香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Multi-CGRA Computation Abstraction & Architecture Support

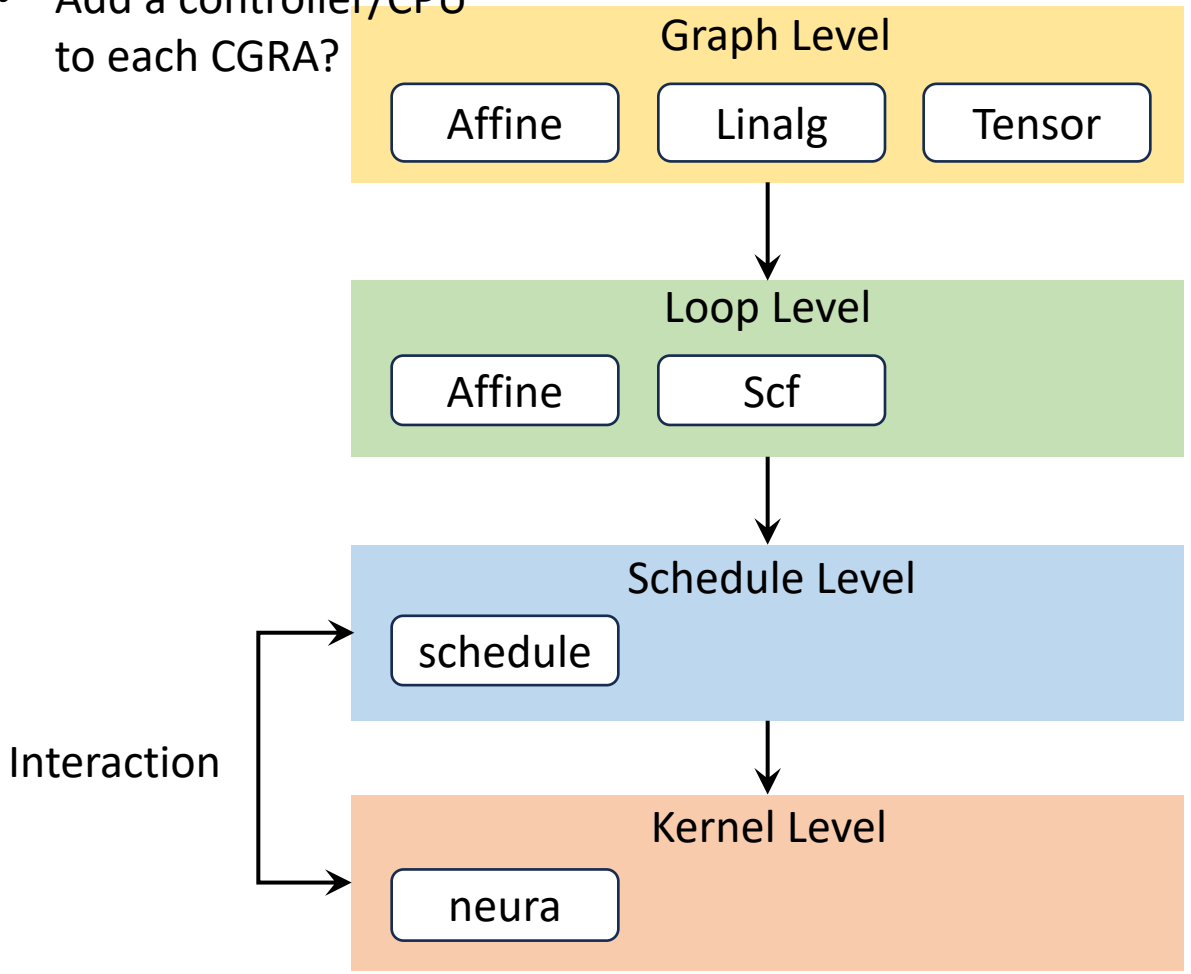
Higher Level Optimization Integration

- Function Level **[Graph Level]**
  - Define the function content/granularity
  - [One loop as a function, N loop as a function]
  - [Change the function as a neura operation, a region to wrap the kernel for dataflow transformations]
  - [A programming model to assign a warp for 1 or 2 CGRA, we need to correctly define, need to consider the yield operation]
- Linalg Level/ONNX Level **[Loop Level]**
  - Fuse/tile/…
  - [CPU-CGRA interface latency sensitive]
  - Polyhedral transform
  - Affine transform
  - [Polyhedral model investigation]
- Acceleration on CGRAs **[Kernel Level]**

Higher Level Optimization Integration

Design Choice:

- Add a controller/CPU to each CGRA?



