# ApproxHPVM: Accuracy-aware Optimizations for Heterogeneous SoCs

**Vikram Adve**

***With:***

**Maria Kotsifakou, Hashim Sharif, Adel Ejjeh, Prakalp Srivastava,**

**Yasmin Sarita, Nathan Zhou, Sarita Adve and Sasa Misailovic**

***University of Illinois at Urbana-Champaign***

*Supported by: NSF, SRC, DARPA, Amazon, Intel*

# Software Multiplexing: Full-system Code Deduplication

*[OOPSLA 2018]*
*[Submission, 2019]*

**Sean Bartell, Will Dietz and Vikram Adve**

***University of Illinois at Urbana-Champaign***

*Supported by: ONR, NSF*

# Sources of Code Duplication Across Programs

- **Duplicated libraries across applications**
- **Duplicated functions within / across applications**
- **Duplicated code fragments within / across applications**

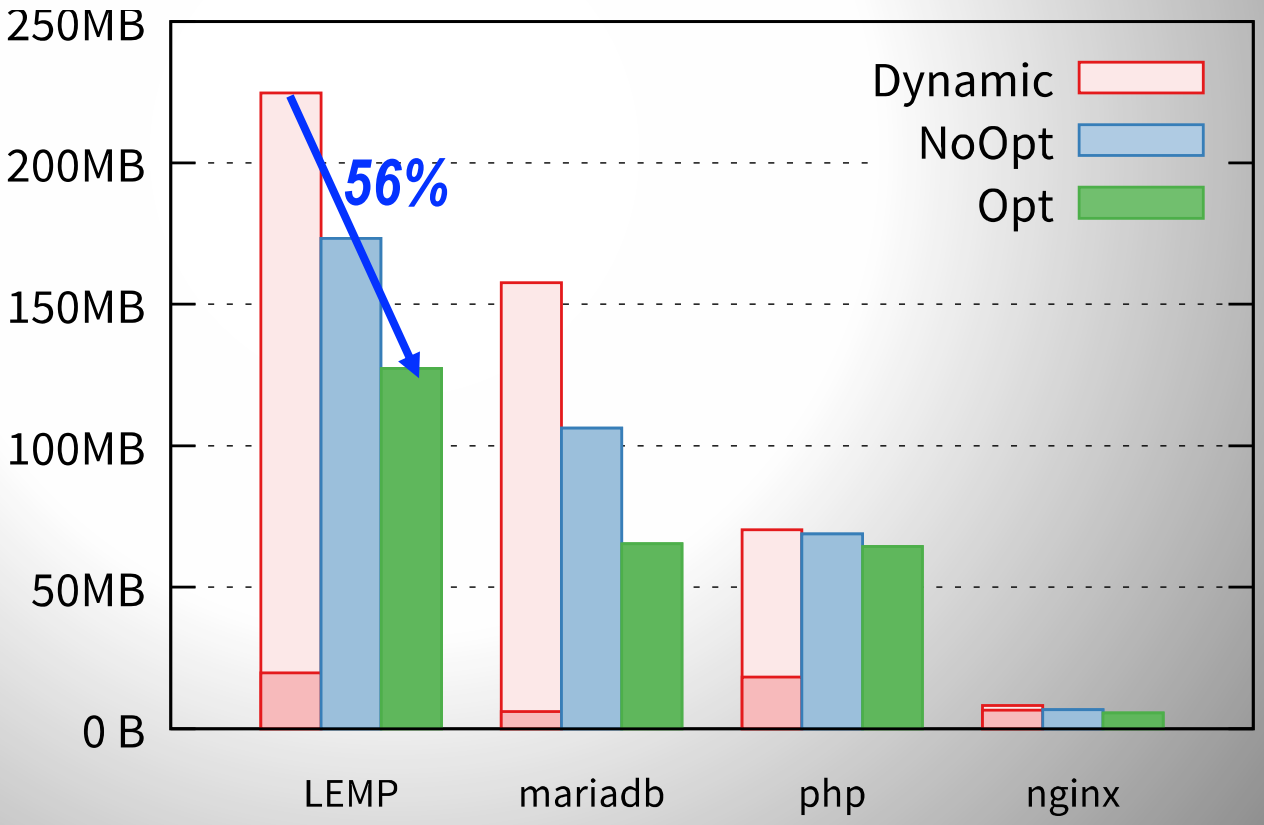
***Software multiplexing is a compiler strategy to address all three issues (current system addresses first two)***

# Example: LEMP Stack for Web Servers

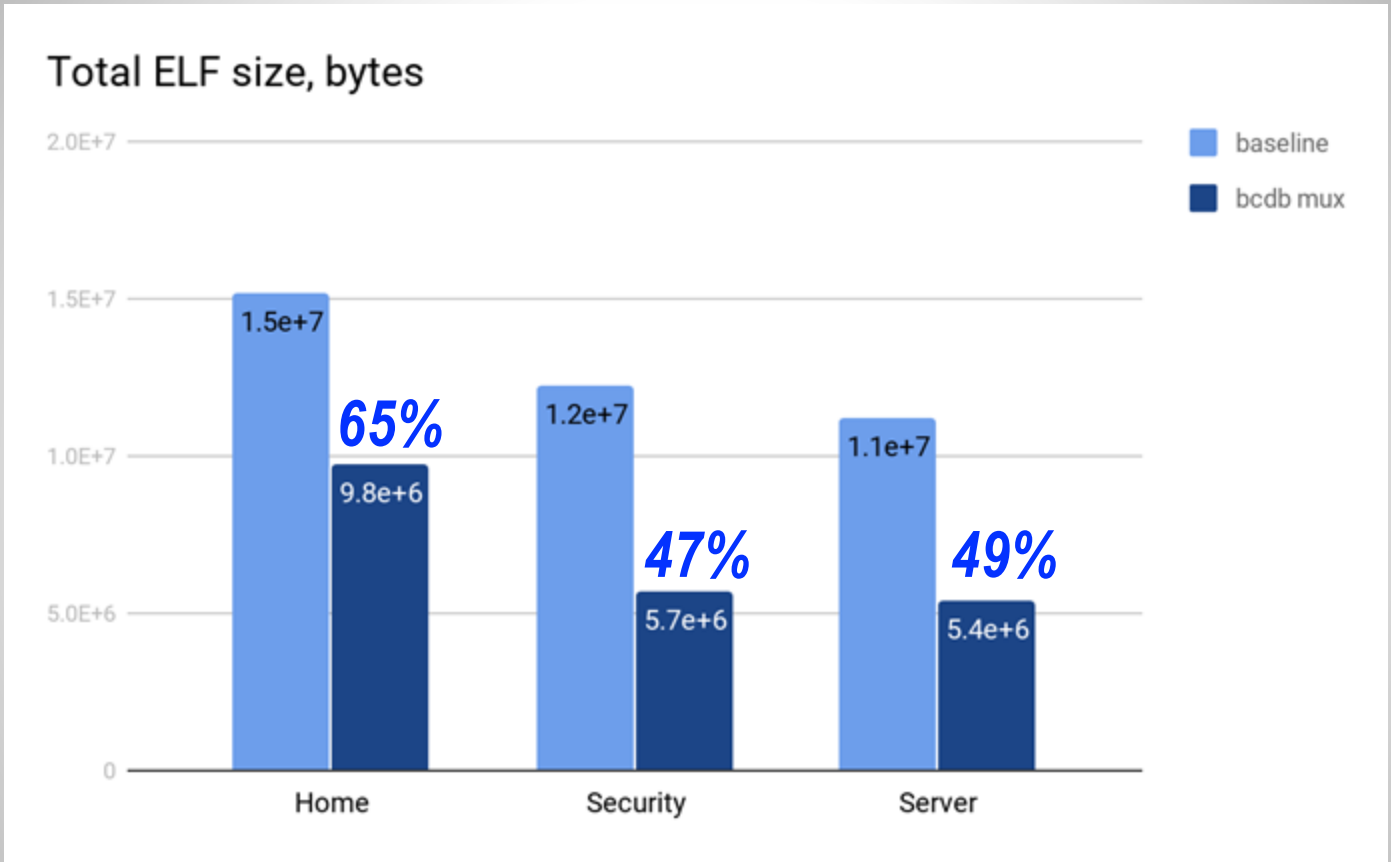
**LEMP = Linux + nginx + mariadb + PHP**

