DietCode: Automatic Code Generation for Dynamic Tensor Programs

Bojian Zheng¹, ², ³, Ziheng Jiang⁴, Cody Yu², Haichen Shen², Josh Fromm⁵, Yizhi Liu², Yida Wang², Luis Ceze⁵, ⁶, Tianqi Chen⁵, ⁷, Gennady Pekhimenko¹, ², ³

* Equal Contribution
Background: Current Auto-Scheduler Design
An operator can have infinitely many possible schedules.

Example

Compute:

```java
for (int i = 0; i < 50; ++i) {
    A[i] = ... 
}
```
An operator can have infinitely many possible schedules.

Background: Current Auto-Scheduler Design

Example

Schedule:
```c
for (int io = 0; io < \lfloor 50/t \rfloor; ++io) {
    for (int ii = 0; ii < t; ++ii) {
        if (io*t + ii < 50) A[io*t + ii] = ...
    }
}
```

\( t \in [2, \infty)! \)
Background: Current Auto-Scheduler Design

Limit the candidates to perfect factors

Schedule:
```java
for (int io = 0; io < Math.ceil(50/t); ++io) {
    for (int ii = 0; ii < t; ++ii) {
        if (io*t + ii < 50) A[io*t + ii] = ...;
    }
}
```

Example

\[ t \in \{2, 5, 10, 25\} \]
Background: Current Auto-Scheduler Design

Auto-Scheduler

Operator Specification

Static Shape Description

Shape-Dependent Search Space

Complete Program Cost Model

predicts

Schedule
Background: Current Auto-Scheduler Design
Background: Current Auto-Scheduler Design
Challenges Faced by the Current Design

- Challenge #1:
  - Hard to share schedules across different shapes of the same operator.

**Example**

Schedule:

```c
for (int io = 0; io < \[50/t\]; ++io) {
    for (int ii = 0; ii < t; ++ii) {
        A[io\times t + ii] = ...
    }
}
```

\[ t \in \{2, 5, 10, 25\} \]
Challenges Faced by the Current Design

• Challenge #1:
  • Hard to share schedules across different shapes of the same operator.

Example

Schedule:

```java
for (int io = 0; io < [49/t]; ++io) {
   for (int ii = 0; ii < t; ++ii) {
      A[io\times t + ii] = ...  
   }
}
```

\[ t \in \{7\} \cap \{2, 5, 10, 25\} = \emptyset \]
Challenges Faced by the Current Design

• Challenge #1:
  • Hard to share schedules across different shapes of the same operator.

Example

Schedule:
for (int io = 0; io < \lfloor \frac{49}{t} \rfloor; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    A[io\times t + ii] = ... 
  }
} 
\[ t \in \{7\} \cap \{2, 5, 10, 25\} = \emptyset \]

Prohibitively expensive auto-scheduling time for dynamic-shape workloads.
Challenges Faced by the Current Design

- Challenge #2:
  - Can deliver sub-optimal performance for not considering non-perfect candidates.

Example

Schedule:

```c
for (int io = 0; io < [49/t]; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    if (io*t + ii < 49) A[io	+t + ii] = ... 
  }
}
```

$t \in \{7\}$  
$t = 2, 3, ...$ might be better candidates

Observation: Performance overhead of if-checks is negligible with local padding (i.e., pad tensors locally by the size of local and/or shared memory variables).
DietCode: A New Auto-Scheduler Framework
DietCode: Key Ideas

• Key Idea #1: Shape-Generic Search Space
  • Composed of *micro-kernels*. Each does a tile of the entire compute.
  • A micro-kernel can be ported to *all* shapes of the same operator.
  • Sampled from *hardware* constraints instead of shape factors (i.e., shape-generic).

Example:
\[ Y = XW^T \]
\[ X: [1024, 768], W: [2304, 768] \]
with micro-kernel dense_{128x128}, which evaluates

\[ Y = XW^T \]
\[ X: [128, 768], W: [128, 768] \]
DietCode: Key Ideas

• Key Idea #2: **Micro-Kernel-based Cost Model**
  - Observation: A cost model trained on one shape can be **inaccurate** on other shapes.
  - Compute throughputs exhibit **predictable linear** trend w.r.t. shape dimensions.
  - Decompose the cost model into:
    \[ f_{MK} \cdot f_{spatial} \]
DietCode: Key Ideas

• Key Idea #2: **Micro-Kernel-based Cost Model**
  • Observation: A cost model trained on one shape can be **inaccurate** on other shapes.
  • Compute throughputs exhibit **predictable linear** trend w.r.t. shape dimensions.
  • Decompose the cost model into:
    \[ f_{\text{MK}} \cdot f_{\text{spatial}} \]
    • Trainable Micro-Kernel Cost
DietCode: Key Ideas

• Key Idea #2: **Micro-Kernel-based Cost Model**
  • Observation: A cost model trained on one shape can be **inaccurate** on other shapes.
  • Compute throughputs exhibit **predictable linear** trend w.r.t. shape dimensions.
  • Decompose the cost model into:
    \[ f_{\text{MK}} \cdot f_{\text{spatial}} \]
    • Trainable Micro-Kernel Cost
    • Analytical Spatial Generalization Cost (linear function)
DietCode: A New Interface

