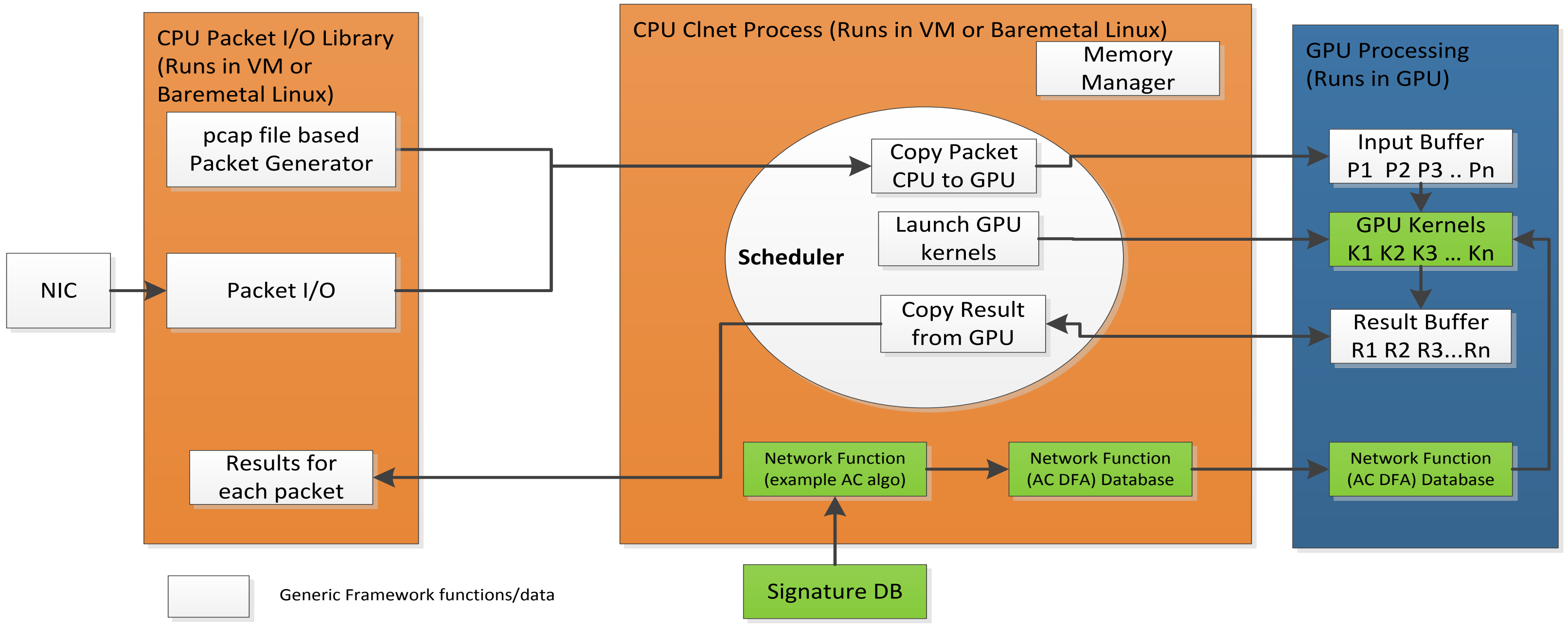
NFV Compute Acceleration APIs and Evaluation

draft-perumal-nfvrg-nfv-compute-acceleration-00.txt

Bose Perumal
Wenjing Chu
R. Krishnan
S. Hemalatha
Dell
Peter Wills
BT

NFV Compute Acceleration– Software Architecture



CPU Packet I/O Library (Runs in VM or Baremetal Linux)
pcap file based Packet Generator
NIC
Packet I/O
Results for each packet
CPU Clnet Process (Runs in VM or Baremetal Linux)
Memory Manager
Scheduler
Copy Packet CPU to GPU
Launch GPU kernels
Copy Result from GPU
Network Function (example AC algo)
Network Function (AC DFA) Database
Signature DB
GPU Processing (Runs in GPU)
Input Buffer P1 P2 P3 .. Pn
GPU Kernels K1 K2 K3 ... Kn
Result Buffer R1 R2 R3...Rn
Network Function (AC DFA) Database