**37 programs combined in ONE!**

**Fully automatic and no change in functionality**



# Example 2: OpenWRT Router Configurations

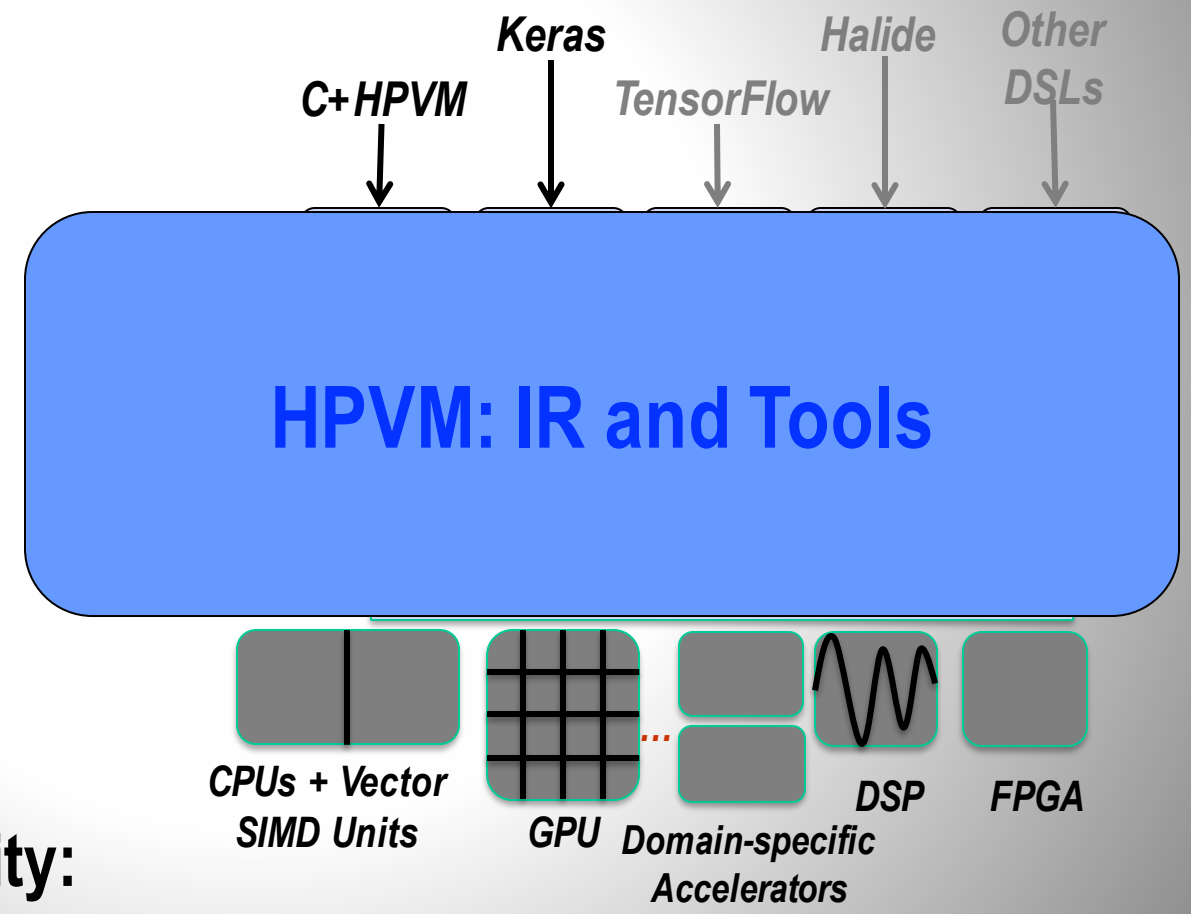



- **3 package sets for OpenWRT: 62 (Home), 103 (Server)**
- **Baseline: “clang -Oz” + LLVM outliner on each binary**

# Heterogeneous Parallel Virtual Machine

***Goal: Programmability for Heterogeneous Parallel Systems***

Mobile / embedded SoCs
Supercomputers
Cloud with accelerators




**Use HPVM for:**

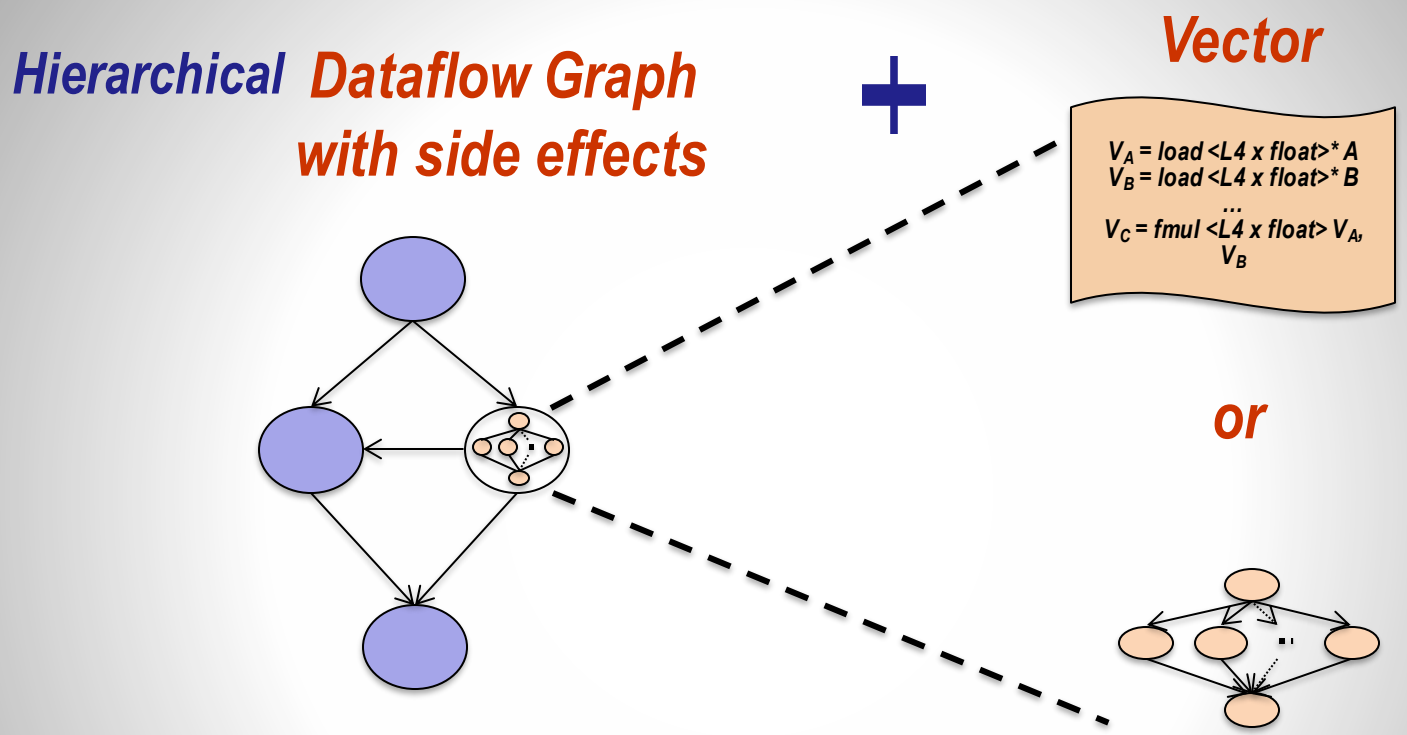
1. Portable *object* code
2. Retargetable parallel compiler IR and system
3. Run-time scheduling