• Supports dynamic-shape workloads with its new interface.
  • E.g., $Y = XW^T$ $X$: $[16 \times T, 768]$, $W$: $[2304, 768]$, $T \in [1, 128]$  
    
    ```python
    T, T_vals = tir.ShapeVar('T'), list(range(1, 128))
    
    task = SearchTask(func=Dense, args=(16*T, 768, 2304),  
                        shape_vars=(T,), wkl_insts=(T_vals,)  
                        wkl_inst_weights=(1. for _ in T_vals),)
    ```
DietCode: A New Interface

• Supports dynamic-shape workloads with its new interface.
  • E.g., \( Y = XW^T \) \( X: [16 \times T, 768], W: [2304, 768], T \in [1, 128] \)
  
  ```python
  T, T_vals = tir.ShapeVar('T'), list(range(1, 128))
  task = SearchTask(func=Dense, args=(16*T, 768, 2304),
                     shape_vars=(T,), wkl_insts=(T_vals,)
                     wkl_inst_weights=[1. for _ in T_vals],)
  ```

• Define a dynamic shape variable \( T \) and its instances.
DietCode: A New Interface

• Supports dynamic-shape workloads with its new interface.
  • E.g., $Y = XW^T \ X$: $[16 \times T, 768], W: [2304, 768], T \in [1, 128]$  
    $T, T\_vals = tir.\text{ShapeVar}(\text{'T'}), \text{list}(\text{range}(1, 128))$

    task = SearchTask(func=Dense, args=(16*T, 768, 2304),
    shape_vars=(T,), wkl\_insts=(T\_vals,),
    wkl\_inst\_weights=(\[1. for _ in T\_vals],))

• Define a dynamic shape variable $T$ and its instances.
• Pass the variable and its instances to the workload function.
DietCode: A New Interface

• Supports dynamic-shape workloads with its new interface.
  • E.g., $Y = XW^T \quad X: [16\times T, 768], W: [2304, 768], T \in [1, 128]$
    
    $T, T_{vals} = \text{tir.\_ShapeVar('T')}, \text{list(range(1, 128))}$

    ```python
    task = SearchTask(func=Dense, args=(16*T, 768, 2304),
                      shape_vars=(T,), wkl_insts=(T_vals,))
    wkl_inst_weights=[[1. for _ in T_vals]],)
    ```

  • Define a dynamic shape variable $T$ and its instances.
  • Pass the variable and its instances to the workload function.
  • [Optional] Assign weight to each shape instance.
Evaluation

Hardware: NVIDIA Tesla T4 GPU

Software: TVM + CUDA + cuDNN
Evaluation

Performance: 30.5% better than Ansor; 5.3% better than Vendor
Auto-Scheduling Time: 5.6× less than Ansor
Evaluation

What about multiple dynamic axes?

Better

24.2% better than Ansor; 15.4% better than Vendor
Summary

• DietCode: An auto-scheduler for dynamic-shape workloads.

• Based on 2 key ideas:
  (1) Shape-Generic Search Space and
  (2) Micro-Kernel-based Cost Model

• Key Features:
  • **Auto-Schedule Once and For All Shapes**
    • Large reduction in the auto-scheduling time.
  • **Better Performance**
    • Up to 30.5% speedup than Ansor, up to 15.4% than Vendor.

• Working on integrating into the TVM main branch ...
DietCode: Automatic Code Generation for Dynamic Tensor Programs

Bojian Zheng*1, 2, 3, Ziheng Jiang*4, Cody Yu2, Haichen Shen2, Josh Fromm5, Yizhi Liu2, Yida Wang2, Luis Ceze5, 6, Tianqi Chen5, 7, Gennady Pekhimenko1, 2, 3

* Equal Contribution
Backup
Scratchpad
DietCode: Key Ideas

• Key Idea #1: **Shape-Generic Search Space**
  • Composed of *micro-kernels*. Each does a tile of the entire compute.
  • A micro-kernel can be ported to *all* shapes of the same operator.
  • Sampled from *hardware* constraints instead of shape factors (i.e., shape-generic).

Example:

\[ Y = XW^T \]

- \( X: [16 \times 1, 768], W: [2304, 768] \)
- with micro-kernel `dense_128x128`, which evaluates

\[ Y = XW^T \]

- \( X: [128, 768], W: [128, 768] \)
Challenges Faced by the Current Design

• Challenge #2:

  • **Can deliver sub-optimal performance** for not considering non-perfect candidates.

Example

Schedule (Loop Tiling):

```java
for (int io = 0; io < [49/t]; ++io) {
    for (int ii = 0; ii < t; ++ii) {
        if (io*t + ii < 49) A[io*t + ii] = ...
    }
}
```

\( t \in \{7\} \)

\( t = 2, 3, ... \) might be better candidates