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

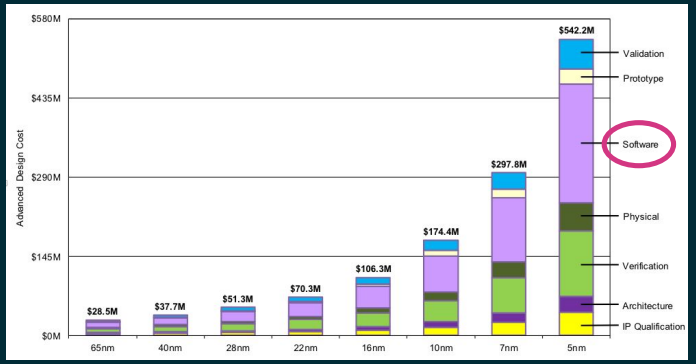
- Protobuf accelerator (Sagar's talk earlier today)
- Cryptocurrency mining: "ASICs are the standard technology found in every large-scale facility"[1]
- Google VCU: specialized hardware for video encoding - "20-33x improved efficiency over our prior well-tuned non-accelerated baseline"[2]
- F1: hardware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

Sources: [1] https://spectrum.ieee.org/why-the-biggest-bitcoin-mines-are-in-china [2] https://dl.acm.org/doi/abs/10.1145/3445814.3446723 [3] https://arxiv.org/abs/2109.05371

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 release[4]



Cost breakdown of developing new chips. Source: IBS [4]

Sources: [4] https://semiengineering.com/how-much-will-that-chip-cost/ [5] https://octoml.ai/blog/on-the-apple-m1-beating-apple-s-core-ml-4-with-50-model-performance-improvements/

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)

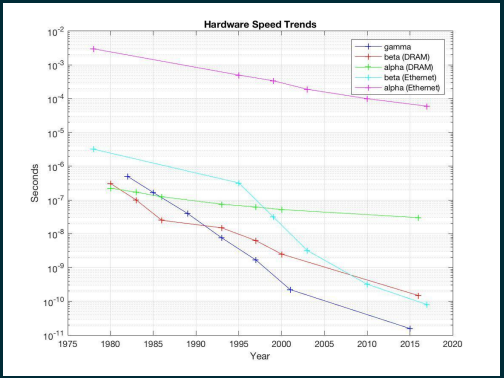
Our approach: develop model for performance, cast scheduling as **numerical optimization**, and solve. Often can get **strong lower bounds** in the model.

Don’t Communicate 😶

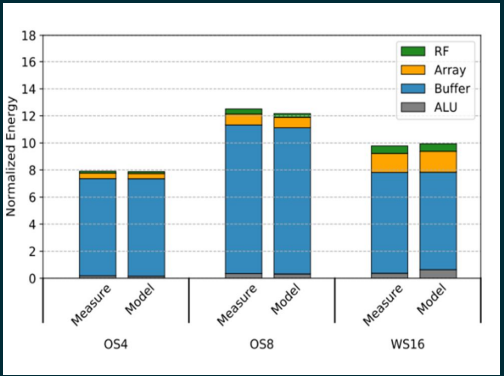
Algorithms have two costs (measured in time or energy). Arithmetic (FLOPS), and communication - moving data between

- 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

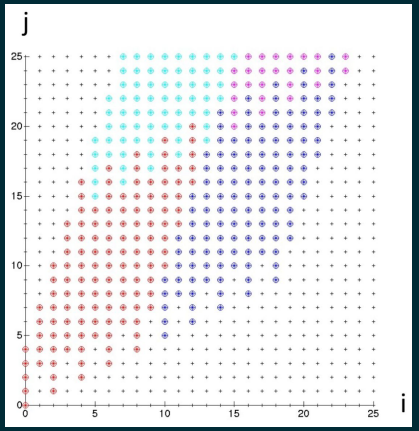
for (i1,i2,...,ik) in $S \subset Z^k$:

Access array locations indexed by affine function of indices, eg φ_C (i1,i2,...,ik) = (i1+2*i3-i7)

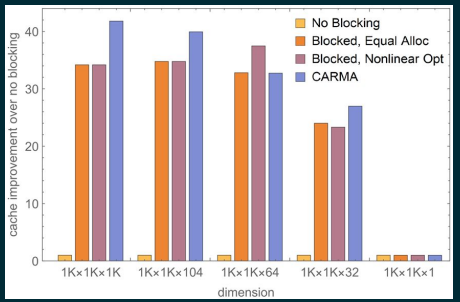
Theorem [Christ et al. '13]: for any reordering, #words moved $\geq \Omega(|S| / M^{e-1})$, for some problem-dependent e

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

Theorem [**D.** - Demmel SPAA '20]: For loops subscripts only dependent on one index (φ projective, "looks like dense linear algebra"): lower bound attainable by efficiently computable tile *regardless of problem size* (i.e. including small dimensions, e.g. "inner product" like computations)



Above: "twisted parallelepiped" tile
Below: communication cost of matmul tiled with DD20 vs. algorithm-specific optimized algorithm (CARMA).

