Center for Domain-Specific Computing

From JVM to FPGA: Bridging Abstraction Hierarchy via Optimized Deep Pipelining

Jason Cong, Peng Wei and Cody Hao Yu

University of California, Los Angeles

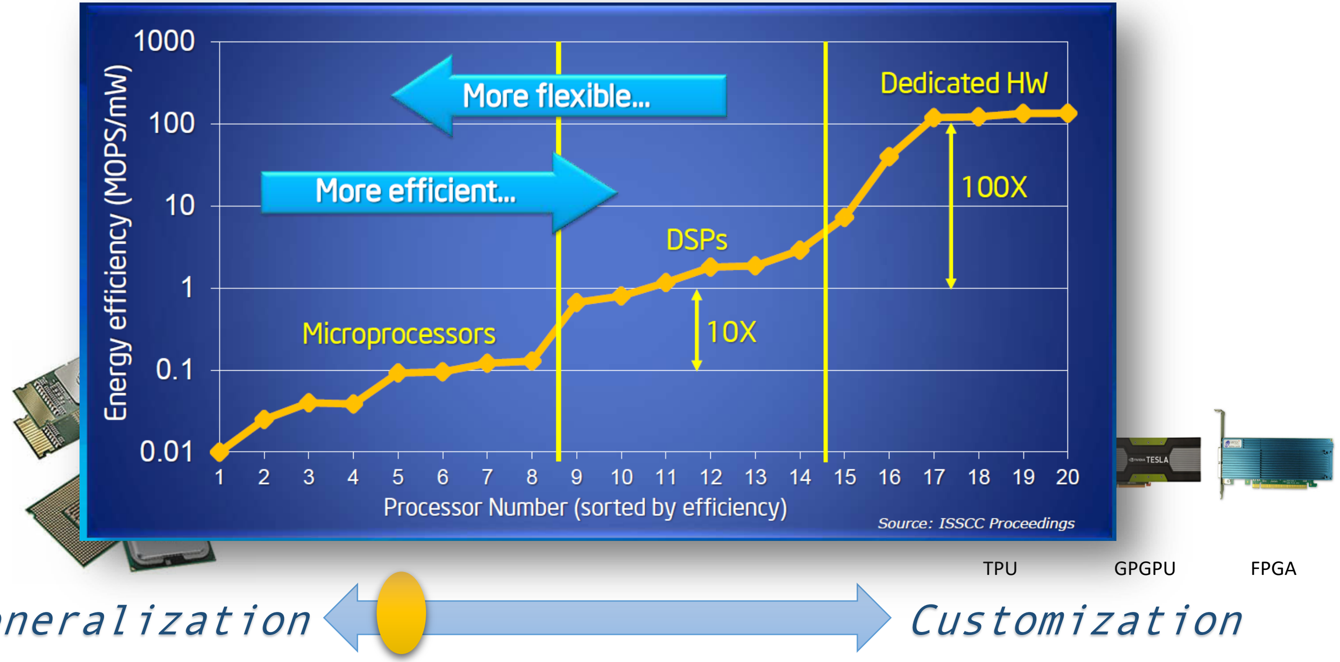
UCLA COMPUTER SCIENCE DEPARTMENT

Motivation:

Harnessing FPGAs in Datacenter

Harnessing FPGAs in Datacenters: Why?

- Heterogeneous architecture: an "agreement" from the hardware community



Sources: Google, NVIDIA, Intel, Bob Broderson, Berkeley Wireless group

Harnessing FPGAs in Datacenters: Why?

