RoofTune: Accelerating the Tuning Process Based on Roofline Model

Hui Zhong, HaiWen Fu, XiaoHua Shi
钟辉，付海文，史晓华

Beihang University
Motivation & Related Work

- The current tuning process is extremely time-consuming...
- Current work can be summarized in two aspects:

  **Based on Search**
  - **CHAMELEON**: RL (reinforcement learning) + AS (adaptive sampling)
  - **AdaTune**: random forest + coefficient of variation (CV)
  - **...**

  **Based on Static Analysis**
  - **Tuna**: Parsing high-level program IR and low-level assembly code
  - **...**

  **RoofTune**

Operator Performance

Time Consume
Overall Design of RoofTune

- **Cost Model**: Manual designed, based on Roofline Model, easily to be deployed

- **Search Algorithm**: Combined with the cost model, two stage search
Roofline Model

Roofline Model tells us:

*Operational intensity* can be used to predict the operator’s performance.

**Ideas:**

We introduce Roofline Model to kernel’s level (fine grained):

- *Operational intensity* can be expressed by some specific schedule parameters.

![Diagram of Roofline Model](image)
Cost Model Design Principle

\[ \text{Score} = I_{\text{kernel}} \times \text{Concurrency} \]

kernel’s arithmetic intensity:
\[ I_{\text{kernel}} = \frac{W_{\text{kernel}}}{T_{\text{kernel}}} \]

kernel’s arithmetic amount:
\[ W_{\text{kernel}} = f_w(\text{knob0}, \text{knob1}, \text{knob2} \ldots) \]

kernel’s execution concurrency:
\[ \text{Concurrency} = f_C(\text{knob0}, \text{knob1}, \text{knob2} \ldots) \]

kernel’s memory traffic:
\[ T_{\text{kernel}} = f_T(\text{knob0}, \text{knob1}, \text{knob2} \ldots) \]
Conv2d Cost Model Design on GPU

**Score** = \( I_{kernel} \times \text{Concurrency} \)

**kernel’s arithmetic intensity:**
\[ I_{kernel} = \frac{W_{kernel}}{T_{kernel}} \]

**kernel’s execution concurrency (consider blocks in GPU):**
\[ \text{Concurrency} = \begin{cases} 1, & \text{if blocks > SPs \times 3} \\ \frac{\text{blocks}}{(SPs \times 3)}, & \text{if blocks \leq SPs \times 3} \end{cases} \]

**kernel’s arithmetic amount:**
\[ W_{kernel} = \prod(f[3], y[3], x[3], f[1], y[0], x[1], rf[0], f[1], y[0], ry[1], rx[0], rx[1]) \]

**kernel’s memory traffic:**
\[ T_{kernel} = \text{data-io} + \text{weight-io} + \text{out-io} \]

**kernel’s data memory traffic:**
\[ \text{data-io} = \text{data-shape}[2] \times \text{weight-shape}[2] \div \prod(f[2], y[2], x[2], y[0], x[0]) \]
Conv2d Cost Model Design on Huawei NPU

\[ S = I \times C \]

- **kernel’s arithmetic amount:**
  \[ W_{kernel} = \text{out-shape} \times \text{weight-shape} \]

- **kernel’s memory traffic:**
  \[ T_{kernel} = \text{data-\text{io}} + \text{weight-\text{io}} + \text{out-\text{io}} \]

- **kernel’s data memory traffic:**
  \[ \text{data-\text{io}} = \text{data-shape} \div \text{AL0-matrix} \]

- **kernel’s arithmetic intensity:**
  \[ I_{kernel} = \frac{W_{kernel}}{T_{kernel}} \]

- **kernel’s data memory traffic:**
  \[ \text{data-\text{io}} = \text{data-shape} \div \text{AL0-matrix} \]

Score = \( I_{kernel} \times \text{Concurrency} \)

### Knobs in the Schedule Config Space to Optimize Convolution

<table>
<thead>
<tr>
<th>KNORS</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL1_shape,BL1_shape</td>
<td>Allocate Buffer L1 to data, weights</td>
</tr>
<tr>
<td>AL0_matrix,BL0_matrix, CL0_matrix</td>
<td>Allocate Buffer L0 to data, weights, outputs</td>
</tr>
<tr>
<td>AUB_shape,BUB_shape, CUB_shape</td>
<td>Allocate Unified Buffer to data, weights, outputs</td>
</tr>
<tr>
<td>(AL1,BL1,AL0,BL0,CL0, AUB,BUB,CUB)_pbuffer</td>
<td>Decide whether to set double buffer at given every on-chip buffers</td>
</tr>
<tr>
<td>block_dim</td>
<td>Conv’s data factors for tiling and binding on each AICore</td>
</tr>
<tr>
<td>(A,B)_overhead_opt_flag, n_bef_group_flag, n_bef_batch_flag</td>
<td>Some schedule adjustments that are difficult to estimate</td>
</tr>
</tbody>
</table>

Concurrency = \( \text{block\_dim} \)
Randomly select a large number of configs from the all schedule space

- Y-axis: operator execution time
- X-axis: scores rank from high to low
RoofTune Two-stage Search Algorithm

First Stage:
To find out the promising scores

1. Random select 500 configs
2. Rank scores from high to low
3. Select one config from the same scores
4. Measure on device
5. Select scores from the top 10 actual performance
RAFT based Two-stage searching algorithm

Second Stage:
To limit the schedule space to score_list

Random select 500 configs

RoofTune Cost Model

Select scores only in score_list

Configuration

Measure on device

Update the best config and get the final result
Performance Evaluation

End To End Performance

RoofTune vs XGB on AutoTVM(GPU)

RoofTune vs GA on TBE’s AutoTune(NPU)
Acknowledgment

• This work is funded by Huawei Technology.