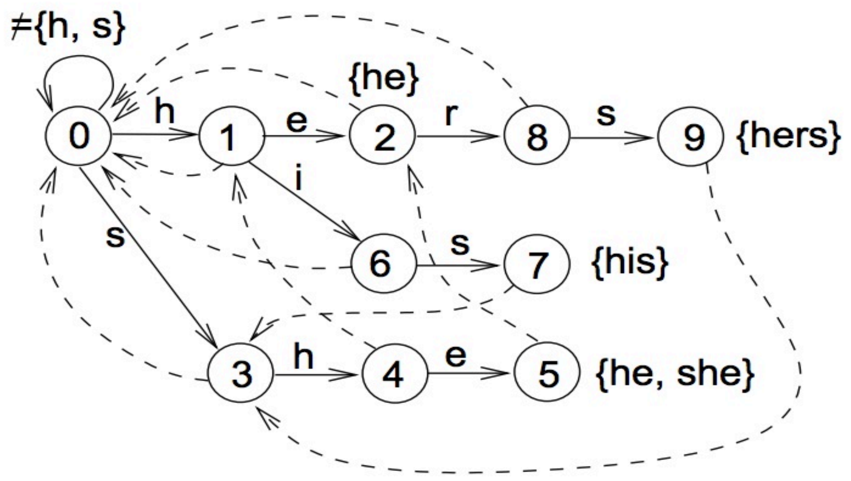
⬜	Generic Framework functions/data
🟩	Service specific functions/data
🟧	CPU Process
🟦	GPU Process

Compute Acceleration for NFV

- **Motivation**
  - Network functions are being virtualized.
  - Networks packet based architecture provides scope for parallel processing.
  - Parallel processing can be done in GPUs, Coprocessors like Intel Xeon Phi and multicore CPUs.
  - Parallel processing happens (implicitly in HW) in implementations such as Intel QuickAssist etc.
- **Generic Software Architecture and APIs**
  - Generic software architecture provides the framework components
  - APIs provides ease of adding network functions, traffic streams and directing traffic flow
- **Evaluation**
  - Multi string matching on all incoming packets using Aho-Corasick algorithm
  - Implementation in OpenCL to avoid hardware dependency.

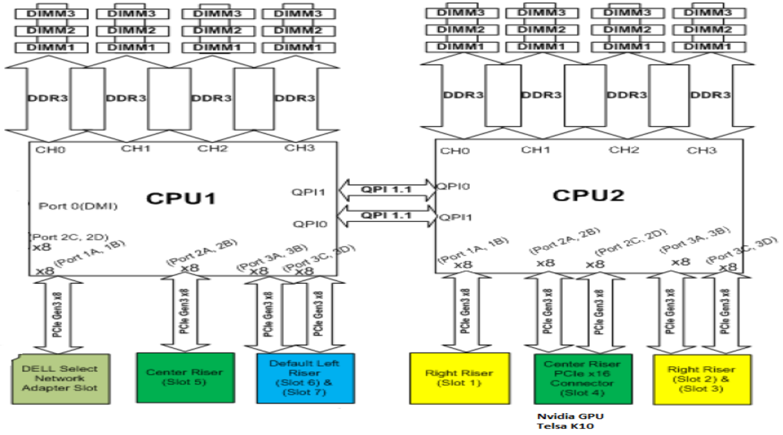
Aho Corasick – Multistring matching Algorithm

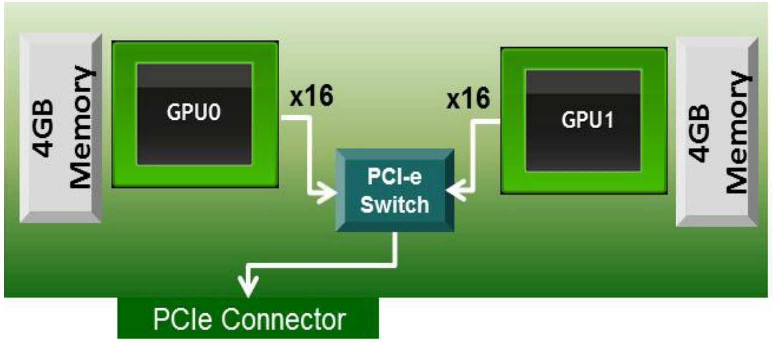
- **Aho Corasick Algorithm used for multistring matching**
  - Initial state machine is built with the keyword strings
  - Input string is matched using the state machine
  - Number of keyword strings do not have big impact on processing time.



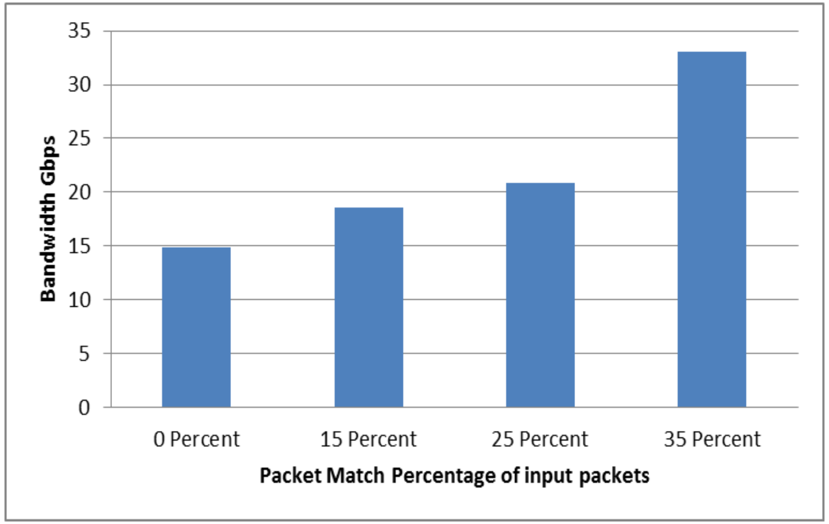
Test Setup –Dell R720 with Nvidia Tesla K10

