CGRA Virtualization

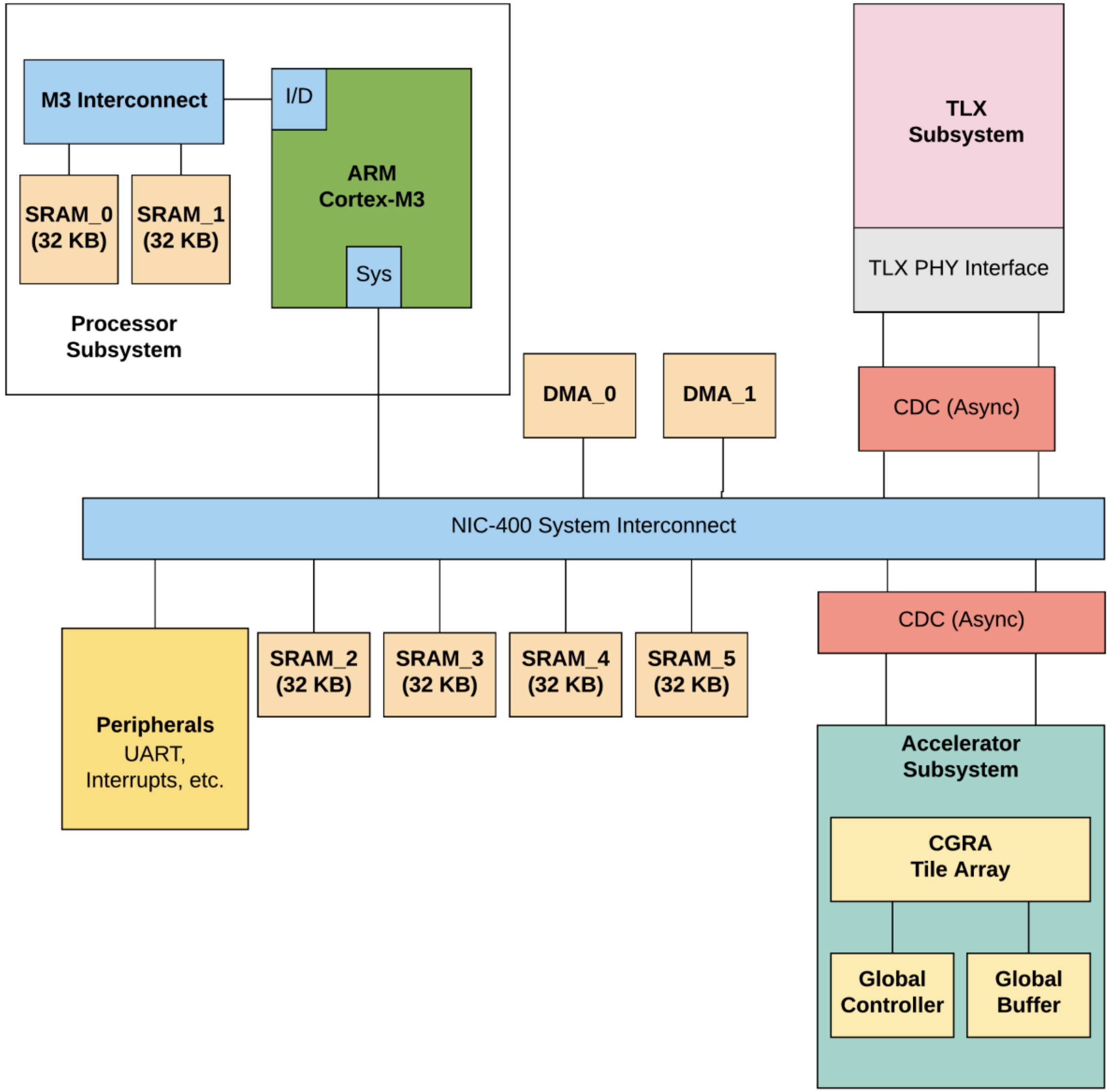
Kalhan Koul

Stanford University

Outline

- Amber SoC Review
  - Quick Overview of the SoC
  - Quick Overview of the CGRA Architecture
  - Application Walk Through
- Motivation for Virtualization
- Hardware/Software Enabling Improvements
- Temporal Virtualization
- Spatial Virtualization
- Ongoing Work

Amber SoC Architecture



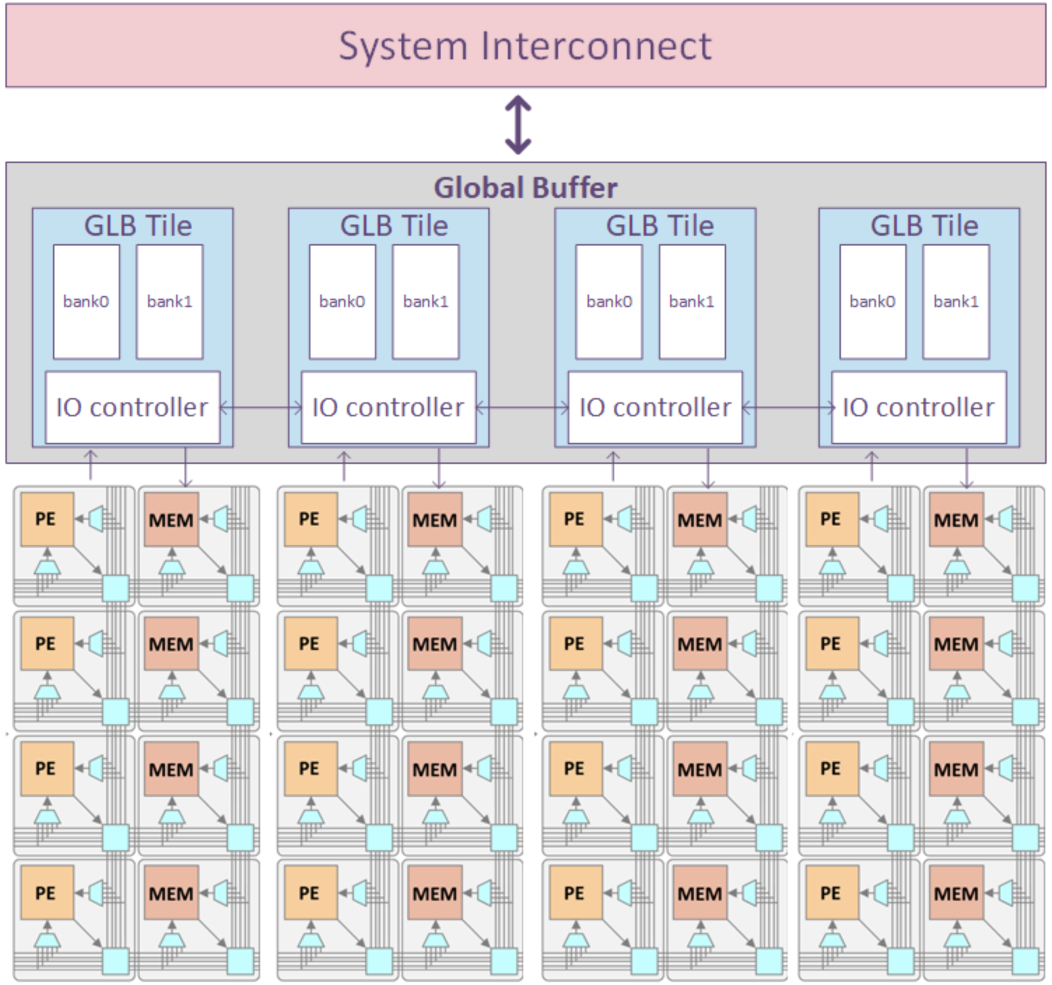
CGRA Architecture

Global Buffer

- Communicates with the SoC system over AXI4 and AXI4-Lite interface
- 1D array of GLB Tiles each with two banks
- I/O controller streams data into/out of CGRA
- Streams the bitstream into CGRA

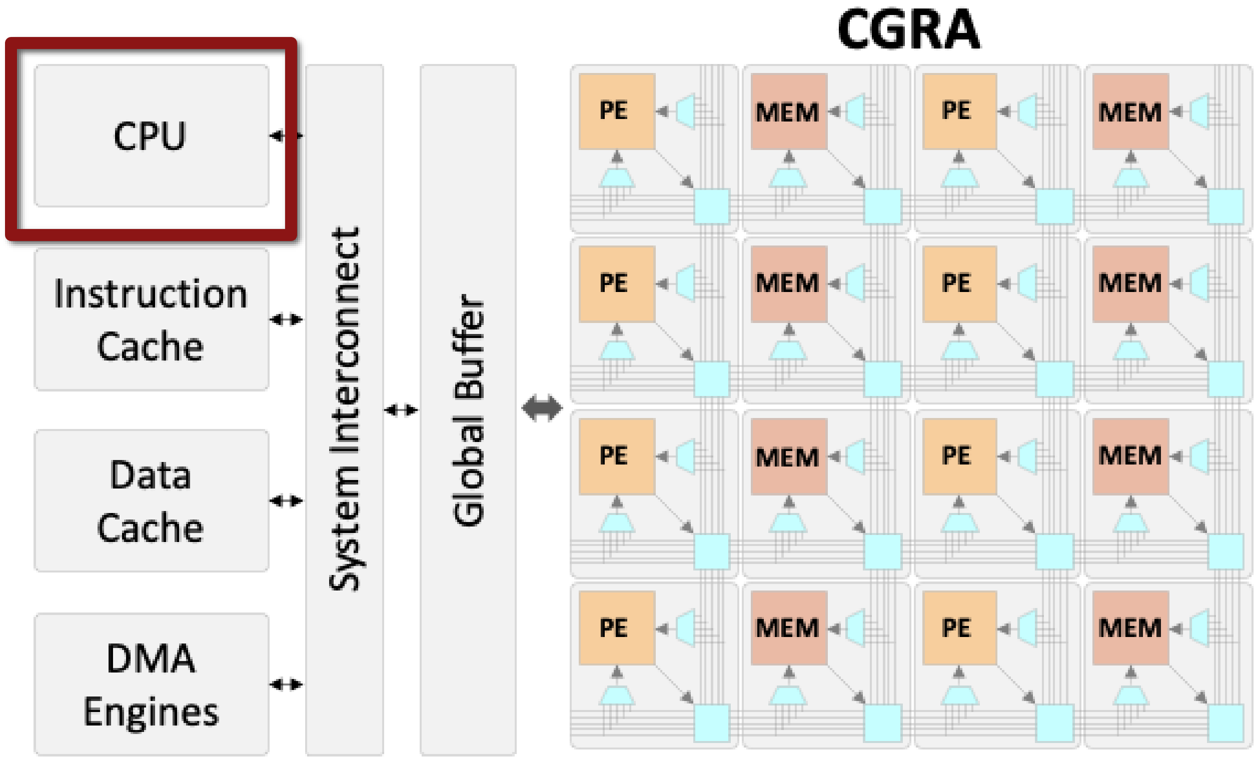
CGRA

- Island-style mesh network with PE tiles and MEM tiles
- Divided into groups of columns



