

Ansor: Generating High-Performance Tensor Programs for Deep Learning

Author: Lianmin Zheng et. al; Presenter: Xavier Chen (zc344)

November 28, 2024

~~Ansor: Generating High-Performance Tensor Programs for Deep Learning~~

Ansor in Hindsight 2024

Author: Lianmin Zheng et. al; Presenter: Xavier Chen (zc344)

November 28, 2024

Outline

Problem to Solve

Proposed Solution

Evaluation

Conclusion

Problem to Solve

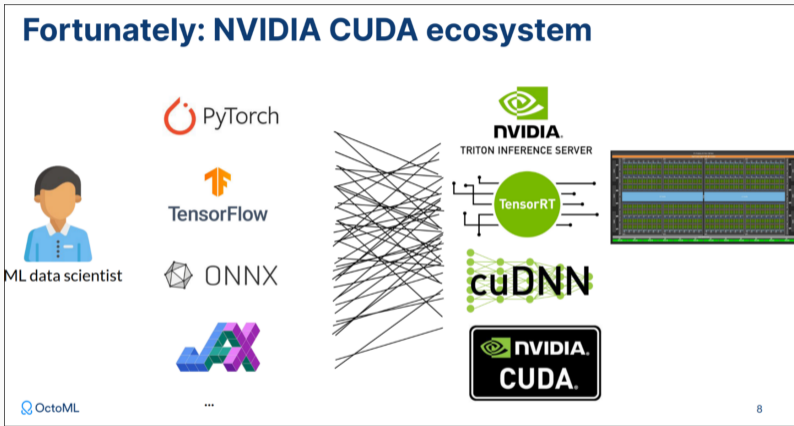
Why TVM?

Motivation

Motivation

- Multiple **frontends** the model comes with: PyTorch, JAX, TF, ONNX.
- Many different **backends**: LLVM, CUDA, ROCm.
- Multiple **devices** they can run on: CPU, GPU, Edge Computing Devices.
- Efficient end-to-end optimization for **deployment**.

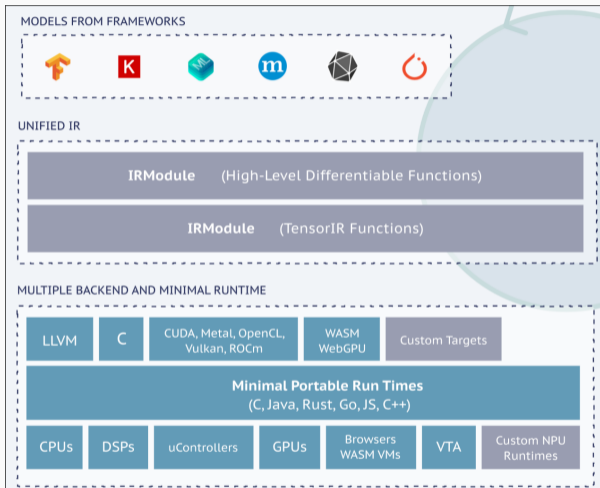
Combinatorial Match between ML Frontend and Backends



¹Using Apache TVM for Automatic, Machine Learning-powered TensorCore Code Generation
NVIDIA. URL:

<https://www.nvidia.com/en-us/on-demand/session/gtcfall21-a31090/>
(visited on 11/27/2024).

TVM Architecture Overview



²Apache TVM. URL: <https://tvm.apache.org/#about> (visited on 11/27/2024).

The TVM line of work largely divides the problem into two parts:

- Search Space generation: what is the set of program semantically equivalent to the input?

The TVM line of work largely divides the problem into two parts:

- Search Space generation: what is the set of program semantically equivalent to the input?
- Tuning/Optimisation: what is the most performant program in the search space.

The TVM line of work largely divides the problem into two parts:

- Search Space generation: what is the set of program semantically equivalent to the input?
- Tuning/Optimisation: what is the most performant program in the search space.

Also it is important to note that Ansor mostly deals with subprogram/operator level optimizations.

