Compiler Final

Dinh Duy Kha

May 27, 2021

Contents

1 Parallel Architecture

1.1 Shared memory (SMP)

- **Symetric multi-processors (SMP)**
- All CPUs use single address space
- Threads, OpenMP
- Communication through load/store (`implicit`)
- Synchronization using locks and barriers
- Requires cache coherence

1.1.1 Cache Coherency

1. Snooping
    - Cache controller snoop on the bus
    - Multiple read-only copies on CPUs
    - Single modified copy on a CPU
        - *Invalidate* other copies when a shared cacheline is update
    - Use 3 states (MSI):
        - Modified: writable
        - Shared: read-only
        - Invalid: no data

    (a) Write Invalidate
        - Invalidate copies on update
        - On cache miss:

- Write-through: up-to-date
- Write-back: Force of most recent copy to update memory

(b) Write Update

- Update copies on update
- On cache miss:
  - Write-through: up-to-date
  - Write-back: Force one of the sharer update memory

(c) MESI Protocol

- Add Exclusive: line exclusive to the CPU
- S -> M : invalidate traffic
- E -> M : no invalidate traffic

2. Directory

1.2 Distributed memory

- Each CPU access partial data
- Remaining through communication
- NUMA, Cluster
- MPI
- Communication using message passing (`explicit`)
- No cache coherence

1.3 Multithreading architectures

1.3.1 Superscalar

1.3.2 Multithreading

1.3.3 Simultaneous Multithreading (SMT)

2 Synchronization

2.1 Mutual exclusion

2.2 Event synchronization

2.2.1 Point-to-point

2.2.2 Barrier

1. Centralized
2. Combining tree

2.3 Lock

2.3.1 Test & test & set

2.3.2 LL-SC

2.3.3 Ticket Lock

2.3.4 Array-based

2.4 Lock-free algorithms

2.4.1 List

2.4.2 Circular queue

3 Dependence Analysis

3.1 Instruction scheduling / reordering

3.2 Parallelization

4 Loop Transformations

4.1 1D loop

4.2 2D loop

4.3 3D loop

4.4 Loop Transformations

4.4.1 Permutation

4.4.2 Reversal

4.4.3 Skew

4.4.4 Tiling (Blocking)

4.4.5 Fusion

4.4.6 Distribution (fission)

4.4.7 Index set splitting (loop peeling)

4.4.8 Unrolling

4.5 Unimodular Transformation

5 DNN Compilers

5.1 TVM

- Ingest models from Pytorch, Tensorflow, ONNX, MxNet
- Target archtecture x86, ARM, GPUs, MIPS, RISC-V
- Optimized for target platform

5.1.1 Operator Optimization

1. Halide progamming model:
    - Functional definition: What the function do
    - Schedule definitions: How should the function do it
2. TVM Scheduling Primitive
    - Primitives for optimizations

5.1.2 Automated optimization search: AutoTVM

- Use ML to learn the best code to be generated

5.1.3 Graph-level Optimization: Use Relay IR

1. Operator fusion
    - Fuse operators together to minimize

5.2 XLA: Accelerated Linear Algebra compiler

- Take tensorflow graphs, split out optimzied assembly
- TF Graph -> XLA Graph -> LLVM IR -> ASM code

5.2.1 TF2XLA

- Old TF: Look for optimized kernel in runtime library
- XLA: Look for tf2xla kernel to plug into TF graph

5.2.2 JIT

- Compile TF clusters of nodes into XLA graph
- Execute the whole cluster

5.2.3 Ahead-of-time compilation

- Use **Graph Compiler**
- Compile the entire XLA graph