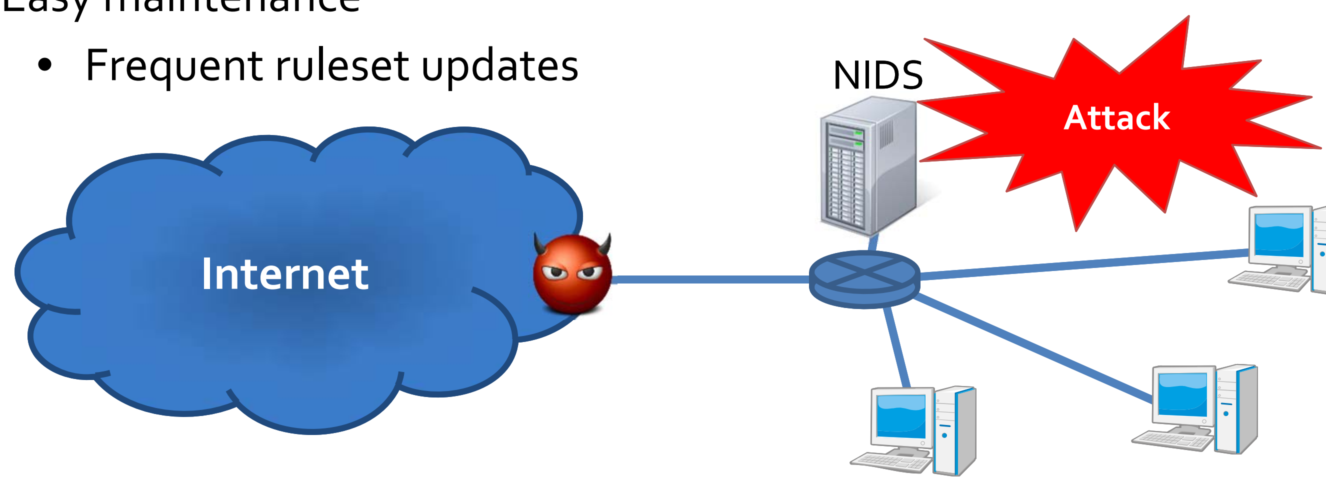
Kargus: A Highly-scalable Software-based Intrusion Detection System

M. Asim Jamshed*, Jihyung Lee†, Sangwoo Moon†, Insu Yun*, Deokjin Kim‡, Sungryoul Lee‡, Yung Yi†, KyoungSoo Park*

* Networked & Distributed Computing Systems Lab, KAIST
† Laboratory of Network Architecture Design & Analysis, KAIST
‡ Cyber R&D Division, NSRI

Network Intrusion Detection Systems (NIDS)

- Detect known malicious activities
  - Port scans, SQL injections, buffer overflows, etc.
- Deep packet inspection
  - Detect malicious signatures (rules) in each packet
- Desirable features
  - High performance (> 10Gbps) with precision
  - Easy maintenance
    - Frequent ruleset updates



Hardware vs. Software

- H/W-based NIDS
  - Specialized hardware
    - ASIC, TCAM, etc.
  - High performance
  - Expensive
    - Annual servicing costs
  - Low flexibility
- S/W-based NIDS
  - Commodity machines
  - High flexibility
  - Low performance
    - DDoS/packet drops

Cisco: IDS/IPS Sensors (10s of Gbps)

~ US$ 20,000 - 60,000

McAfee: IDS/IPS M8000 (10s of Gbps)