**Key to Programmability:**

**Common abstractions for heterogeneous parallel hardware**

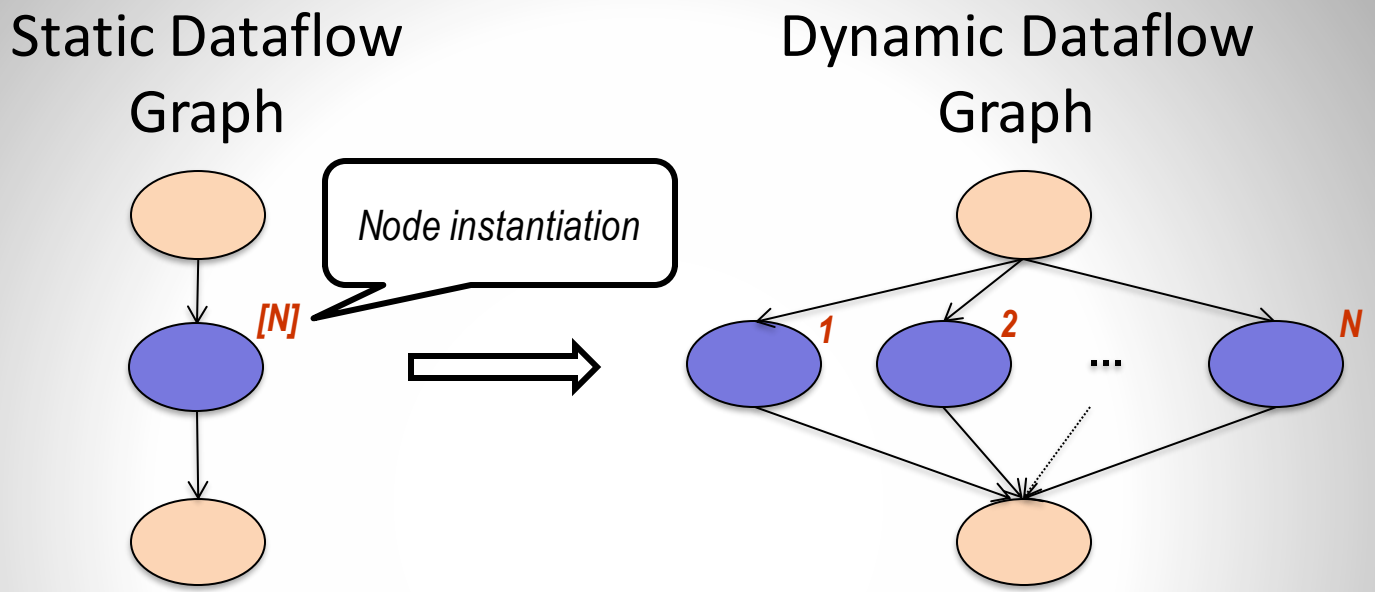
*Kotsifakou et al., PPOPP 2018*

# Abstraction of Parallel Computation




- Graph nodes – coarse-grain or fine-grain computational tasks
- Graph edges – explicit data transfer between nodes
- Loads and stores – implicit communication via shared memory
- Hierarchical – multiple levels of nested parallelism

# HPVM Abstractions




- ✓ **Graph Structure – coarse grain task parallelism, streams, pipelines**
- ✓ **Graph hierarchy – nested parallelism**
- ✓ **Node Instantiation – captures SPMD-style data parallelism**
- ✓ **Vector instructions in leaf nodes – fine grain vector parallelism**
- ✓ **Supports high-level optimizations**
- ✓ **Captures FPGAs, some semi-custom hardware**

**N different parallelism models ⇨ single unified parallelism model!**

# Simplifying Approximate Computing

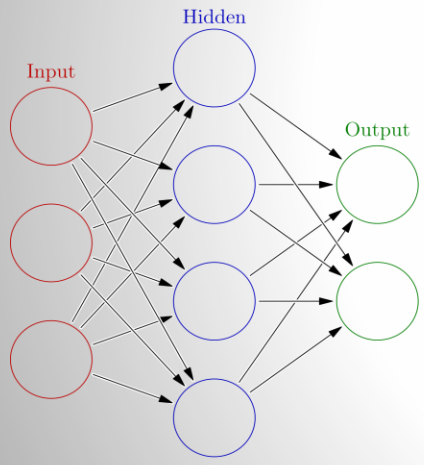
We know applications can be **2x-10x** more efficient by slightly relaxing accuracy / quality requirements.

Can we make these strategies **easier to use**?

*Sharif et al., OOPSLA 2019*

# Domains for Approximate Computing

An approximate answer is enough in some important domains




*Machine Learning*

*Data Sciences*

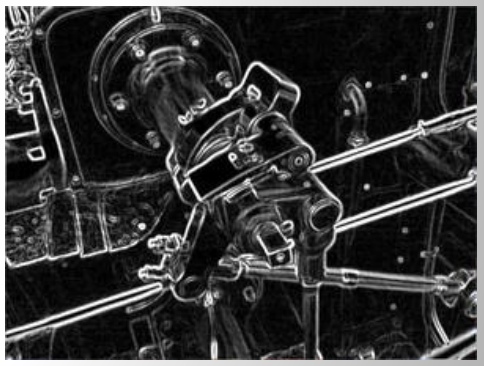


*Image Processing*

# Heterogeneity of Approximation Methods

- **Diverse approximate computing methods**

  **Software:** Loop perforation, barrier elision, reduction sampling, function substitution, relaxed synchronization, …

  **Hardware:** Numerical precision, caches, custom accelerators

- **Example 1: GPU**
  - ➢ Choices of **precision:** FP32, FP16, Int8, Int4, …

- **Example 2: PROMISE(*)**
  - ➢ Programmable deep in-memory analog ML accelerator
  - ➢ Choices of **bit-line swing voltage:** 7 different levels of accuracy

*(*) Kang et al., ISCA 2018*

# ApproxHPVM: Key Goals

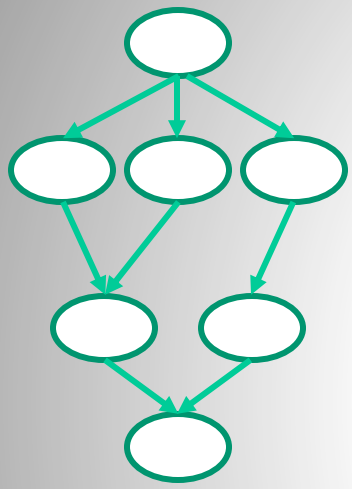
- **Applications should only specify *high-level goals***
  - Maximum acceptable loss in quality, e.g., inference error, PSNR
  - End-to-end metrics, not per function or pipeline stage or …
  - **System should select and optimize approximation techniques**

- **(Often) Want object-code portability**
  - Approximation choices are highly system-dependent
  - Can make orders-of-magnitude difference in performance, energy

# How to Map to Heterogeneous Approx. SoC?

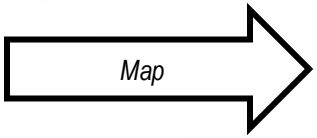
- **SoC with processing elements A, B, C and D**
- **Application with end-to-end quality spec (e.g., max error)**

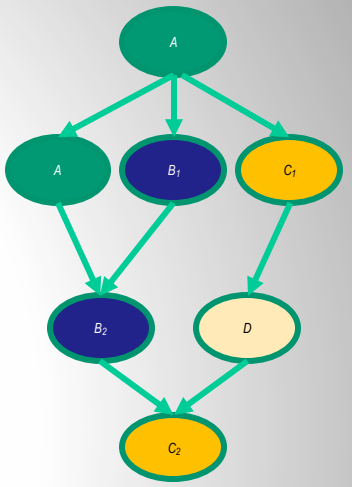
# Three Specific Challenges



+

***End-to-End Quality Specification***




- *Map end-to-end quality spec to individual task quality specs*
- *Cost of mapping*
- *Flexible code-gen to heterogeneous h/w*