- Heterogeneous architecture: an “agreement” from the hardware community
- The FPGA-based cluster is a promising paradigm
  - Standalone FPGA accelerators demonstrate orders-of-magnitude performance/watt improvement

CNN

Dynamic Prog

Encryption

String Matching

2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines

Accelerating SpMV on FPGAs by Compressing Nonzero Values

A Scalable High-Bandwidth Architecture for Lossless Com

Enabling High Throughput and Virtualization for Traffic Classific

Accelerating Interconnect Analysis using High-Level HDLs and FPGA SpiNNaker as a Case Study

Fast Design Space Exploration using Vivado HLS: Non-Binary L

Zedwulf: Power-Performance Tr 32-node Zynq SoC clu

Design of A Distributed Compressor for Astronomy SSD

Scalable Key/Value Search in Datacenters

FFT

Analytics

Compression

GEMM/SPMV

Harnessing FPGAs in Datacenters: Why?

- Heterogeneous a   ne hardware community
- The FPGA-based
  - Standalone FPGA a   tude performance/watt improvement
  - FPGAs are reconfig
    - A relatively “gene
    - It is now in the cl

Microsoft Azure

amazon web services™ EC2

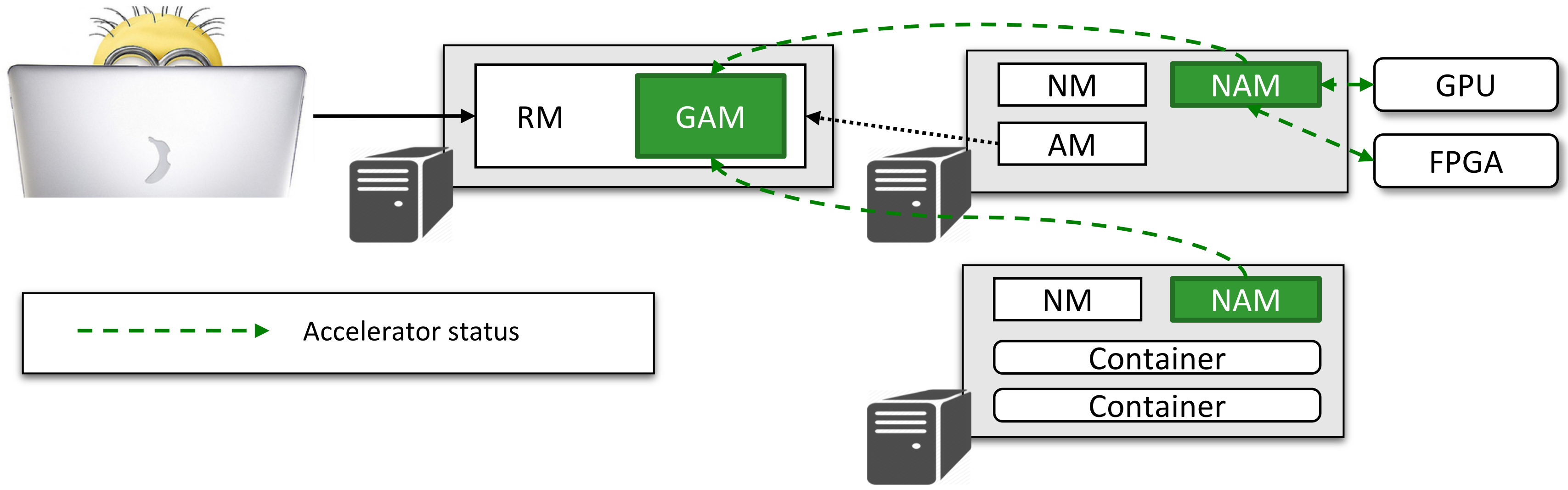
AliCloud

Baidu Cloud

Challenge:

system integration - from kernel speedup to system acceleration

Accelerator (FPGA)-as-a-Service

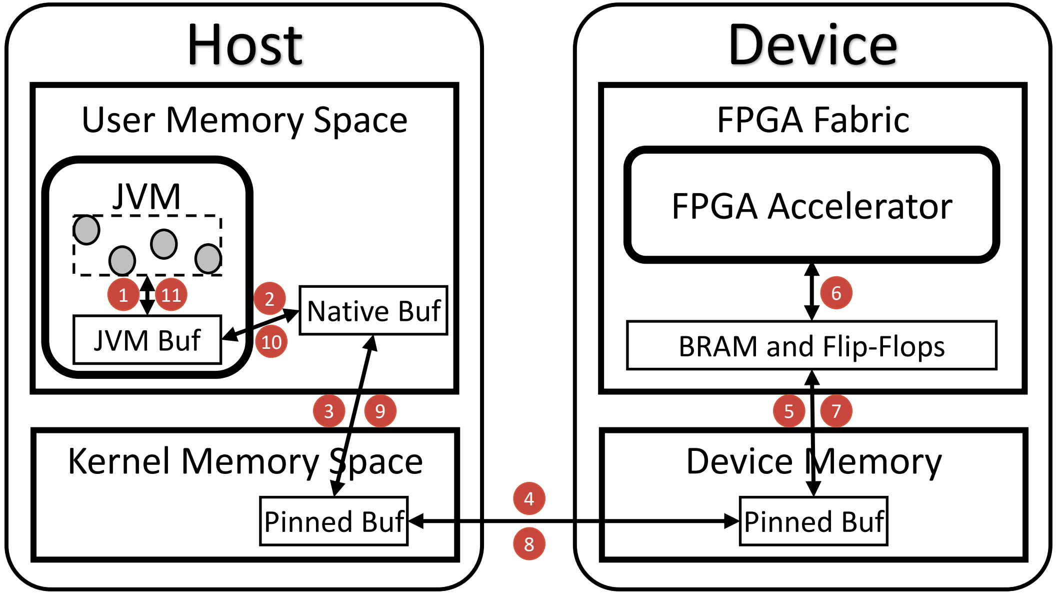


GAM Global Accelerator Manager: accelerator-centric scheduling

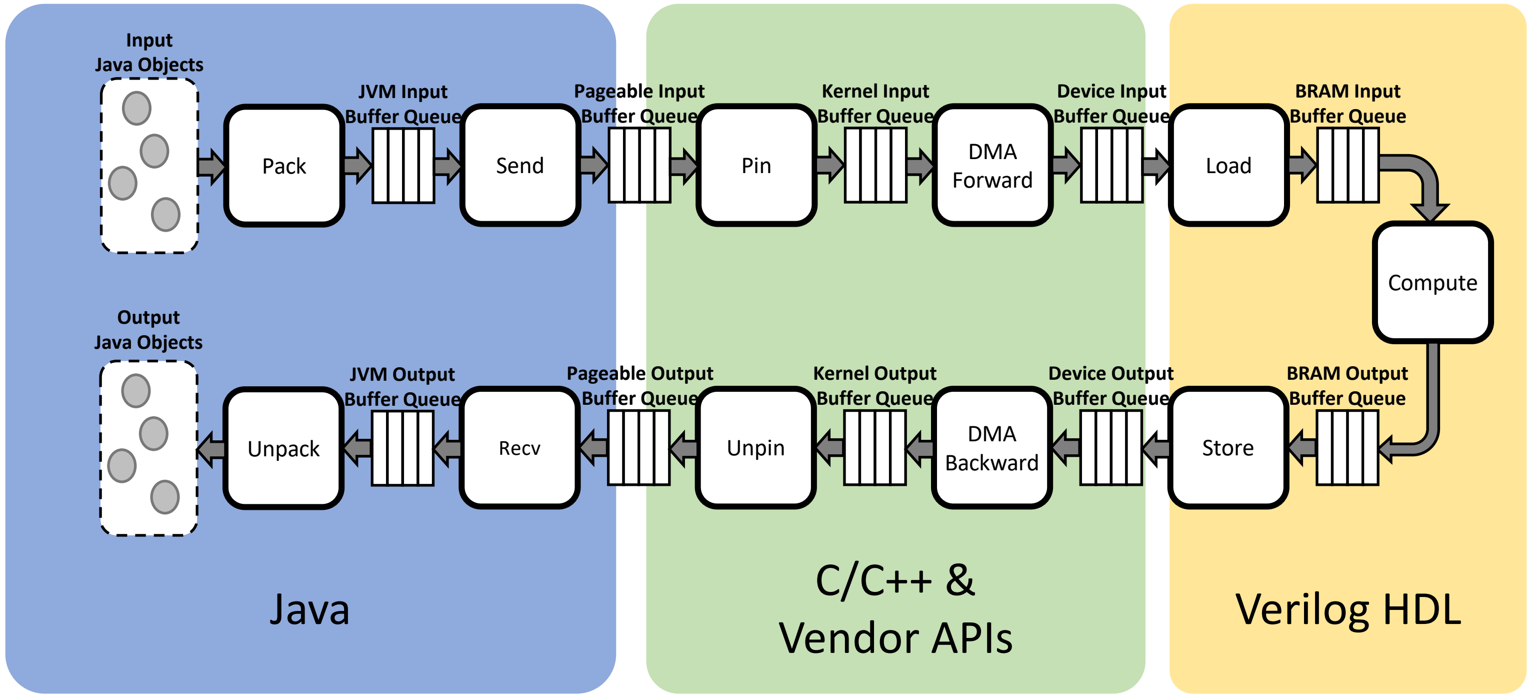
NAM Node Accelerator Manager:
local accelerator service management, JVM-to-ACC communication optimization