~ US$ 10,000 - 24,000

Snort: Open-source S/W

≤ ~2 Gbps

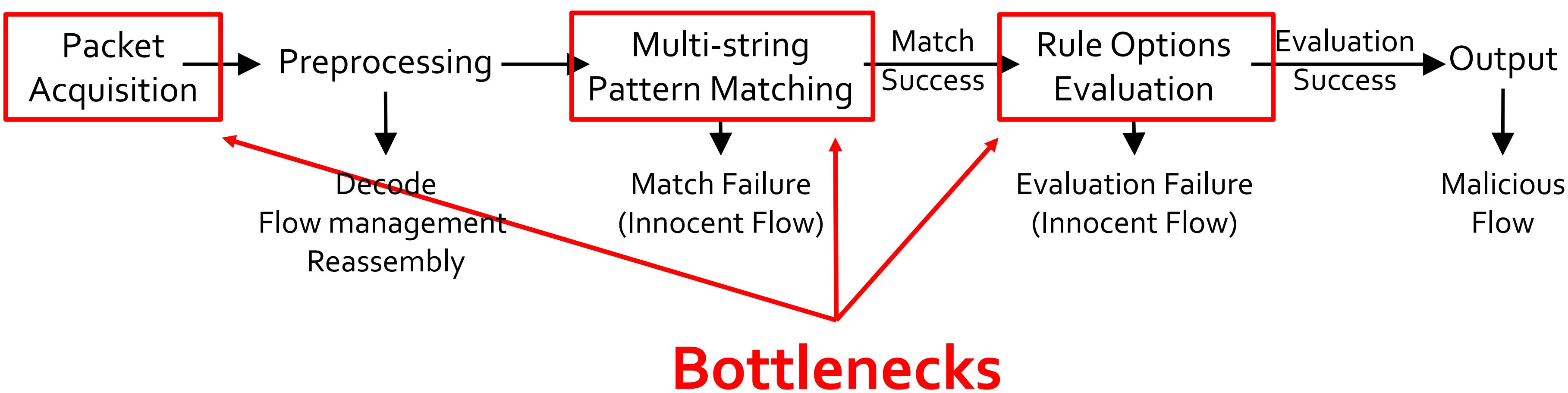
Goals

– High performance

- S/W-based NIDS
  - Commodity machines
  - High flexibility

Typical Signature-based NIDS Architecture

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80
(msg:"possible attack attempt BACKDOOR optix runtime detection"; content:"/whitepages/page_me/100.html";
pcre:"/body=\x2521\x2521\x2521Optix\s+Pro\s+v\d+\x252E\d+\S+sErver\s+Online\x2521\x2521\x2521/")
```



* PCRE: Perl Compatible Regular Expression

Contributions

Goal A highly-scalable software-based NIDS for high-speed network

***Slow* software NIDS** → ***Fast* software NIDS**

Bottlenecks		***Solutions***
Inefficient packet acquisition	→	Multi-core packet acquisition
Expensive string & PCRE pattern matching	→	Parallel processing & GPU offloading

Outcome

- Fastest S/W signature-based IDS: **33Gbps**
- 100% malicious traffic: **10 Gbps**
- Real network traffic: **~24 Gbps**

Challenge 1: Packet Acquisition

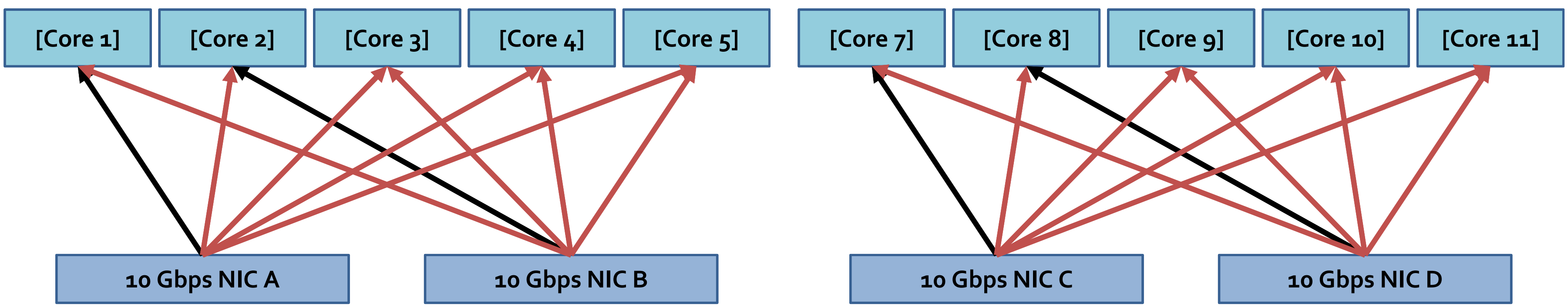
- Default packet module: Packet CAPture (PCAP) library
  - Unsuitable for multi-core environment
  - Low performing
  - More power consumption
- Multi-core packet capture library is required

Packet RX bandwidth*

0.4-6.7 Gbps

CPU utilization

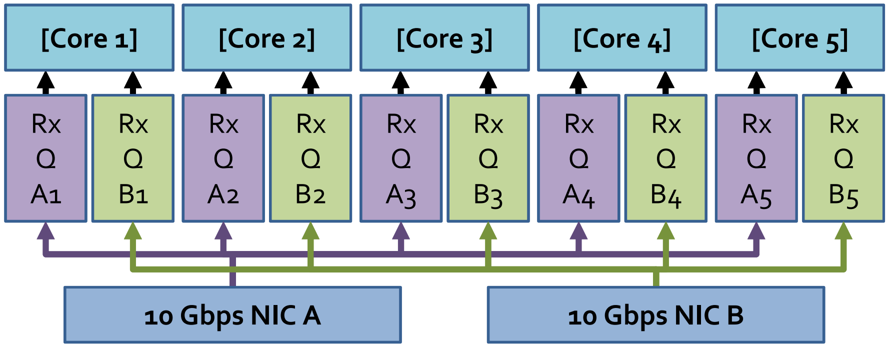
100 %



* Intel Xeon X5680, 3.33 GHz, 12 MB L3 Cache

Solution: PacketShader I/O

- PacketShader I/O
  - Uniformly distributes packets based on flow info by RSS hashing
    - Source/destination IP addresses, port numbers, protocol-id
  - 1 core can read packets from RSS queues of multiple NICs
  - Reads packets in batches (32 ~ 4096)
- Symmetric Receive-Side Scaling (RSS)
  - Passes packets of 1 connection to the same queue



Packet RX bandwidth
~~0.4 ~ 6.7 Gbps~~
40 Gbps