Goal: Min Data Transfer & Max Parallelism

- Global Operator Fusion
- Layout Optimization

Goal: Granularity Search & Wrapping

- Granularity search
- Loop Tiling/Unroll/Interchange/Permutation

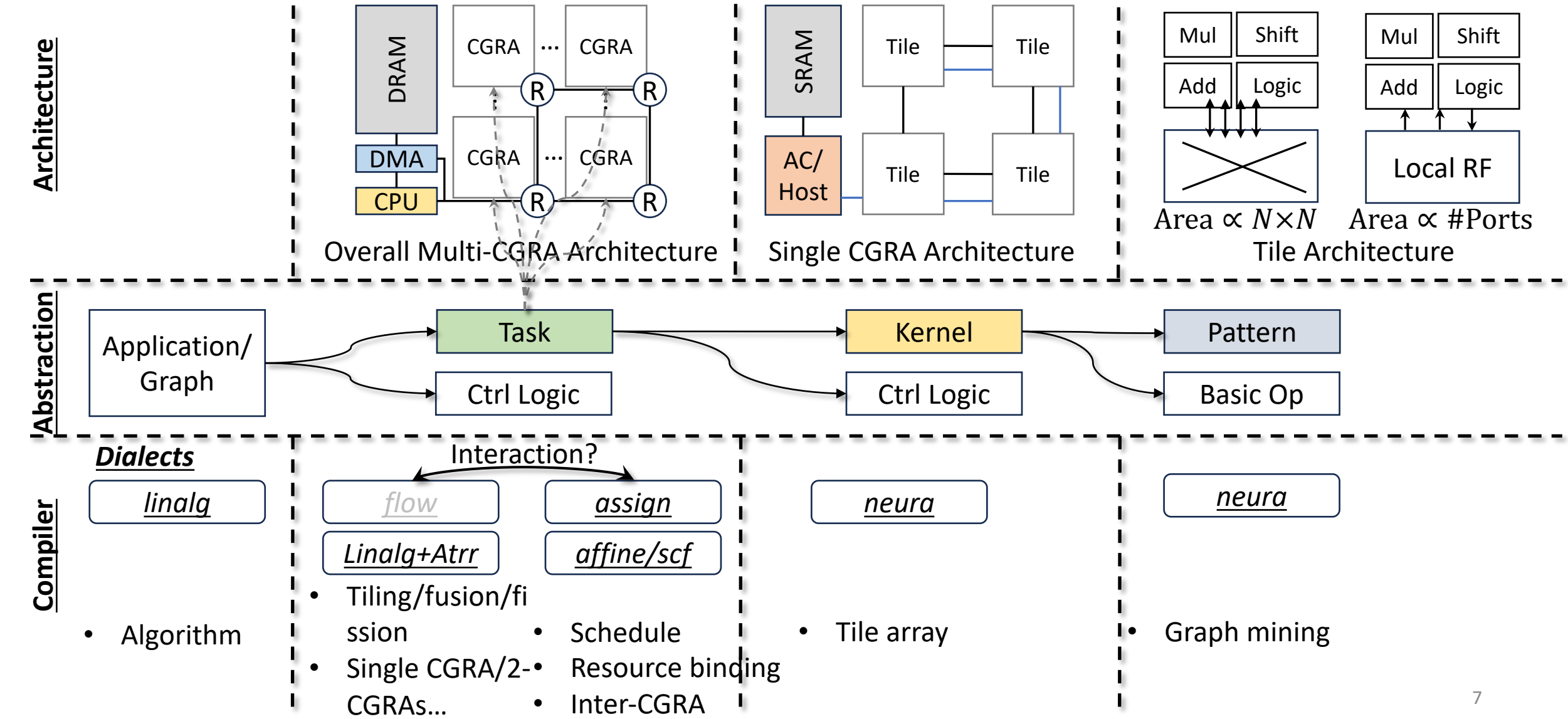
Goal: Multi-CGRA Task Mapping

- Inter-task dependency graph construction
- Temporal task switching for resource reuse

Goal: Single-CGRA DFG Mapping & Fusion

- Subgraph mining & fine-grained Op Fusion
- Heuristic Mapping & Custom Mapping

Scale CGRA Architecture



AC/Host in Existing Arch

Plasticine

- Control block: reconfigurable combinational logic + programmable up-down counters for state machines
- Support 3 control protocols:
  - Sequential execution:
  - Coarse-grained pipeline:
  - Streaming:
- Tree-based control management:
  - Controllers w/o children controllers are mapped to PCUs' control blocks
  - Others are mapped to control blocks in switches for synchronization
- **Drawbacks:** Low utilization (rigid lane numbers, rigid pattern-based programming)
  - The length of each lane is determined, need to partition long data path across multiple PCU for computing
  - Long latency when cross multi-PCU, because they need NoC for data transfer
  - Mismatch between the lane length and the data path
  - Inefficiency on multi-independent small loops (each corresponds to a PCU)
  - Inefficiency when facing imperfect-nested loops (use a PCU for the computation between loops)

