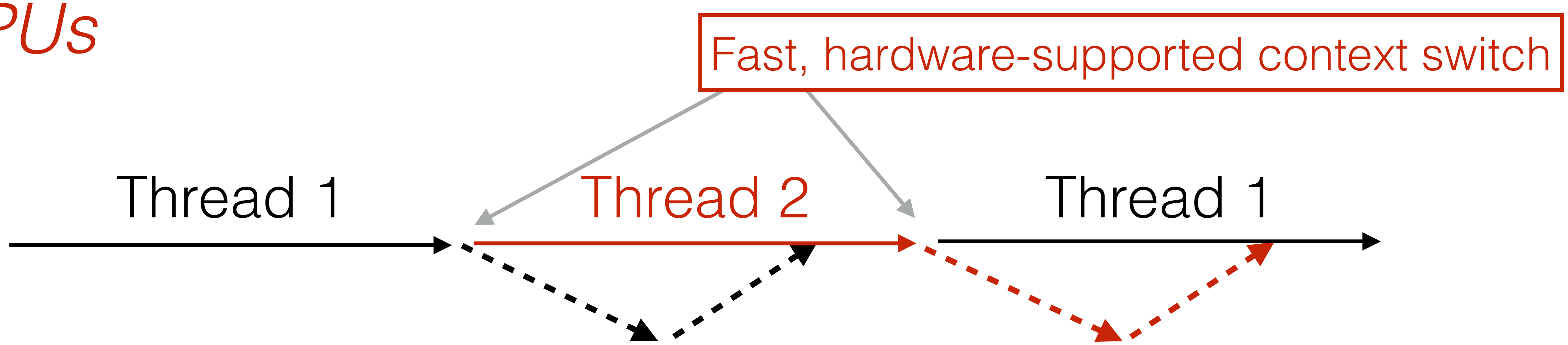
~~Higher memory bandwidth~~

Most applications not memory intensive

Memory latency hiding ✓

# Memory latency hiding

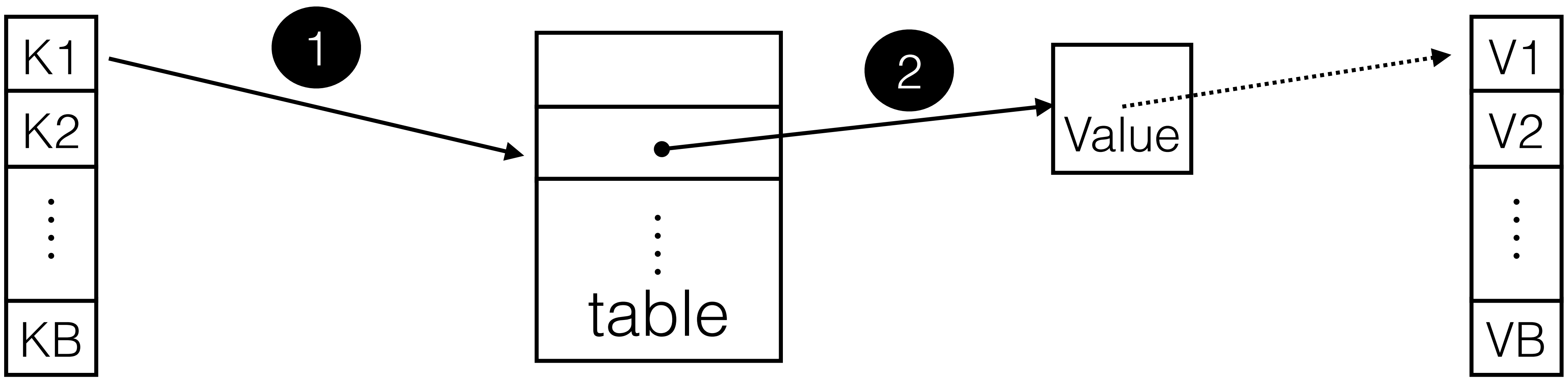
*GPUs*




*CPUs*

- CuckooSwitch [CoNEXT 13]: manual group-prefetching
- Grappa [U. Washington]: lightweight context switching to hide RDMA latency

