Tensor Program Optimization with Probabilistic Programs

Junru Shao, Xiyou Zhou, Siyuan Feng, Bohan Hou, Ruihang Lai, Hongyi Jin, Wuwei Lin, Masahiro Masuda, Cody Hao Yu, Tianqi Chen

Presenter: Qianyi Liu
Current deep learning frameworks relies on **vendor-specific operator libraries** (e.g. CuDNN) to optimise deployment of neural networks on hardware
- Choose from logically equivalent programs with significantly different performance
- Huge engineering effort + domain knowledge

**Automatic program optimisation** — machine learning
- Two crucial components
- A search space (loop transformation, vectorisation, threading patterns, and hardware acceleration)
- Learning-based search algorithms
A typical workflow for automatic tensor program optimization

Key elements in automatic tensor program optimization

Examples

```
for i in range(1024):
    for j in range(1024):
        for k in range(1024):
            C[i, j] += A[i, k] * B[j, k]
```

```
for i_0, j_0 in grid(16, 8):
    for i_1, j_1 in grid(8, 16):
        for k_0 in range(1024):
            for k_1 in range(16):
                C[...[, ...]] += ...
```

```
for i_0, j_0 in grid(64, 8):
    for i_1, j_1 in grid(4, 32):
        for k_0 in range(64):
            for k_1 in range(4):
                C[...[, ...]] += ...
```

...
The search space itself fundamentally limits the best possible performance search algorithms can get.

Defining the search space for a wide range of tensor programs is challenging:
- $S(e0)$ is highly dependent on $e0$
- Differs in different hardware domains
- Hardware and model settings evolve -> update $S(e0)$

This paper aims to provide a programmable abstraction to construct $S(e0)$ in a composable and modular way.

**MetaSchedule**: a domain-specific probabilistic programming language abstraction to construct a search space of tensor programs
Stochastic Search Space Construction

- Parameterize an optimisation search space by the initial program followed by a sequence of transformations on the program.
- Allow further parameterization of each transformation step with random variables, drawn from sampling distributions.

Parameterization

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Parameterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>① Split</td>
<td>1, 32, 8, 4</td>
</tr>
<tr>
<td>② Parallelize</td>
<td>$i_0$</td>
</tr>
<tr>
<td>③ Vectorize</td>
<td>$i_2$</td>
</tr>
</tbody>
</table>

Initial tensor program: $e_0$


<table>
<thead>
<tr>
<th>Equivalent Programs Induced by Parameterized Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivlent intermediate program: $e_1$</td>
</tr>
<tr>
<td>for $i_0$ in range(32):</td>
</tr>
<tr>
<td>B[i] = ReLU(A[i])</td>
</tr>
<tr>
<td>for $i_1$ in range(8):</td>
</tr>
<tr>
<td>for $i_2$ in range(4):</td>
</tr>
<tr>
<td>i = $i_0 \times 32 + i_1 \times 4 + i_2$</td>
</tr>
<tr>
<td>B[i] = ReLU(A[i])</td>
</tr>
<tr>
<td>Parameterized by: $e_0 + ①$</td>
</tr>
</tbody>
</table>

| Equivalent optimized program: $e^*$                        |
| parallel for $i_0$ in range(32):                           |
| for $i_1$ in range(8):                                    |
| i = $i_0 \times 32 + i_1 \times 4$                       |
| B[i : i + 4] = ReLU(A[i : i + 4])                         |
| Parameterized by: $e_0 + ①②③$                            |
Defining stochastic transformation in MetaSchedule

Probabilistic Program

```
def Probabilistic-Program():
    # ① Loop tiling for Dense
    θ₀, θ₁ ~ Sample-Tile(1, parts=2)
    θ₂, θ₃ ~ Sample-Tile(j, parts=2)
    i₀, i₁ = Split(i, [θ₀, θ₁])
    j₀, j₁ = Split(j, [θ₂, θ₃])
    Reorder(i₀, j₀, i₁, j₁)
    # ② ReLU fusion
    θ₀ReLU ~ Sample-Compute-Location(ReLU)
    Compute-At(ReLU, θ₀ReLU)
```

Stochastic Transformation

```
# Dense:
  For i in range(512):
    for j in range(256):
      C[... | ← ...
    # ReLU:
    for i' in range(512):
      for j' in range(256):
        D[...] = ...
# Fused Dense + ReLU
  # θReLU : Deep fusion under j₁
  for i₀, j₀ in grid(θ₀, θ₁):
    for i₁, j₁ in grid(θ₂, θ₃):
      for k in range(16):
        C[... | ← ...
        for i', j' in grid(θ₀, θ₁):
          D[...] = ...
# Fused Dense + ReLU
  # θReLU : Shallow fusion under j₀
  for i₀, j₀ in grid(θ₀, θ₁):
    for i₁, j₁ in grid(θ₂, θ₃):
      for k in range(16):
        C[... | ← ...
        for i', j' in grid(θ₀, θ₁):
          D[...] = ...
```
Modular Search Space Composition

- Aim: make transformation reusable, make MetaSchedule more easy to use
- Introduce **transformation module**
  - Atomic stochastic transformation
  - Composition of program analysis, sampling as well as smaller transformations
A generic learning-driven framework to find an optimized program

1. Search algorithm samples the MetaSchedule program to obtain a collection of traces
2. An evolutionary search algorithm that proposes a new variant of the trace by mutating the RV -> validator + cost model -> accept
3. Proxy cost model: a tree-boosting-based cost model – updated throughout the process
Experiment 1: Expressiveness to cover common optimisation techniques

Target: a diverse set of operators and subgraphs

- MetaSchedule: our approach
- TVM (AutoTVM and Ansor) — SOTA tensor program optimisation system
- PyTorch — optimised with vendor libraries
Experiment 2: optimising End-to-End deep learning models

Conclusion: MetaSchedule performance is on parity with TVM, while surpassing PyTorch in all cases -> the MetaSchedule framework delivers end-to-end performance
Experiment 3: Search space composition and hardware-specific modules

- By progressively enriching the search space, the performance of optimized tensor programs consistently increases -> translate to end-to-end model performance
- Convenience of customization and composition

(a) Performance with different search spaces.

(b) BERT-Large Performance.
Takeaways

Pros:

- A novel piece of work — MetaSchedule, probabilistic programmable abstraction
- Decouples the search space construction from the search — enabling further customisation without surgical changes to the system
- A simple yet powerful generalisation of existing tensor program optimisation methods

Cons:

- Lack of further evaluation on the search space construction process and program optimisation process
- Didn't explain in detail the advantage over previous deterministic approaches using other DSLs
Discussion