PLANT：基于多面体模型的张量编译器

李晨昊

清华大学计算机科学与技术系

2021 年 6 月 10 日

背景

- 张量计算在深度学习，图像处理，科学计算等领域占据重要地位 [20]
- 以深度学习为例，随着体系结构和高层框架复杂性的增加，为异构平台生成高效的张量代码面临巨大的工程挑战。

多种多样的深度学习框架和硬件

背景

- 张量编译器通过编译优化技术自动化代码生成过程
- 多面体模型 [10, 12] 是编译优化领域的重要技术，有许多将它应用于张量编译器领域的尝试
- 集合：整数元组的集合
  - $\{c[i,j] : 0 \le i \le 1 \wedge 0 \le j \le 2\}$
  - 语句每次执行，包围它的循环变量取值形成元组，它们组成的集合表示了循环的迭代范围
- 映射：两个集合间的关系，即整数元组序对的集合
  - $\{c_1[i,j] \to c_2[j,i] : 0 \le i \le 1 \wedge 0 \le j \le 2\}$
  - 映射用于表示循环调度指令，访问下标等概念

相关工作

1. 人工调度的非多面体编译器：Halide [15]，TVM [4]
   - 综合性能较好
   - 自动化程度低，AutoTVM [5]，Ansor [19] 等拓展工作实现一定的自动化
   - 程序表达和分析能力有限
2. 自动调度的多面体编译器：PLuTo [2]，PPCG [17]
   - 自动化程度高
   - 生成的代码性能不理想
   - 调度指令覆盖面有限
   - 用户难以介入编译过程
3. 人工调度的多面体编译器：CHiLL [7]，AlphaZ [18]，TIRAMISU [1] 。兼有前两者的部分优缺点

简介

- PLANT: PoLyhedral bAsed teNsor opTimizer
- 主体是人工调度的多面体编译器，应用非多面体编译器的自动调度方法
- 丰富的调度指令，多面体模型的强大表达能力，高效的编译速度和优化效果，一定程度的自动化

分层设计

- 借鉴 TIRAMISU [1] 的分层中间表示思想，将程序定义分离为算法描述，循环调度，内存调度
- 按顺序应用调度指令，简化编译器实现

```
let (n, s, m) = (1024, 1024, 1024);
let f = Func::new("matmul");
// 算法描述
let a = f.buf("a", I32, In, x![n, s]);
let b = f.buf("b", I32, In, x![s, m]);
let c_init = f.comp("C_init", x![n, m], x!(0));
let c = f.comp("C", x![n, m, s], x!(0));
c.set_expr(x!(a(i0, i2) * b(i2, i1) + c(i0, i1, i2 - 1)));
// 循环调度
c_init.tile(0, 1, 32, 32);
c.tile(0, 1, 32, 32);
c.after(c_init, 4);
c.tag(0, Parallel);
// 内存调度
let buf_c = f.buf("c", I32, Out, x![n, m]);
c_init.store(buf_c);
c.store_at(buf_c, x![i0, i1]);
// 定义完成，最终生成代码
f.codegen(&[a, b, buf_c]);
```

循环调度

指令	功能
split	将循环分裂为嵌套的两层
fuse	合并两个相邻的嵌套循环
reorder	调整循环的嵌套顺序
tile	循环分块
skew	循环倾斜
shift	循环索引偏移
after	控制循环体位置
separate	将循环拆分为两个并列的循环
inline	将表达式内联到使用它的计算中
tag	为循环添加标记
apply_sch	底层原语，执行任意仿射变换

内存调度

指令	功能
store	控制计算结果的保存位置
cache_identity	自动完成恒等映射的缓存
cache	用户手动控制的缓存
set_loc	将缓冲区映射到内存层次结构
alloc_at	控制缓冲区申请和释放的位置
auto_transfer	控制 GPU 缓冲区自动传输数据