# CPU memory latency hiding

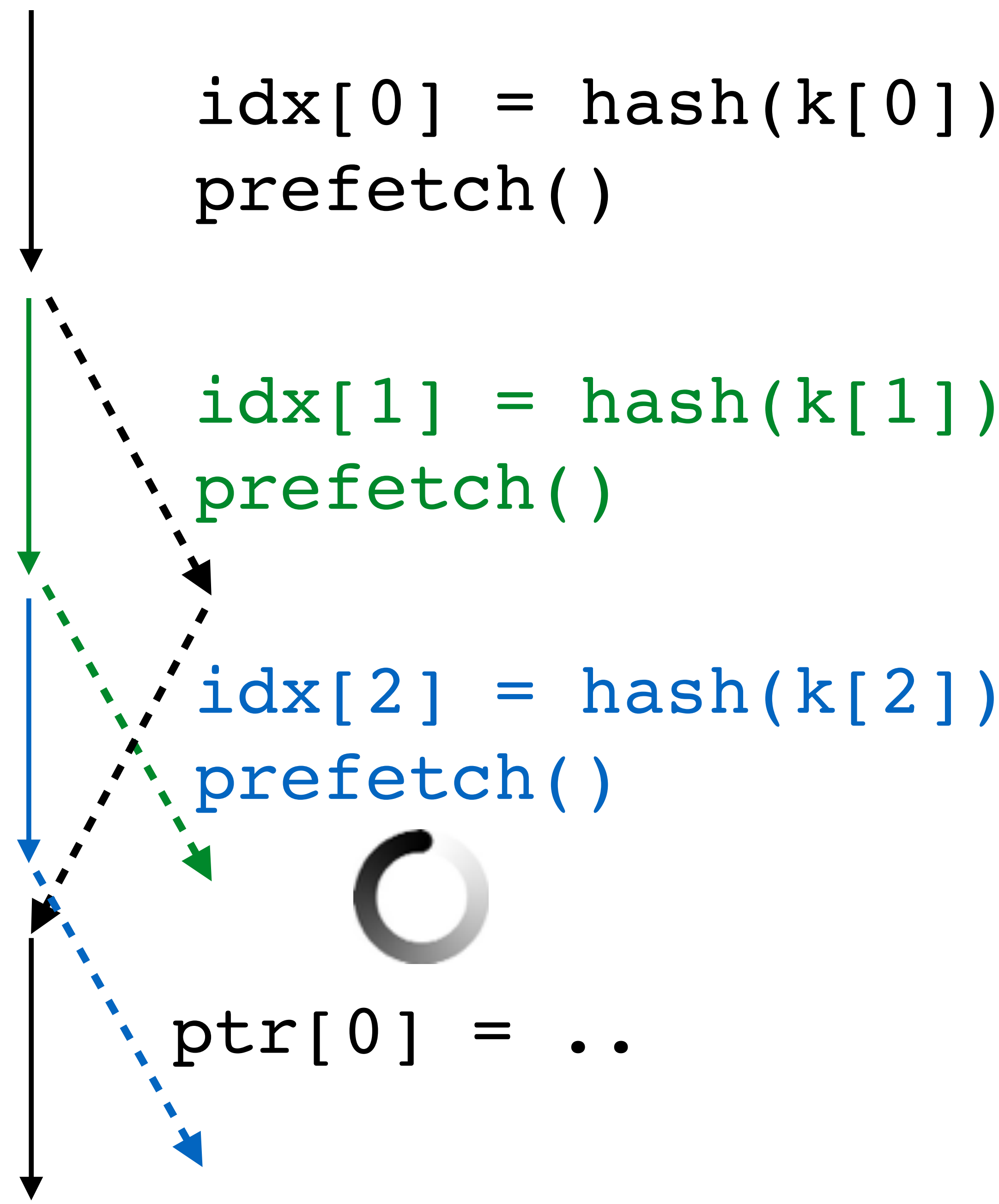



```
find(key *K, value *V) {
   int i
   for(i = 0; i < B; i++) {
 1  int idx = hash(K[i])
 2  value *ptr = table[idx].ptr
    V[i] = *ptr
   }
}
```