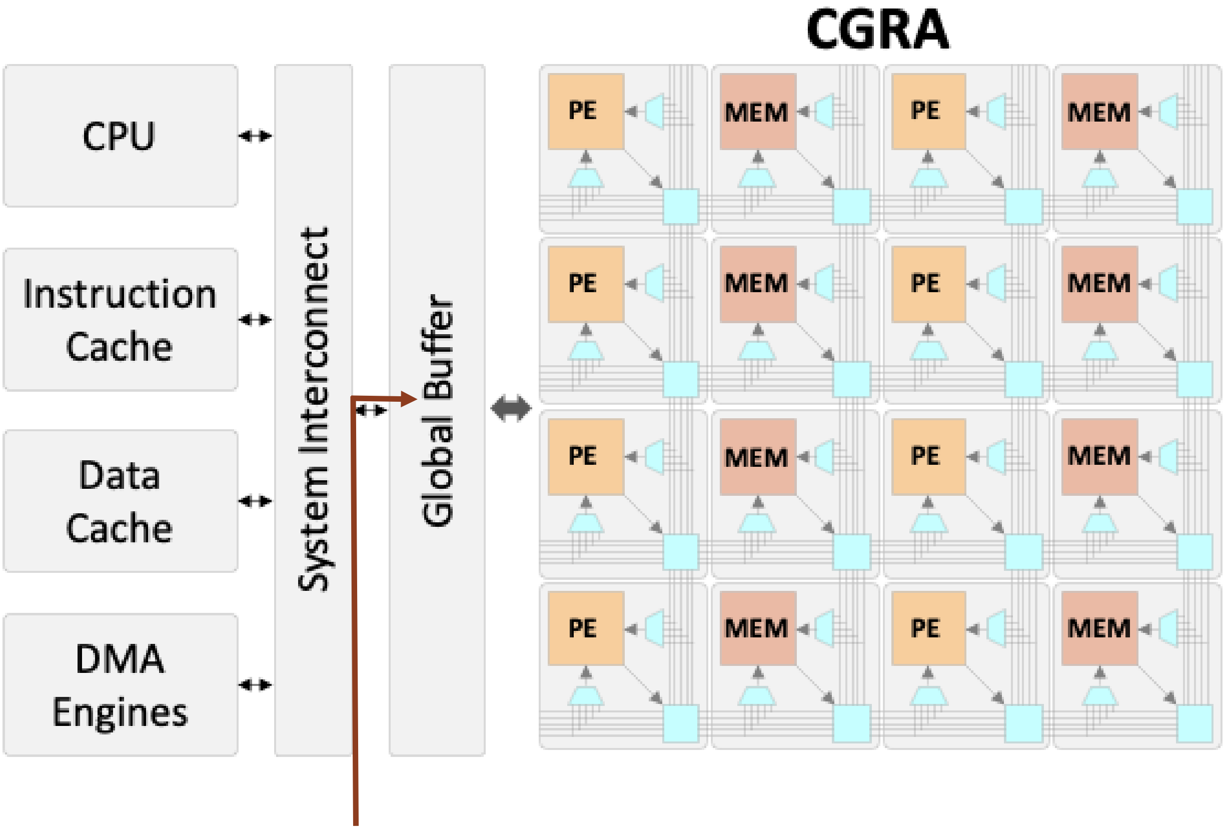
Conducting an Application with the M3 Core

1. **Initialize System**
   - **Enable UART, DMA engines, etc.**
   - **Select Clocks, Interrupts, etc.**
2. Bring in Configuration and Input Data
   - Use DMA engines to move data (bitstream, images, etc.)
   - From DRAM/SRAMs to Global Buffer
3. Configure Global Buffer and Run Kernel
   - Use Parallel Configuration to send the bitstream to the CGRA
   - Stream in input data and stream out output data
4. Repeat Step 2 and 3 per Application
5. Finish Application
   - DMA Output Data to DRAM
   - Send done information to Host Processor



Conducting an Application with the M3 Core

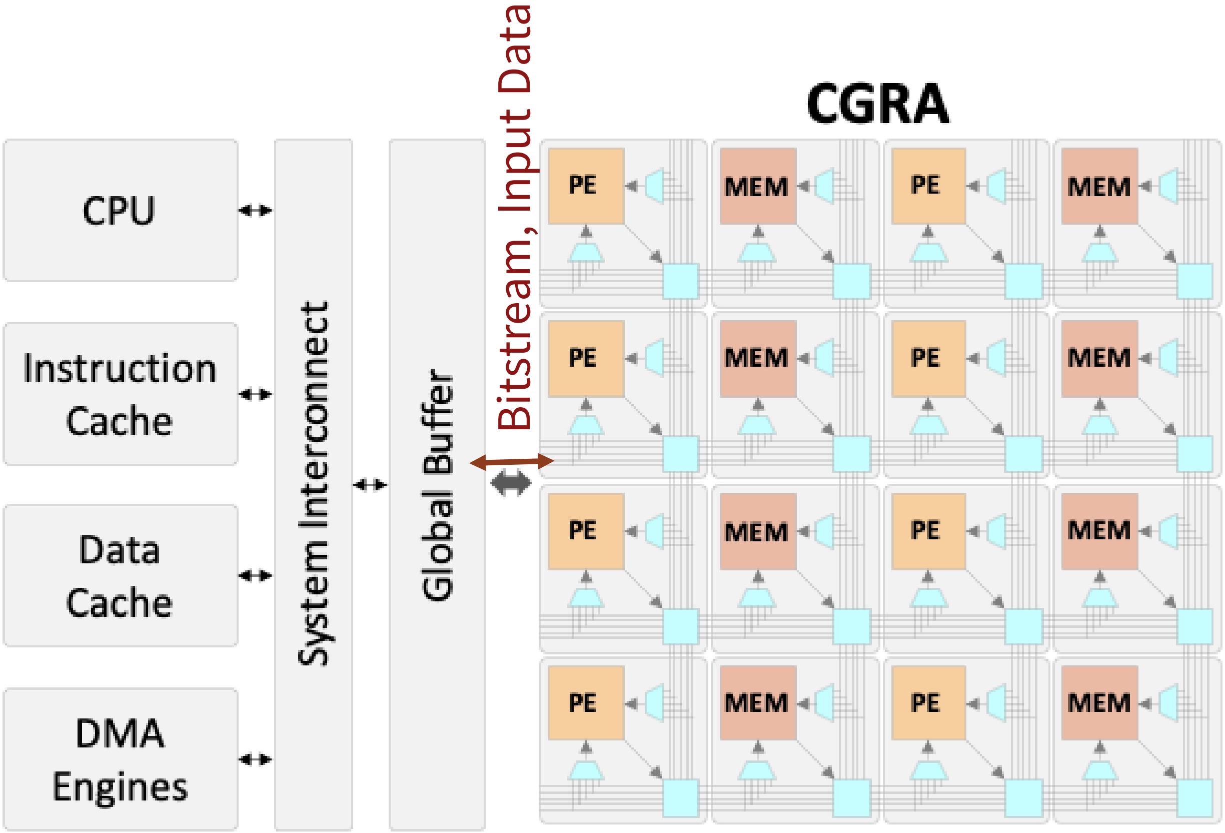
1. Initialize System
   - Enable UART, DMA engines, etc.
   - Select Clocks, Interrupts, etc.
2. **Bring in Configuration and Input Data**
   - **Use DMA engines to move data (bitstream, images, etc.)**
   - **From DRAM/SRAMs to Global Buffer**
3. Configure Global Buffer and Run Kernel
   - Use Parallel Configuration to send the bitstream to the CGRA
   - Stream in input data and stream out output data
4. Repeat Step 2 and 3 per Application
5. Finish Application
   - DMA Output Data to DRAM
   - Send done information to Host Processor



Off Chip: Bitstream, Input Data

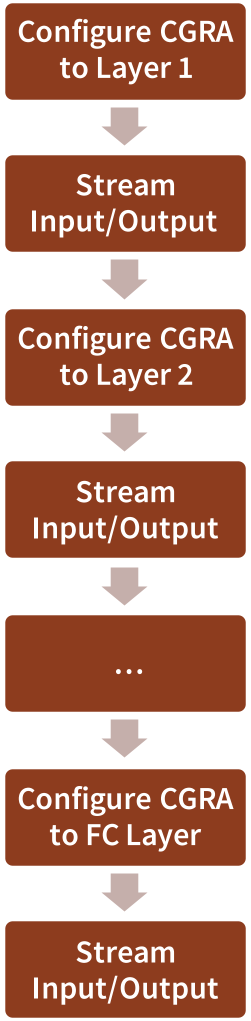
Conducting an Application with the M3 Core