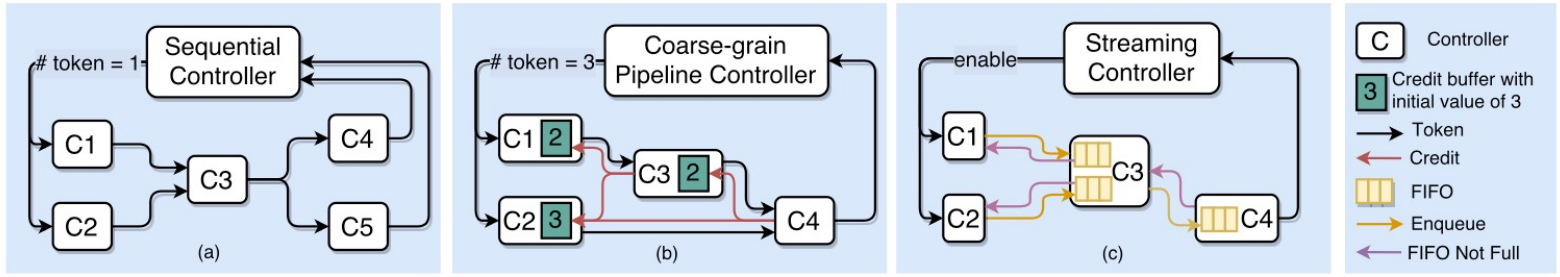


Figure 6: Sequential, coarse-grained pipelining, and streaming control schemes.

AC/Host in Existing Arch

Amber

- Use the iteration domain counter in each PE tile & MEM tile to determine the exit condition
- Can map multiple kernels on the big array (spatially & temporally)
- Use predicate for condition execution
- **Drawbacks:** Lack of flexibility
  - Extreme complexity in compiler/scheduler design (No hand shaking scheme, hard for general kernels)
  - Inserting bubbles for inter-GLB tile latency: like introducing global stall, degrading performance
  - No multi-kernel streaming support

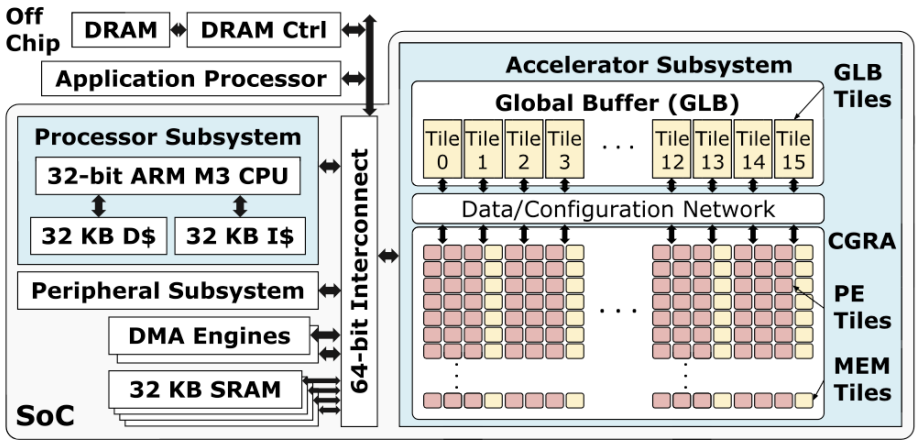


Fig. 1. Amber SoC architecture. The accelerator consists of a GLB and a CGRA. The processor subsystem controls application execution.

AC/Host in Existing Arch

Softbrain

- Target workloads' features:
  - High computational intensity with **long phases** (one data, long data path)
  - Small instruction footprints with simple ctrl flow (w/o nested ctrl handling)
  - Straightforward memory access and reuse patterns
- Provide affine/stream memory access to reduce the address calculation/generation
- 3 Stream engine enables stream data access (No need for address calculation)
  - Scratchpad, Memory, and Recurrence Stream engines
- **Drawbacks:**
  - When facing high-dimention nested loops, how to stream them (even with different nested levels)
  - Need Control Core and Stream Engine for control flow handling
  - No Multi-kernel support (both independent & streaming)

AC/Host in Existing Arch

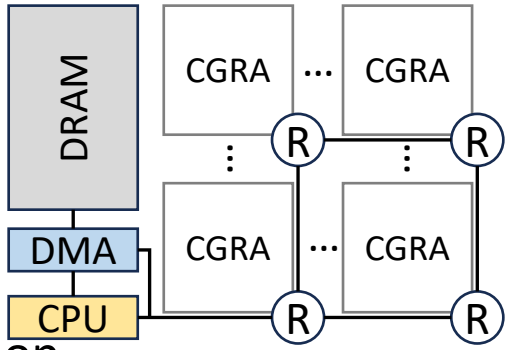
Fifer

- Target workloads' features:
  - Irregular workloads, like bfs
- Partition the app as multiple stages
- Time-multiplexing the CGRA to execute these multiple stages
- **Drawbacks:**
  - Not clear yet