The (Manual) TVM Approach

It is merely a language used for domain experts to express transformations.

```
# Designate a set of tile sizes
i_tiles = [16, 8, 8, 8]
j_tiles = [16, 8, 8, 8]
k_tiles = [256, 8]
# Tile the loops according to the tile sizes
i_0, i_1, i_2, i_3 = sch.split(loop=i, factors=i_tiles)
j_0, j_1, j_2, j_3 = sch.split(loop=j, factors=j_tiles)
k_0, k_1           = sch.split(loop=k, factors=k_tiles)
# Organize the loops into "SSRSRS" 6-level tiles
sch.reorder(
    i_0, j_0, # S
    i_1, j_1, # S
    k_0,      # R
    i_2, j_2, # S
    k_1,      # R
    i_3, j_3, # S
)
```

The AutoTVM Approach

Replace fixed tile, loop, vector sizes with tunable parameters

```
# Sample tile sizes
i_tiles = sch.sample_perfect_tile(i, n=4)
j_tiles = sch.sample_perfect_tile(j, n=4)
k_tiles = sch.sample_perfect_tile(k, n=2)
# Tile the loops according to the random variables
i_0, i_1, i_2, i_3 = sch.split(loop=i, factors=i_tiles)
j_0, j_1, j_2, j_3 = sch.split(loop=j, factors=j_tiles)
k_0, k_1           = sch.split(loop=k, factors=k_tiles)
# Organize the loops into "SSRSRS" 6-level tiles
sch.reorder(
    i_0, j_0, # S
    i_1, j_1, # S
    k_0,      # R
    i_2, j_2, # S
    k_1,      # R
    i_3, j_3, # S
)
```

Search Space: handwritten templates potentially for specific architectures.

Optimization: XGBoost (Gradient Tree Boosting) or TreeRNN.

Problem with Approach

Manually written templates requires significant domain knowledge.

*For example, the code repository of TVM already contains more than 15K lines of code for these templates. This number continues to grow as new operators and new hardware platforms emerge.*³

Supporting more devices becomes a challenge.

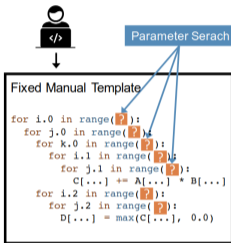
³Lianmin Zheng et al.

Ansor: Generating High-Performance Tensor Programs for Deep Learning. Oct. 15, 2023. DOI: [10.48550/arXiv.2006.06762](https://doi.org/10.48550/arXiv.2006.06762). arXiv: [2006.06762](https://arxiv.org/abs/2006.06762) [cs]. URL: <http://arxiv.org/abs/2006.06762> (visited on 11/26/2024).

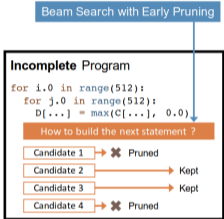
Proposed Solution

Search Space: procedurally generated using hierarchical rule.

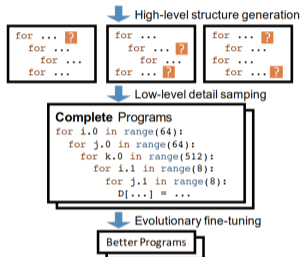
Optimization: learned model for initial filtering + evolutionary search.



(a) Template-guided Search



(b) Sequential Construction Based Search



(c) Ansor's Hierarchical Approach

Figure 2: Search strategy comparison. The pseudo-code shows tensor programs with loop nests. The orange question marks denote low-level parameters.

Hierarchical Sample Generation

No	Rule Name	Condition	Application
1	Skip	$\neg IsStrictInlinable(S, i)$	$S' = S; i' = i - 1$
2	Always Inline	$IsStrictInlinable(S, i)$	$S' = Inline(S, i); i' = i - 1$
3	Multi-level Tiling	$HasDataReuse(S, i)$	$S' = MultiLevelTiling(S, i); i' = i - 1$
4	Multi-level Tiling with Fusion	$HasDataReuse(S, i) \wedge HasFusableConsumer(S, i)$	$S' = FuseConsumer(MultiLevelTiling(S, i), i); i' = i - 1$
5	Add Cache Stage	$HasDataReuse(S, i) \wedge \neg HasFusableConsumer(S, i)$	$S' = AddCacheWrite(S, i); i = i'$
6	Reduction Factorization	$HasMoreReductionParallel(S, i)$	$S' = AddRfactor(S, i); i' = i - 1$
...	User Defined Rule


4

Rule 1, 2 determines inlinability. Rule 3, 4, 5 deals with multi-level tiling and node fusion. Rule 6 allows transformation of space loops to reduction loops.