1. Initialize System
   - Enable UART, DMA engines, etc.
   - Select Clocks, Interrupts, etc.
2. Bring in Configuration and Input Data
   - Use DMA engines to move data (bitstream, images, etc.)
   - From DRAM/SRAMs to Global Buffer
3. **Configure Global Buffer and Run Kernel**
   - **Use Parallel Configuration to send the bitstream to the CGRA**
   - **Stream in input data and stream out output data**
4. Repeat Step 2 and 3 per Application
5. Finish Application
   - DMA Output Data to DRAM
   - Send done information to Host Processor



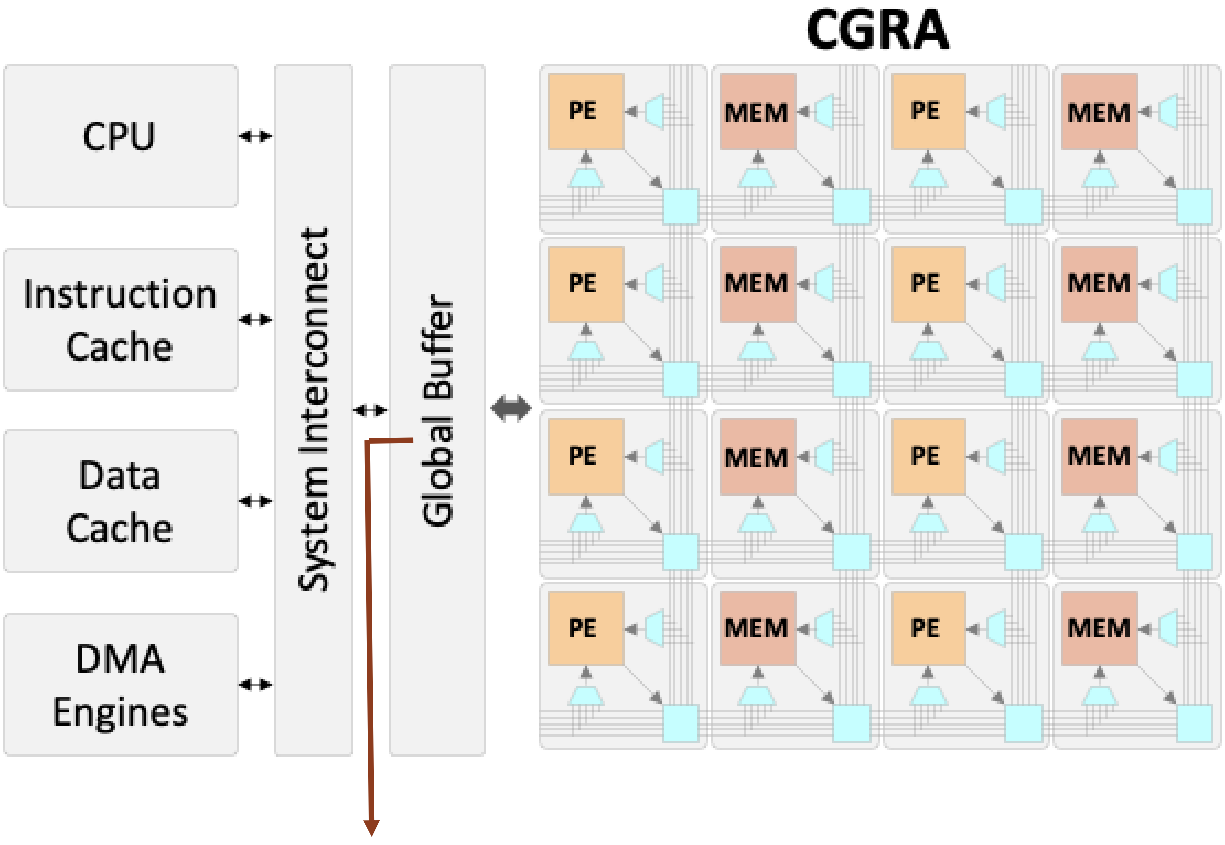
Conducting an Application with the M3 Core

1. Initialize System
   - Enable UART, DMA engines, etc.
   - Select Clocks, Interrupts, etc.
2. Bring in Configuration and Input Data
   - Use DMA engines to move data (bitstream, images, etc.)
   - From DRAM/SRAMs to Global Buffer
3. Configure Global Buffer and Run Kernel
   - Use Parallel Configuration to send the bitstream to the CGRA
   - Stream in input data and stream out output data
4. **Repeat Step 2 and 3 per Application**
5. Finish Application
   - DMA Output Data to DRAM
   - Send done information to Host Processor


Configure CGRA to Layer 1
Stream Input/Output
Configure CGRA to Layer 2
Stream Input/Output
…
Configure CGRA to FC Layer
Stream Input/Output

Conducting an Application with the M3 Core

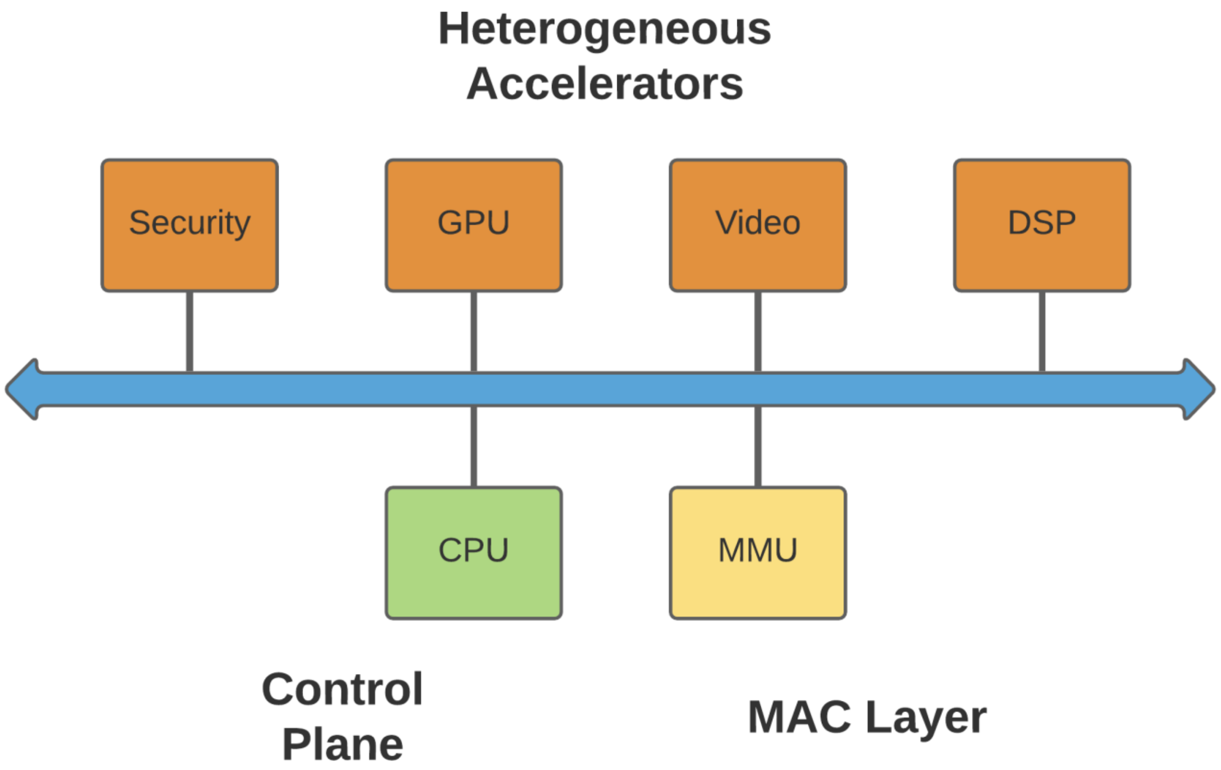
1. Initialize System
   - Enable UART, DMA engines, etc.
   - Select Clocks, Interrupts, etc.
2. Bring in Configuration and Input Data
   - Use DMA engines to move data (bitstream, images, etc.)
   - From DRAM/SRAMs to Global Buffer
3. Configure Global Buffer and Run Kernel
   - Use Parallel Configuration to send the bitstream to the CGRA
   - Stream in input data and stream out output data
4. Repeat Step 2 and 3 per Application
5. **Finish Application**
   - **DMA Output Data to DRAM**
   - **Send done information to Host Processor**



