Theoretical Methods for Optimizing Structured Array Computations

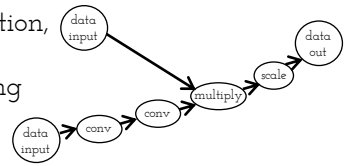
Grace Dinh
dinh@berkeley.edu

Motivation, Model, and Problem Statement

Structured operations on tensors are common and increasingly important building blocks for modern workloads. Example: CNNs, image processing, etc. How to schedule such workloads efficiently? General model: d-nested loop operating on n multidimensional arrays X_i indexed by functions $\boldsymbol{\phi_i}$ of the loop indices $\mathbf{x}_i$, each bounded by L_i.

$$\text{for } x_1 \in [1..L_1], ..., \text{for } x_d \in [1..L_d]:$$
$$\text{perform operations on } A_1[\phi_1(x_1, ..., x_d)], ..., A_n[\phi_n(x_1, ..., x_d)]$$

Structured in a DAG-like computation, with nodes representing one such computation and edges representing dependencies between them (e.g. successive NN layers).



Communication costs tend to dominate, and trend is increasing, esp. w.r.t. energy.

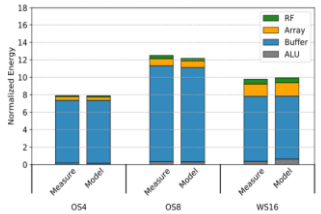
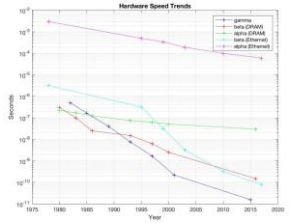


Image from Interstellar [Yang et al. ASPLOS '20]



gamma: flop latency.

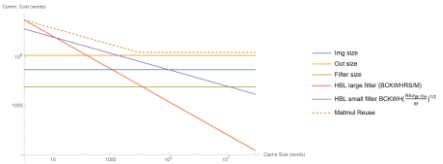
How do we reorder and schedule a workload like the one above to run efficiently w.r.t. communication, latency, throughput?

Two approaches in this poster: tiling and scheduling a **single** layer at a time (and finding lower bounds) - useful for optimizing execution on a chip, and simultaneously scheduling and tiling **entire** DAG at once - useful for optimizing on/off chip data movement.

Goal: methods backed by proofs of lower bound, more reliable and faster to run than ML/autotuning or heuristics.

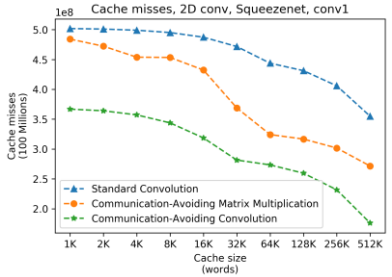
Single Task Communication Lower Bounds (and Optimal Tilings)

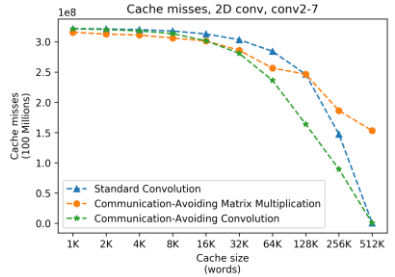
For CNNs: **Theorem** [Demmel-**D** MDS '20]: *Any* rearrangement of a standard batched 2D convolution with filter size R*S and strides $\sigma_H\sigma_W$ using a cache of size M must communicate at least this many words between cache and main memory:



$$\#\text{words moved} \geq \Omega(\max($$
$$\text{size of input, output,}$$
$$(\#\text{arithmetic ops})/M, \quad \text{...(for large filters)}$$
$$(\#\text{arithmetic ops})/(M^{1/2}\,(RS/(\sigma_H\sigma_W))^{1/2}\,)) \quad \text{...(for small filters)}$$

In comparison, any matmul based approach must communicate at least $(\#\text{arithmetic ops})/M^{1/2}$ words, even assuming free replication.





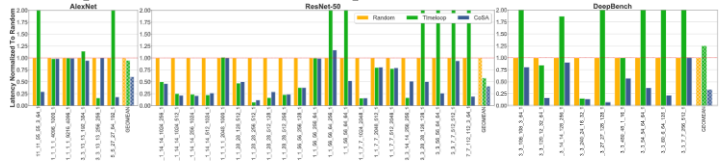
Lower bound achievable using an efficiently constructible tiling by solving an appropriate LP (or plugging in problem size into piecewise LP). Work ongoing on benchmarking, finding constant factors.

For things like matmul, tensor contraction, etc., this result generalizes prior problem specific work: **Theorem** [**D**-Demmel SPAA '20]: [*Suppose the maps are projections (i.e. each $\boldsymbol{\phi_i}$ maps $\mathbf{x}_1,...,\mathbf{x}_d$ to a subset of itself).* Then the **largest possible tile is a rectangle** of dimensions $M^{\lambda_1} \times \cdots \times M^{\lambda_d}$, where $\boldsymbol{\lambda_{1..d}}$ are given by the following linear program:

$$\max \sum_i \lambda_i \text{ s.t.}$$
$$\sum_{i \text{ s.t. } x_i \in \text{supp}(\phi_j)} \lambda_i \leq 1 \quad \forall j \in [n]$$
$$\lambda_i \leq \beta_i \quad \forall i \in [q] \qquad \text{where } \boldsymbol{\beta_i} = \log_M L_i.$$

Accelerator-Specific Optimizations

What about mapping these problems onto accelerators with multiple cores? How to decide where to parallelize? [HK**D**NKWS '20, under submission] Split up loops into components by factoring (maybe with padding), then use similar (I)LP based approach in order to decide how to reorder loops and assign parallelism to certain loops, optimizing for latency/communication/computation.



Current Work: Simultaneous Partitioning/Tiling

With DAGs of problems with dependencies (e.g. neural nets) there's often advantage to doing several layers in a tile to reduce interlayer communication. Currently: can optimally (subject to restrictions on tiling imposed by HW restrictions) tile and partition a **chain graph.**

Theorem: In order to tile a particular chain graph from top to bottom, Can, at no (asymptotic) penalty, restrict ourselves to tiles generated by single rectangular tile at some level of chain and its dependencies (up to a point).

Finding tiling is nonlinear but in practice easily (numerically) solvable optimization problem, similar to above.

Can find a globally optimal partitioning/tiling by tiling all possible subgraphs (~1hr at compilation time right now, can be improved) and performing DP to find optimal partitioning.