In terms of performance, there is much to say...

- Time breakdown of AES
  - Pack: app.-dep.; ~4GB/s
  - Send (via socket): ~3GB/s
  - Usr->Kernel: ~6GB/s
  - DMA: ~5GB/s
  - Load: ~6GB/s (shd w/ Store)
  - Compute: 12.8GB/s
    - >100x over CPU
  - ...
  - 1/(1/4+1/3+...) = 0.47 GB/s
  - ***27x performance loss!!!***

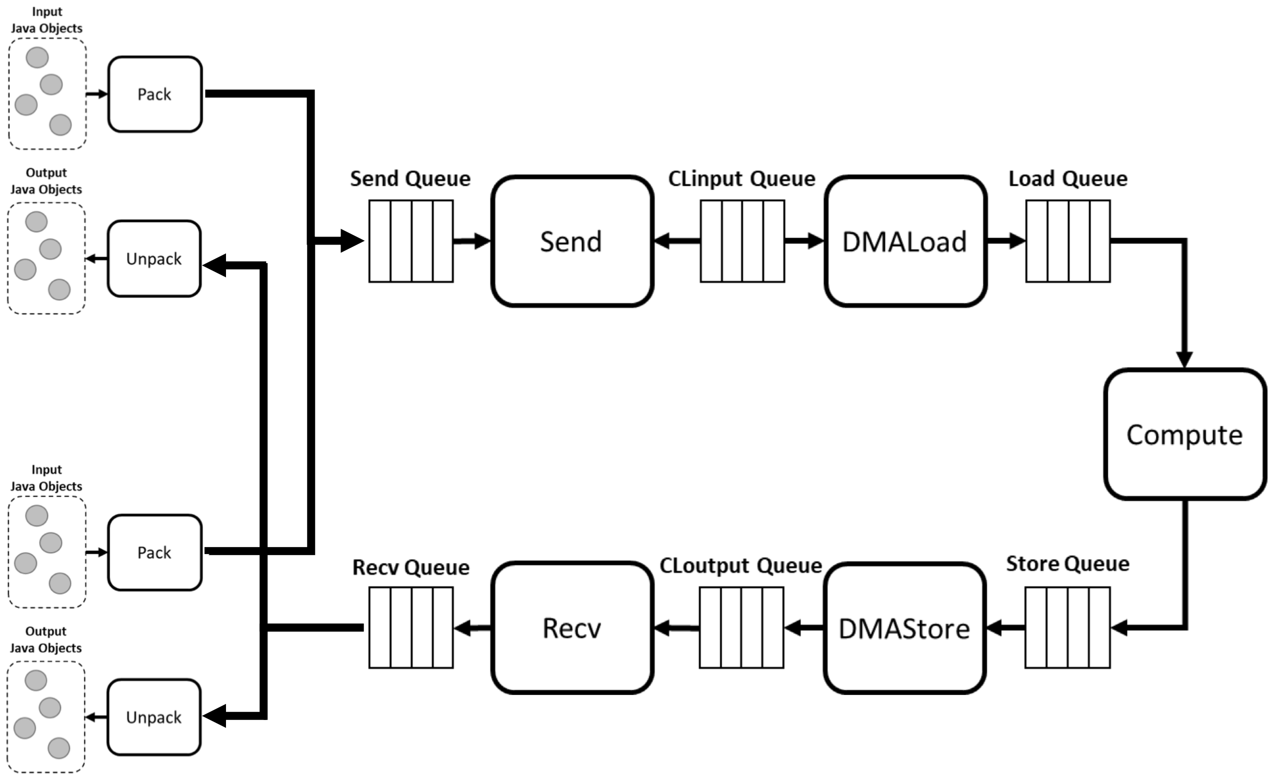


JVM-FPGA Communication Pipelining



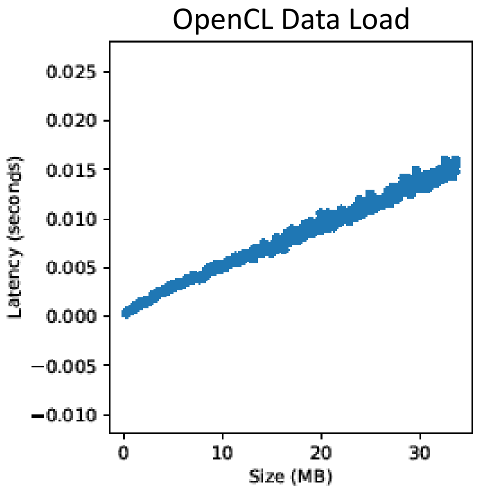
JVM-FPGA Communication Pipelining

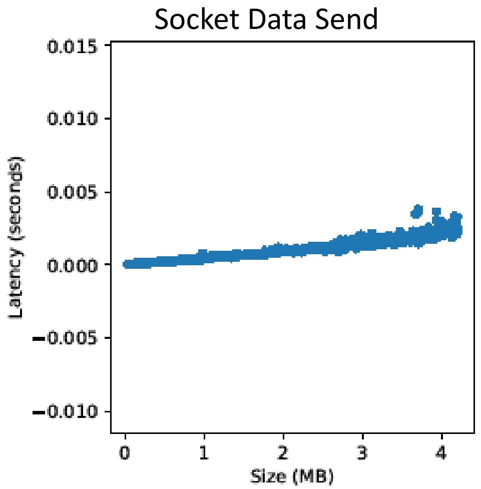
- Send + Pin => Send
  - Limitation of vendor APIs
- Load/Compute/Store => Compute
  - Overlapping comm. & comp.
- Programmer's responsibility
  - Pack and unpack
  - Implementing an iterator interface to supply input data
  - Header + payload



In terms of performance, there is still much to say...

- The pipeline efficiency is bounded by the slowest stage
- In general, **latency = time_setup (one-time) + payload_size * time_unit (linear)**
- Adjust the payload sizes of different pipeline stages to balance their throughputs
- … but how to?
- Linear with constraints => **linear programming**





Linear Programming Formulation

Problem Formulation:

maximize the pipeline throughput, i.e.,

$$T_K = Min(T_{pack},\ T_{send},\ \dots,\ T_{unpack})$$

$$T_{stage} = \frac{1}{L_{stage}} = \frac{1}{f_{stage}(S_{stage})}$$

Modeling of Compute Stage:

profile a set of payload sizes (power of two), and model the latency of the compute stage into a selection equation with a set of binary variables:

$$L_{compute} = \sum_i p_i \times L_{S_i},\ where \sum_i p_i = 1,\ p_i \in \{0,1\}$$