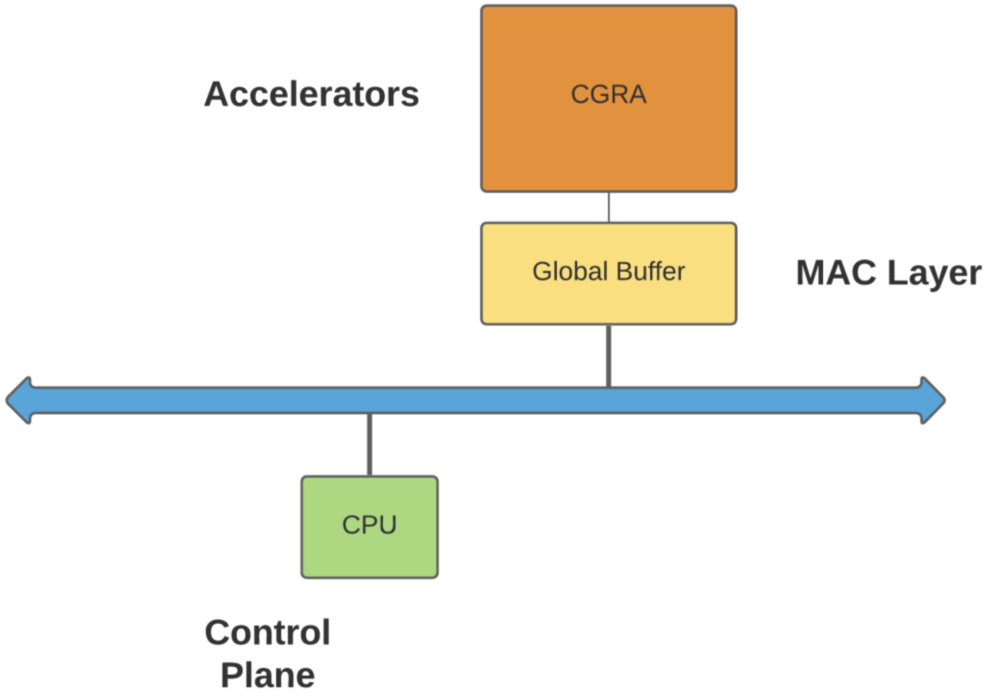
Output Data

Motivation for Virtualization

Traditional View



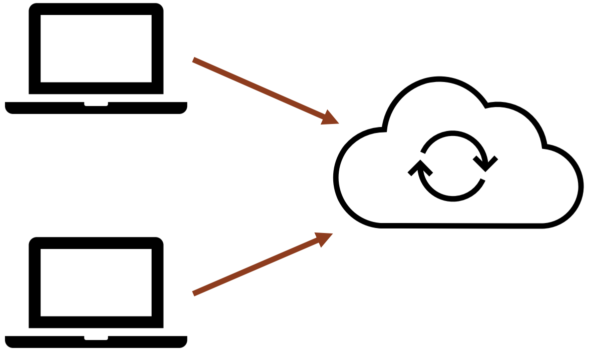
Our View

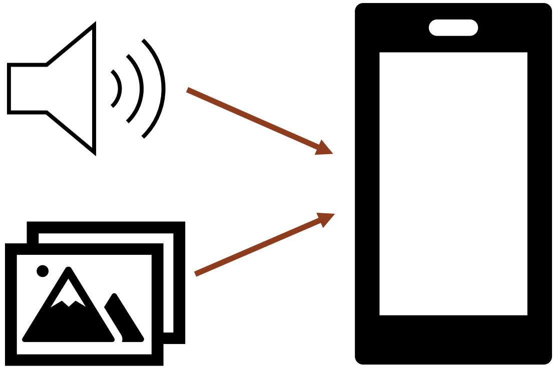


Motivation for Virtualization

1. **Cloud**: Two users access server for application acceleration
   - Most domain specific accelerators will schedule one after the other with high PE utilization
   - Large CGRA fabric gives us opportunity to do both in parallel, to achieve high PE utilization

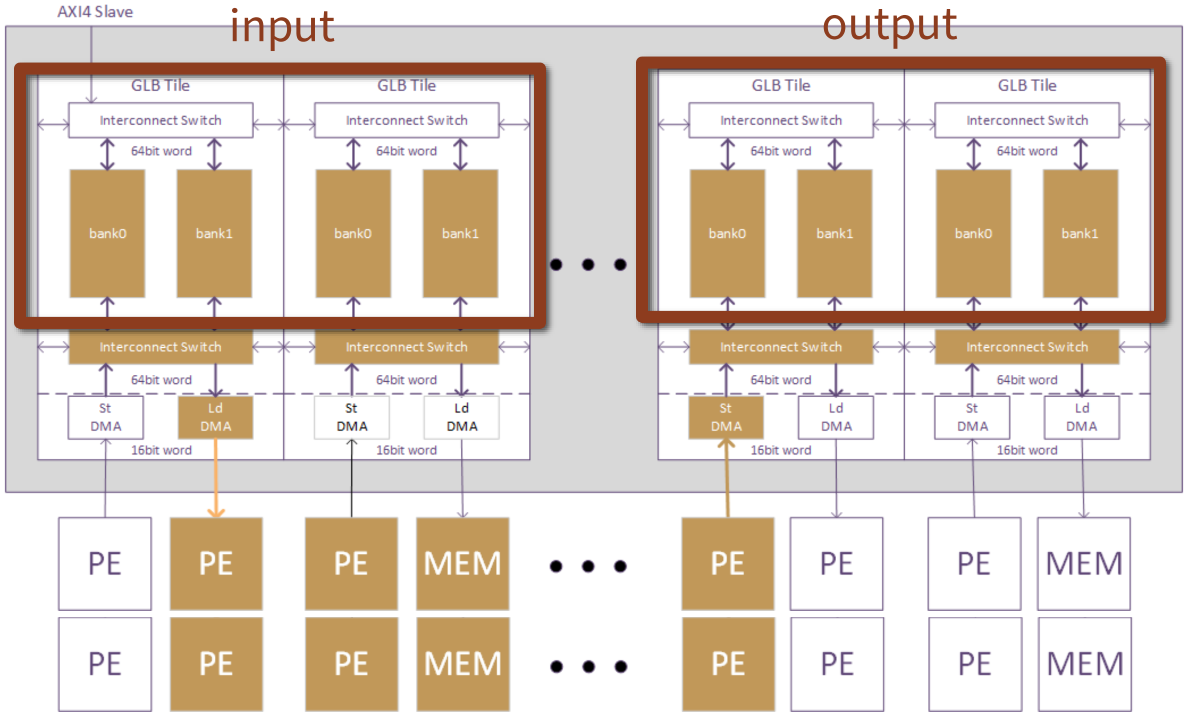
2. **Edge**: Two sensors need immediate processing
   - Most edge devices have several accelerators they need to manage and schedule
   - CGRA gives us opportunity to do both in parallel on the same accelerator, simplifying SoC architecture and scheduling management





Hardware/Software Features Implemented

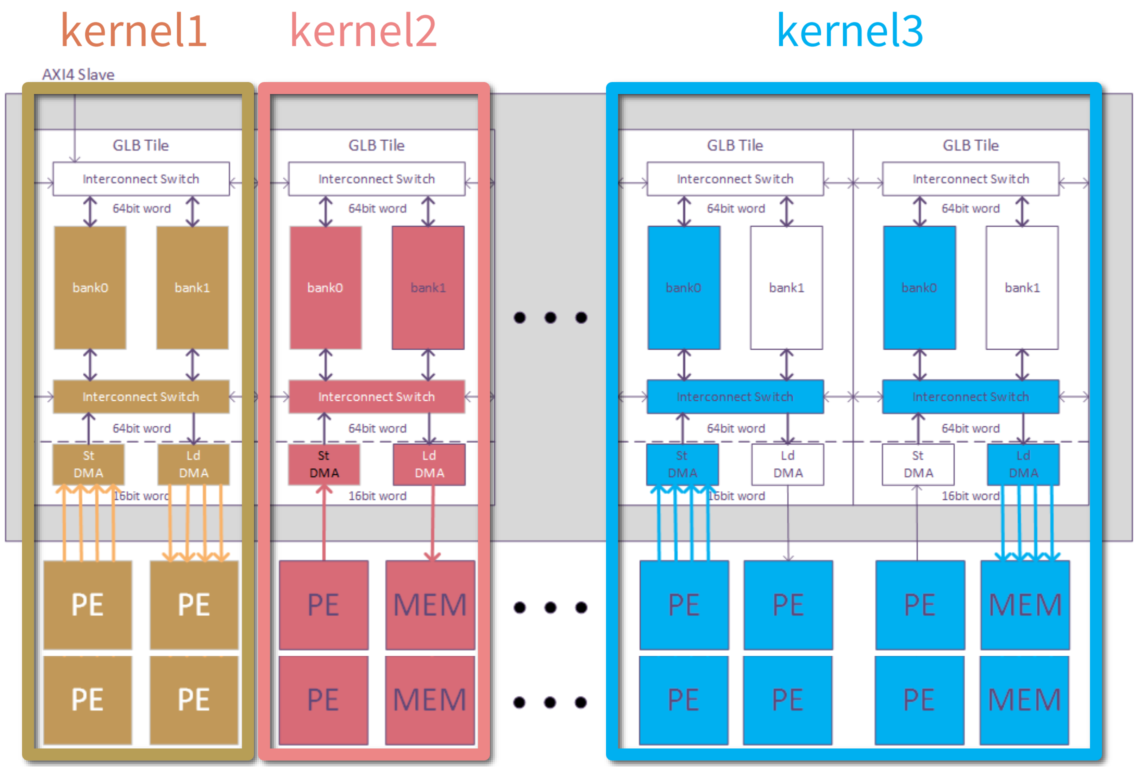
Flexible Global Buffer Allocation



For a Single Kernel

- Input buffer (4 banks)
- Output buffer (4 banks)
- Flexibility in mapping applications with different input/output data bandwidth

Multiple Kernels in Parallel



Up to 8 Kernels on 32x16 CGRA Fabric

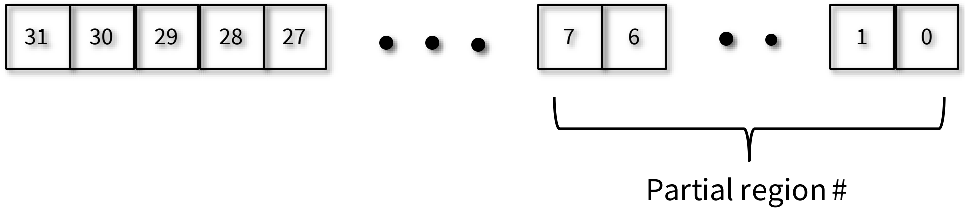
- Different kernels run in different regions of the CGRA
- Independent control of each kernel (start, interrupt)