CPU utilization
~~100 %~~
16-29%

* S. Han et al., "PacketShader: a GPU-accelerated software router", ACM SIGCOMM 2010

Challenge 2: Pattern Matching

- CPU intensive tasks for serial packet scanning

- Major bottlenecks
  - Multi-string matching (Aho-Corasick phase)
  - PCRE evaluation (if 'pcre' rule option exists in rule)

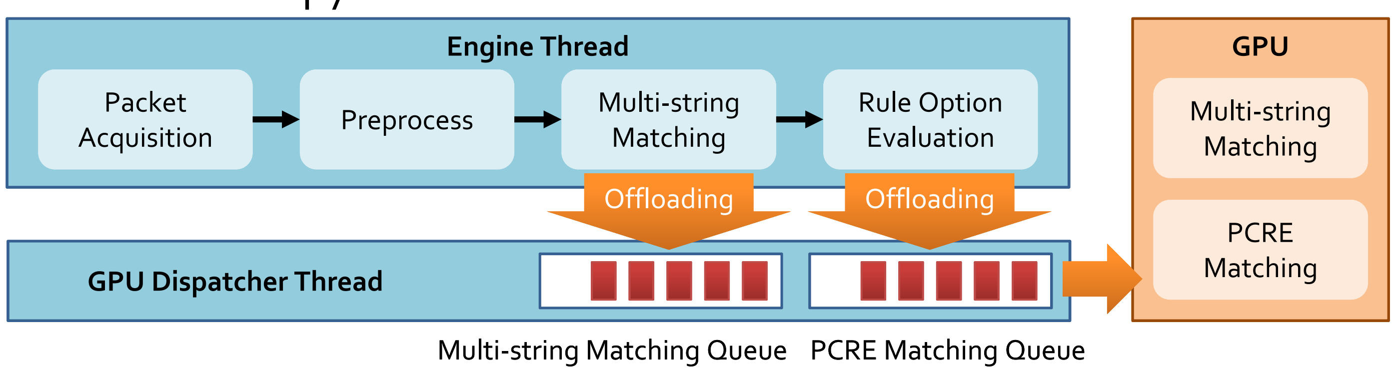
- On an Intel Xeon X5680, 3.33 GHz, 12 MB L3 Cache
  - Aho-Corasick analyzing bandwidth per core: **2.15 Gbps**
  - PCRE analyzing bandwidth per core: **0.52 Gbps**

Solution: GPU for Pattern Matching

- GPUs
  - Containing 100s of SIMD processors
    - 512 cores for NVIDIA GTX 580
  - Ideal for parallel data processing without branches
- DFA-based pattern matching on GPUs
  - Multi-string matching using Aho-Corasick algorithm
  - PCRE matching
- Pipelined execution in CPU/GPU
  - Concurrent copy and execution

Aho-Corasick bandwidth
~~2.15 Gbps~~
39 Gbps

PCRE bandwidth
~~0.52 Gbps~~
8.9 Gbps



Optimization 1: IDS Architecture

- How to best utilize the multi-core architecture?
- Pattern matching is the eventual bottleneck

Function	Time %	Module
acsmSearchSparseDFA_Full	51.56	multi-string matching
List_GetNextState	13.91	multi-string matching
mSearch	9.18	multi-string matching
in_chksum_tcp	2.63	preprocessing

* GNU gprof profiling results

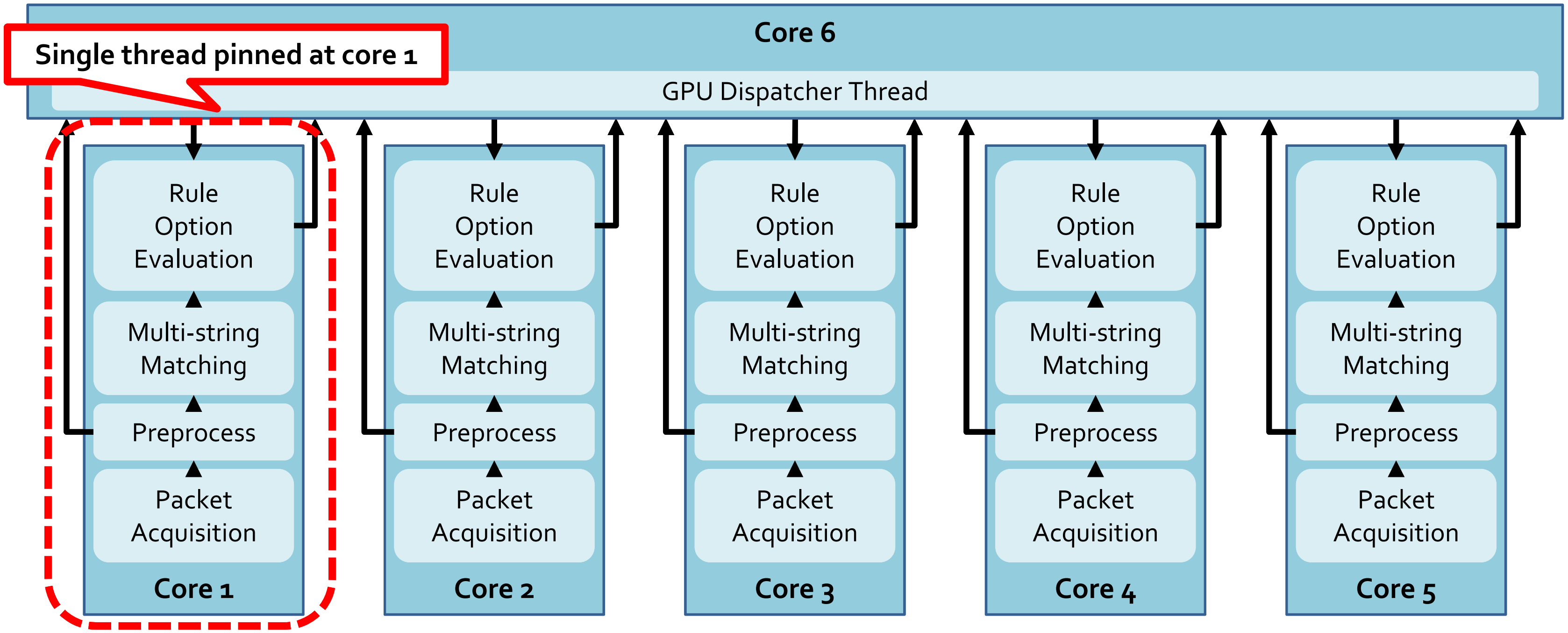
- Run entire engine on each core

Solution: Single-process Multi-thread

- Runs multiple IDS engine threads & GPU dispatcher threads concurrently
  - Shared address space
  - Less GPU memory consumption
  - Higher GPU utilization & shorter service latency

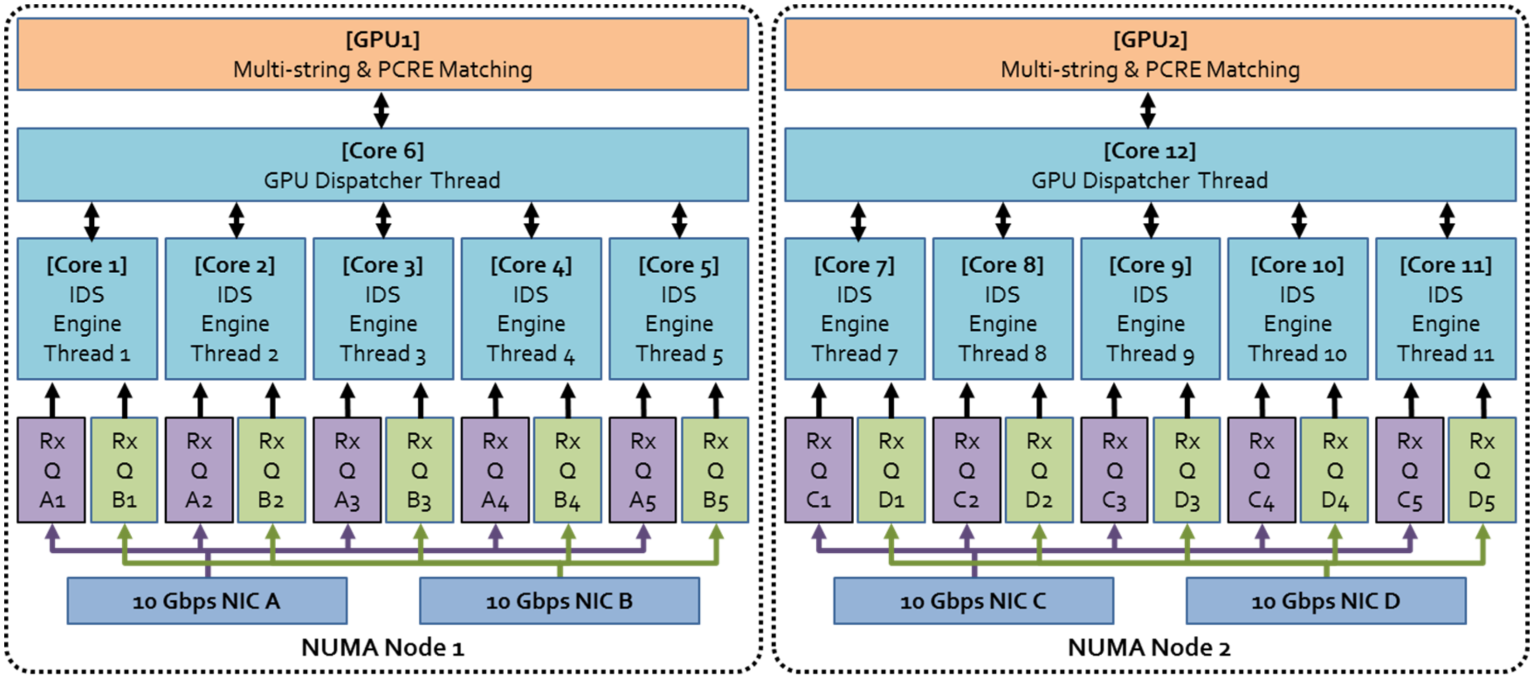
GPU memory usage

1/6



Architecture

- Non Uniform Memory Access (NUMA)-aware
- Core framework as deployed in dual hexa-core system
