Towards a GPU SDN Controller

Eduard Gibert Renart
Eddy Zheng Zhang
Badri Nath
Overview of the presentation

• Introduction
• Gpu Architecture
• Our approach to a GPU SDN Controller handling warp divergence
• Results
• Future work
Introduction

• My team:
  • Myself, Eduard Gibert Renart, 2nd year PhD student
  • Dr. Zheng Zhang, GPU Expert
  • Dr. Badri Nath, Networking Expert
GPU

ARCHITECTURE
GPU Architecture

- SM - Streaming multiprocessors with multiple processing cores
  - Perform the actual computations
  - Each SM contains 32 processing cores
  - Up to 16 SMs on a card for a maximum of 512 compute cores
GPU Execution Model

• A grid is composed of blocks which are completely independent
• A block is composed of threads which can communicate within their own block
• 32 threads from a warp
Previous Work

- PacketShader a GPU Software Router [SIGCOMM ’10].

- SSLShader: Cheap SSL Acceleration with Commodity Processors [NSDI ’11].

- Multi-Layer Packet Classification with Graphics Processing Units [CoNEXT ’14].
SDN Packet Heterogeneity

Switch

Controller

PacketIn

32 bits

header

buffer_id

total_len

in_port

reason

pad

data

network byte order

Controller

Switch

EchoReq

EchoRes

EchoReq

EchoRes

header

data (arbitrary length)

network byte order
Warp Divergence

```cpp
switch (threadIdx.x % N) {
    case 0: ... 
    case 1:
        ...
}
```
Our approach to a GPU SDN

CONTROLLER HANDLING

WARP DIVERGENCE
Basic Idea

17,000,000 p/s !!!
GPU speedup 5.4x
Step 1 - Packet Classification

```
recv(socket, reply, 2000, 0);
```
Step 2 - Host to Device

cudaMemcpy(devArray, hostArray, bytes, cudaMemcpyHostToDevice);
Step 3 - Kernel Execution

kernel<<<gridDimensions,numberOfThreads>>>(dataOut, dataIn);
Step 4 - Device to Host

cudaMemcpy(hostArray, devArray, bytes, cudaMemcpyDeviceToHost);
It scales vertically!!

We only need to add more GPUs.
RESULTS
Results (Kernel only)

CPU
GTX 680
Tesla K40C

15,360,000 p/s
16,384,000 p/s
17,408,000 p/s
Packet Size Heterogeneity

- 100% 82B
- 50% - 50%
- 60% - 40%
- 70% - 30%
- 80% - 20%
- 90% - 10%
- 100% 62B
Towards a GPU SDN Controller

Results (Full Process)

- Naïve approach: 13,312,000 p/s
- Compression approach: 14,336,000 p/s, 15,360,000 p/s
Future

WORK
Hybrid Controller

Traditional SDN Controller only CPU.
Only activate the GPU when the RX exceeds 40% of the capacity.

The GPU solves the incast and latency problems.
Any

QUESTIONS?