# ApproxHPVM IR: Intrinsics & Quality Metrics

## ApproxHPVM IR Tensor Intrinsics:

| *Tensor Intrinsic* | *Description* |
|---|---|
| i8* **@hpvm.tensor.mul**(i8* lhs, i8* rhs) | Performs a matrix multiply operation on the input tensors. |
| i8* **@hpvm.tensor.conv**(i8* input, i8* filter, i32 stride, i32 padding) | Applies a convolution filter on input tensor with given stride and padding. |
| i8* **@hpvm.tensor.add**(i8* lhs, i8* rhs) | Element-wise addition on input tensors. |
| i8* **@hpvm.tensor.reduce_window**(i8* input, i32 reduction_type, i32 window_size) | Performs a (configurable) reduction operation over a specified window size on the input tensor. |
| i8* **@hpvm.tensor.relu**(i8* input) | Element-wise relu activation function. |
| i8* **@hpvm.tensor.clipped.relu**(i8* input) | Element-wise clipped relu activation function. |
| i8* **@hpvm.tensor.tanh**(i8* input) | Element-wise tanh activation function. |

## ApproxHPVM IR Portable Quality metrics:

$L_1$ Error: $L_1^e = \frac{L_1(A - G)}{L_1(G)}$

$L_2$ Error: $L_2^e = \frac{L_2(A - G)}{L_2(G)}$

# Keras -> ApproxHPVM

*Conv2D(Num_filters, Kernel_sizes, activation = “relu”, padding = “same”))*

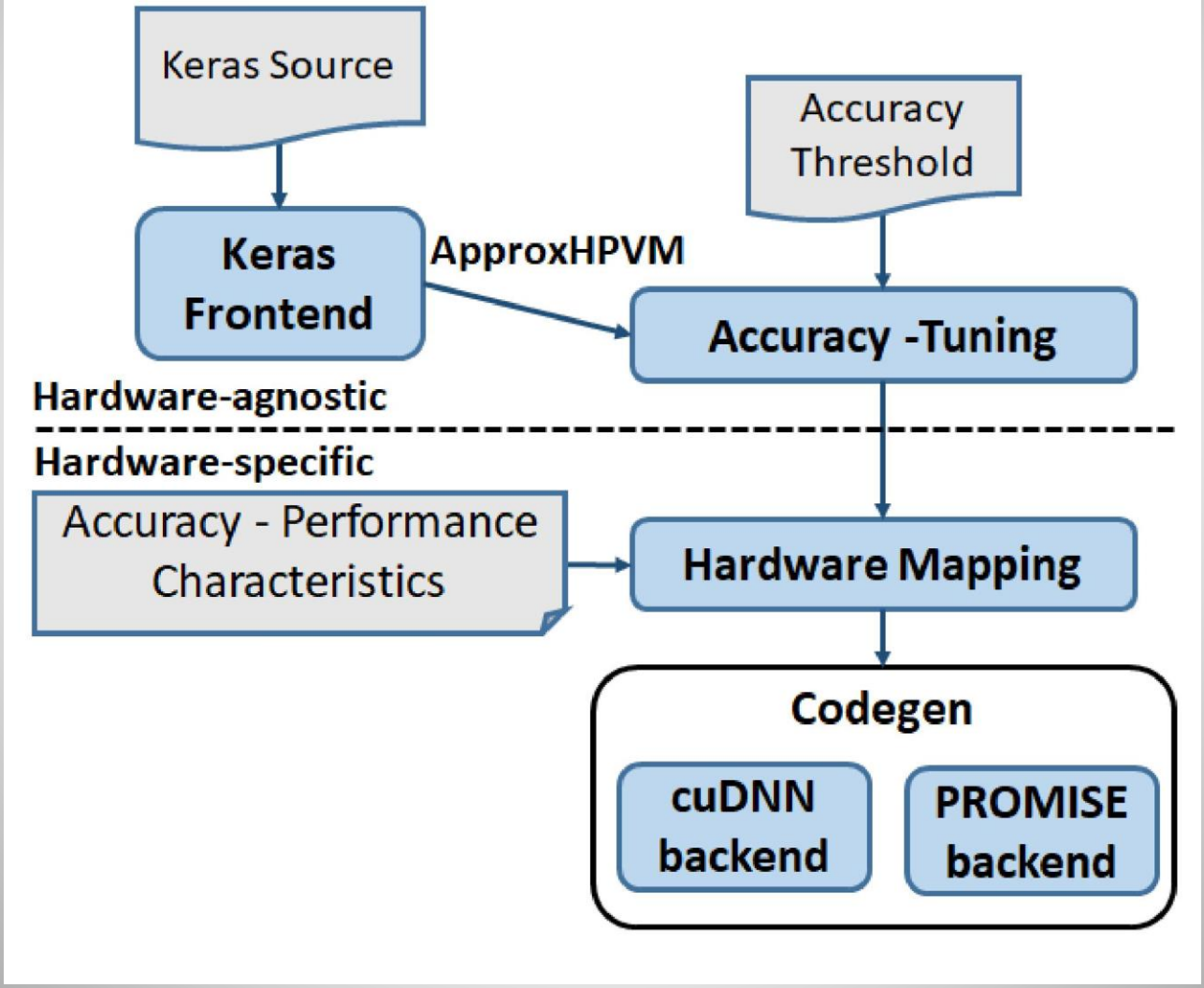
***Convolution layer In Keras***

# Keras -> ApproxHPVM

*Conv2D(Num_filters, Kernel_sizes, activation = "relu", padding = "same"))*

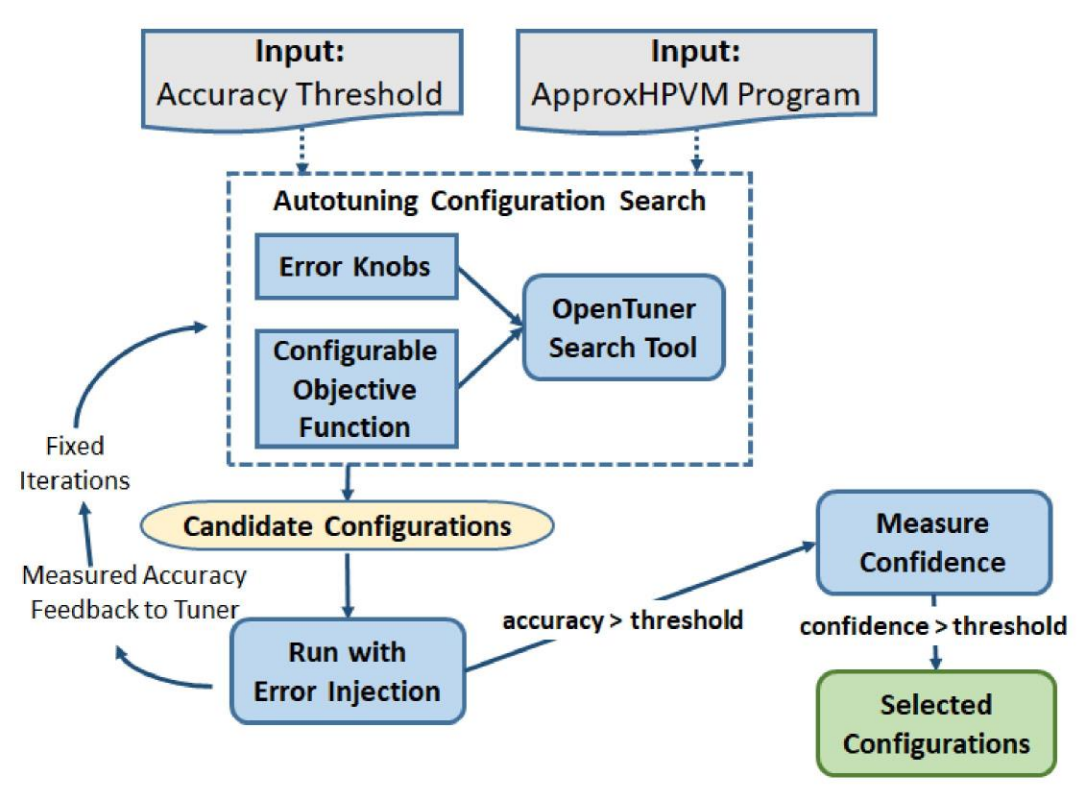
```
define i8* @tensorConvNode(i8* %input, i8* %filter) {
    %result = call i8* @tensor.conv(i8* %input, i8* %filter, i32* %padding)
    return i8* %result
}
define i8* @tensorAddNode(i8* %input, i8* %bias_weights) {
    %result = call i8* @tensor. add(i8* %input, i8* bias_weights)
    return i8* %result
}
define i8* @tensorReluNode(i8* %input) {
    %result = call i8* @tensor.relu(i8* %input)
    return i8* %result
}
```