- Can be configured to various NUMA set-ups accordingly



▲ Kargus configuration on a dual NUMA hexanode machine having 4 NICs, and 2 GPUs

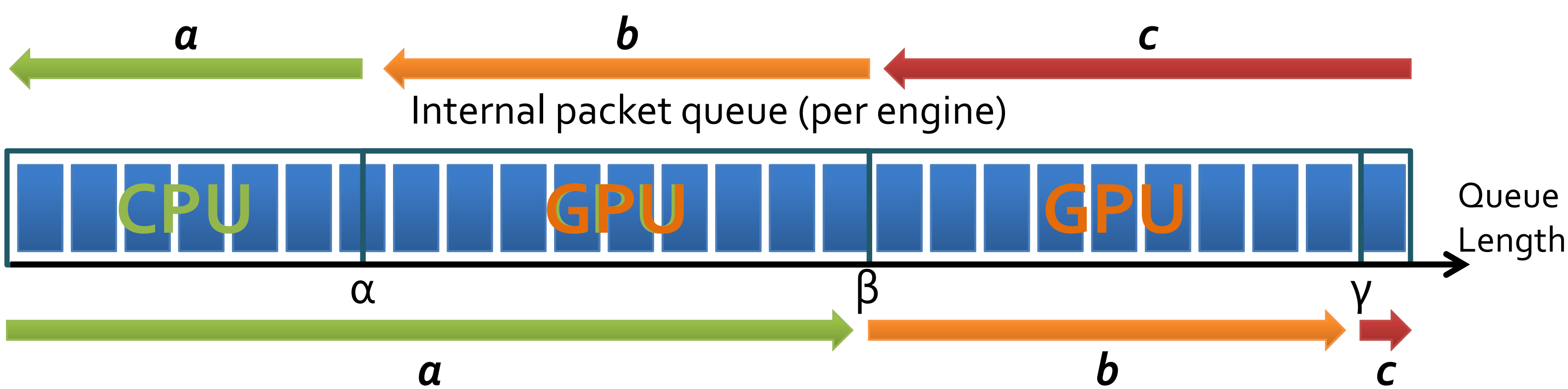
Optimization 2: GPU Usage

- Caveats
  - Long per-packet processing latency:
    - Buffering in GPU dispatcher
  - More power consumption
    - NVIDIA GTX 580: 512 cores
- Use:
  - CPU when ingress rate is low (idle GPU)
  - GPU when ingress rate is high

Solution: Dynamic Load Balancing

- Load balancing between CPU & GPU
  - Reads packets from NIC queues per cycle
  - Analyzes smaller # of packets at each cycle (***a*** < ***b*** < ***c***)
  - Increases analyzing rate if queue length increases
  - Activates GPU if queue length increases

Packet latency with
~~GPU : 640 μsecs~~
CPU: **13 μsecs**



Optimization 3: Batched Processing

- Huge per-packet processing overhead
  - > 10 million packets per second for small-sized packets at 10 Gbps
  - reduces overall processing throughput
- Function call batching
  - Reads group of packets from RX queues at once
  - Pass the batch of packets to each function

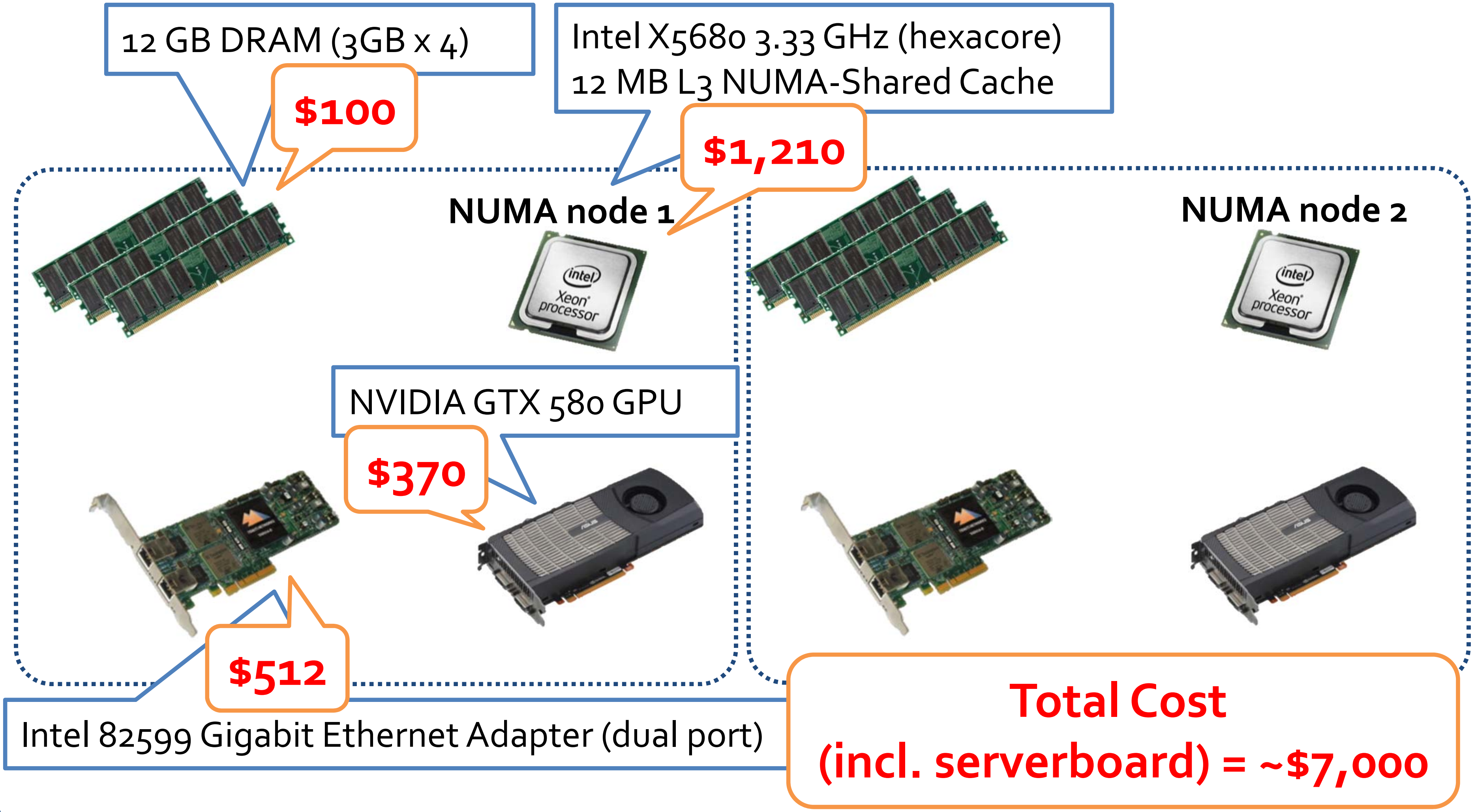
Decode(p) → Preprocess(p) → Multistring_match(p)

↓

Decode(list-p) → Preprocess(list-p) → Multistring_match(list-p)

2X faster processing rate

Kargus Specifications



IDS Benchmarking Tool

- Generates packets at line rate (40 Gbps)
  - Random TCP packets (innocent)
  - Attack packets are generated by attack rule-set
- Support packet replay using PCAP files
- Useful for performance evaluation

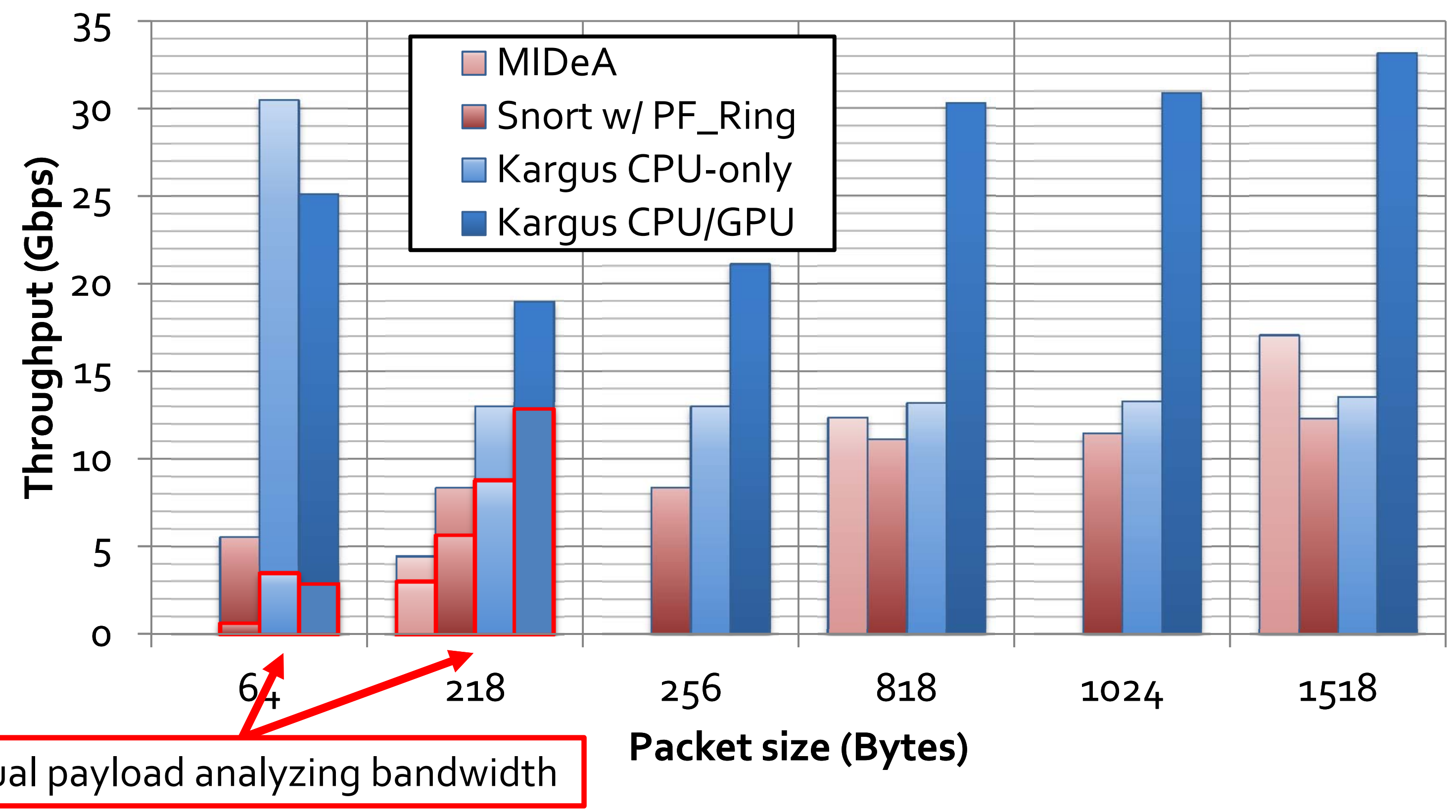
Kargus Performance Evaluation

- Micro-benchmarks
  - Input traffic rate: 40 Gbps
  - Evaluate Kargus (~3,000 HTTP rules) against:
    - Kargus-CPU-only (12 engines)
    - Snort with PF_RING
    - MIDeA*
- Refer to the paper for more results