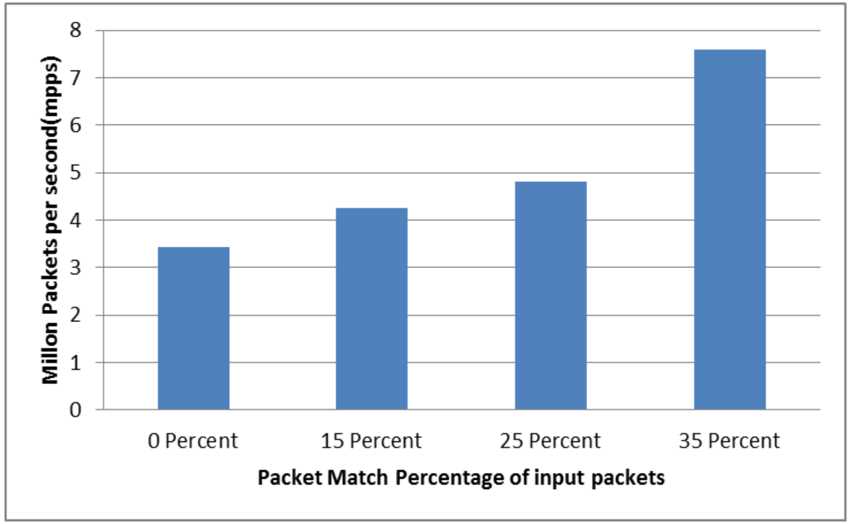
- **Server**
  - Intel Xeon E5-2665 @2.40GHz
  - 2 processors with 16 cores each
  - 2.5MB Cache per core.
  - RAM 96 GB
  - 1 PCIe x16 full length, full height slot which supports PCIe3.0.
- **GPU Card**
  - Nvidia GPU Tesla K10 @745 MHz
  - 2 GPU processors with 1536 cores each
  - 8GB memory(4GB per processor) (256-bit GDDR5)
  - PCI 3.0 x16 interface
- **Development Environment**
  - Linux distribution - Fedora 20
  - Compiler - OpenCL with Nvidia 6.0 libraries