Stanford University

Hardware Support for Reconfiguration

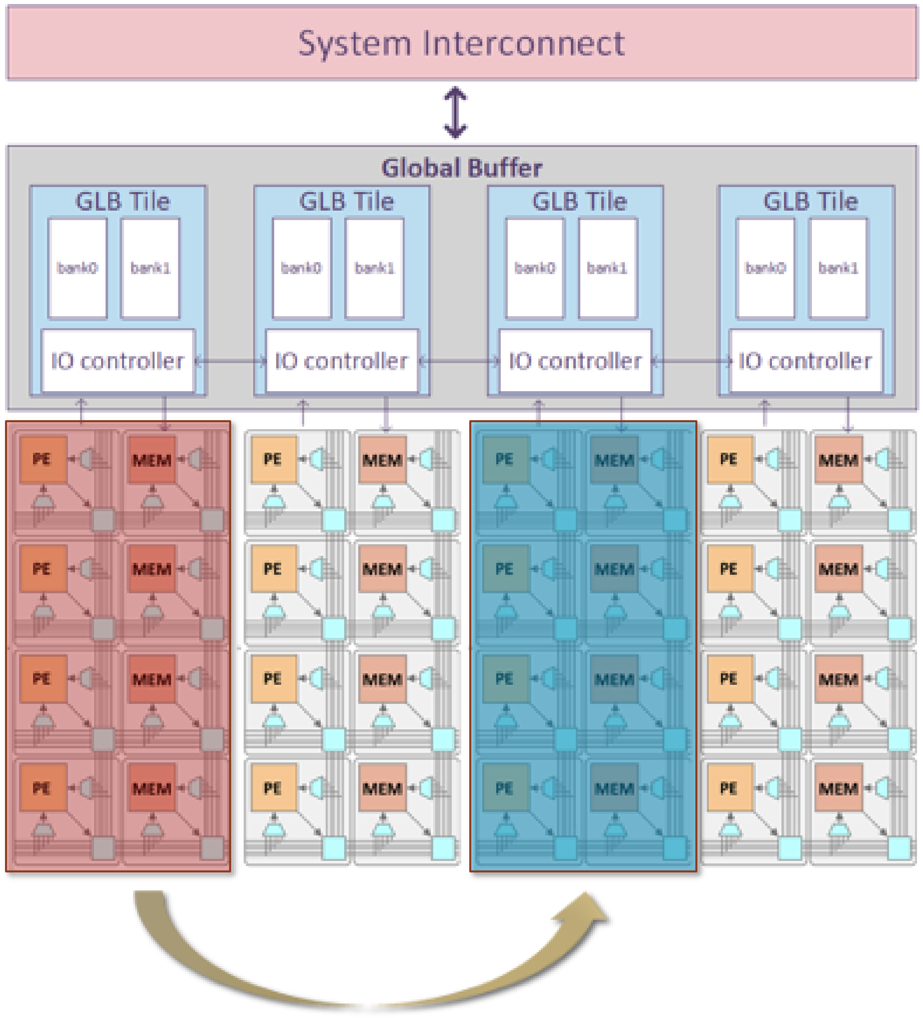
- #1 Bitstream is relocatable
  - This allows flexible hardware allocation for each kernel without software overhead

- Bitstream address breakdown



  - LSB 8bits indicate the partial region
  - Rest of bitstream bits remain unchanged

- Bitstream address relocation
  - Currently, this is done on the M3, which is slow
  - Hardware to handle bitstream relocation will be ready soon



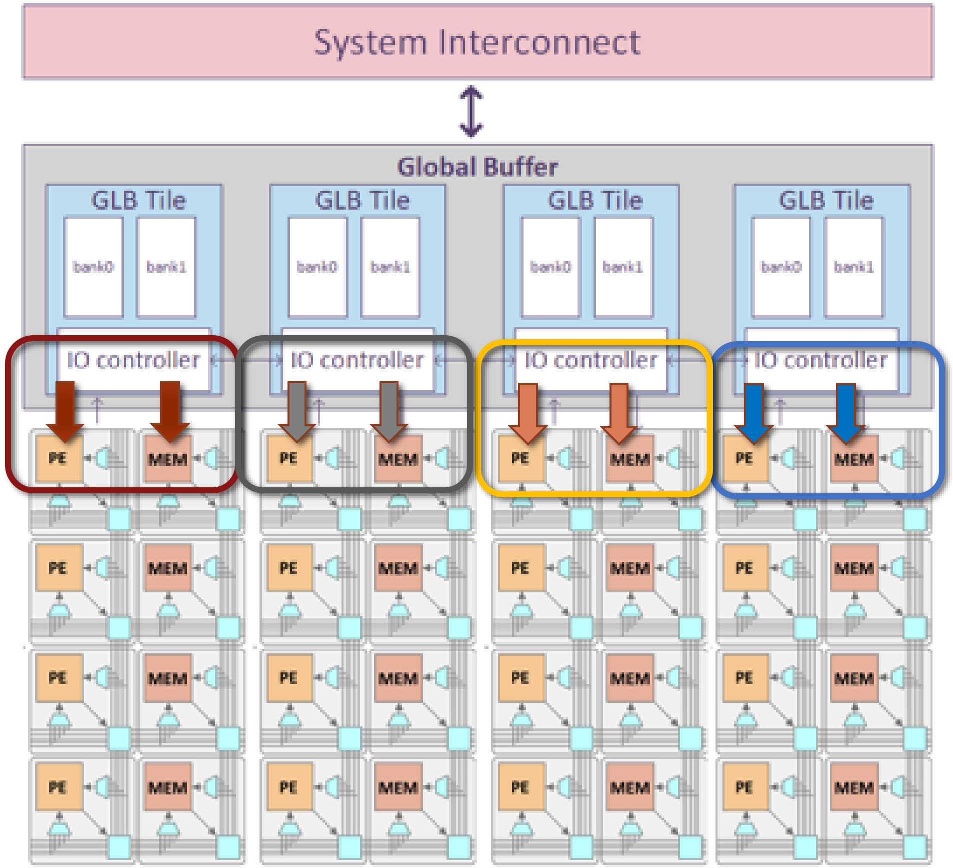
Relocatable

Hardware Support for Dynamic Partial Reconfiguration

- #2 Dynamic Partial Reconfiguration (DPR) is fast, parallel, independent

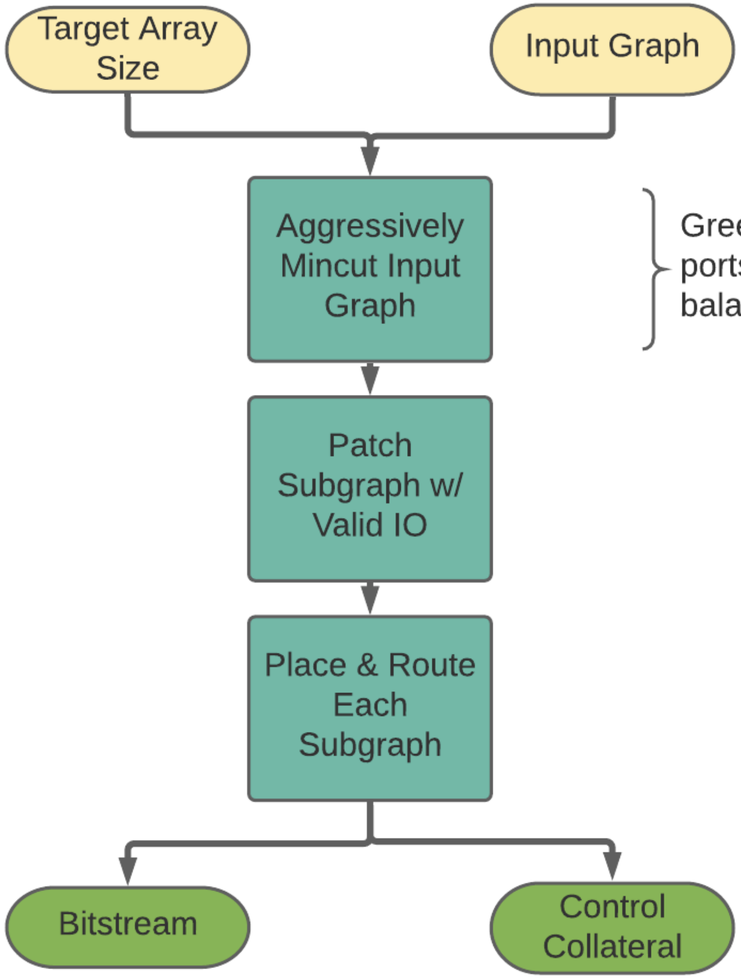
- Bitstream is stored in global buffer tile
  - Fast: Global buffer streams burst of bitstream
  - Parallel: Up to 16 controllers can work in parallel
  - Independent: Each controller is independent

- Configuration time (75,520 registers)
  - JTAG: 108.7ms
  - AXI4-Lite: 604us
  - DPR: 4.72us



Stanford University

Partitioning Algorithm



Target Array Size

Input Graph

Aggressively Mincut Input Graph

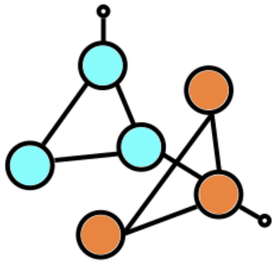
Patch Subgraph w/ Valid IO

Place & Route Each Subgraph

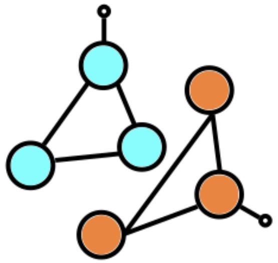
Bitstream

Control Collateral

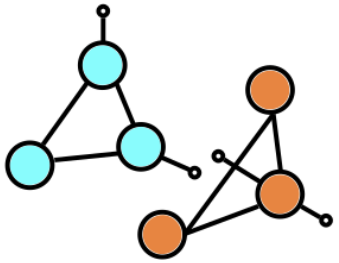
Greedy Approach: Use IO ports as cost function to balance area



