# Agile Hardware: Rethinking DSSOC Design

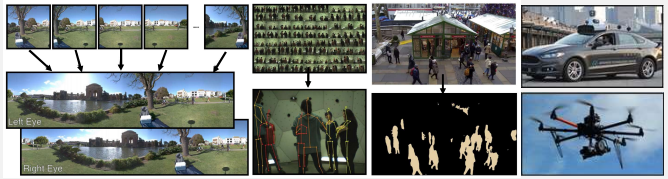
## *Rapidly Mapping Applications to Specialize Hardware and Doing It Over and Over Again*

Clark Barrett, Kayvon Fatahalian, Pat Hanrahan, Mark Horowitz, & Priyanka Raina

Stanford University

## Architectures Thrust: Domain-Specific System on Chip (DSSoC)

### Driving Application Domain: Video Analysis




**Cloud-scale video data mining**
Image processing and DNN inference on millions of hours of video

**Real-time video stream processing (including fusion of multiple video streams)**
Online DNN training/inference, registration, reconstruction, flow/depth estimation

**Ultra-low latency/power deployments**
Always on sensing/wakeboarding, sense-process-display loops

### DSSoC Design

**Today**

**IP Based Design**

- Get base IP blocks
- Create custom accelerators
  - Study applications to accelerate
  - Design accelerators
  - Create firmware for accelerators
- Create system software
  - For IP blocks, and accelerators
- 2-3 years later
  - Hope it works
  - Hope it solves the right problem

**Potential Future**

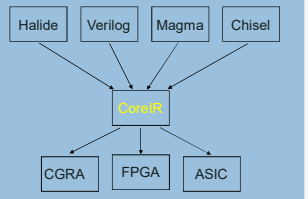
**Agile Hardware/Software Design**

- Bring rapid learning to hardware
  - Tighter coupling between apps/hardware
  - Faster design cycles
    - Takes many iterations to reach final design
- Don't start w/ accelerator design
  - Start with applications
    - Compile to hardware
    - Map to programmable substrate (universal)
  - Incrementally improve substrate
    - While maintaining the programming system
    - Latest application can **always** be mapped

### CoreIR : LLVM for Hardware

Lessons from LLVM
- Multiple language front-ends
- Multiple back-end targets

Goal: Build complete compiler
- From HDL to Verilog/bitstream
- Compile and link to libraries




CoreIR Compiler Pass Types

Generators
Modules
Instances

Process different node types
- NamespacePass
- ModulePass
- InstancePasses

Pass manager
- Executes passes in order
- Caches intermediates

Example Passes
- Constant folding
- Dead circuit elimination
- Mux optimization
- Flattening and elaboration
- Clock and power domains
- Backends
- Formal verification

### Input: Domain Specific Languages

- Halide: Declarative, data-parallel expression of image processing algorithms
- Widely used and maintained by industry (Google, FB, Instagram, Adobe)
- Now considering new extensions for video (time varying data)

**Functional Halide algorithm description:**

```
// 3x3 separable convolution in Halide
blurx(x,y) = (in(x-1,y)    + in(x,y)    + in(x+1,y))    / 3;
out(x,y)   = (blurx(x,y-1) + blurx(x,y) + blurx(x,y+1)) / 3;
```

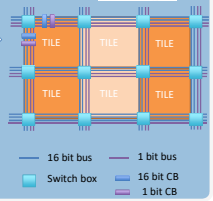
**Compact language for describing algorithm's mapping to parallel machine**

```
out.tile(x, y, xi, yi, 256, 32);      compute output in tiled order
out.vectorize(xi, 8);                 vectorize innermost loop
out.parallelize(y);                   parallelize loop across cores
blurx.compute_at(xi);                 loop fusion
blurx.vectorize(x, 8);                vectorize innermost loop
```

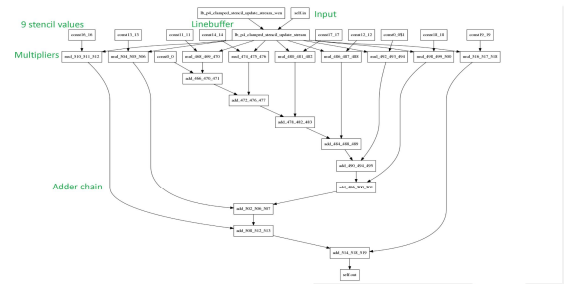
### Agile Hardware (AHA!)

How to easily add a cup-holder to your system

- Goal today is to extend working systems
  - Want to add "cup holder" to complex system
    - Cup holder = small addition to complex products with large value
- Must start with working end-to-end system
  - Create a hardware playground – we created a CGRA
  - Tools to allow application programmers to use the system
  - This system won't be perfect for your task
    - Learn from these results to improve most critical features
- Use this system to build accelerators
  - Quickly generate data on where bottlenecks really are
    - Improve software/hardware/tools to remove bottlenecks

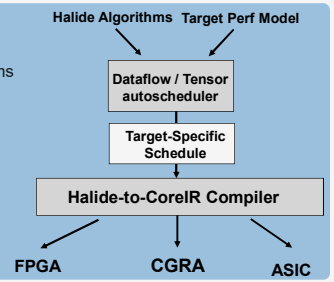



### Example: Compiling Halide to CoreIR



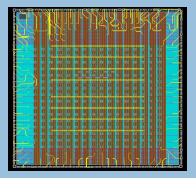

### Auto-Scheduling Halide to Hardware

Early work auto-scheduled Halide programs
- For CPUs and GPUs
- Now targeting DSP, and hardware
  - ASIC, FPGA, and CGRA




### Philosophy – Out Reach

- Practice what we preach
  - Use agile practices in the design of our tools and hardware
    - Continuous integration, rapid design cycles
    - Plan multiple tapeouts / year
  - Eat our own dogfood
    - Use our tools to create prototypes
    - Use our prototypes to create new systems
- Run a focused project
  - Already have taped out our first test chip
- It is all open source
  - Have a set of large companies supporting this effort
  - Intel, NVIDIA, Amazon, Facebook, + …



### COSA – CoreIR Symbolic Analyzer

Goal: Prove compiler transformations generate correct code
By: Leveraging Satisfiability Modulo Theories (SMT) solvers for formal checking

1. CoreIR primitives based on SMT-LIB BitVector and Arrays which have a formal semantics
2. Compiler passes insert additional lemmas that describe the IR transformations, e.g., state to state mapping
3. Leads to much faster model checking proofs using SMT (e.g. 10 secs. vs 2 hrs.)

DNN: Deep Neural Network
FB: Facebook
CPU: Central Processing Unit
GPU: Graphics Processing Unit
ASIC: Application-Specific Integrated Circuit
FPGA: Field Programmable Gate Array

CGRA: Coarse-Grained Reconfigurable Architecture
IP: Intellectual Property
LLVM: Low-Level Virtual Machine
HDL: Hardware Description Language

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.
Distribution Statement A – Approved for Public Release, Distribution Unlimited

THE ELECTRONICS RESURGENCE INITIATIVE