Friedrich-Alexander-Universität
Technische Fakultät

Deep Learning Compiler

Huiqiao Zhang, Jianchang Su
Hardware-Software-Co-Design, Friedrich-Alexander-Universität Erlangen-Nürnberg
October 16, 2022

Agenda

Intros

Vanilla TVM

AutoTVM

Auto-scheduler(a.k.a. Ansor)

Conclusion

Outline

Classical Compiler

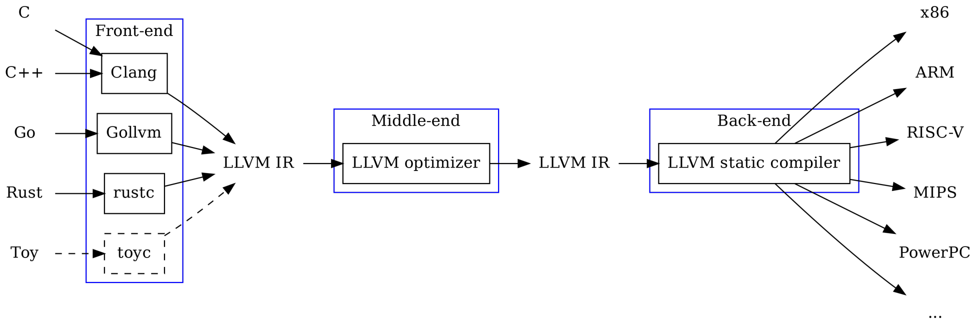

Figure: Three Major Components of a Three-Phase Compiler [1]

Deep Learning Compiler

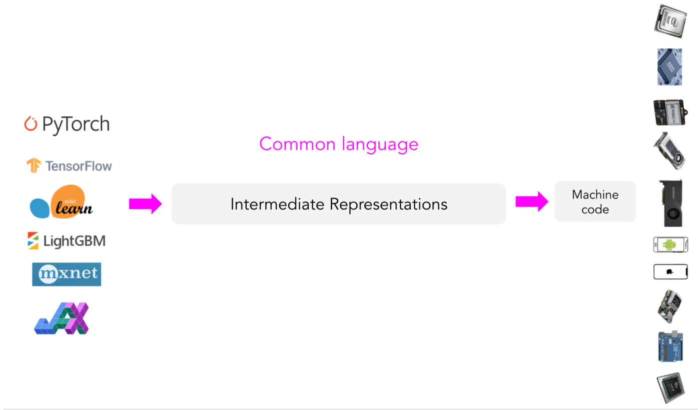

Figure: Deep learning Compiler Architecture [2]

Apache TVM

- Apache TVM [3] is an End to End Machine Learning Compiler Framework
- It aims to optimize and run computations efficiently on any hardware back-end.

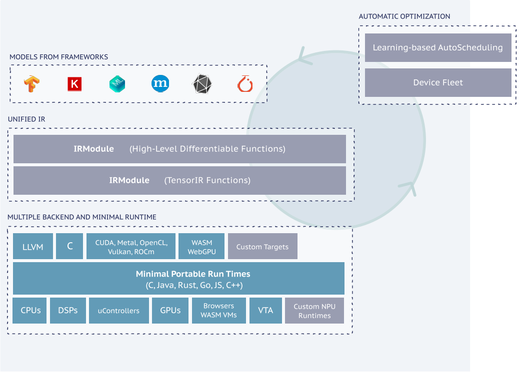

Figure: TVM Compiler [4]

Outline

TVM vs TensorRT

- **Platforms:**
  –Intel CPU (i7-8700)
  –NVIDIA GPU (RTX 2080)
- **Network:**
  –Resnet-101
  –Input shape(NCHW): [1, 3, 244, 244]

Table: Vanilla-TVM V.S TensorRT (Lower is better)

	Vanilla-TVM	TensorRT FP32	TensorRT FP16	TensorRT INT8
Cost time [ms]	9.18	6.10	1.90	1.37

Outline

Auto TVM

- Auto TVM [5] is the search module of Machine Learning Compiler Framework TVM
- It employs **a template-based** search algorithm to find efficient implementations for a given tensor computation.
    - still requires implementing **a non-trivial manual template**(more than 15k lines of code) on every platform
    - have **inefficient** and **limited** search spaces, unable to achieve optimal performance
- **Procedure:**
    - Step 1: Define the search space
    - Step 2: Search through the space

Tuning

Table: Tuning data

Autotune trial num	0	16	32	64	128	256	512	1024
TVM FP32	9.18	13.48	14.26	10.69	7.97	7.54	6.50	6.02
TRT FP32	6.10	-	-	-	-	-	-	-
TRT FP16	1.90	-	-	-	-	-	-	-
TRT INT8	1.37	-	-	-	-	-	-	-

- The time cost of TensorRT on resnet-101 is 6.10ms. So we are a little faster.