# Keras Compiler Workflow



# Autotuning Workflow

**Use autotuning to determine per-task quality metrics**



# Experimental Setup

## Hardware Platform

- Nvidia Jetson TX2 GPU: direct execution
- PROMISE: simulations, validated against previous chips

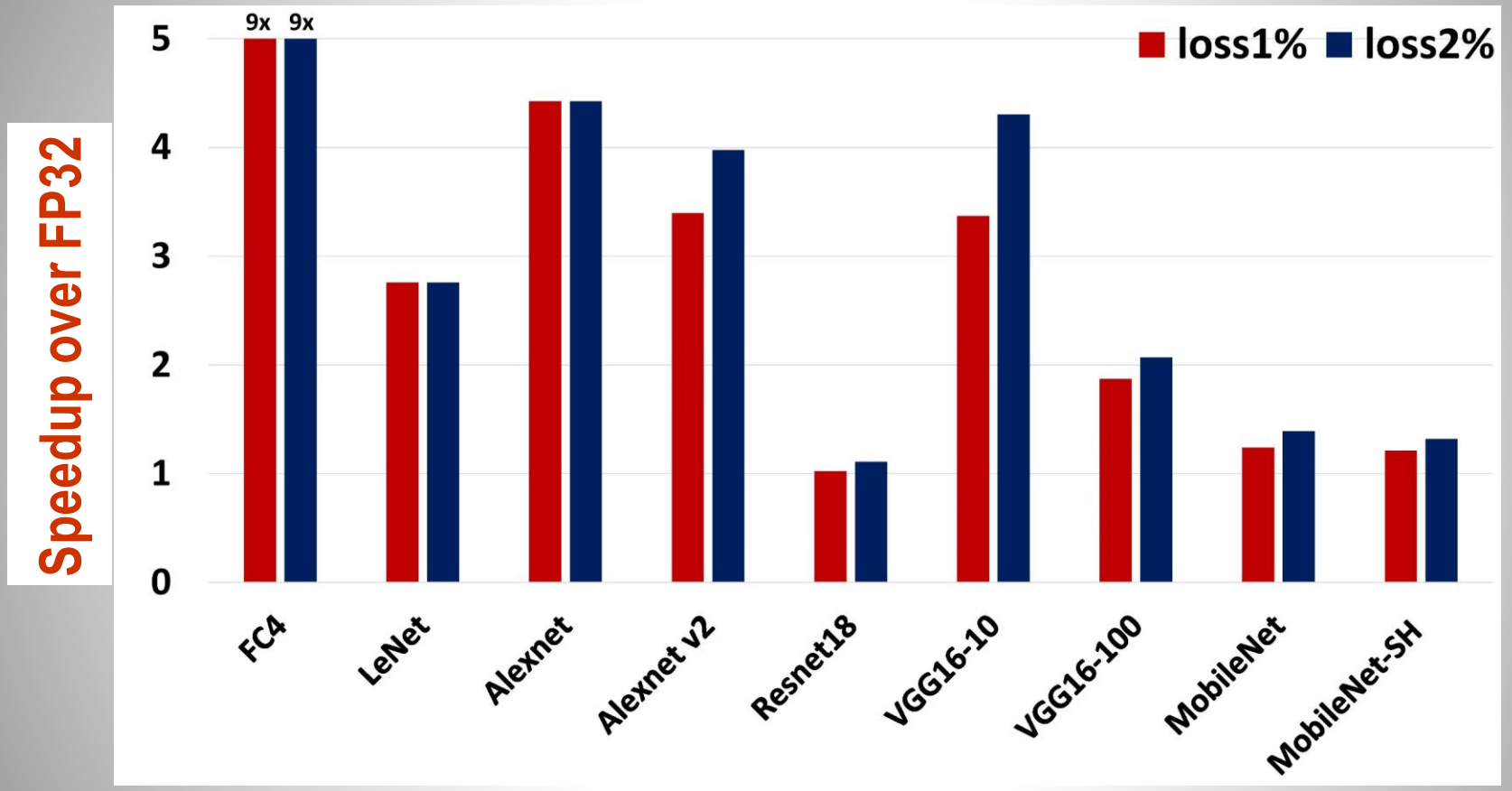
## Benchmarks:

- DNNs: FC-4, LeNet, Alexnet, ResNet, VGG-16, MobileNet, Shallow MobileNet
- Image processing pipelines of 3-4 filters:
  **G**aussian, **M**otion blur, **S**harpen, **O**utline, **E**mboss

## Quality comparisons:

- Baseline: FP32
- DNNs application goal: **Inference accuracy loss <= %1, loss <= %2**
- Image processing application goal: **PSNR=30, PSNR=20**