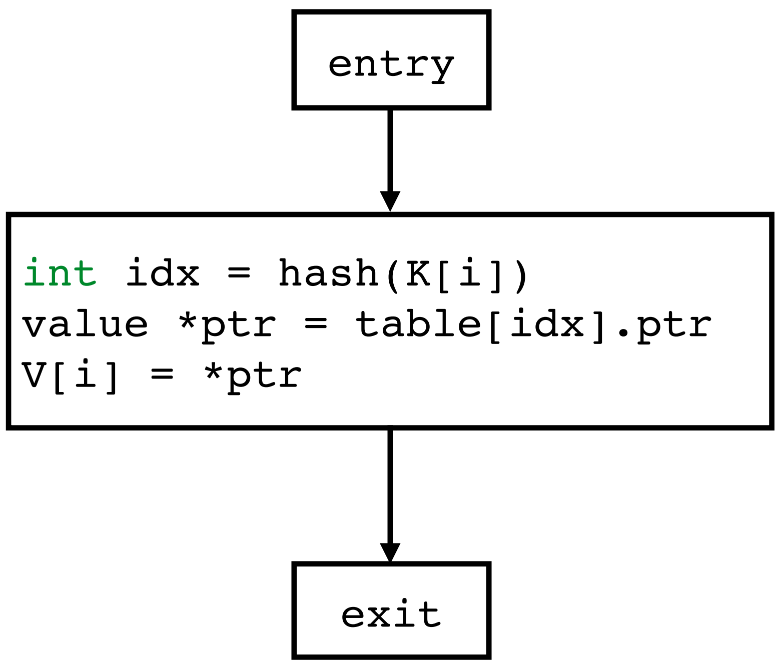
**Cache misses!**

# Strawman: Group Prefetching

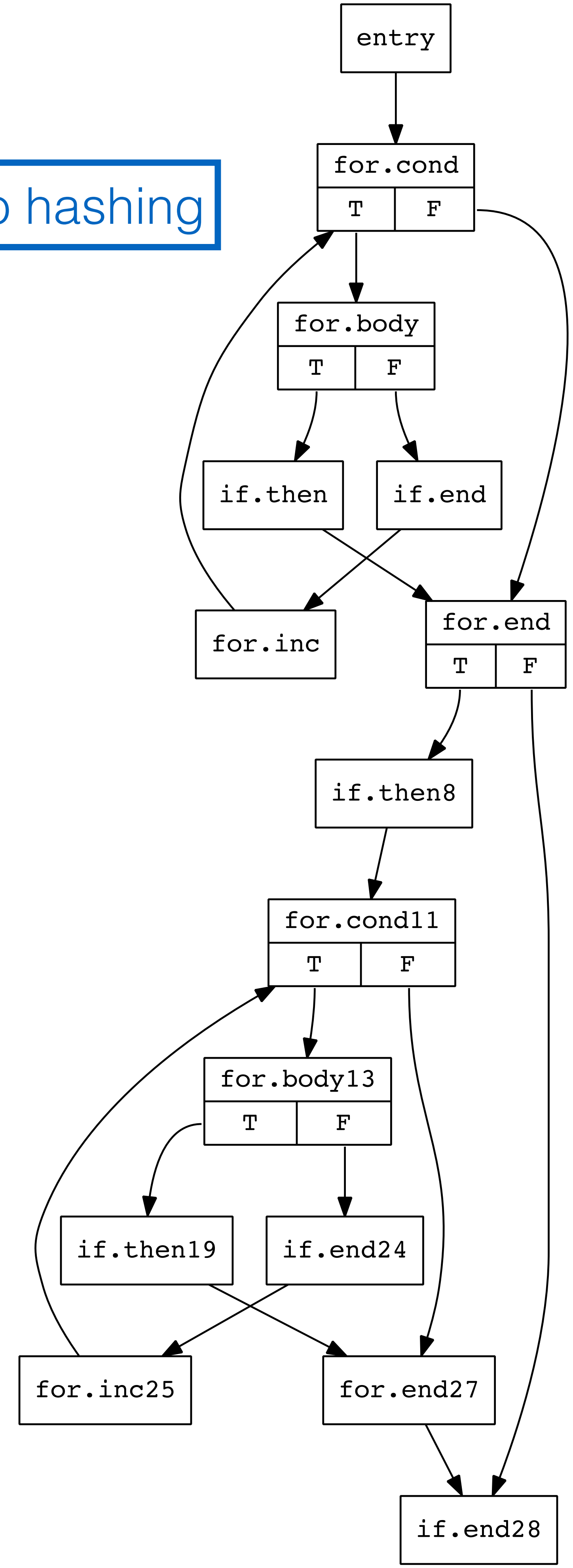
```
find(key *K, value *V) {
  int i, idx[B]
  value *ptr[B]

  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]
}
```