What exactly AutoTVM did


```
{"input": ["cuda -keys=cuda,gpu -arch=sm_75 -max_num_threads=1024 -model=unknown -
thread_warp_size=32", "conv2d_nchw.cuda", [["TENSOR", [1, 256, 14, 14], "float32"],
["TENSOR", [512, 256, 1, 1], "float32"], [2, 2], [0, 0, 0, 0], [1, 1], "float32"], {}],
"config": {"index": 11312, "code_hash": null, "entity": [["tile_f", "sp", [-1, 4, 32,
2]], ["tile_y", "sp", [-1, 1, 1, 7]], ["tile_x", "sp", [-1, 1, 1, 1]], ["tile_rc", "sp",
[-1, 8]], ["tile_ry", "sp", [-1, 1]], ["tile_rx", "sp", [-1, 1]],
["auto_unroll_max_step", "ot", 0], ["unroll_explicit", "ot", 0]]}, "result":
[[0.00010343405303678358], 0, 1.5358951091766357, 1658184289.7048202], "version": 0.2,
"tvm_version": "0.9.dev0"}
```


Figure: Tiling and splitting

Outline

Auto-scheduler(a.k.a. Ansor)

- It aims at **a fully automated auto-scheduler** for generating code for tensor computations.
  - Input: only tensor expressions
  - Output: high-performance code **without manual templates**
  - Search strategy: using heuristic search algorithms
- Ansor [6] constructs **a hierarchical search space** that decouples high level structure from low level details and **automatically** constructs the search space for computing graphs without manually developing templates.

Ansor’s Hierarchical Approach

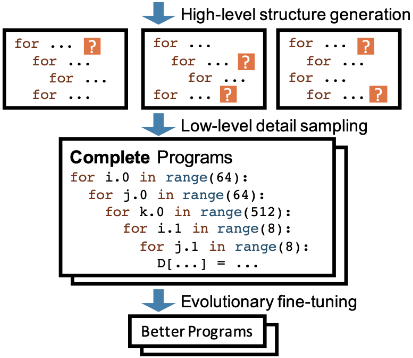

Figure: Ansor’s Hierarchical Approach [6]

Auto-scheduler(a.k.a. Ansor)

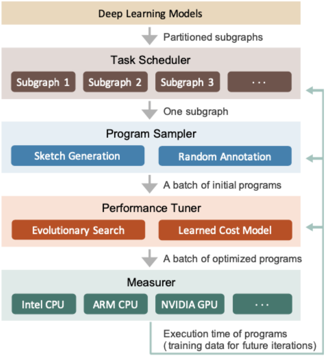

Figure: Search system Overview [7]

4 Procedures

- **Task Scheduler:** Cut the entire computational graph into **subgraphs**, find the hot subgraphs by gradient descent, and then focus on optimizing the hot subgraphs.
- **Sketch:** Extracts the features of the higher-level operators in the subgraphs, performs coarse-grained optimization of the operators, and determines the basic structure of the optimization.
- **Annotation:** Randomly initialize the Tilling Size and some for-loop strategies to obtain a complete representation of the subgraph.
- **Performance fine-tuning:** Improving the search performance by evolving the search and cost models and getting an efficient implementation of the final code.

Task

```
========== Task 4  (workload key: ["677e98c19c9fb48e26...) ==========
placeholder = PLACEHOLDER [1, 2048, 7, 7]
pad_temp(i0, i1, i2, i3) = placeholder[i0, i1, i2, i3]
placeholder = PLACEHOLDER [512, 2048, 1, 1]
compute(nn, ff, yy, xx) += (pad_temp[nn, rc, (yy + ry), (xx + rx)]*placeholder[ff, rc, ry, rx])
placeholder = PLACEHOLDER [1, 512, 1, 1]
T_add(ax0, ax1, ax2, ax3) = (compute[ax0, ax1, ax2, ax3] + placeholder[ax0, ax1, 0, 0])
T_relu(ax0, ax1, ax2, ax3) = max(T_add[ax0, ax1, ax2, ax3], 0f)
```