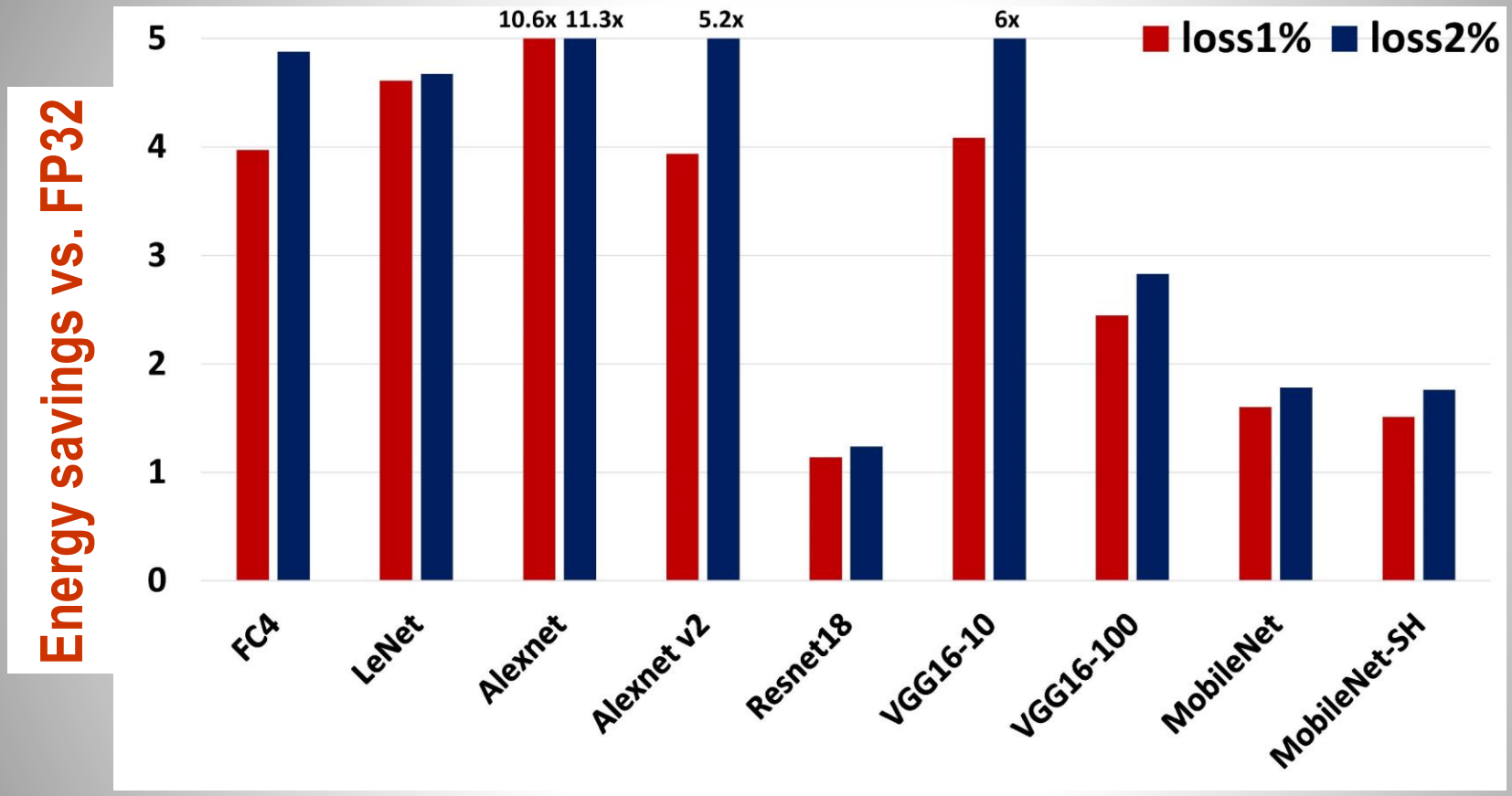
# DNN Speedup




**1-2% loss of inference accuracy gives**

**2x-9x speedups in most networks**

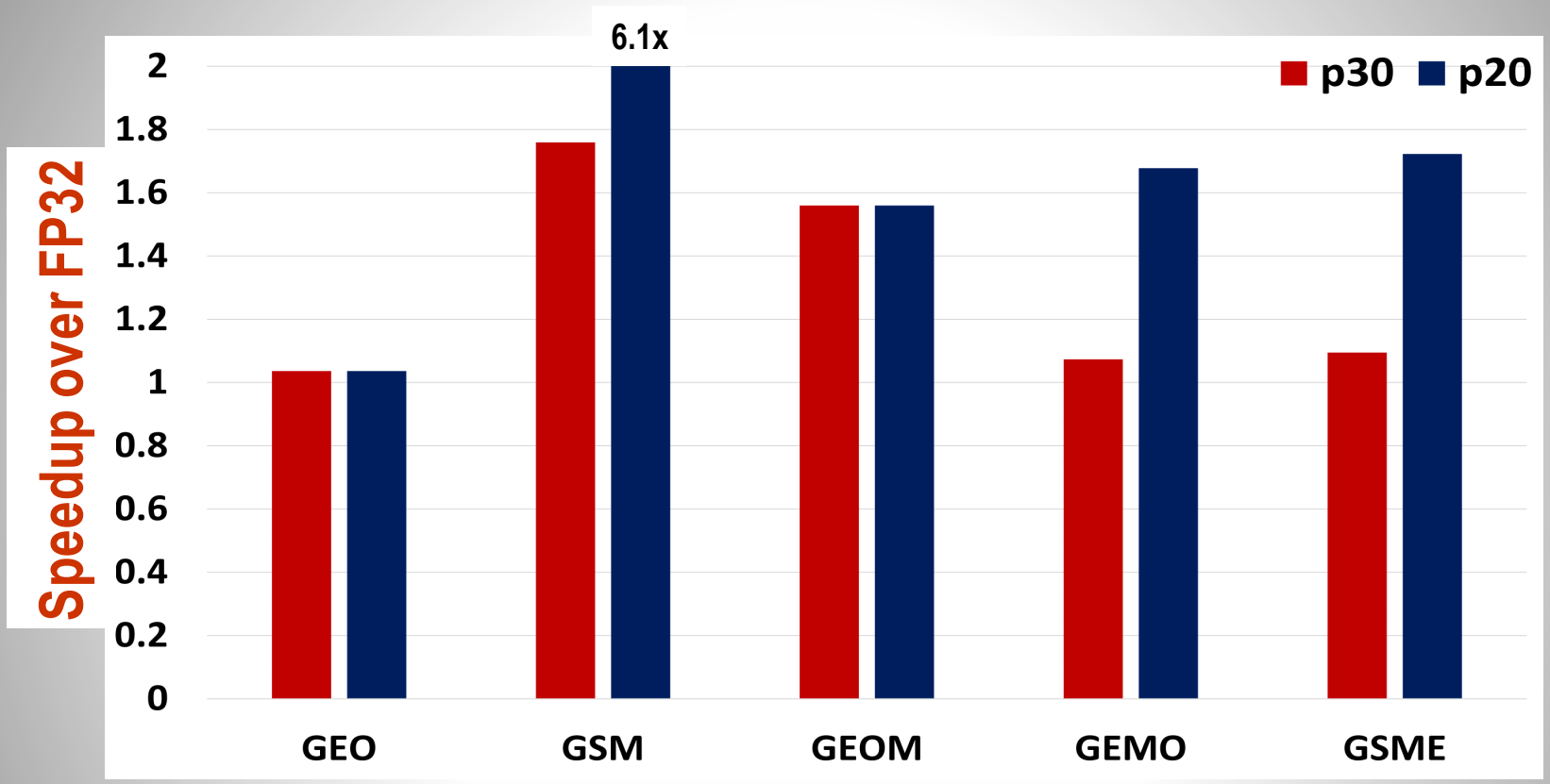
# DNNs: Energy Savings




**1-2% loss of inference accuracy gives**

**2x-11x energy savings in most networks**

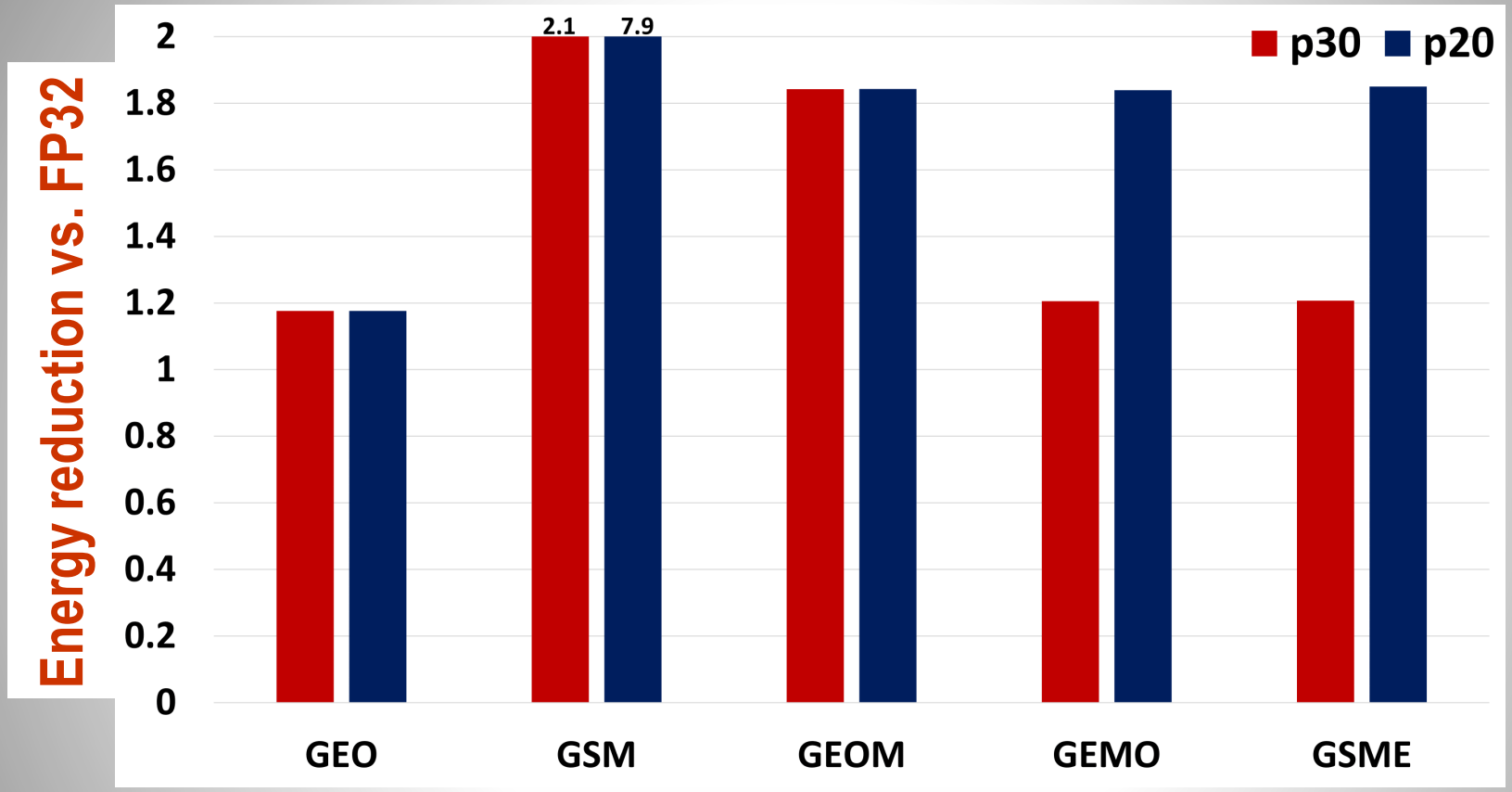
# Image Benchmarks Speedup




**PSNR of 30dB or 20dB gives**

**1x-6x speedups in image processing pipelines**

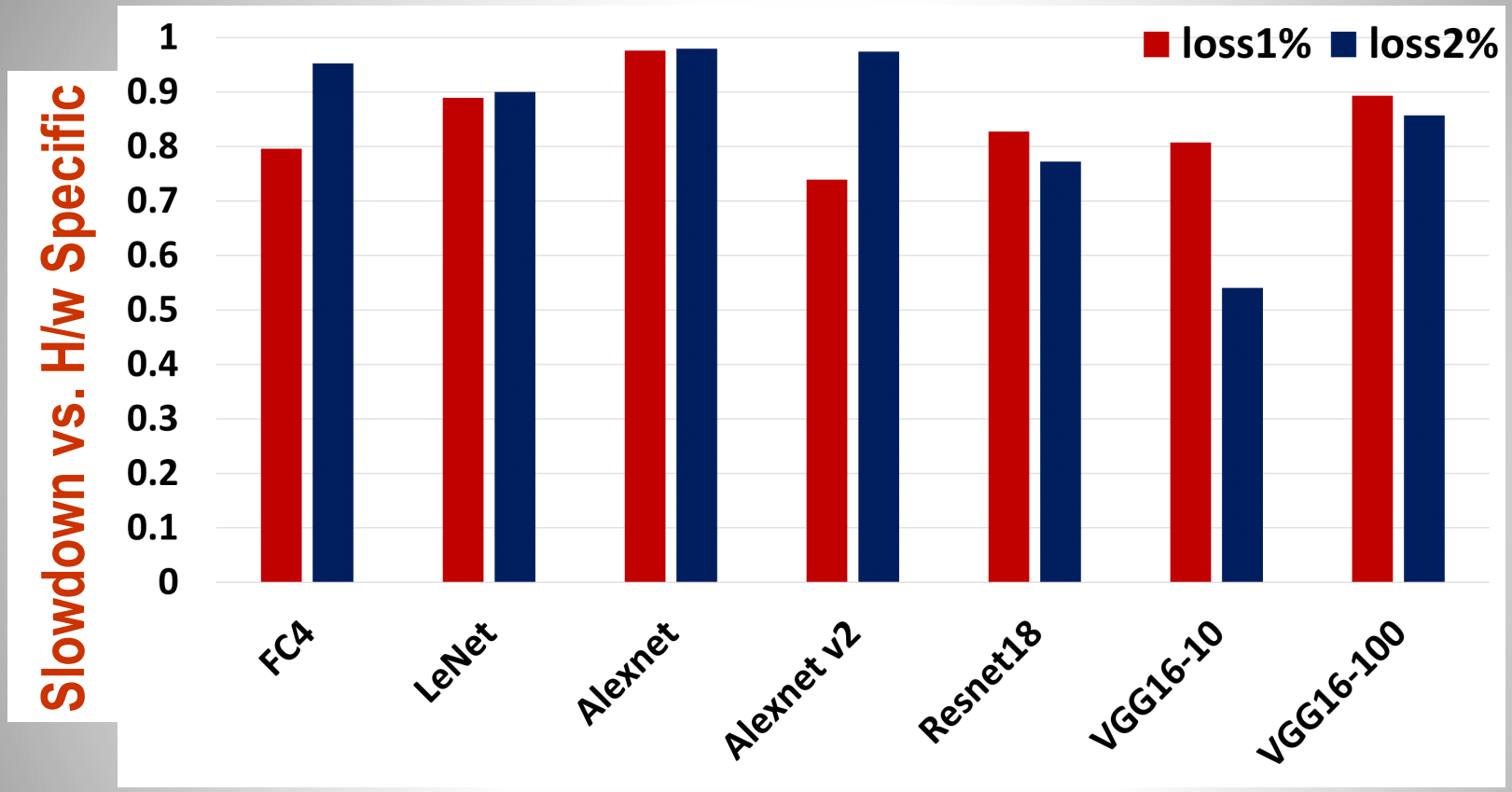