Modeling of Data Transfer Stages:

for each individual data transfer stage, impose the payload size constraint, and model the relation between the payload size and the latency via linear equations:

$$L_{stage} = L_{stage}^{setup} + S_{stage} \times L_{stage}^{unit}$$

$$S_{stage} <= S_{stage}^{max}$$

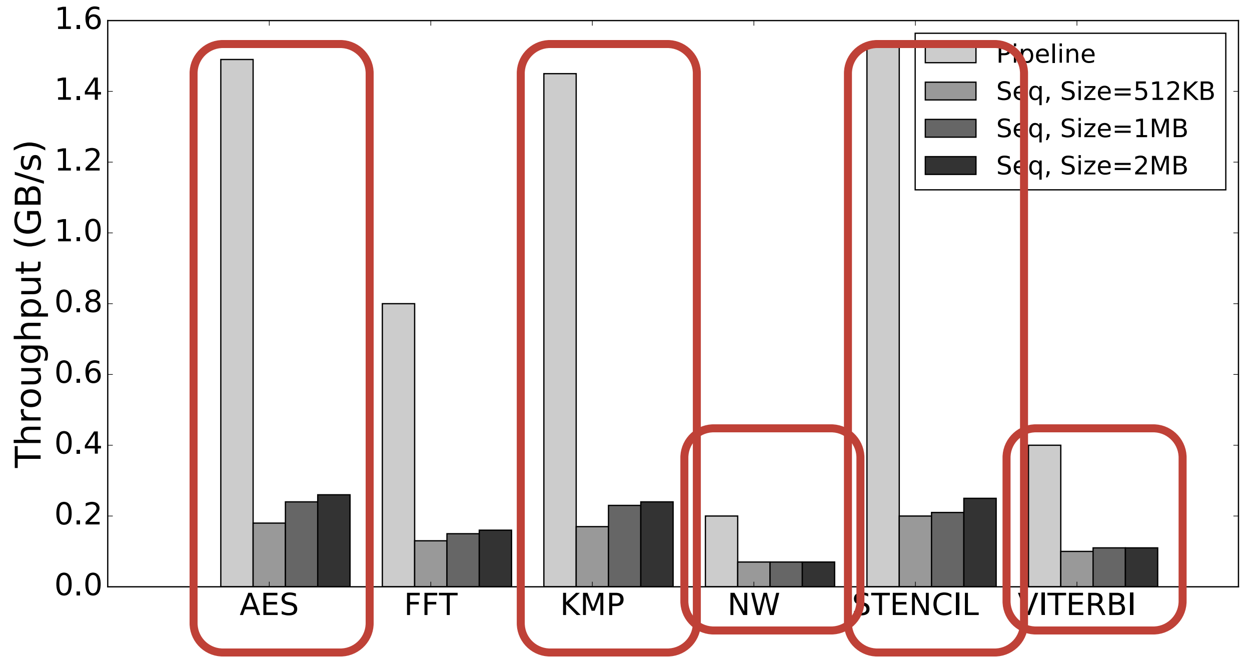
Modeling of Memory Constraints:

constrain the memory usage of the pipeline in both the CPU and the FPGA sides for separate-memory platforms, and in only the CPU side for shared-memory platfroms:

$$\sum S_{Q_{stage}} = \sum (S_{stage} \times D_{stage}) \leq S_{capacity}$$

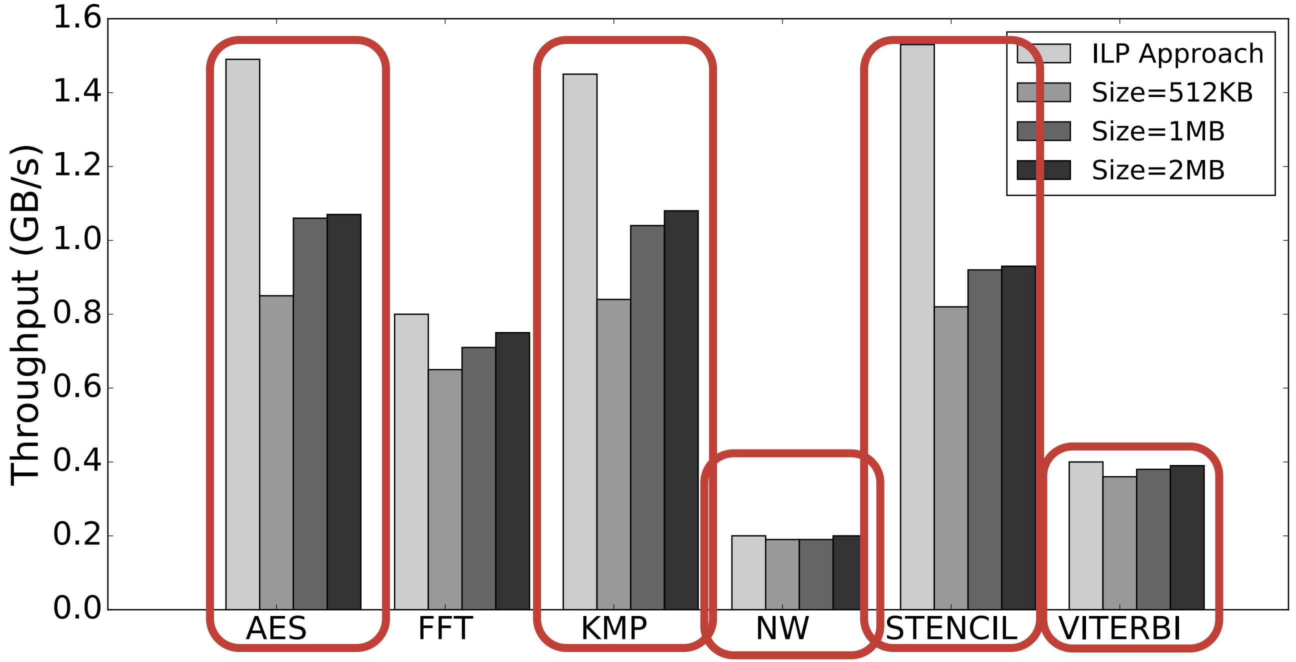
Experimental Results

- A set of computation kernels as benchmarks
- Each with a Java program as the host
- Currently single-threaded, and will showcase the real-application results in the near future



Experimental Results

- A set of computation kernels as benchmarks
- Each with a Java program as the host
- Currently single-threaded, and will showcase the real-application results in the near future



Lessons Learned and Future Work

- Single thread -> multiple threads -> Mainstream frameworks
  - Modeling in the multithreaded scenario
  - Integration with frameworks like Apache Hadoop and Spark
- Adapt to various platforms
  - Latest platforms support FPGA's direct access of user-space data, like IBM CAPI and Intel Xeon+FPGA
  - Amazon EC2 F1 instance brings virtualization into consideration
- JVM related improvement
  - Fast and safe allocation and management of native-space memory

THANKS FOR YOUR ATTENTION.