Toy hash table




Cuckoo hashing



# GPU programming model

*Programmer writes batched independent code*

```
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!

**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

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

# A G-Opt example

```
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

```
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]

}
```

# State = Arrays + saved PPs

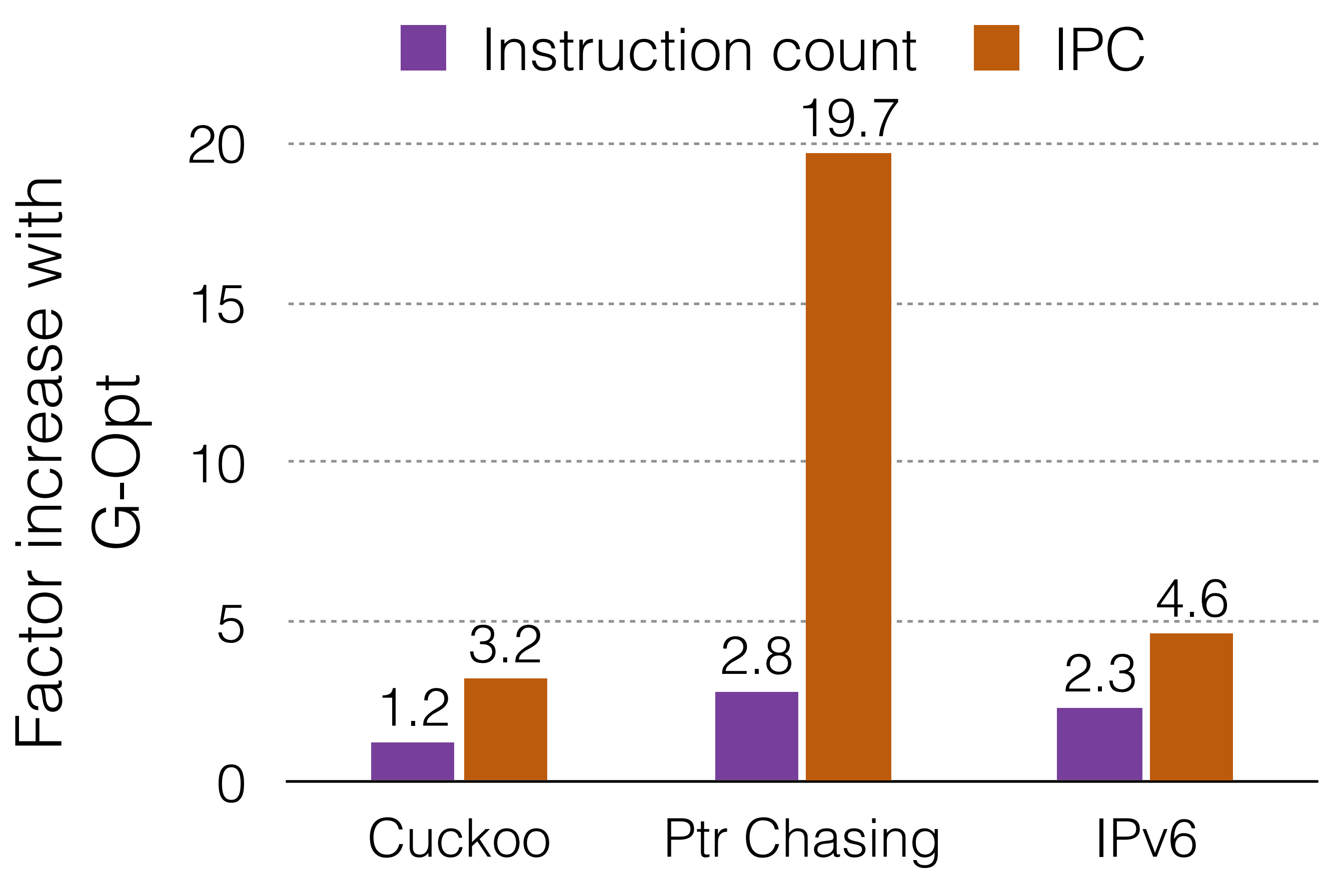
```
find(key *K, value *V) {
  int idx[B]
  value *ptr[B]

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

}
```

