

Contents

1	TVM	1
1.1	Goals	1
1.2	Motivations	1
1.3	Contributions	1

1 TVM

1.1 Goals

- Automatically generate deployable codes that are performance-competitive with state-of-art vendor-specific libraries.
- Through automatically generate codes, address the problem that handcrafting operator kernels for the massive space of back-end specific operators and operator combinations.

1.2 Motivations

- Current deep learning framework rely on a computation graph representation.
- Challenges and Goals
 1. High-level dataflow rewriting.
 - kernel fusion
 - data layout optimization
 2. Memory reuse across threads.
 - cooperation among threads on shared memory
 3. Tensorized computation intrinsics.
 4. Latency Hiding.

1.3 Contributions

- TVM separate the algorithm description, schedule, and hardware interface .
- TVM presents two stage optimization
 1. computation graph level optimization
 1. operator fusion
 2. data layerout transformation
 2. **tensor level optimization**
 1. **Tensor Expression Language** takes cues from Halide.
 - describe both the users' intended compute description and the abstractions that the hardware exposes.
 - commutative reduction operator
 - high-order scan operator : *to form recurrent computation*
 2. introduce **schedule primitives** to decouple computation description and schedule.
 - adopt useful primitives from Halide and *introduce new ones (?)* to tackle the challenges introduced by GPU and specialized hardware accelerators.
 3. Nested parallelism with cooperation
 - traditional solution for parallelism: **shared-nothing nested parallelism (fork-join parallelism)**
 - introduce the concept **memory scope** so that a stage canbe marked as shared.
 - * the shared task needs to compute the dependencies of all the working threads.
 - * use persist threads
 - * memory synchronization barriers need to be properly inserted.
 4. Tensorization: (1) inputs are *ndarrays*; (2) dicdate different data layerout.
 1. challenges:
 1. DL workloads have high arithmetic intensity.
 2. cannot resort to a fixed set of primitives.
 2. separate the hardware interface from the schedule
 - **declare the behavior of each new hardware intrinsic.**

3. introduce *a tensorize schedule primitive*
 - replace a unit of computation with the corresponding tensor intrinsics.
5. Latency hiding: decoupled-access/execute philosophy
 1. assume the hardware pipeline consists of memory and compute stages that can execute concurrently.
 2. use FIFO queues to implement explicit dependency tracking.
 3. introduce *virtual thread schedule primitive*: programming at low-level is difficult and painstaking.