非仿射程序

- 仿射的程序要求循环范围表示成外层循环变量的仿射表达式
- 动态形状/稀疏算子等不满足条件，借助 ISL [16] 的参数机制，配合人工指示参数范围以描述程序

```
let b0 = f.comp("b0", x![m,], x!(ptr(i0)));
let b1 = f.comp("b1", x![m,], x!(ptr(i0 + 1)));
let y = f.comp("y", x![m, b1 - b0, m], x!(0));
y.set_expr(x!(val(i1 + b0) * b(idx(i1 + b0), i2) + y(i0, i1, i2 - 1)));
f.set_constraint(x![m > 0, b0 > 0, b1 > 0, b1 > b0]);
y.split(1, 32);
y.cache_identity(val, 1, Local);
y.cache_identity(idx, 1, Local);
```

```
for (int i0 = 0; i0 < m; i0 += 1) {
  int b0 = ptr[i0], b1 = ptr[i0 + 1];
  for (int i1 = 0; i1 <= (-b0 + b1 - 1) / 32; i1 += 1) {
    int cache_val[32];
    for (int i2 = 0; i2 <= 31; i2 += 1)
      if (b1 >= (b0 + i2 % 32) + 1)
        cache_val[i2] = val[(b0 + i1 * 32) + i2];
    ...
  }
}
```

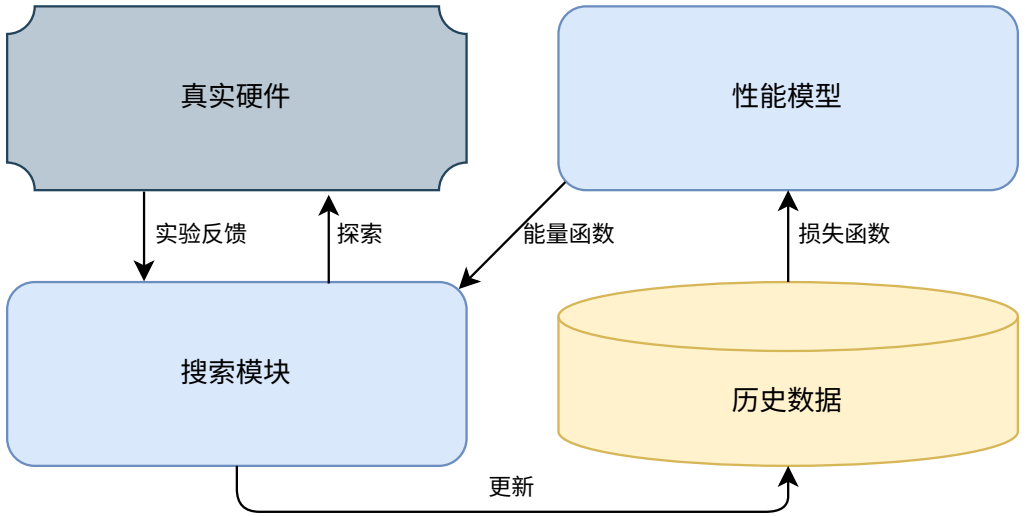
自动调度

- 调度包含大量参数，编写高效的调度仍需理解体系结构
- 设计借鉴 AutoTVM [5]，基于模板的自动调度，在用户定义的搜索空间上调优参数
  - Knob: 用户提供可选取值
  - Split: 计算循环分裂可能参数，循环跨度因子或 2 的幂次
  - Tag: 调优循环标记
  - Reorder: 调优循环嵌套顺序

```
// 定义搜索空间
space.define_split("sp", SplitPolicy::new(oc)
  .set_pow2(true).set_n_output(4));
...
// 应用搜索空间中的调度
let sp = cfg.get("sp");
b.split(0, sp[0]).split(0, sp[1]).split(0, sp[2]);
...
```

自动调度

- 对程序抽取特征
  - Knob: 直接用参数值作特征
  - Iter: 循环特征，包括浮点运算，访存，并行特征等
- 使用 XGBoost [3] 算法建立基于统计数据的性能模型
  - Reg: 拟合绝对耗时
  - Rank: 拟合不同程序耗时排序
- 复杂算子参数空间无法穷尽搜索，以性能模型的输出为能量函数，使用模拟退火 [11] 挑选有希望的程序进行实验