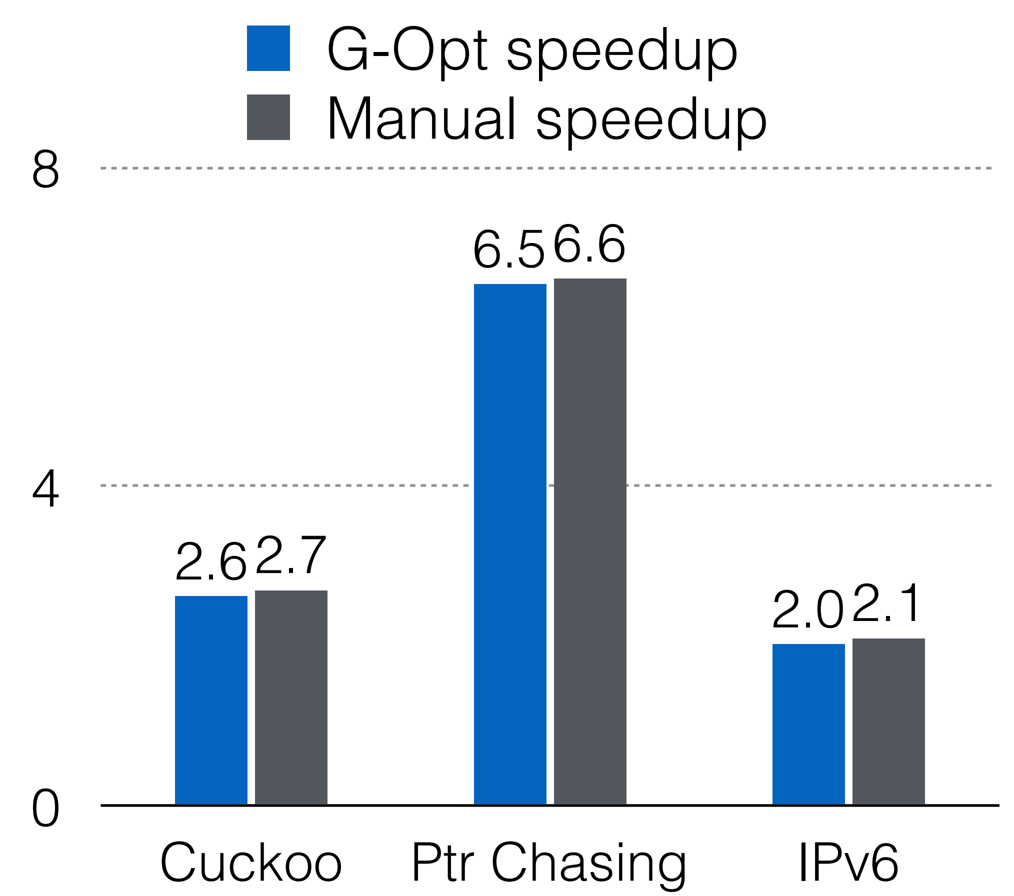
```
Prefetch, Save, Switch

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"?*

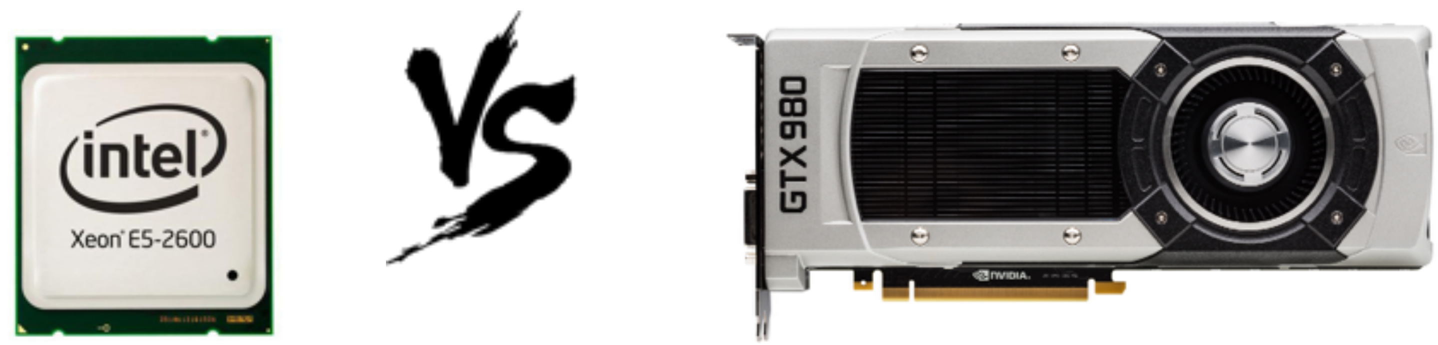


# G-Opt for Packet Processing

| Application | Code | Lines of code | Annotations |