* G. Vasiliadis et al., “MIDeA: a multi-parallel intrusion detection architecture”, ACM CCS ‘11

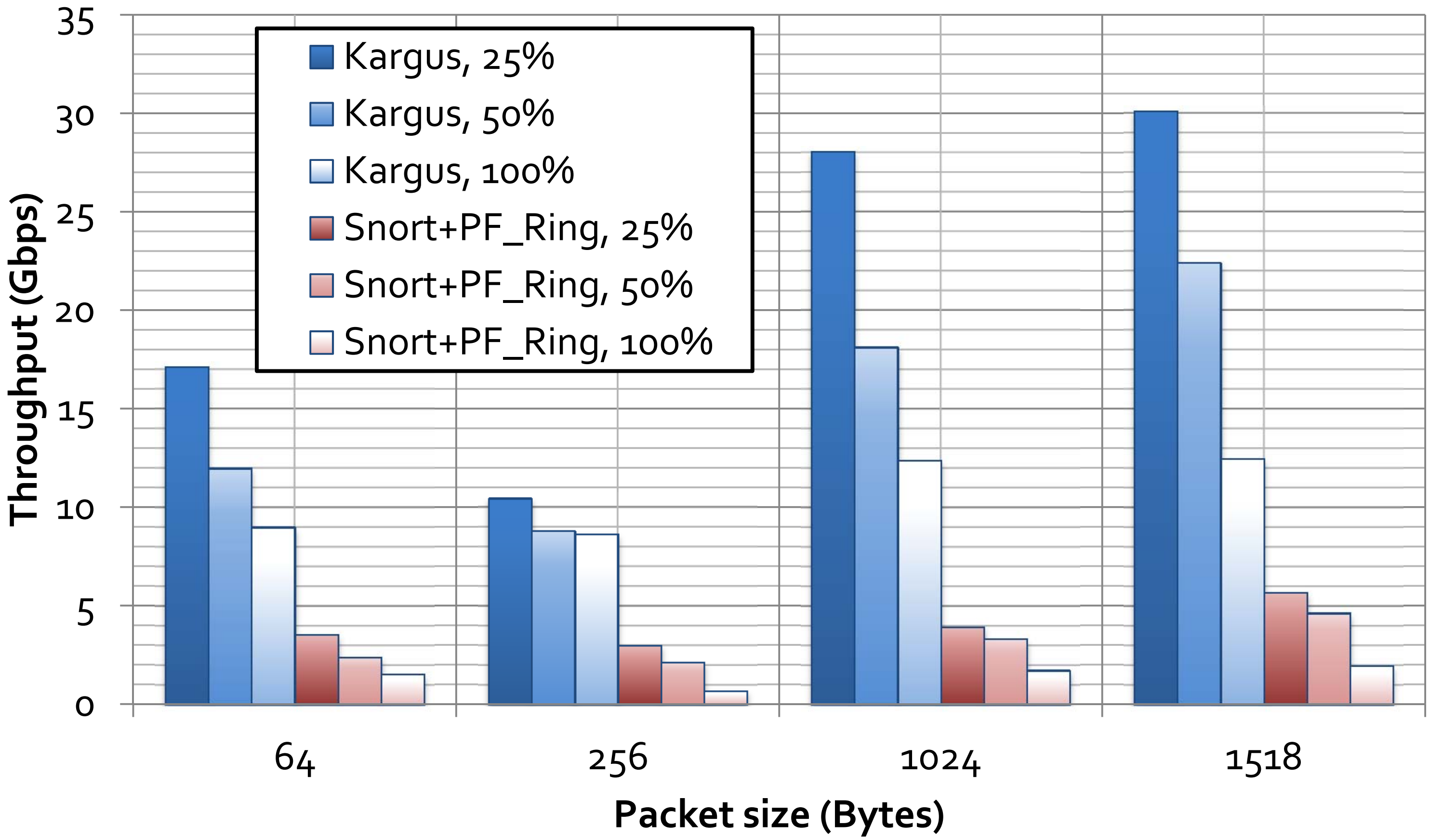
Innocent Traffic Performance

- 2.7-4.5x faster than Snort
- 1.9-4.3x faster than MIDeA



Malicious Traffic Performance

- 5x faster than Snort



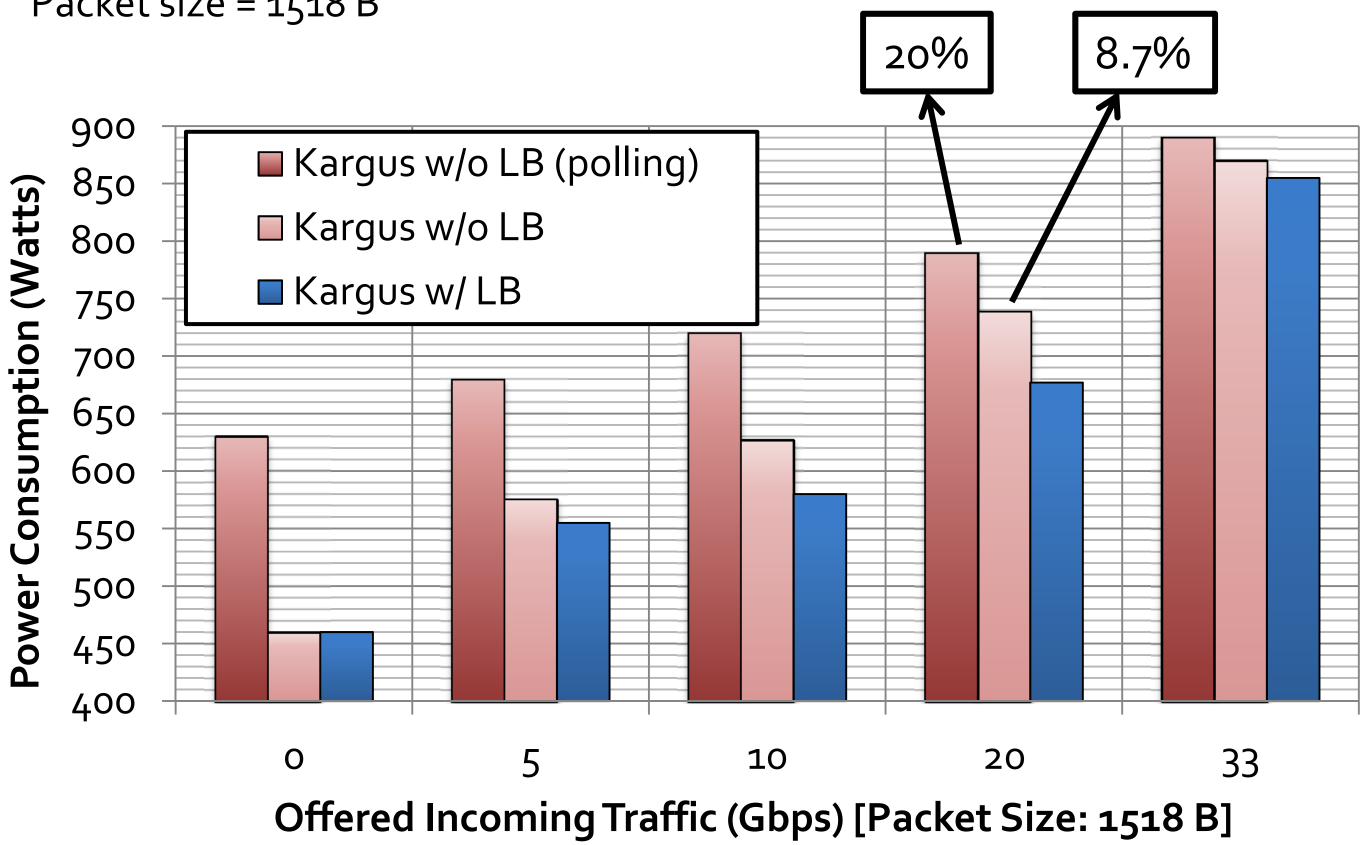
Real Network Traffic

- Three 10Gbps LTE backbone traces of a major ISP in Korea:
  - Time duration of each trace: 30 mins ~ 1 hour
  - TCP/IPv4 traffic:
    - 84 GB of PCAP traces
    - 109.3 million packets
    - 845K TCP sessions

- Total analyzing rate: **25.2 Gbps**
  - Bottleneck: Flow Management (preprocessing)

Effects of Dynamic GPU Load Balancing

- Varying incoming traffic rates
  - Packet size = 1518 B



Conclusion

- Software-based NIDS:
  - Based on commodity hardware
    - Competes with hardware-based counterparts
  - 5x faster than previous S/W-based NIDS
  - Power efficient
  - Cost effective

> 25 Gbps (real traffic)
> 33 Gbps (synthetic traffic)
US $~7,000/-

Thank You

fast-ids@list.ndsl.kaist.edu

https://shader.kaist.edu/kargus/

Backup Slides

Kargus vs. MIDeA

UPDATE	MIDEA	KARGUS	OUTCOME

* G. Vasiliadis, M.Polychronakis, and S. Ioannidis, "MIDeA: a multi-parallel intrusion detection architecture", ACM CCS 2011

Kargus vs. MIDeA

UPDATE	MIDEA	KARGUS	OUTCOME
Packet acquisition	PF_RING	PacketShader I/O	70% lower CPU utilization

* G. Vasiliadis, M.Polychronakis, and S. Ioannidis, "MIDeA: a multi-parallel intrusion detection architecture", ACM CCS 2011

Kargus vs. MIDeA

UPDATE	MIDEA	KARGUS	OUTCOME
Packet acquisition	PF_RING	PacketShader I/O	70% lower CPU utilization
Detection engine	GPU-support for Aho-Corasick	GPU-support for Aho-Corasick & PCRE	65% faster detection rate

* G. Vasiliadis, M.Polychronakis, and S. Ioannidis, "MIDeA: a multi-parallel intrusion detection architecture", ACM CCS 2011