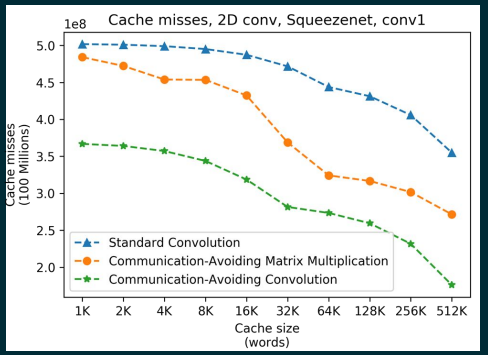


For *specific algorithms*

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

For convolutions:

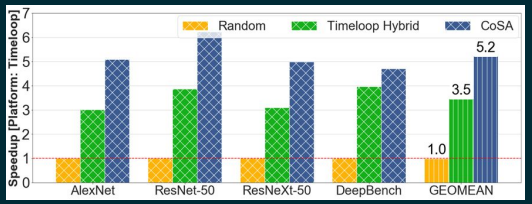
- [Demmel-**D.** MDS '20]: tight, efficiently attainable asymptotic lower bound for convolutions.
  - Computer generated proof, automatic code generation of tiling attaining the lower bound for all cases
  - Implication: well-tiled direct conv more communication-efficient than im2col, regardless of optimizations applied to latter.
- [CD**D**HH, under submission] - tighten communication bounds *including constant factors* and for parallel machines

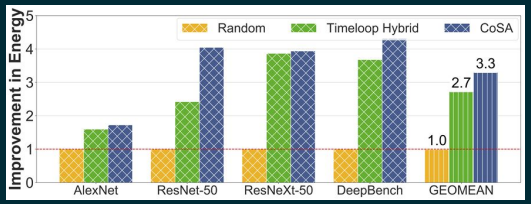


Making Models More Realistic

No lower bounds, but more closely matches real architectures

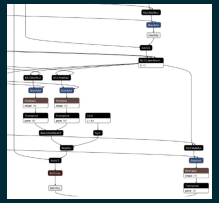
[HK**D**KDWS ISCA '21]: recast tiling, loop ordering (dataflow), and parallelization as constrained optimization problem (easily solvable with Gurobi/cvxpy). ~50% speedup, ~20% energy efficiency improvement w/90x faster time-to-solution vs. tuning



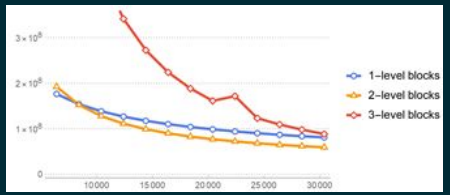


In-progress work: fusing operators to save communication between layers (to be tested on GEMMINI)

A BERT computation graph:
(image source: https://octoml.ai/blog/on-the-apple-m1-beating-apple-s-core-ml-4-with-50-model-performance-improvements/)



Modeled cache misses for a simulated NN vs. cache sizes (lower better). Colors indicate layers per block



In practice…?

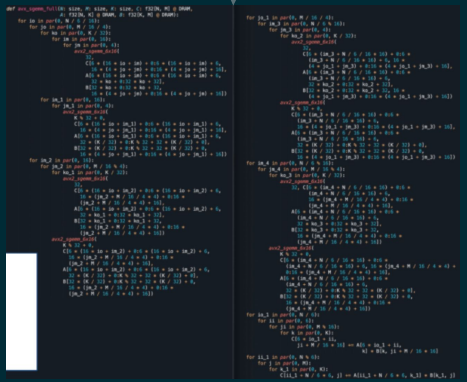
```
matmul_c_i8 = matmul_c_i8.split('i #0',16,['i','i_in'], perfect=True)
matmul_c_i8 = matmul_c_i8.reorder('i_in #0','j')
matmul_c_i8 = matmul_c_i8.split('j #0',16,['j','j_in'], perfect=True)
matmul_c_i8 = matmul_c_i8.lift_alloc('res : _ #0', n_lifts=1)
matmul_c_i8 = matmul_c_i8.lift_alloc('res : _ #0', n_lifts=1, mode='col', size=16)
matmul_c_i8 = matmul_c_i8.lift_alloc('res : _ #0', n_lifts=2)
matmul_c_i8 = matmul_c_i8.fission_after('res[_] = 0.0 #0', n_lifts=2)
matmul_c_i8 = matmul_c_i8.fission_after('for k in _:_ #0', n_lifts=2)
matmul_c_i8 = matmul_c_i8.reorder('i_in #0','k')
matmul_c_i8 = matmul_c_i8.reorder('j_in #0','k')
```

```
def new_sgemm():
    @proc
    def sgemm_full(
        N: size,
        M: size,
        K: size,
        C: f32[N, M] @ DRAM,
        A: f32[N, K] @ DRAM,
        B: f32[K, M] @ DRAM,
    ):
        for i in par(0, N):
            for j in par(0, M):
                for k in par(0, K):
                    C[i, j] += A[i, k] * B[k, j]
    return sgemm_full
```

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