|---|---|---|---|
| **IPv4 forwarding** | DPDK library | 42 | 1 |
| **IPv6 forwarding** | DPDK library | 43 | 1 |
| **Layer-2 switch** | Our own | 54 | 2 |
| **NDN forwarding** | Our own | 79 | 2 |

# Experiment Setup

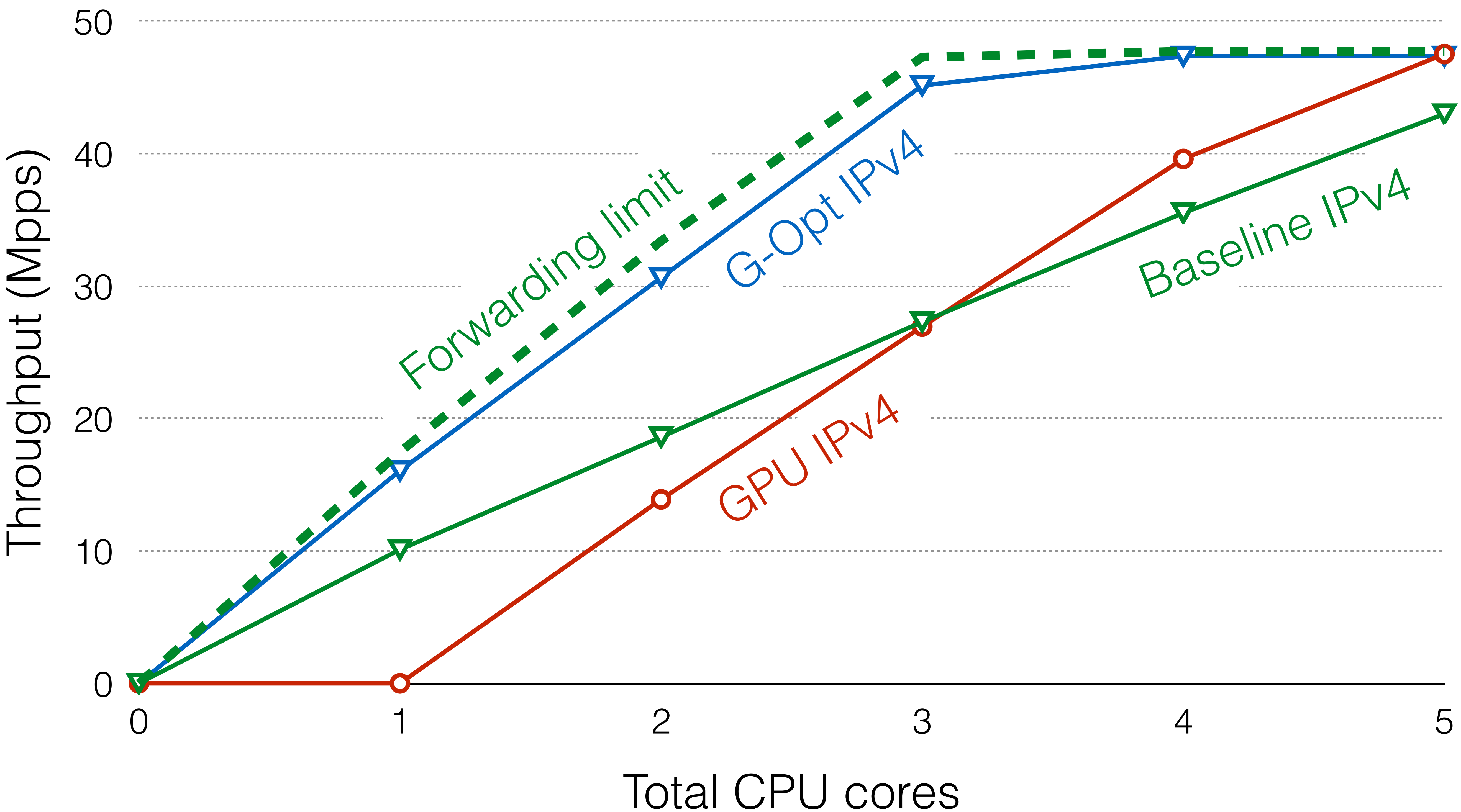