⁴Lianmin Zheng et al.

Ansor: Generating High-Performance Tensor Programs for Deep Learning. Oct. 15, 2023. DOI: [10.48550/arXiv.2006.06762](https://doi.org/10.48550/arXiv.2006.06762). arXiv: 2006.06762 [cs]. URL: <http://arxiv.org/abs/2006.06762> (visited on 11/26/2024).

Example

<p>Example Input 1:</p> <p>* The mathematical expression:</p> $C[i,j] = \sum_k A[i,k] \times B[k,j]$ $D[i,j] = \max(C[i,j], 0.0)$ <p>where $0 \leq i,j,k < 512$</p> <p>* The corresponding naive program:</p> <pre>for i in range(512): for j in range(512): for k in range(512): C[i, j] += A[i, k] * B[k, j] for i in range(512): for j in range(512): D[i, j] = max(C[i, j], 0.0)</pre> <p>* The corresponding DAG:</p>  <pre>graph LR A((A)) --> C((C)) B((B)) --> C((C)) C((C)) --> D((D))</pre>	<p>Generated sketch 1</p> <pre>for i.0 in range(TILE_I0): for j.0 in range(TILE_J0): for i.1 in range(TILE_I1): for j.1 in range(TILE_J1): for k.0 in range(TILE_K0): for i.2 in range(TILE_I2): for j.2 in range(TILE_J2): for k.1 in range(TILE_I1): for i.3 in range(TILE_I3): for j.3 in range(TILE_J3): C[...] += A[...] * B[...] for i.4 in range(TILE_I2 * TILE_I3): for j.4 in range(TILE_J2 * TILE_J3): D[...] = max(C[...], 0.0)</pre>	<p>Sampled program 1</p> <pre>parallel i.0@j.0@i.1@j.1 in range(256): for k.0 in range(32): for i.2 in range(16): unroll k.1 in range(16): unroll i.3 in range(4): vectorize j.3 in range(16): C[...] += A[...] * B[...] for i.4 in range(64): vectorize j.4 in range(16): D[...] = max(C[...], 0.0)</pre> <p>Sampled program 2</p> <pre>parallel i.2 in range(16): for j.2 in range(128): for k.1 in range(512): for i.3 in range(32): vectorize j.3 in range(4): C[...] += A[...] * B[...] parallel i.4 in range(512): for j.4 in range(512): D[...] = max(C[...], 0.0)</pre>
--	--	---

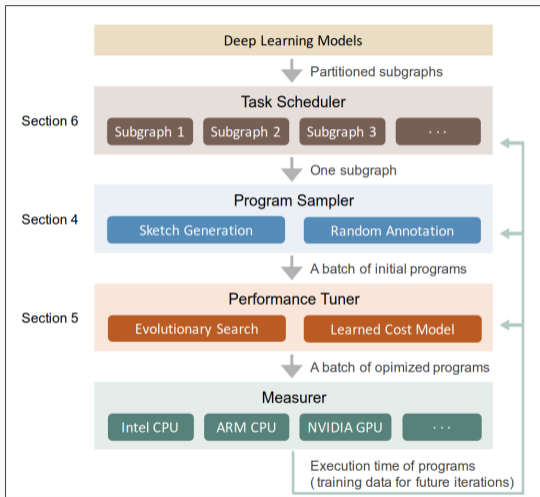
Input1 -> Rule 1 -> Rule 4 -> Rule 1 -> Sketch 1. (i.e. Performing tiling on node C and ignore the rest).