系统优化

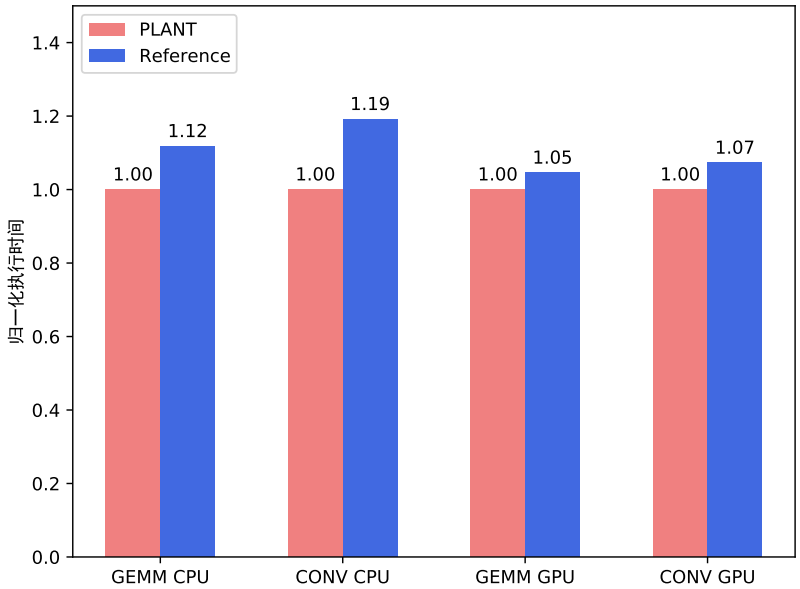
- 对代码生成和运行时进行了大量优化
  - 插入 assume，restrict 等编译器 hint
  - 实现并行库，考虑超线程，核心绑定，线程池等因素
  - 自动调度算子时刷新缓存
  - 选择合适的向量化，循环展开参数
  - ...

其他功能

- 远程执行
  - 移动和嵌入式设备资源有限，需要在主机设备上完成编译，在目标设备上运行算子并返回结果
  - 实现了轻量级的网络协议，与 TVM 的 RPC 模式相比，无需安装各种依赖，只使用 C 标准库
- Python 接口
  - 调用编译好的算子，便于快速调试和迭代

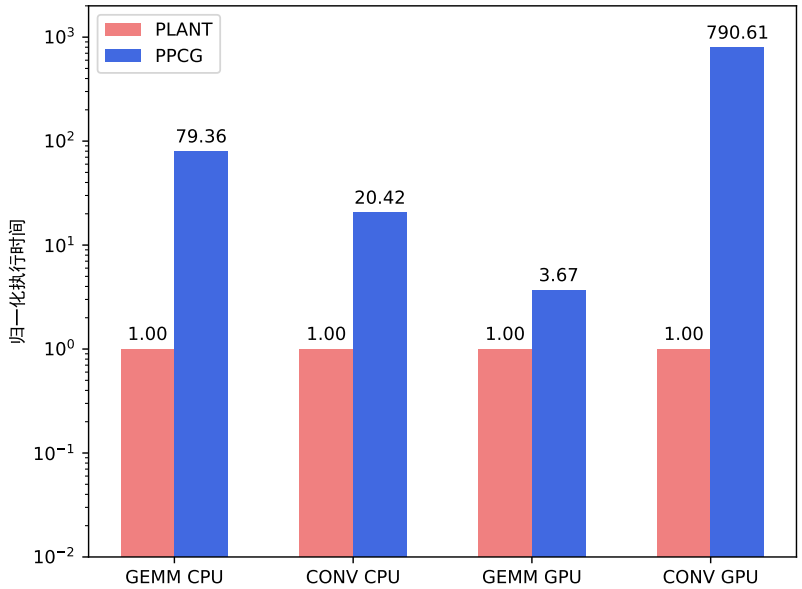
```
a = np.random.uniform(size=(M, N)).astype(np.float32)
b = np.random.uniform(size=(N, K)).astype(np.float32)
c = np.dot(a, b)

a_gpu = Array.from_np(a).to_gpu()
b_gpu = Array.from_np(b).to_gpu()
c_gpu = Array.alloc(c.shape, ty=plant.F32, loc=plant.GPU)
f = Func("./matmul_gpu.so")
f(a_gpu, b_gpu, c_gpu)
```