Kargus vs. MIDeA

UPDATE	MIDEA	KARGUS	OUTCOME
Packet acquisition	PF_RING	PacketShader I/O	70% lower CPU utilization
Detection engine	GPU-support for Aho-Corasick	GPU-support for Aho-Corasick & PCRE	65% faster detection rate
Architecture	Process-based	Thread-based	1/6 GPU memory usage

* G. Vasiliadis, M.Polychronakis, and S. Ioannidis, "MIDeA: a multi-parallel intrusion detection architecture", ACM CCS 2011

Kargus vs. MIDeA

UPDATE	MIDEA	KARGUS	OUTCOME
Packet acquisition	PF_RING	PacketShader I/O	70% lower CPU utilization
Detection engine	GPU-support for Aho-Corasick	GPU-support for Aho-Corasick & PCRE	65% faster detection rate
Architecture	Process-based	Thread-based	1/6 GPU memory usage
Batch processing	Batching only for detection engine (GPU)	Batching from packet acquisition to output	1.9x higher throughput

* G. Vasiliadis, M.Polychronakis, and S. Ioannidis, “MIDeA: a multi-parallel intrusion detection architecture”, ACM CCS 2011

Kargus vs. MIDeA

UPDATE	MIDEA	KARGUS	OUTCOME
Packet acquisition	PF_RING	PacketShader I/O	70% lower CPU utilization
Detection engine	GPU-support for Aho-Corasick	GPU-support for Aho-Corasick & PCRE	65% faster detection rate
Architecture	Process-based	Thread-based	1/6 GPU memory usage
Batch processing	Batching only for detection engine (GPU)	Batching from packet acquisition to output	1.9x higher throughput
Power-efficient	Always GPU (does not offload only when packet size is too small)	Opportunistic offloading to GPUs (Ingress traffic rate)	15% power saving

* G. Vasiliadis, M.Polychronakis, and S. Ioannidis, “MIDeA: a multi-parallel intrusion detection architecture”, ACM CCS 2011

Receive-Side Scaling (RSS)

- RSS uses Toeplitz hash function (with a random secret key)

Algorithm: RSS Hash Computation

