CS 598CM: ML for Compilers and Architecture

Instructor: Charith Mendis
Brief Announcements

• **Paper Selections:** Due on today, presentations start Sept 13th

• **Paper Reviews:**
  • Due for each paper 2 days prior to the discussion date
  • Instructions will emailed before class next week

• **Piazza:** Let’s use Piazza to discuss papers and more ML related work!
Recap

- Anatomy of a compiler optimization pass
- Domain Specific Languages
Lecture 4: DSLs + ML in Architecture
Deep Learning Stack

Workload

Language

Compiler

Hardware
Computational Graphs

- A data-flow graph with tensor operations
Example: Inception

Can grow to 1000s of nodes

Szegedy et. al “Going Deeper with Convolutions”
Optimization: Operator Fusion

Zhou et. al “Transferable Graph Optimizers for ML Compilers”
HW: Tensor Processing Unit

Jouppi et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit”
HW: Simplified Version

- Scratch pads are not caches
- Software Programmable
- Uses Direct Memory Access transfers

**Size:** scratch pad $<<<$ HBM

**Latency:** scratch pad $<<<$ HBM

Processing Element ↔ Scratch pad Memory ↔ HBM (High Bandwidth Memory)
• K1 executes
• Writes back results to HBM
• K2 executes
• Writes back results to HBM
• Fused Operator Executes
• Writes back results to HBM
• Intermediates can be stored in HBM or Scratchpad
Operator Fusion

Which is faster? Usually 2
Operator Fusion

Written to HBM
Operator Fusion

Fusion is not always profitable!
Typical type II Optimization
Operator Fusion

HLO IR

Sample HLO ops
- Elementwise math
  - Add, Tanh, Map
- Spezialized math for neural nets
  - Dot, Convolution, Reduce
- Re-organize data
  - Reshape, Broadcast, Concat, Tuple
- Control flow
  - While, Call, CustomCall
- Data transfer
  - Parameter, Constant

Sample data types
- Primitive types
  - PRED
  - F16
  - F32
- Composite types
  - array: F32[2,3], F16[]
  - tuple: TUPLE(F32[16], F16)
Graph Simplifications

- expressed as computational graph rewrites

(a) Associativity of matrix multiplication.

(b) Fusing two matrix multiplications using concatenation and split.

TASO [SOSP'19]
Operator Placement

Figure 4. RL-based placement of Neural MT graph. Top: encoder, Bottom: decoder. Devices are denoted by colors, where the transparent color represents an operation on a CPU and each other unique color represents a different GPU. This placement achieves an improvement of 19.3% in running time compared to the fine-tuned expert-designed placement.

11/03 Reading
Mirhoseini et. al “Device Placement Optimization with Reinforcement Learning”
Graph Computing

- Models computations on graph structured data
- Vertex centric or Edge centric
- Plenty of domain specific optimizations
  - Push / Pull optimizations
  - Vertex Reordering
  - Graph Segmentation etc…
Optimizations on Graphs

Better than Frequency-based Reordering

#hits: 4
#misses: 3

Cache

#hits: 5
#misses: 2

Graph Processing
Auto-tuning for graphs

• We have a separate lecture on auto-tuning

• Challenging
  • Highly input sensitive (e.g. power-law graphs vs road graphs)
  • Dependent on the graph algorithm (e.g. Page rank vs BFS)
  • Dependent on hardware (e.g. GPUs vs CPUs)

• 9/29: Meng et. al “A pattern based algorithmic autotunes for graph processing on GPUs”
ML in Architecture
Memory Subsystem

• Usually CPU arithmetic capacity is far greater than the memory bandwidth or latency

• We can optimize memory performance by exploiting
  • Spatial Locality
  • Temporal Locality

• This gave rise to caches and cache hierarchies
Caches

Size: Registers < L1 cache < L2 cache < L3 cache < DRAM
Latency: Registers < L1 cache < L2 cache < L3 cache < DRAM
Caches

• Hold a portion of the data from memory for faster access
• The space is caches is limited, so determining what data it holds is important
• We want to maximize cache-hit rate
• Cache replacement policies are important in determining the cache-hit rate
• 12/01 - we are going to discuss learned replacement policies
ML in Architecture: good idea?

- Depends

- Assume you are including a Neural Network (NN) in HW
  - Implementing NN in takes area
  - Adds execution latency / clock cycles get longer
  - May be a too heavy of a hammer

- Designing new hardware: Great Choice!

- We are reading papers on 11/15 and 11/17 about Design Space Exploration
Branch Prediction

- Determines which instructions to fetch on a conditional branch
- If wrong instructions are fetched and executed entire processor pipeline should be flushed — Costly!
- Early work on using ML for branch prediction, Jimenez and Lin “Dynamic Branch Prediction with Perceptrons”

HPCA 2019 Test of Time award
Dynamic Branch Prediction with Perceptrons

- Why perceptrons? Can be efficiently implemented in hardware

\[ y = w_0 + \sum_{i=1}^{n} x_i w_i. \]

('PERSONAL TECH')

'Neural network' spotted deep inside Samsung's Galaxy S7 silicon brain
Secrets of Exynos M1 cores spilled

https://www.theregister.com/2016/08/22/samsung_m1_core/
Dynamic Branch Prediction with Perceptrons

\[ y = w_0 + \sum_{i=1}^{n} x_i w_i. \]

if \( \text{sign}(y_{out}) \neq t \) or \( |y_{out}| \leq \theta \) then

\[
\begin{align*}
&\text{for } i := 0 \text{ to } n \text{ do} \\
&w_i := w_i + tx_i \\
&\text{end for}
\end{align*}
\]

end if
Dynamic Branch Prediction with Perceptrons

If the predictor is right

No training needed!

If the predictor is wrong or within threshold

\[
\text{if } \text{sign}(y_{out}) \neq t \text{ or } |y_{out}| \leq \theta \text{ then}
\]

for \( i := 0 \) to \( n \) do

\( w_i := w_i + tx_i \)

end for

end if

\( W_i \uparrow \) When \( t \) and \( x_i \) same sign

\( W_i \downarrow \) When \( t \) and \( x_i \) opposite signs
Any Questions?