| | Intel Xeon E5-2680 | NVIDIA GTX 980 |
|---|---|---|
| **Execution units** | 8 SandyBridge cores | 2048 CUDA cores |
| **Sequential memory bandwidth** | 51.2 GB/s | 224 GB/s |

40 Gbps network (2x dual port 10GbE)

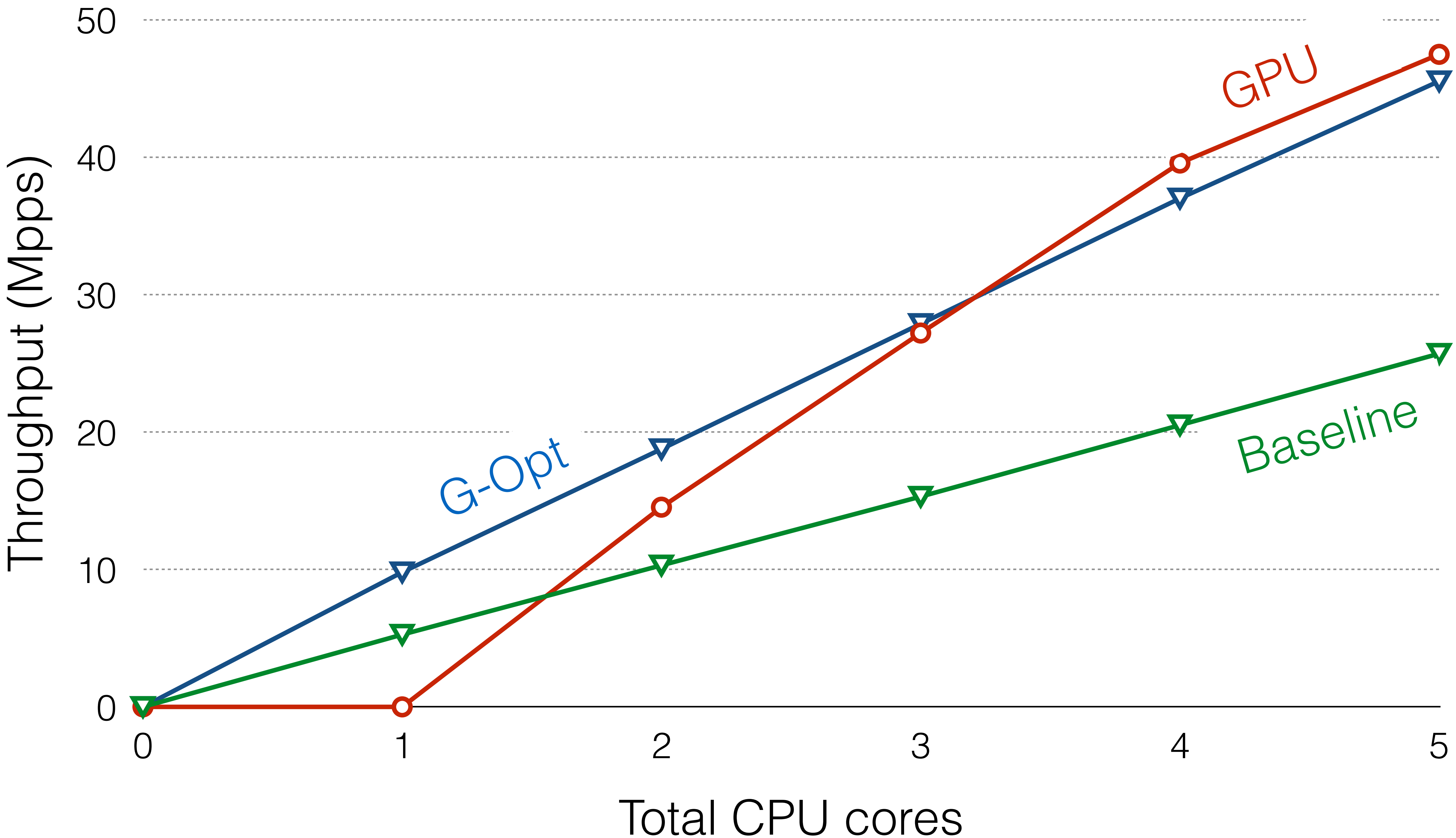