Fill in the Blanks (Random Annotation)

<p>Example Input 1:</p> <p>* The mathematical expression:</p> $C[i, j] = \sum_k A[i, k] \times B[k, j]$ $D[i, j] = \max(C[i, j], 0.0)$ <p>where $0 \leq i, j, k < 512$</p> <p>* The corresponding naive program:</p> <pre>for i in range(512): for j in range(512): for k in range(512): C[i, j] += A[i, k] * B[k, j] for i in range(512): for j in range(512): D[i, j] = max(C[i, j], 0.0)</pre> <p>* The corresponding DAG:</p>  <pre>graph LR A((A)) --> C((C)) B((B)) --> C C --> D((D))</pre>	<p>Generated sketch 1</p> <pre>for i.0 in range(TILE_I0): for j.0 in range(TILE_J0): for i.1 in range(TILE_I1): for j.1 in range(TILE_J1): for k.0 in range(TILE_K0): for i.2 in range(TILE_I2): for j.2 in range(TILE_J2): for k.1 in range(TILE_I1): for i.3 in range(TILE_I3): for j.3 in range(TILE_J3): C[...] += A[...] * B[...] for i.4 in range(TILE_I2 * TILE_I3): for j.4 in range(TILE_J2 * TILE_J3): D[...] = max(C[...], 0.0)</pre>	<p>Sampled program 1</p> <pre>parallel i.0@j.0@i.1@j.1 in range(256): for k.0 in range(32): for i.2 in range(16): unroll k.1 in range(16): unroll i.3 in range(4): vectorize j.3 in range(16): C[...] += A[...] * B[...] for i.4 in range(64): vectorize j.4 in range(16): D[...] = max(C[...], 0.0)</pre> <p>Sampled program 2</p> <pre>parallel i.2 in range(16): for j.2 in range(128): for k.1 in range(512): for i.3 in range(32): vectorize j.3 in range(4): C[...] += A[...] * B[...] parallel i.4 in range(512): for j.4 in range(512): D[...] = max(C[...], 0.0)</pre>
---	--	---

“randomly fill out tile sizes, parallelize some outer loops, vectorize some inner loops, and unroll a few inner loops.”

Tuning with Evolutionary Search



⁵Lianmin Zheng et al.

Ansor: Generating High-Performance Tensor Programs for Deep Learning. Oct. 15,

2022. doi: 10.48550/arXiv.2006.06762 arXiv: 2006.06762 [cs] URL:

Remember we are performing subgraph level optimisations.

Additional Performance Concerns

Remember we are performing subgraph level optimisations.

We also need to infer which subgraph gives the most marginal benefit for improving end-to-end performance.

Optimisation Scheduling

(Approximated) gradient descent on a selection of cost functions.

$$f_1 = \sum_{j=1}^m \sum_{i \in S(j)} w_i \times g_i(t)$$

$$f_2 = \sum_{j=1}^m \max(\sum_{i \in S(j)} w_i \times g_i(t), L_j)$$

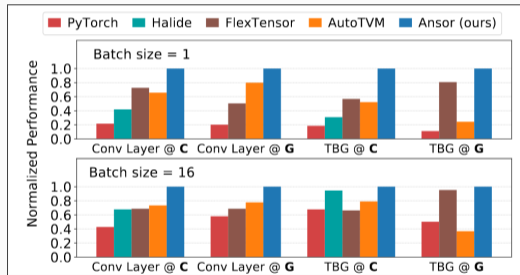
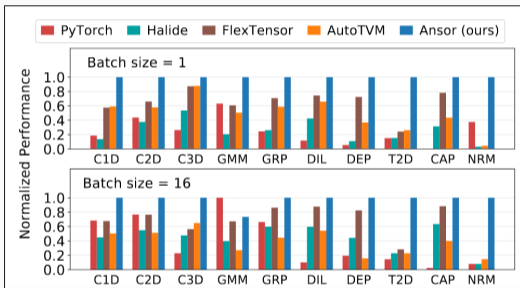
$$f_3 = -\left(\prod_{j=1}^m \frac{B_j}{\sum_{i \in S(j)} w_i \times g_i(t)}\right)^{\frac{1}{m}}$$