Comp. Abstraction & Execution Model

Target Architecture

- Multi-CGRA Systems



Features for High Performance, Utilization & Generality

- Multi kernel on Single CGRA
- Irregular control workloads: Two parallel loops nested inside an imperfect loop
- Regular control workloads: like ResNet, Bert, etc.
- Combine multi-CGRAs as big single CGRA

The workload mapping abstraction

- App: the input is function for now
- Task: how to define this? MLIR Graph Region for streaming, MLIR SSACFG Region for general
- Kernel: Wrapped in a neura.kernel's region in MLIR

Comp. Abstraction & Execution Model

The workload description abstraction

Level 1: Application Level (Inter-CGRA Flow)

- Structure: Task Graph (Node-Task, Edge-Tensor/Dependency)
- Execution Model: Independent, Sequential, Coarse-Pipeline, Stream

Level 2: Task Level (Host-Device Interface)

- Structure:
  - Host: Stream & Event descriptions, Start/Wait, Data Communication (DRAM-SRAM, Inter-CGRA)
  - Tile Array: compute configurations
- Execution Model: Host sends event signals (e.g., config, load, compute, etc,.) [这个host的trigge可以是streaming trigger，也可以是config trigger-就是配置好循环条件], Tile array triggers kernel execution

Level 3: Kernel Level (Inter-Tile Execution)

- Structure: DFG
- Execution Model: Spatial-Temporal/Spatial

Level 4: Pattern Level (Intra-Tile Execution)

Comp. Abstraction & Execution Model

Level 1: Application Level (Inter-CGRA Flow)

- Structure: Task Graph (Node-Task, Edge-Tensor/Dependency)
- Execution Model: Independent, Sequential, Coarse-Pipeline, Stream
- Elastic Task Wrapping:
  - Hierarchical Control
  - Fused kernel (with spatial partition)
  - Fused kernel (with fused DFG, like FexMo)
  - One task on multi-CGRA as a single-CGRA
- Task Node: Computation & Memory
- Task Edge: Plasticine的control 方法太过静态，它的control 是基于固定的pattern的，没有hybrid的场景
  - Data Edge
    - Consistency Edge:：当前task结束，后面才能开始
    - Streaming Edge：当前没有结束，后面也可以开始
  - Control Edge
    - Trigger Edge: 用于有控制依赖的task node之间

Distinguish from Plasticine & SARA

Distinguish of Plasticine from our solution

- 他的control flow的表现方式时DHDL，hierarchical pipelining，也并不是他们自己提出来的
- 如何将不同的pipeline task给最好的balance 起来，其实是一个bottleneck，plasticine并没有考虑这个
- Plasticine在不同的control block之间没有采用handshaking，而是采用的Compiler Managed Memory Consistency
- Plasticine用的是hierarchical synchronization scheme，在deeply-nested control hierachies时引入pipeline bubbles。不支持branch
- SARA将控制层级拍扁为了一个扁平的CMMC图中，虽然效率高，但是缺少了层级感，这使得它没有办法完成层级化的优化（比如loop tiling, fusion），SARA的computation abstraction丢失了很多高层语义
- 注意，plasticine的streaming mode只是为了将多个PCU连起来用于加速一个大的loop pattern，而我们可以用hard的线连起来，不用走FIFO，本质上是提供了一个更大的physical的fabric，性能更优
- SARA通过将循环控制的相关东西map到一个单独的VCU上以支持dynamic loop bounds（我们是通过predication computation）

Distinguish from TileFlow & SET

Distinguish of TileFlow & SET from our solution

- 这两个工作是用树形结构来表示调度结果的表示，这天生会导致树的结构中不同的node表示的信息隶属于不同的结构，这其实无法突出node与Spatial architecture的对应关系

Arch Design

Controller Contribution

Adaptable CGRA-Binding Contribution

- Combine multiple-CGRA into a single big-CGRA

Discussion

Motivation

- For Software-defined general purpose spatial computing
- Bridge the efficiency gap between **rigid spatial architectures** and both regular & irregular **complex-nested computational workloads** in general-purpose spatial computing.

Contribution

- Towards Elastic Granularity Acceleration on Multi-CGRA
  - Elastic Granularity from task wrapping side (software)
  - Elastic Granularity from multi-CGRA binding side (hardware)
- Unified Hierarchical Abstraction for holistic workloads optimization & mapping
  - The computation abstraction (software)
  - Abstraction Distinguish with Plasticine & SARA：多种control edge的并存
  - The affine controller support