Programming CPU/FPGA Heterogeneous Systems for Image Processing from Halide DSL

Jing Pu, Steven Bell, Xuan Yang, Stephen Richardson, Jonathan Ragan-Kelley, Mark Horowitz

Motivation
- Image Processing Domain
  - Computer Vision
  - Computational Photography
  - Augmented Reality
- New SoCs with FPGA are promising platforms for acceleration

Challenges: Programmability
- Designing FPGA accelerator is HARD
- Integrating new HW to existing systems is HARD
- Writing software to access HW is HARD

Opportunities: Halide
- Halide splits algorithm from schedule
- Algorithms define essential computation
- Different architectures (CPU, GPU, FPGA) use different schedules for optimization

Design Flow
- Halide App
- Halide Compiler
- C Compiler
- HW HLS C
- HW Interface Config
- FPGA CAD tool
- System Generator
- HW Interface Config
- Target FPGA
- CPU/EPG System (Zynq)
- FPGA Backend
- Prototyping algorithms uses X86 backend.
- Accelerating uses heterogeneous backend.

High-level Language: Halide

```
Func unsharp(Func in) {
    Func gray, blurx, blury, sharpen, ratio, unsharp;
    Var x, y, c, xi, yi;
    // The algorithm
    gray(x, y) = 0.3*in(0, x, y) + 0.6*in(1, x, y) + 0.1*in(2, x, y);
    blurx(x, y) = (gray(x, y-1) + gray(x, y) + gray(x, y+1)) / 3;
    blury(x, y) = (gray(x-1, y) + gray(x, y) + gray(x+1, y)) / 3;
    sharpen(x, y) = 2 * gray(x, y) - blurx(x, y);
    ratio(x, y) = sharpen(x, y) / gray(x, y);
    unsharp(x, y) = ratio(x, y) * input(c, x, y);
    return unsharp;
}
```

Halide algorithm and schedule for unsharp

Compiler and Platform

Compiler based on Halide compiler

Platform based on Xilinx Zynq SoC

System Performance and Energy Efficiency

Our system vs. NVIDIA Tegra K1’s four ARM cores and GPU:

- Baseline
  - Throughput
  - Energy Efficiency

K1 CPUs: 6x (avg. 2.6x)
K1 GPU: 3.5x (avg. 2x)

As image processing pipeline grows deeper, the FPGA acceleration becomes more effective.

Contact Information
Jing Pu, jingpu@stanford.edu