$$f_4 = \sum_{j=1}^m \sum_{i \in S(j)} w_i \times \max(g_i(t), ES(g_i, t))$$

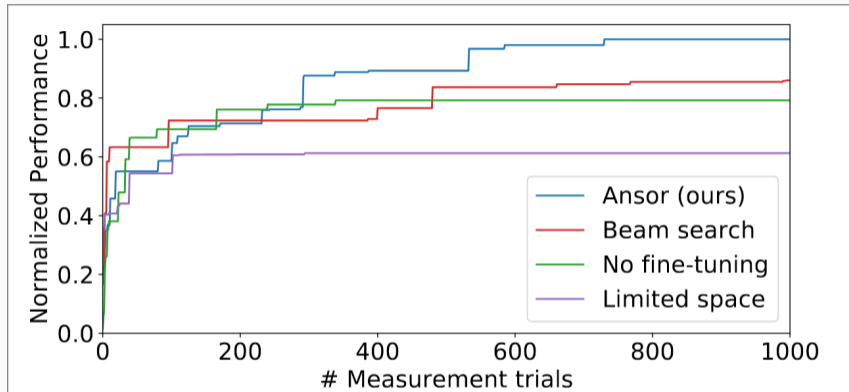
- f_1 sums up the total latency.
- f_2 is basically f_1 but with an estimated bound for “good enough”
- f_3 maximises the geometric mean of end to end latency against “good enough” threshold.
- f_4 estimates the “good enough” threshold based on history.

Evaluation

Performance

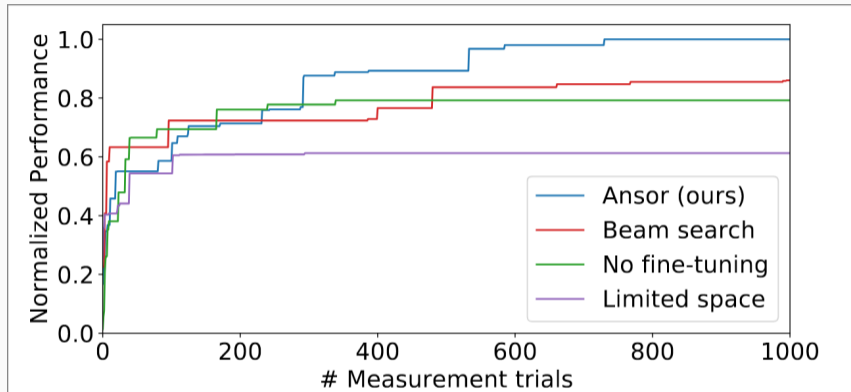


Convergence Time



Each trial takes $k = 80$ programs, 700 trials to converge = 56000 inferences. Plus the time to generate the search space. Empirically from my own experiments that takes 0.5 to 2 times the time taken to perform inferences.

Convergence Time



Each trial takes $k = 80$ programs, 700 trials to converge = 56000 inferences. Plus the time to generate the search space. Empirically from my own experiments that takes 0.5 to 2 times the time taken to perform inferences.

Way too long!

It is non-trivial to create a rewrite rule so that inner most part of the loop fits exactly in a TensorCore.

Addressed by MetaSchedule (Andrew's presentation), Bolt (Gabriel's presentation), Einnet (Sid's presentation)

The New Contribution: Metaschedule

Goal: flexibility of DSL, but with automation

- “Expressiveness, modularity, designed for learning”

Main idea: decouple *search space* from *search algorithm*

- Search space: possible optimized programs
- Search algorithm: finds good optimized program

Second idea: randomly sample parameters

- Probabilistic programming for optimizations

6

⁶Credit: Andrew Krapivin

Conclusion

- + First to take into account the marginal cost of optimising each subgraph.
- + Detailed formulation of cost functions and choices of performance metrics used for prediction.
- + Used by the industry.
- – Highly CPU-centric explanation.
- – Poorly justified choices for using evolutionary search in particular.
- – No simple way to support new intrinsics.

- For systems work, if you are able to inject domain knowledge, it would almost always give you a better search space, and faster convergence.
- Applicability of ML compilers today?