# Image Benchmarks: Energy Savings




**PSNR of 30dB or 20dB gives**

**1.2x-8x energy reduction in image processing pipelines**

# Hardware-agnostic vs Hardware-specific




**Hw-agnostic tuning is 0-25% worse than hw-specific**

**(Up to 45% in one case)**

# Ongoing Research (1)

## Extending ApproxHPVM for flexibility, efficiency

- Algorithmic approximations as well as system-level
- Dynamic selection of approximations
- More DNN architectures: deeper n/ws, recurrent n/ws
- New domains beyond tensor operations

# Ongoing Research (2)

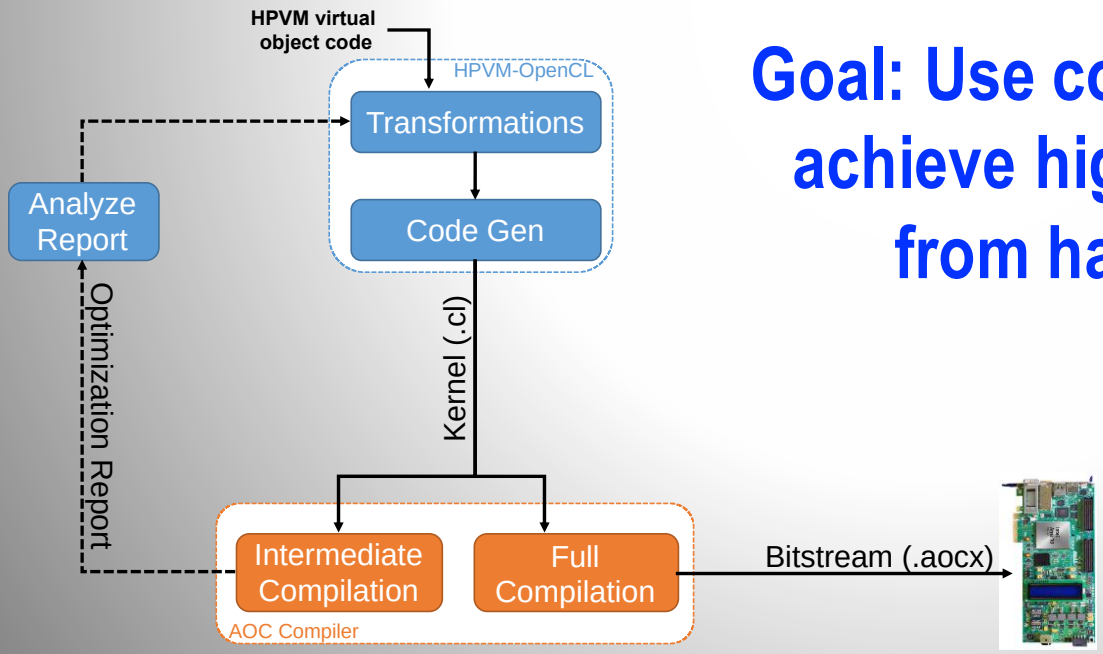
## Domain-specific programming of edge systems