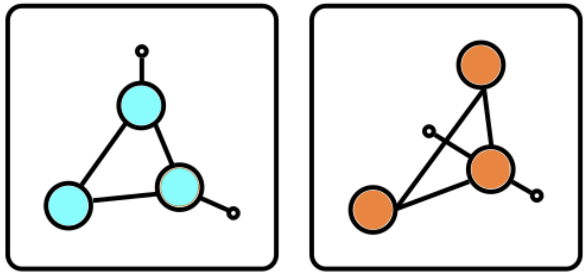
Input Graph



Graph Partition



Fix IO and Control Logic

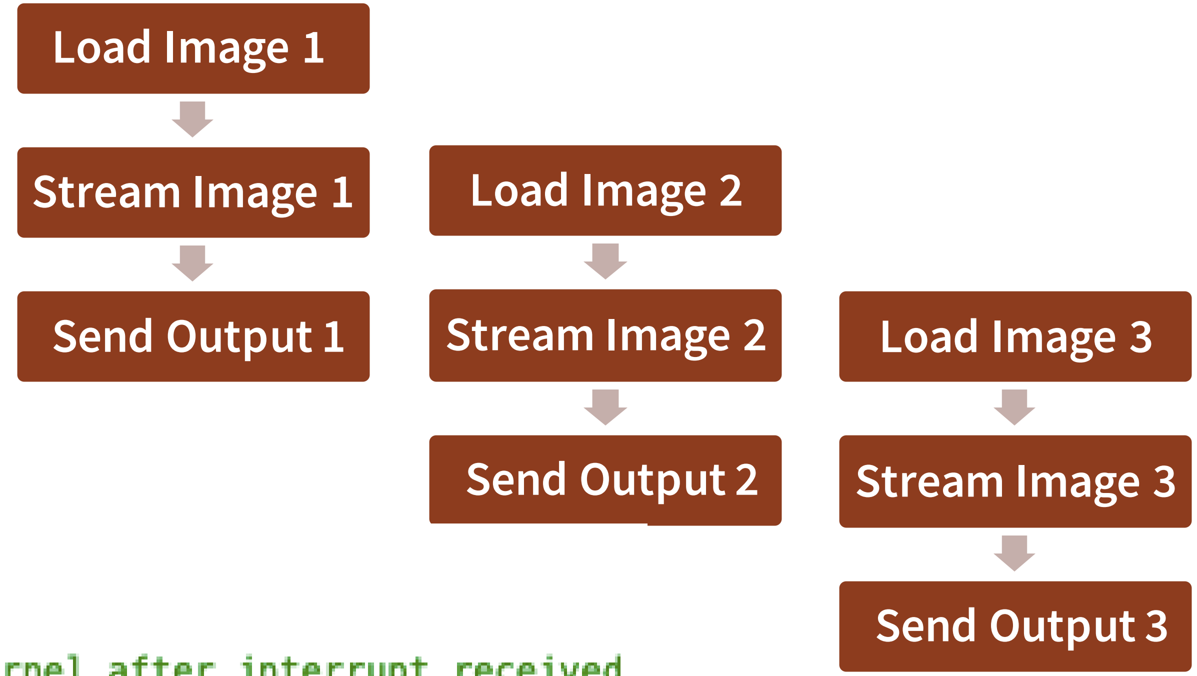


Place and Route in a Virtualized Fabric

Temporal and Spatial Virtualization

Temporal Scheduling

- The M3 and Global Buffer pairing allow us to efficiently schedule applications
- For example: a batch of images with the same kernel
- Cycles to put an image in GB ≈ cycles to stream an image to the fabric
- Can Send Output 1 and Load Image 3 at same time with 2 DMAs



Load Image 1

Stream Image 1

Send Output 1

Load Image 2

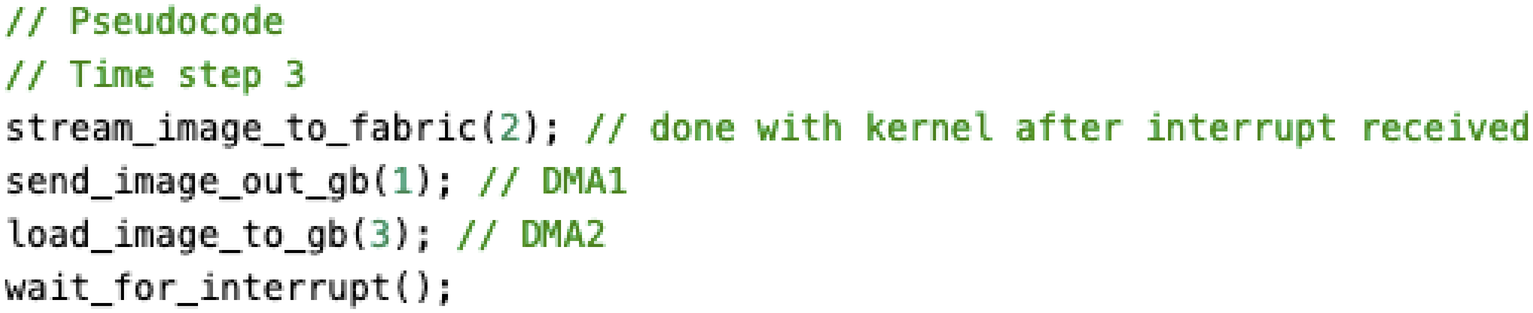
Stream Image 2

Send Output 2

Load Image 3

Stream Image 3

Send Output 3



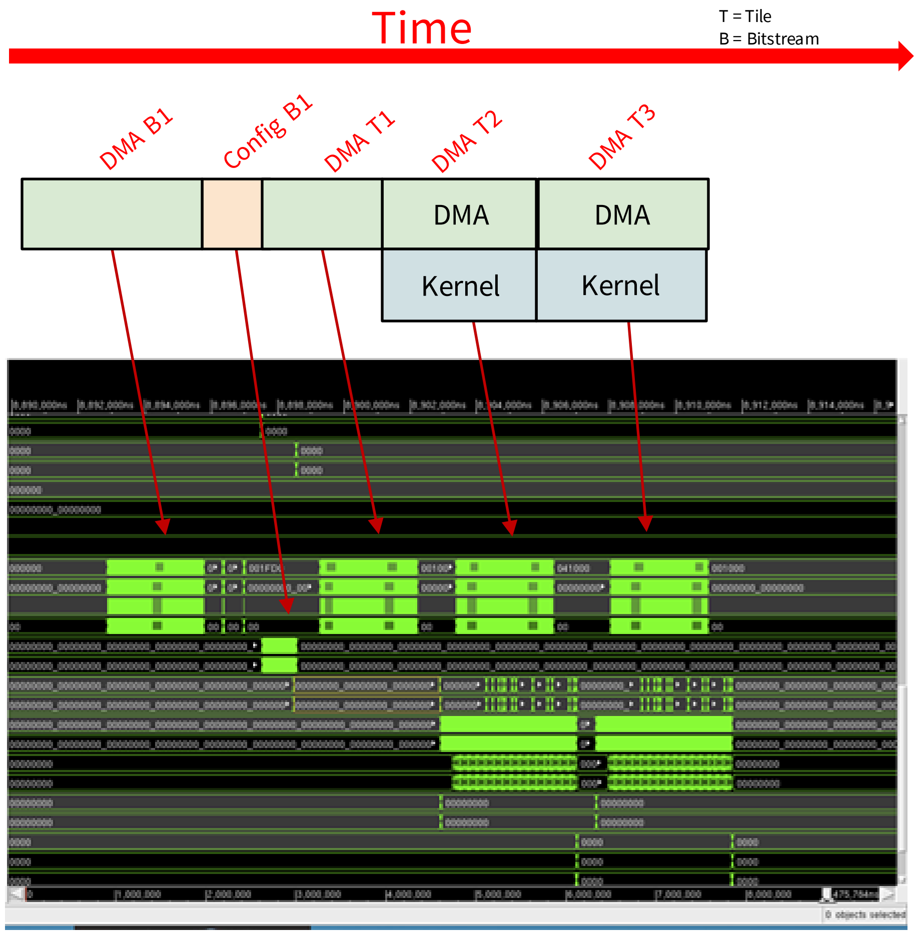
```
// Pseudocode
// Time step 3
stream_image_to_fabric(2); // done with kernel after interrupt received
send_image_out_gb(1); // DMA1
load_image_to_gb(3); // DMA2
wait_for_interrupt();
```

Temporal Scheduling Results

Scenario 1: Stream images into a single kernel (accelerator)

Result: **88% Utilization**

Metric	Description	Time
Programming Time	DMA bitstream and configure	6300 Cycles
Launch Time	Time between kernels	600 Cycles
Processing Time & Data Movement	Run kernel and DMA next tile	4096 Cycles

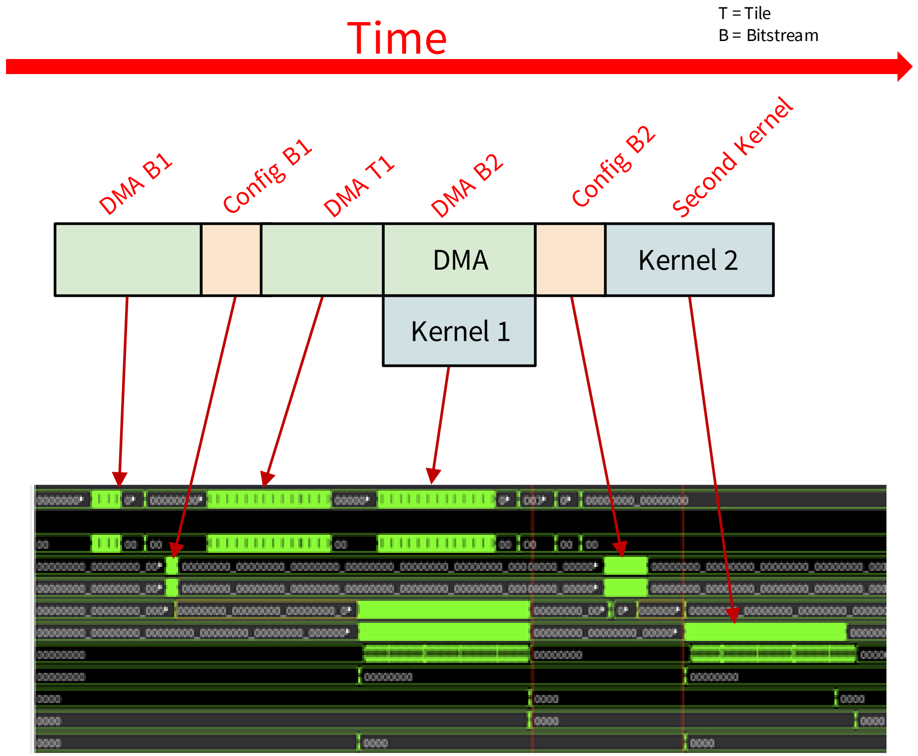


Temporal Scheduling Results Cont.