Figure: Task 4 computation DAG

Program

```
Program:
Placeholder: placeholder, placeholder, placeholder
blockIdx.x ax0.0@ax1.0@ax2.0@ax3.0@ (0,28)
  threadIdx.x ax0.2@ax1.2@ax2.2@ax3.2@ (0,256)
    compute auto_unroll: 512
    for rc.0 (0,32)
      threadIdx.x ax0@ax1@ax2@ax3@.0.1 (0,256)
        vectorize ax0@ax1@ax2@ax3@.1 (0,4)
          placeholder.shared = ...
      threadIdx.x ax0@ax1@ax2@ax3@.0.1 (0,256)
        vectorize ax0@ax1@ax2@ax3@.1 (0,7)
          pad_temp.shared = ...
      for rc.1 (0,4)
        for xx.3 (0,7)
          for rc.2 (0,4)
            for ff.4 (0,2)
              compute = ...
    for ax1.3 (0,2)
      for ax3.3 (0,7)
        T_relu = ...
```

Figure: Task 4 program

Benchmark

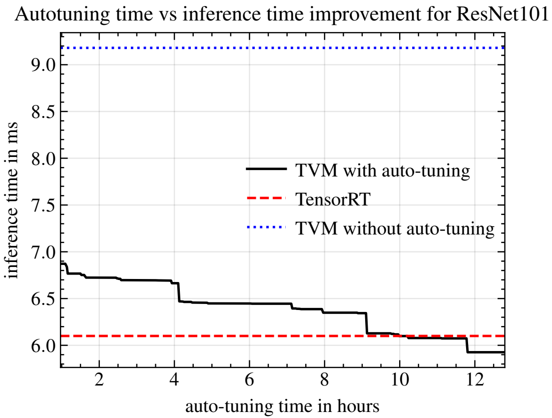

Figure: Estimated latency

Outline

Intros

Vanilla TVM

AutoTVM

Auto-scheduler(a.k.a. Ansor)

Conclusion

Conclusion

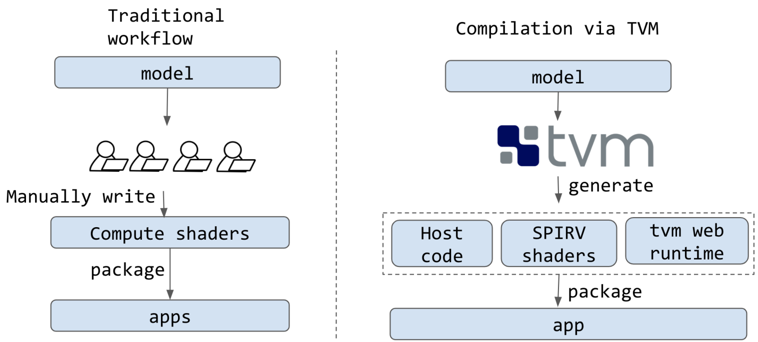

Figure: With TVM, we can spend less effort [8]

Conclusion

- TVM auto-scheduler is a system that **automatically** generates high-performance code for tensor expressions.
- By reconstructing the **search space structure** and **algorithm**, auto-scheduler is capable of generating schedules with better performance in a shorter time.

Thanks for listening.

Any questions?

References I

[1] "The Architecture of Open Source Applications: LLVM," (), [Online]. Available: https://www.aosabook.org/en/llvm.html.

[2] "A friendly introduction to machine learning compilers and optimizers," (), [Online]. Available: https://huyenchip.com/2021/09/07/a-friendly-introduction-to-machine-learning-compilers-and-optimizers.html.

[3] T. Chen, T. Moreau, Z. Jiang, *et al.*, "TVM: An automated End-to-End optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594, ISBN: 978-1-939133-08-3.

[4] "Apache TVM," (), [Online]. Available: https://tvm.apache.org/.

[5] T. Chen, L. Zheng, E. Yan, *et al.*, "Learning to optimize tensor programs," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

References II

[6] L. Zheng, C. Jia, M. Sun, *et al.*, "Ansor: Generating High-Performance tensor programs for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, USENIX Association, Nov. 2020, pp. 863–879, ISBN: 978-1-939133-19-9.

[7] "Introducing TVM Auto-scheduler (a.k.a. Ansor)," (), [Online]. Available: `https://tvm.apache.org/2021/03/03/intro-auto-scheduler`.

[8] "Compiling Machine Learning to WASM and WebGPU with Apache TVM," (), [Online]. Available: `https://tvm.apache.org/2020/05/14/compiling-machine-learning-to-webassembly-and-webgpu`.