- **Example: ARM (+ GPU) (+ DNN) (+ FPGA)**
- **Users: Crop scientists, civil engrs, medical researchers…**
- **Can we enable *non-expert users* to program complex heterogeneous SoCs?**
  - Very high-level DSLs
  - Automatic partitioning, approximation, mapping, code generation
  - Automatic run-time scheduling, performance analysis

# Ongoing Research (3)

## Hardware-agnostic programming of FPGAs

- **FPGAs are becoming widely available in data centers**
- **Application users lack hardware expertise**




**Goal: Use compiler optimizations to achieve high-perf. FPGA designs from hardware-agnostic code**

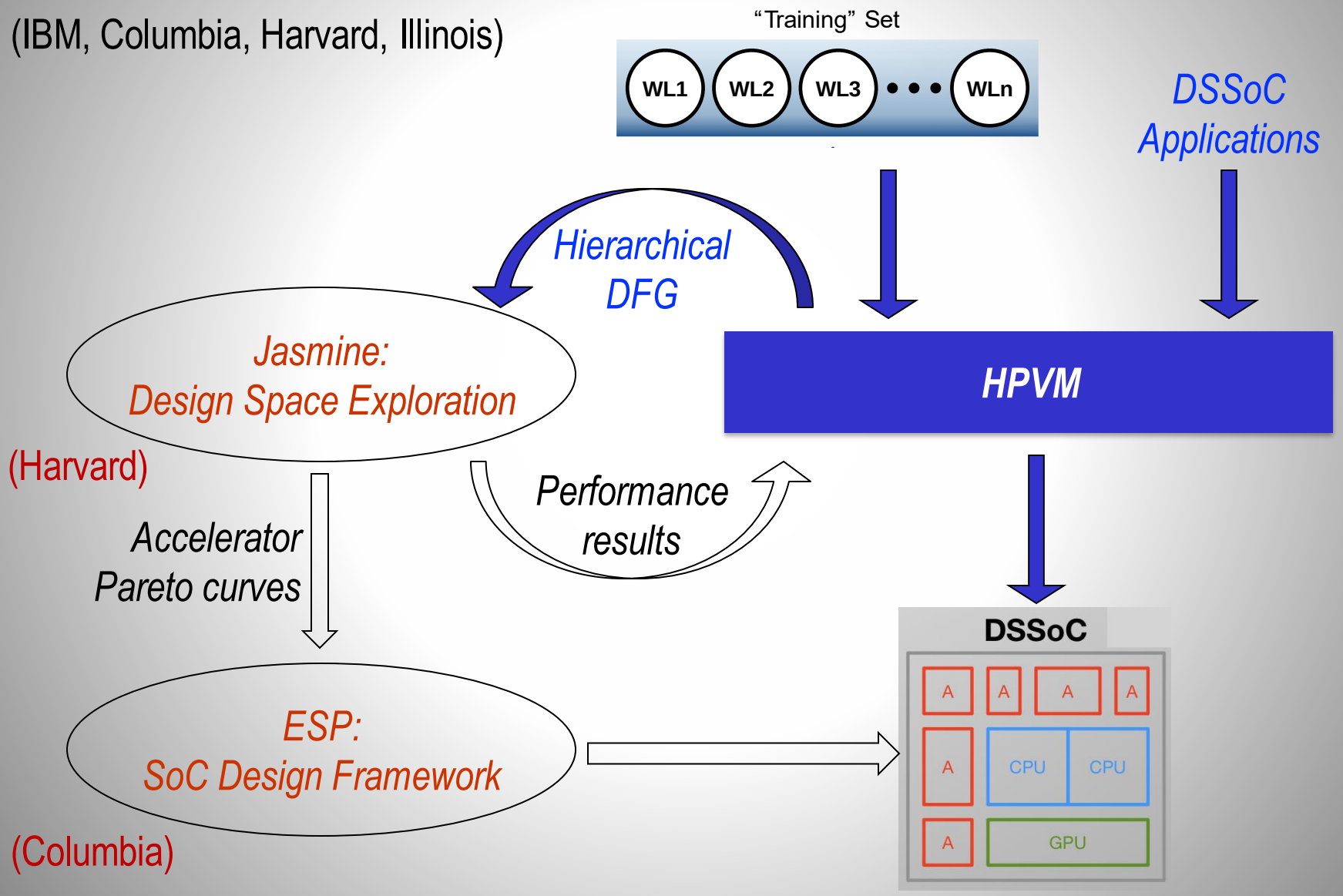
# Ongoing Research (4)

## Integrate ApproxHPVM with Jasmine Toolflow

- Improve hw-agnostic tuning to match hw-specific
- Partition application + iterate through design space
- Explore approximate hw, sw mechanisms

# DSSoC: Hardware Design Space Exploration

(IBM, Columbia, Harvard, Illinois)



# Summary

**HPVM:** portability + performance for heterogeneous systems

**ApproxHPVM:** easy access to approximation techniques

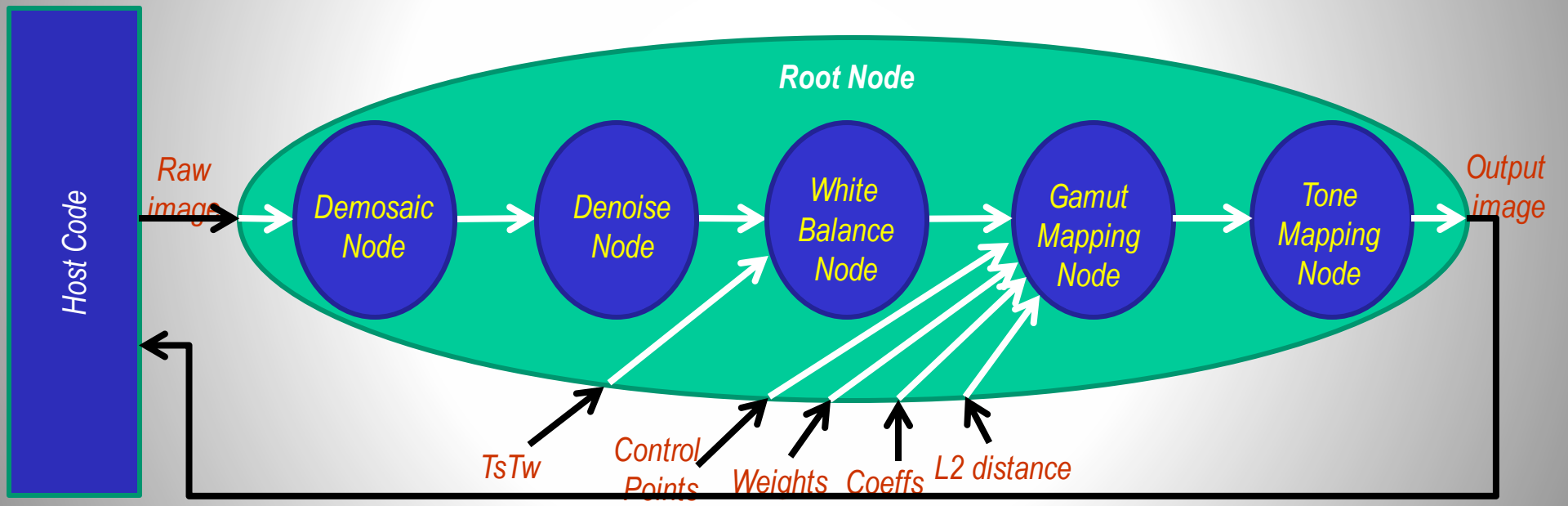
**Ongoing research:**

- Application-driven **hardware design**
- Rich compiler infrastructure for **DSLs**
- **Easy programming** of energy-efficient edge compute systems

***Questions?***

- **EXTRA SLIDES**

# Example: HPVM CAVA DFG Structure



# Example: HPVM CAVA DFG Structure