```
function ComputeRSSHash(Input[], RSK)
        ret = 0;
        for each bit b in Input[] do
                if b == 1 then
                        ret ^= (left-most 32 bits of RSK);
                endif
                shift RSK left 1 bit position;
        end for
end function
```

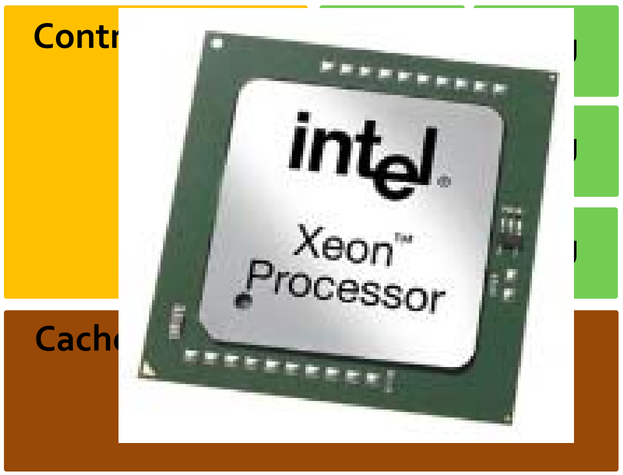
Symmetric Receive-Side Scaling

- Update RSK (Shinae *et al.*)

0x6d5a	0x56da	0x255b	0x0ec2
0x4167	0x253d	0x43a3	0x8fb0
0xd0ca	0x2bcb	0xae7b	0x30b4
0x77cb	0x2d3a	0x8030	0xf20c
0x6a42	0xb73b	0xbeac	0x01fa

0x6d5a	0x6d5a	0x6d5a	0x6d5a
0x6d5a	0x6d5a	0x6d5a	0x6d5a
0x6d5a	0x6d5a	0x6d5a	0x6d5a
0x6d5a	0x6d5a	0x6d5a	0x6d5a
0x6d5a	0x6d5a	0x6d5a	0x6d5a

Why use a GPU?



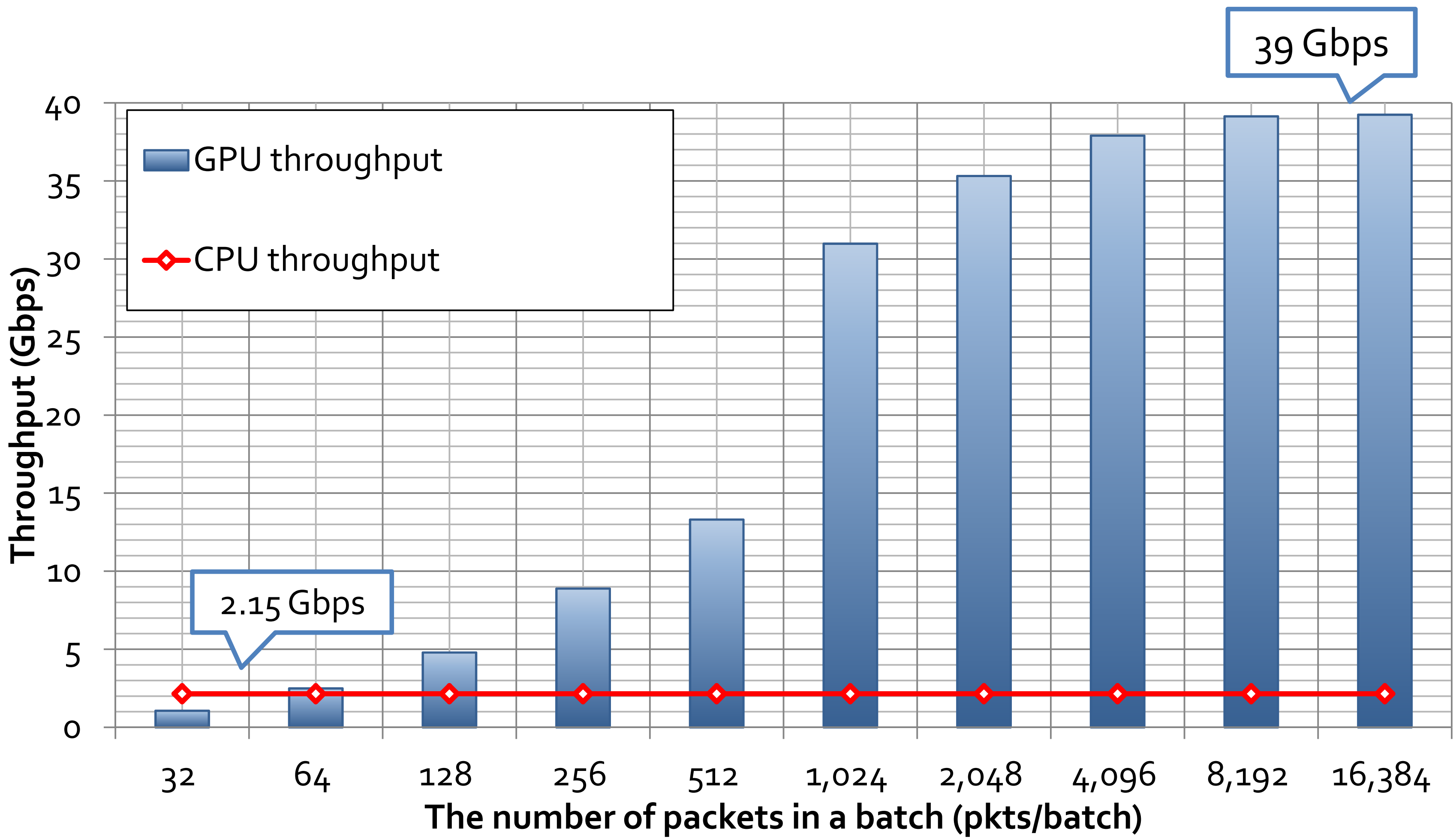
VS

Xeon X5680:
6 cores

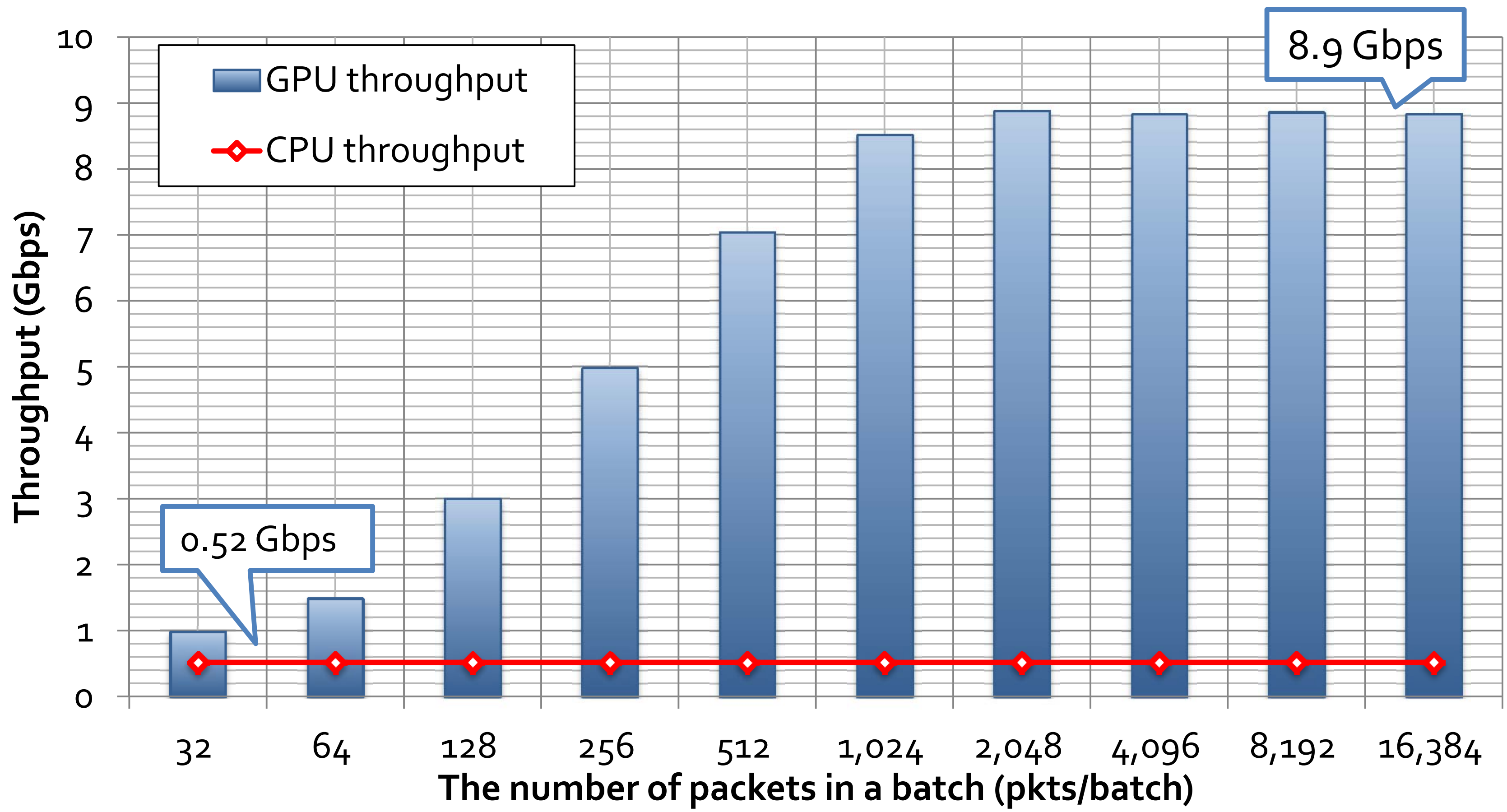
GTX 580:
512 cores