Discussion

Problem Formulation:
maximize the pipeline throughput, i.e.,

$$T_K = Min(T_{pack}, T_{send}, \ldots, T_{unpack})$$

$$T_{stage} = \frac{1}{L_{stage}} = \frac{1}{f_{stage}(S_{stage})}$$

Modeling of Compute Stage:
profile a set of payload sizes (power of two), and model the latency of the compute stage into a selection equation with a set of binary variables:

$$L_{compute} = \sum_i p_i \times L_{S_i},\ where\ \sum_i p_i = 1,\ p_i \in \{0, 1\}$$

Modeling of Data Transfer Stages:
for each individual data transfer stage, impose the payload size constraint, and model the relation between the payload size and the latency via linear equations:

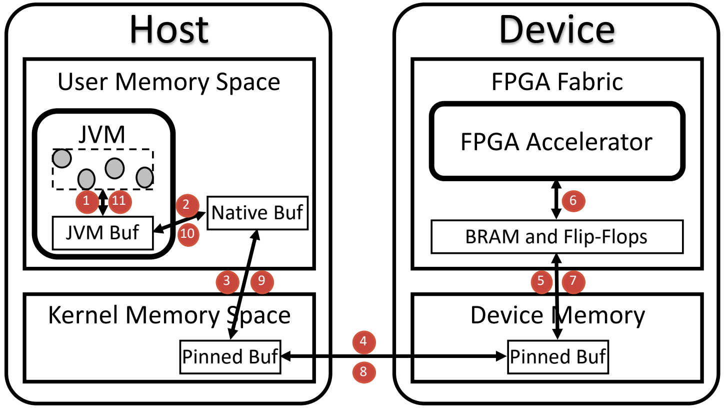
$$L_{stage} = L_{stage}^{setup} + S_{stage} \times L_{stage}^{unit}$$

$$S_{stage} <= S_{stage}^{max}$$

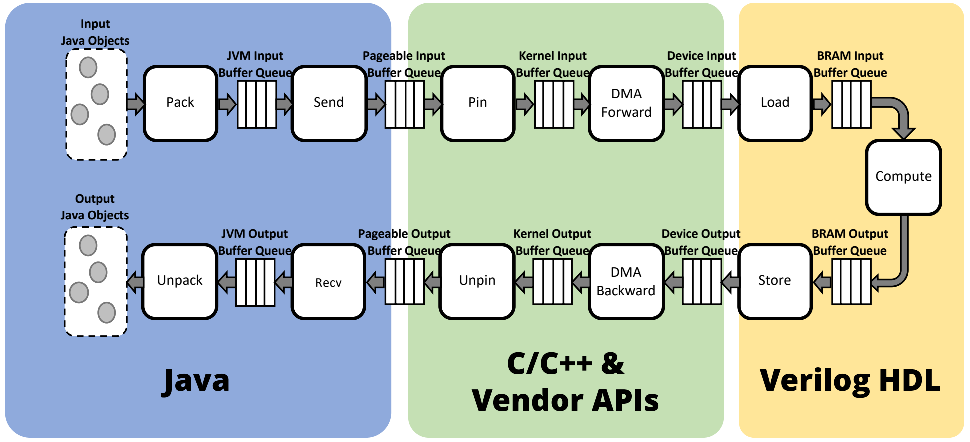
Modeling of Memory Constraints:
constrain the memory usage of the pipeline in both the CPU and the FPGA sides for separate-memory platforms, and in only the CPU side for shared-memory platfroms:

$$\sum S_{Q_{stage}} = \sum (S_{stage} \times D_{stage}) \leq S_{capacity}$$

Linear Programming



Host
User Memory Space
JVM
1
11
JVM Buf
2
Native Buf
10
3
9
Kernel Memory Space
Pinned Buf
4
8
Device
FPGA Fabric
FPGA Accelerator
6
BRAM and Flip-Flops
5
7
Device Memory
Pinned Buf



Input Java Objects
Pack
JVM Input Buffer Queue
Send
Pageable Input Buffer Queue
Pin
Kernel Input Buffer Queue
DMA Forward
Device Input Buffer Queue
Load
BRAM Input Buffer Queue
Compute
BRAM Output Buffer Queue
Store
Device Output Buffer Queue
DMA Backward
Kernel Output Buffer Queue
Unpin
Pageable Output Buffer Queue
Recv
JVM Output Buffer Queue
Unpack
Output Java Objects
Java
C/C++ & Vendor APIs
Verilog HDL