Algorithms and Systems for Scheduling Structured Programs

Grace Dinh (gnd@berkeley.edu)
ADEPT End-of-Lab Talk

09 December 2021
Background

End of Moore’s law ⇒ Accelerators increasingly ubiquitous in both edge and datacenter

Accelerators expanding beyond ML for performance-critical applications, due to plateauing performance from general-purpose chips and falling design costs for specialized chips

- Protobuf accelerator (Sagar’s talk earlier today)
- Cryptocurrency mining: “ASICs are the standard technology found in every large-scale facility”¹
- Google VCU: specialized hardware for video encoding - “20-33x improved efficiency over our prior well-tuned non-accelerated baseline”²
- F1: hardware accelerators for fully homomorphic encryption³ - “accelerating full FHE computations by over 3-4 orders of magnitude”, making it actually practical for real world apps

Background

Software/algorithms play a huge role in performance on accelerators, and still lots of room for improvement.

“the actual performance of new ML-optimized hardware often lags far behind the promise... ML engineers may spend months hand-tuning their models to try to take advantage of what a new hardware target offers.” - from OctoML, which beat Apple’s CoreML performance on Apple M1 by 50% under two months after its release⁴

Dealing with Generality

Many algorithms (in multiple sizes!) and many accelerators (also in multiple sizes - GEMMINI (next talk) programmatically generates accelerators).

Manual optimization too labor-intensive.

Tuning often expensive (esp. benchmarking performance on simulators - ~5min/run)

Our approach: develop model for performance, cast scheduling as numerical optimization, and solve. Often can get strong lower bounds in the model.
Don’t Communicate 😞

Algorithms have two costs (measured in time or energy). **Arithmetic** (FLOPS), and communication - moving data between

- levels of a memory hierarchy (serial case - accelerator scratchpad to cache, on/off chip memory)
- processors over a network (parallel case - also encapsulates on-chip communication, e.g. systolic array traffic).

Goal: minimize communication by rearranging program, without changing what it does

Above: latencies for flops (gamma) vs. communication over time
Below: energy consumption of NN. Arithmetic (ALU) cost is tiny.
HBL Tiling

for \((i_1, i_2, \ldots, i_k) \in S \subseteq \mathbb{Z}^k\):

Access array locations indexed by affine function of indices, eg \(\varphi_c\)

\((i_1, i_2, \ldots, i_k) = (i_1 + 2*i_3 - i_7)\)

Theorem [Christ et al.’13]: for any reordering, #words moved \(\geq \Omega(|S|/M^{e-1})\),
for some problem-dependent \(e\)

Theorem [Demmel-Rusciano ’16]: if loop bounds large enough, exists
optimal tiling algorithm (right, top) to attain this bound.

Theorem [\textbf{D.} - Demmel SPAA ’20]: For loops subscripts only dependent on
one index (\(\varphi\) projective, “looks like dense linear algebra”): lower bound
attainable by efficiently computable tile \textit{regardless of problem size} (i.e.
including small dimensions, e.g. “inner product” like computations)
For specific algorithms

For matmul: variable-aspect ratio tilings (Vivek’s preceding talk)

For convolutions:

- [Demmel-D. MDS ‘20]: tight, efficiently attainable asymptotic lower bound for convolutions.
  - Computer generated proof, automatic code generation of tiling attaining the lower bound for all cases
  - Implication: well-tiled direct conv more communication-efficient than im2col, regardless of optimizations applied to latter.

- [CDDHH, under submission] - tighten communication bounds including constant factors and for parallel machines
Making Models More Realistic

No lower bounds, but more closely matches real architectures

[HDKWS ISCA ’21]: recast tiling, loop ordering (dataflow), and parallelization as constrained optimization problem (easily solvable with Gurobi/cvxpy). ~50% speedup, ~20% energy efficiency improvement w/90x faster time-to-solution vs. tuning

In-progress work: fusing operators to save communication between layers (to be tested on GEMMINI)

A BERT computation graph:
(Image source: https://github.com/laogouche/bert-eye
was a work in progress, 3D model performance
and improvements)

Modeled cache misses for a simulated NN vs. cache sizes (lower better). Colors indicate layers per block
**User-schedulable language**: Given basic kernel and user-supplied scheduling instructions, generate optimized code. (more to come in ~20 minutes, at Gilbert’s SysTL talk)

To generate high-level scheduling code: metaprogramming layer called MoST (Modular Schedule Transforms). Schedule objects represent “high-level” transforms (“block this loop with specified params”)

Algorithms from previous slides implemented as *generator functions* that generate MoST objects.
Thanks for watching!

Questions? Ask on Slack or in person at 4pm, or email gnd@berkeley.edu