**Slide adapted from NVIDIA CUDA C A Programming Guide Version 4.2 (Figure 1-2)*

GPU Microbenchmarks – Aho-Corasick

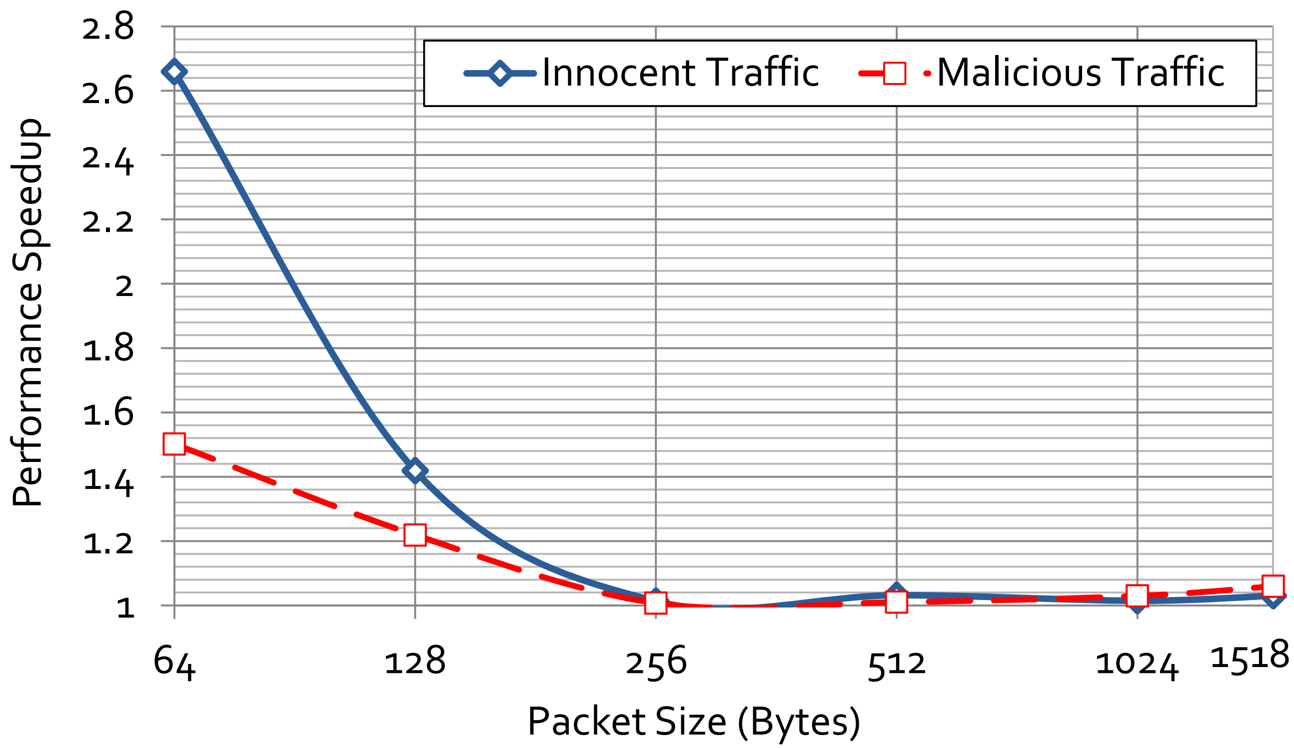


GPU Microbenchmarks – PCRE



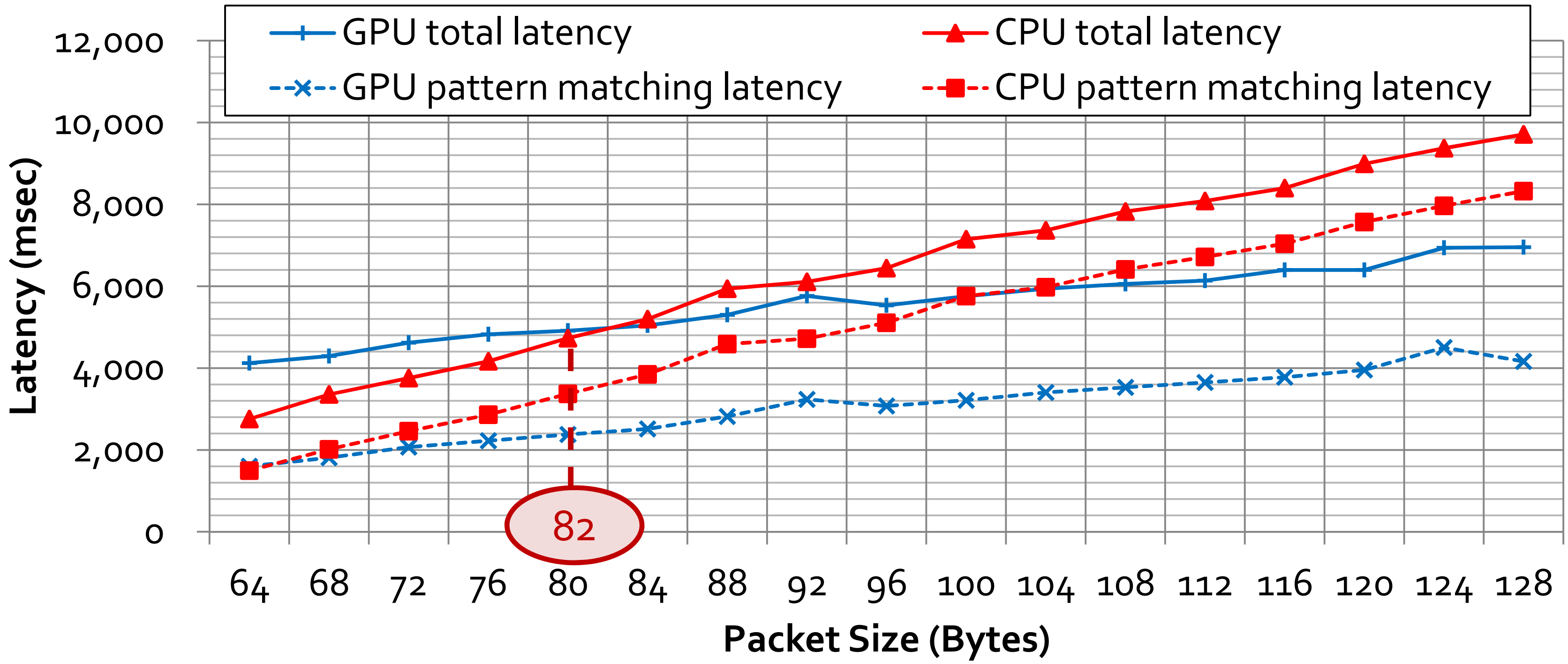
Effects of NUMA-aware Data Placement

- Use of global variables minimal
  - Avoids compulsory cache misses
  - Eliminates cross-NUMA cache bouncing effects



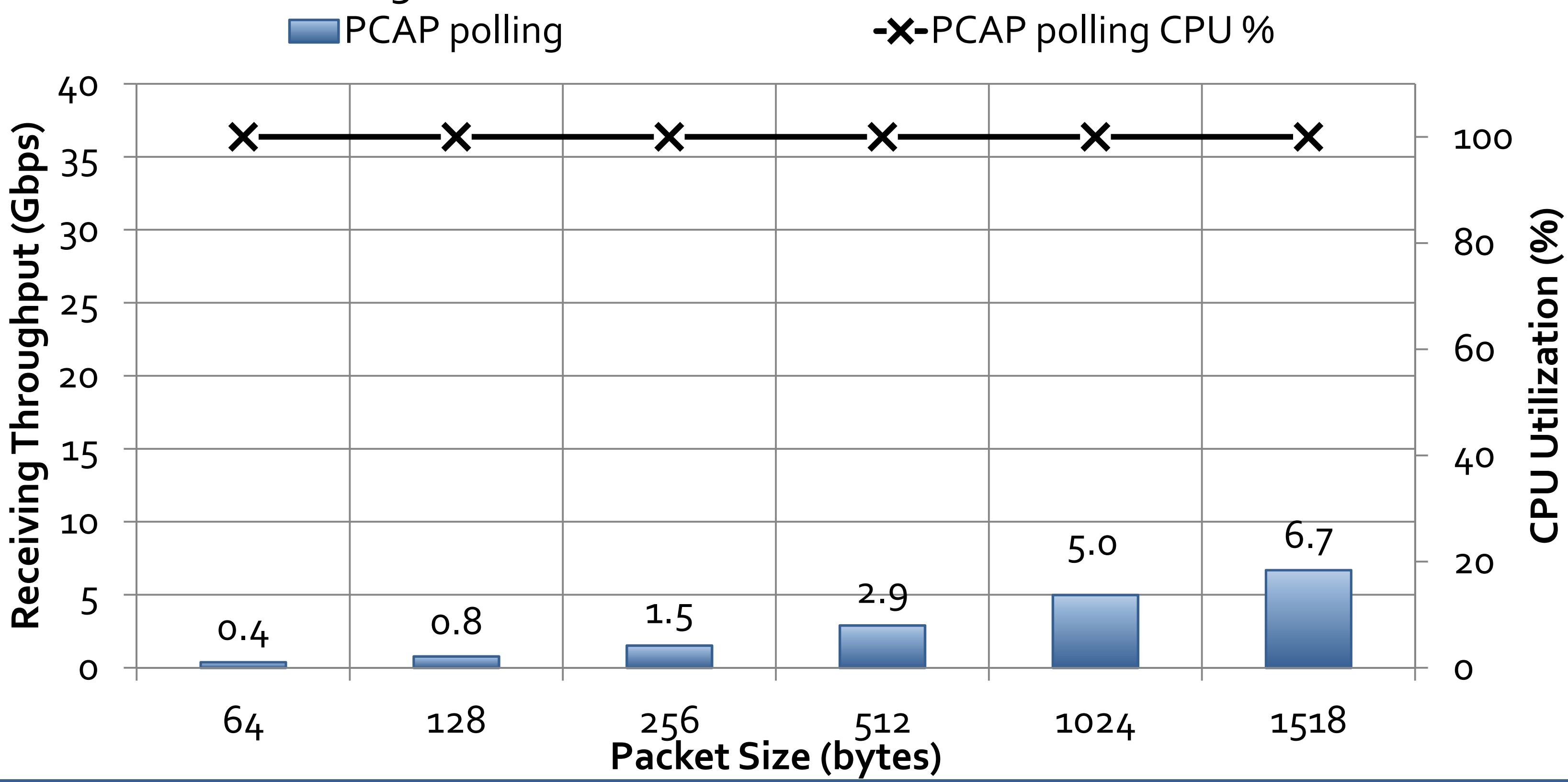
CPU-only analysis for small-sized packets

- Offloading <u>small-sized</u> packets to the GPU is expensive
  - Contention across page-locked DMA accessible memory with GPU
  - GPU operational cost of packet metadata increases



Challenge 1: Packet Acquisition

- Default packet module: Packet CAPture (PCAP) library
  - Unsuitable for multi-core environment
  - Low Performing



Solution: PacketShader* I/O