# IPv4 forwarding

1.6x throughput increase
Cost of IPv4 lookup ~9%

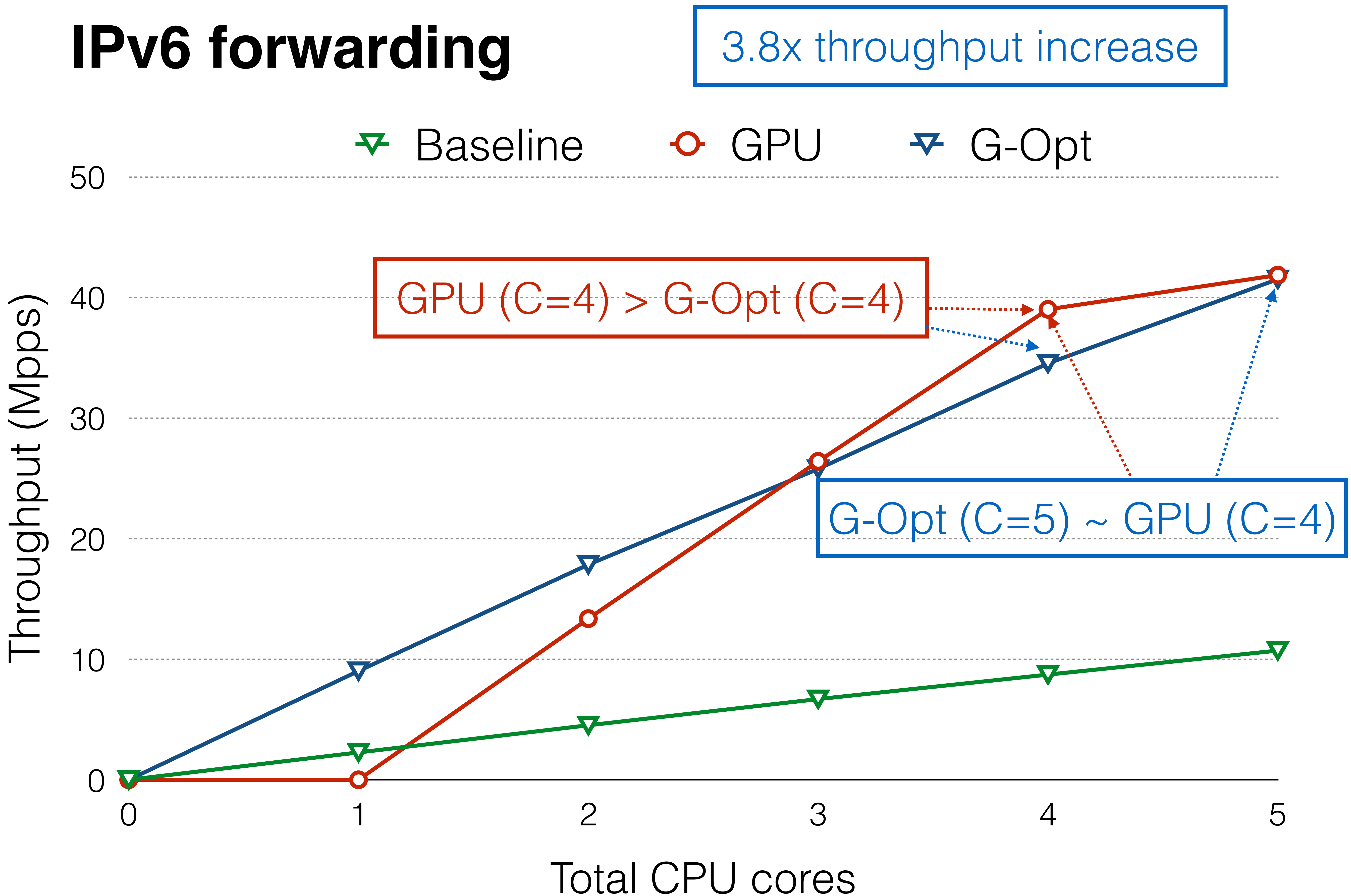


# Layer-2 switch

1.8x throughput increase



# IPv6 forwarding



# GPU assumptions ⇒ 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

Thanks!