Complete Packet – Multi String Search

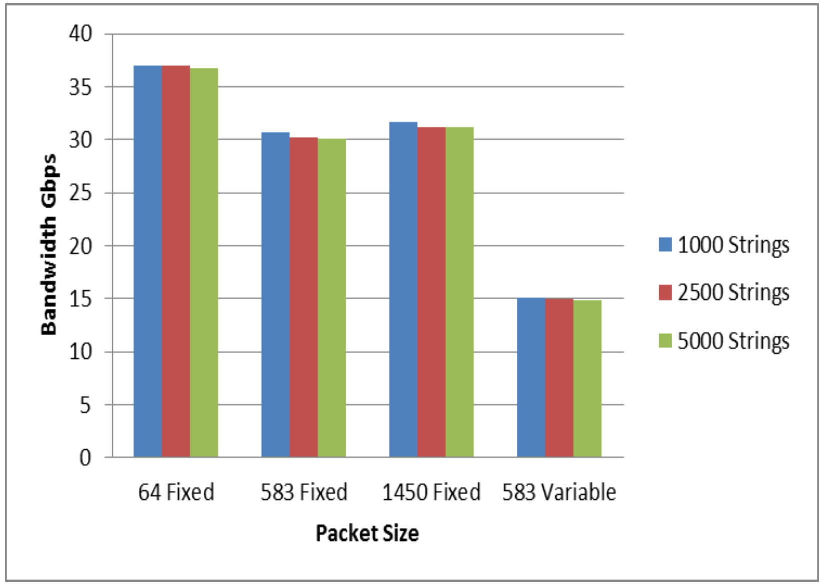


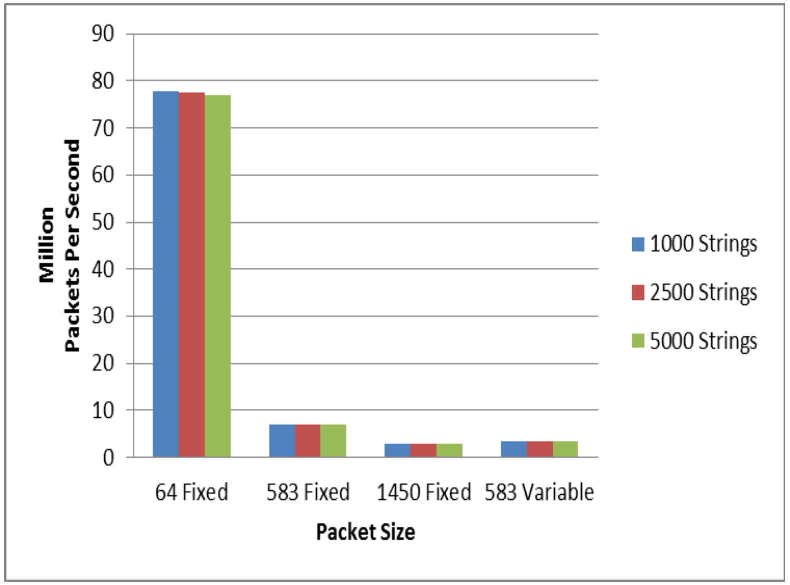


Variable packet sizes – Average packet size 583 bytes

Signature data base - Top 5000 website names

Complete Packet – Worst case
0 string match





Different Signature data bases - 1000 / 2500 / 5000 website names

Different Fixed packet sizes - 64 / 583 / 1450

Variable packet size with averages packet size - 583

Results – GPU multi string matching 35% packets match

- **Traffic generation and string matching**
  - Average packet size 583
  - 16384 packets batched for processing in GPU
  - Each packet checked for 5000 strings(Aho Corasick state machine)
  - 35 % of packets matched
- **Time for single batch processing**
  - Single copy time from CPU to GPU 0.923 milliseconds
  - Execution time 4.38 milliseconds
  - Result buffer copy from GPU to CPU 0.168 milliseconds
- **Operations in one second**
  - 464 iterations executed in one second
  - 7.6 million packets processed in one second
  - 33.02 Gbps processed in one second

Results – GPU multi string matching 0 packets match

- **Traffic generation and string matching**
  - Average packet size 583
  - 16384 packets batched for processing in GPU
  - Each packet checked for 5000 strings(Aho Corasick state machine)
  - 0 packet matched
- **Time for single batch processing**
  - Single copy time from CPU to GPU 0.903 milliseconds
  - Execution time 9.784 milliseconds
  - Result buffer copy from GPU to CPU 0.161 milliseconds
- **Operations in one second**
  - 209 iterations executed in one second
  - 3.42 million packets processed in one second
  - 14.9 Gbps processed in one second

Portability

- **Code written OpenCL is portable to different platforms only with makefile changes.**
- **Implementation tested in different platforms**
  - GPU based acceleration with baremetal Linux Server
  - GPU based acceleration from Virtual Machine on Vmware Hypervisor
  - Intel Xeon Phi based acceleration from baremetal Linux Server
  - Intel Multicore CPU based acceleration on baremetal Linux Server

NFV Compute Acceleration API

- Having a common API for NFV Compute Acceleration (NCA) can abstract the hardware details and enable NFV applications to use compute acceleration.
- API will be a C library, user can compile it along with their code.

No	API	Description
1	nca_add_network_function	Add network function
2	nca_traffic_stream_init	Add traffic stream and attach network functions
3	nca_add_packets	Add packets to buffer
4	nca_buffer_ready	Process packets
5	notify_callback	Event notification
6	nca_read_results	Read results

Future Work

- Refining the APIs and adding more algorithms for network functions such as encryption/decryption and compression/decompression.
- Integration with I/O acceleration, like Intel DPDK, ODP etc.
- Verifying in additional platforms
- The final goal is to have a common API for SW or HW based acceleration schemes in a OpenDaylight/OpenStack/OPNFV framework.

Thank You

Backup Slides

Implemented Optimizations

- **Variable size Packet Packing**
  - Multiple copies between CPU and GPU are costly. Variable size packets are packed into a single buffer.
  - Initial portion of the buffer holds all the packet starting offsets, packet contents follow that.
- **Pinned Memory**
  - Using pinned memory reduces copy time 3x
  - Used pinned memory for CPU to GPU copy and GPU to CPU copy
- **Scheduler for Pipe lining**
  - Written a scheduler to do pipelining with multiple command queues
    - With 3 command queues we are able to hide 99% of copy time.
- **Reducing GPU global memory access**
  - Reduce the GPU global memory access
    - Copy to private memory if needed. Instead of byte by byte copy, copying 32 bytes using vload8 with float type.
  - Move common structures to constant memory
- **Organizing GPU cores**
  - If there are more memory accesses , overload number of kernels to hide the latency.