Scenario 2: Two different kernels back to back

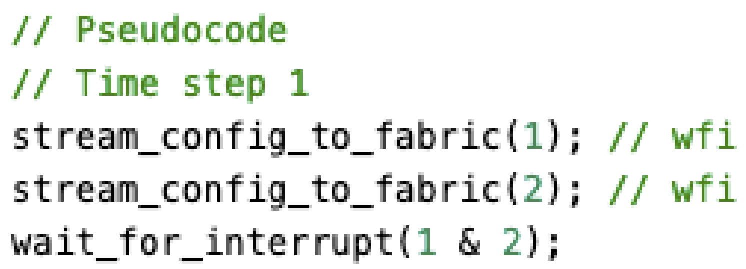
Result: **70% Utilization**

Metric	**Description**	**Time**
Programming Time	DMA bitstream and configure	6300 Cycles
Launch Time	Time between kernels	3500 Cycles
Processing Time & Data Movement	Run kernel and DMA next tile	4096 Cycles

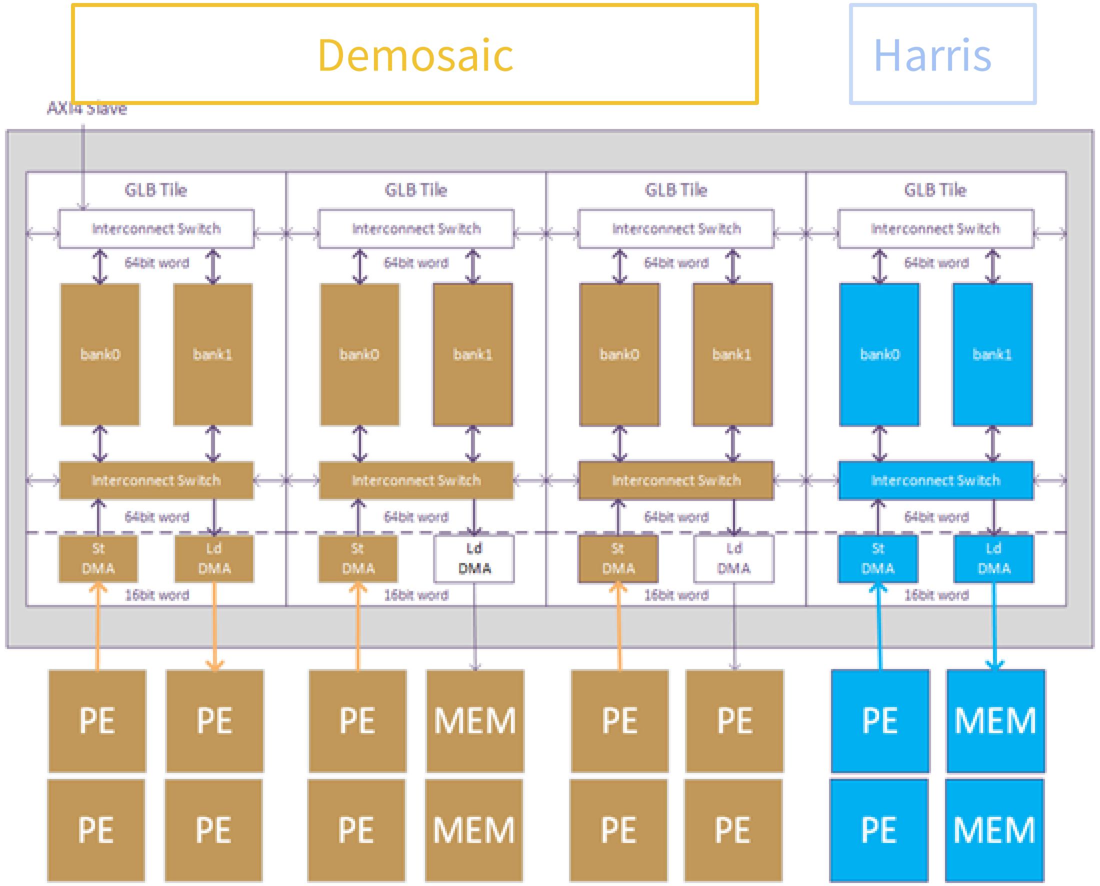


Spatial Scheduling

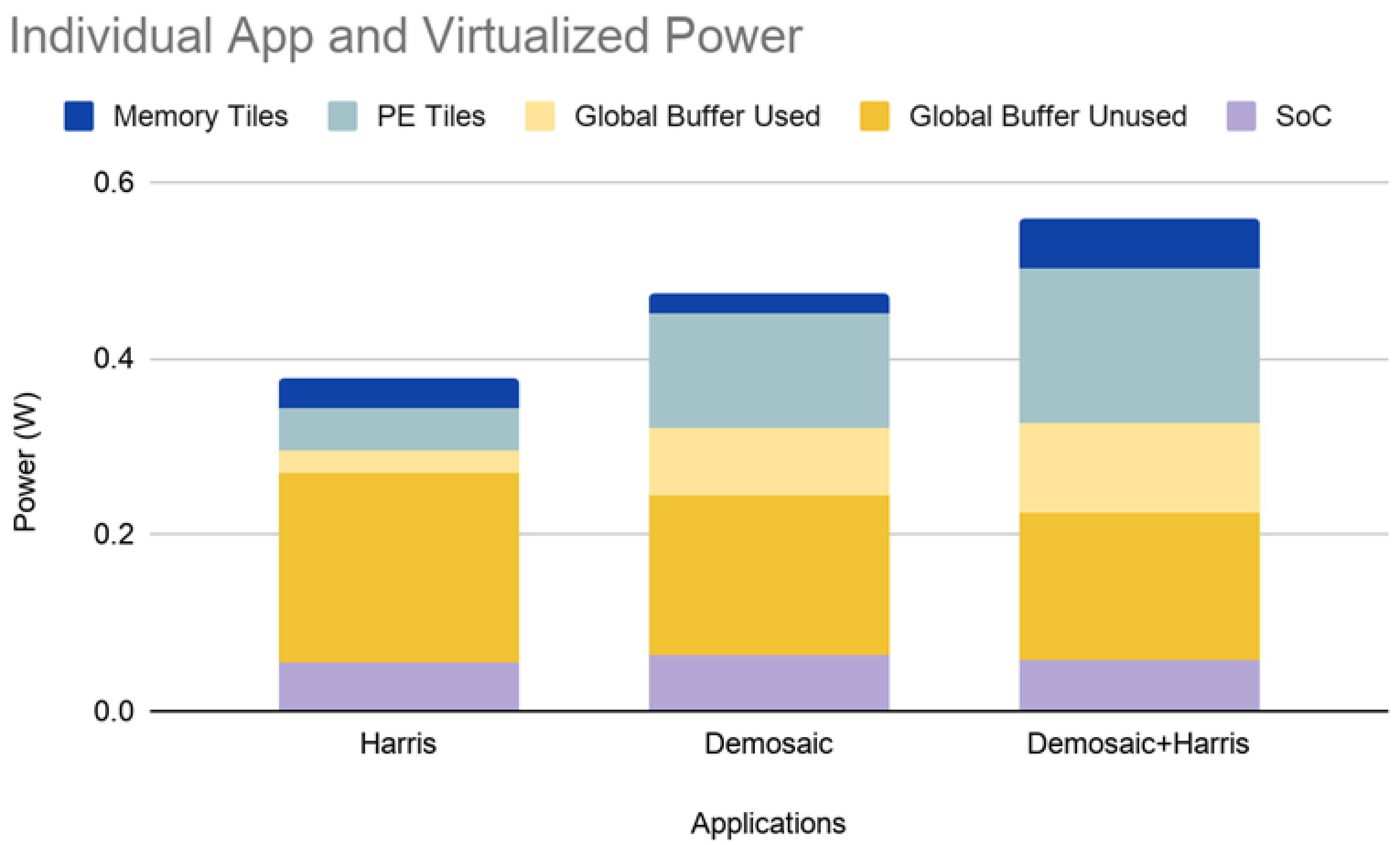
- We can place more than one kernel onto the fabric at once
- For example: Demosaic and Harris configured on the CGRA
- With 2 DMAs we can run both applications independently



```
// Pseudocode
// Time step 1
stream_config_to_fabric(1); // wfi
stream_config_to_fabric(2); // wfi
wait_for_interrupt(1 & 2);
```



Spatial Scheduling Results



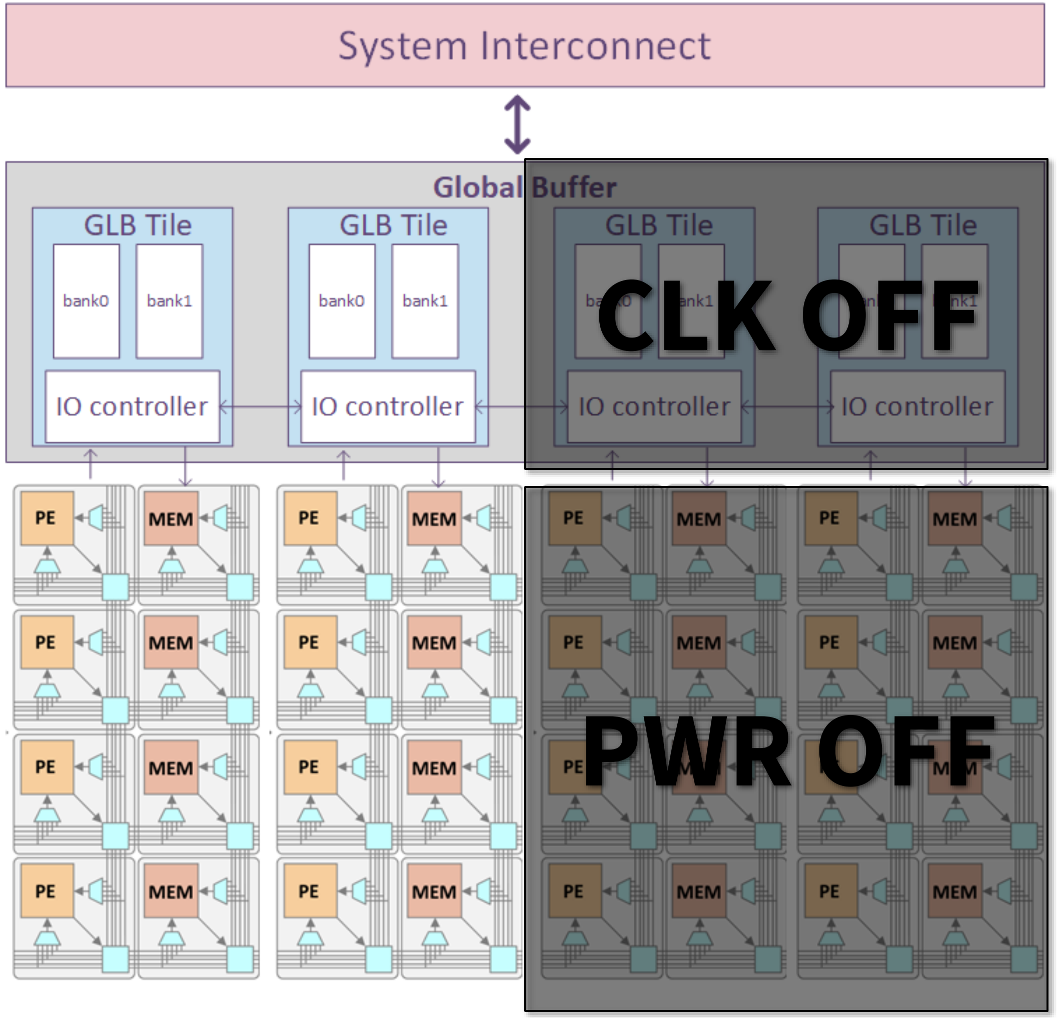
Characterization of Simultaneous Applications

Metric	Demosaic only (unused tiles powered down)	Harris only (unused tiles powered down)	Harris and Demosaic simultaneously (unused tiles powered down)
Image size	480x640	480x640	480x640*2 Images
Tile size	64x64 (80 Tiles)	64x64 (80 Tiles)	64x64 (80 Tiles*2 Images)
Utilization (#PEs, #MEMs, #I/O, #Tracks)	217, 10, 6 16bit-track 14.4% 1bit-track 1.4%	83, 5, 4 16bit-track 11.5% 1bit-track 2.0%	300, 15, 10 16bit-track 13.7% 1bit-track 1.6%
Latency (startup latency, processing latency)	12000 Cycles, 6900 Cycles	6300 Cycle, 6900 Cycles	18300 Cycles, 6900 Cycles
Throughput @750MHz	2000 images/second	2000 images/second	4000 images/second
Energy	2.84pJ/op	7.19pJ/op	2.55pJ/op

Ongoing Work

Clock Gate Global Buffer Tiles

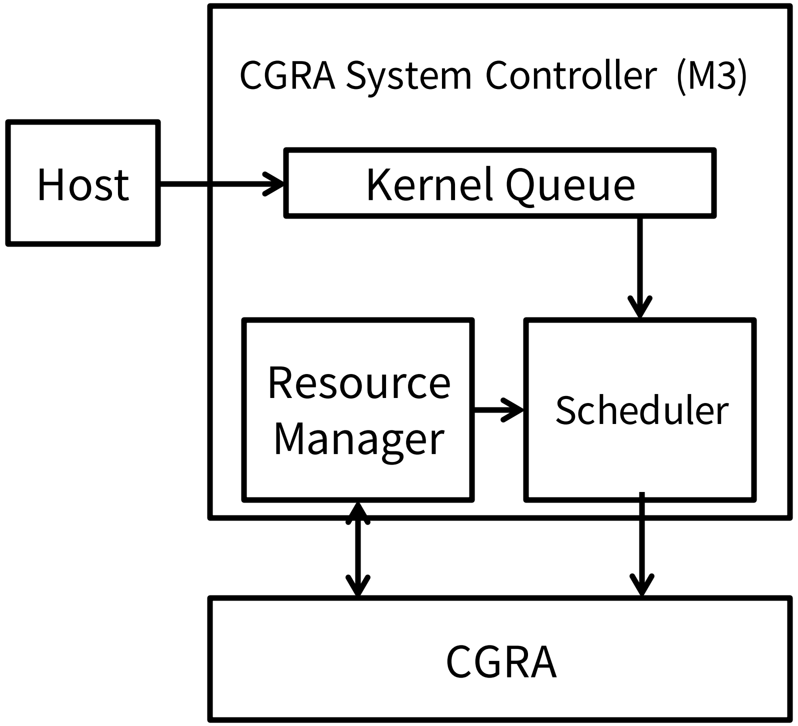
- Global buffer is a complex block with SRAMs, DMAs, and bitstream control logic
- Harris corner detection total power: 377mW
  - Global buffer: 273mW
    - Used global buffer tiles: 41mW
    - Unused global buffer tiles: ***232mW***
  - Currently, Only SRAM macros are gated, but the idle global buffer tiles still consume a lot of power
- Future clock gating
  - Power is mostly consumed by switching, so clock gating will save lots of power
  - Configuration register controls clock-gating for each global buffer tile



Dynamic scheduling

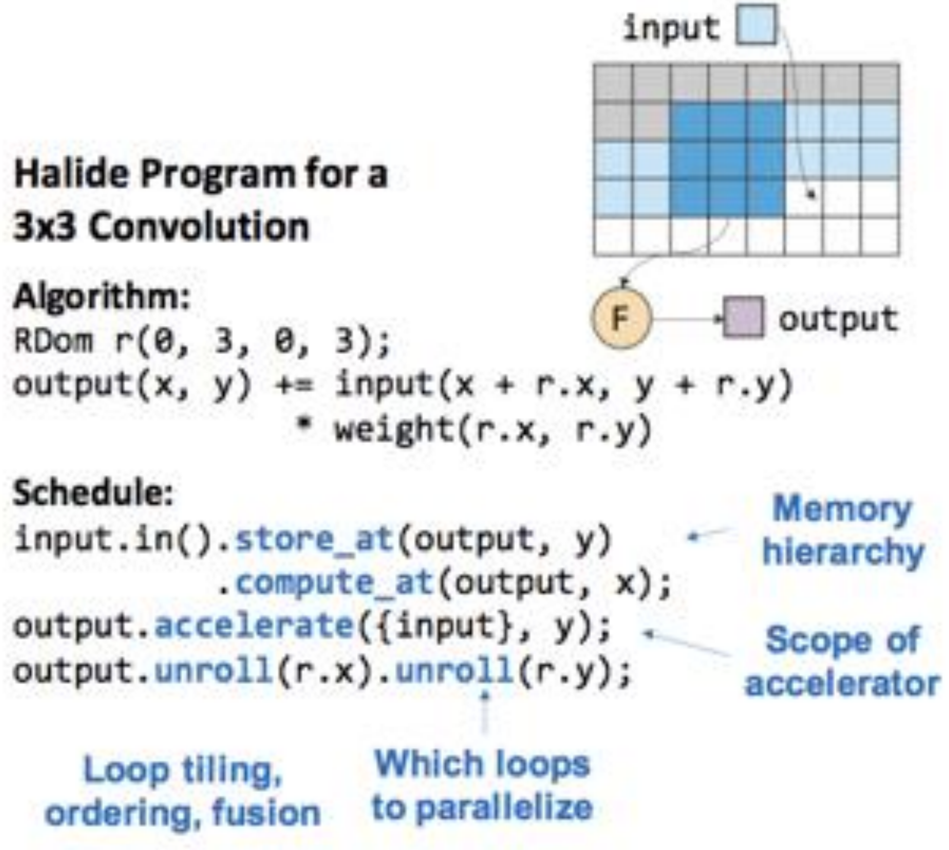
- CGRA System run-time controller on M3
  - Interface between host and CGRA
  - Controls kernels

- CGRA System controller components
  - Queue
    - Outstanding kernels are waiting here
  - Resource manager
    - Checks available hardware resources
  - Scheduler
    - Schedules kernels to hardware
    - Simple first-in-first-out scheduling
    - Complicated out-of-order scheduling



Autoscheduling

- Leverage high degree of determinism on CGRA – utilize per operation and per memory access costs to create a configurable cost function based on performance, energy, and area
- Using an outer loop over different schedules and a cost function, we can determine the optimal schedule
- Can divide scheduling decision types **inter-** and **intra-** kernel



Halide Program for a 3x3 Convolution

Algorithm:

```
RDom r(0, 3, 0, 3);
output(x, y) += input(x + r.x, y + r.y)
              * weight(r.x, r.y)
```

Schedule:

```
input.in().store_at(output, y)
          .compute_at(output, x);
output.accelerate({input}, y);
output.unroll(r.x).unroll(r.y);
```

Virtualization Summary and Next Steps

Summary

- Efficient overlap of data movement and acceleration
- Simultaneous Applications increase accelerator utilization

Next Steps

- Clock gate Global Buffer Unused Tiles
- Given the variety of scheduling decisions due to the complexity of our system (DMA, GLB, CGRA, etc.)
  - Tiling at DRAM level
  - Tiling at Global Buffer level
  - Overlapping of Data Movement, Kernels, Configuration etc.
- Dynamic- and Auto- Scheduling