算子性能



对比算子库

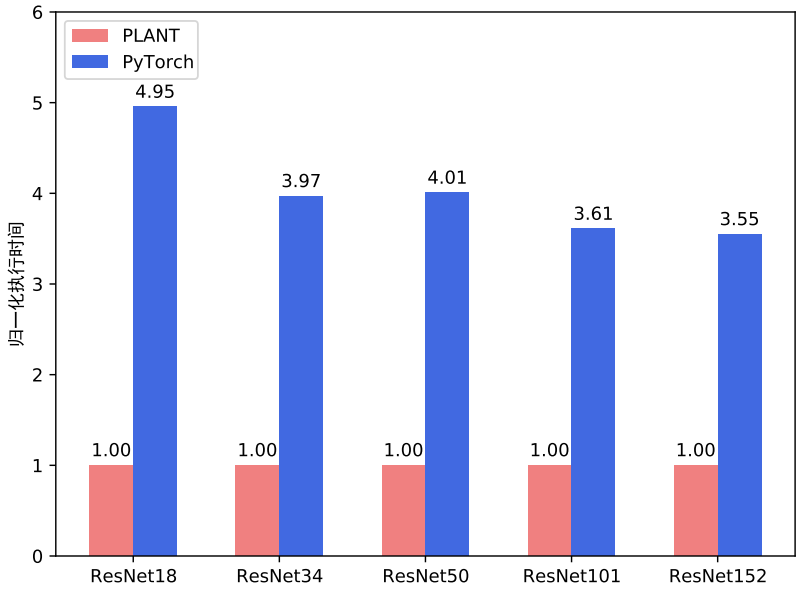
MKL [9] cuBLAS [13] cuDNN [6]



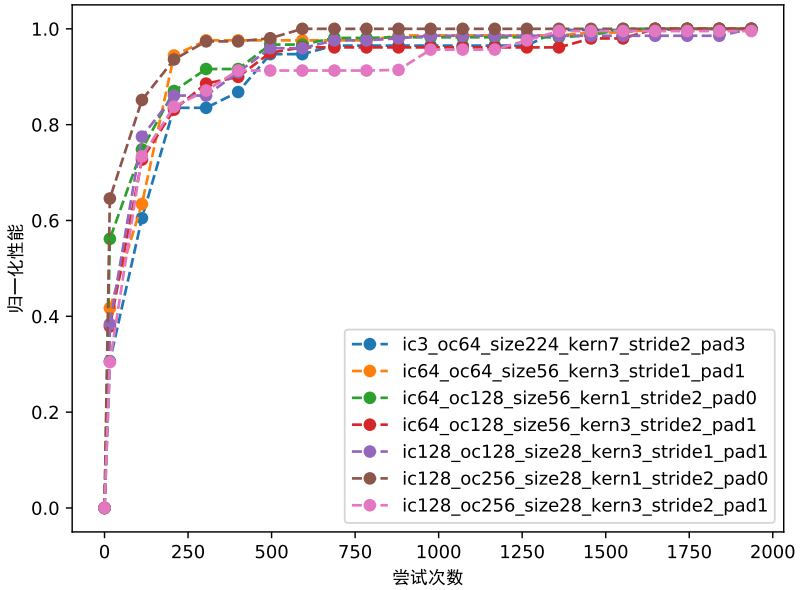
对比 PPCG [17]

神经网络推理

- 使用 CPU 上的 ResNet [8] 评测神经网络推理性能



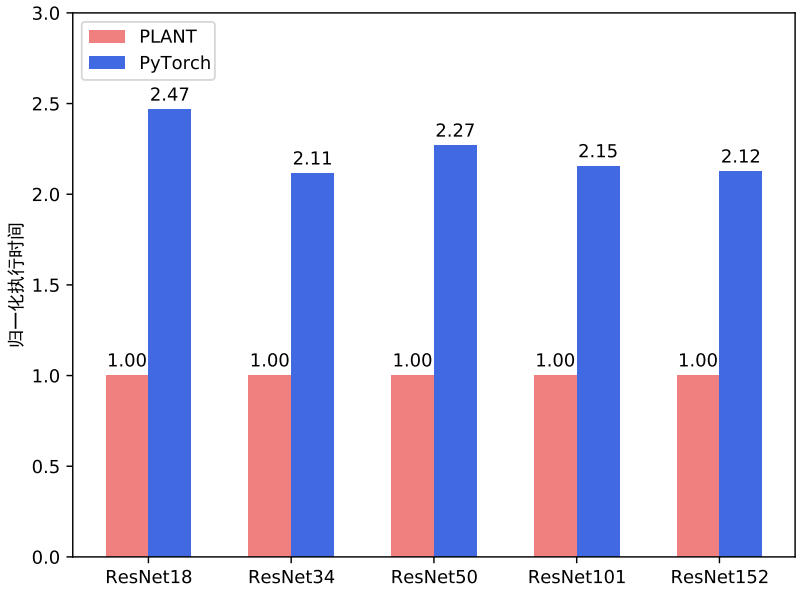
对比 PyTorch [14]



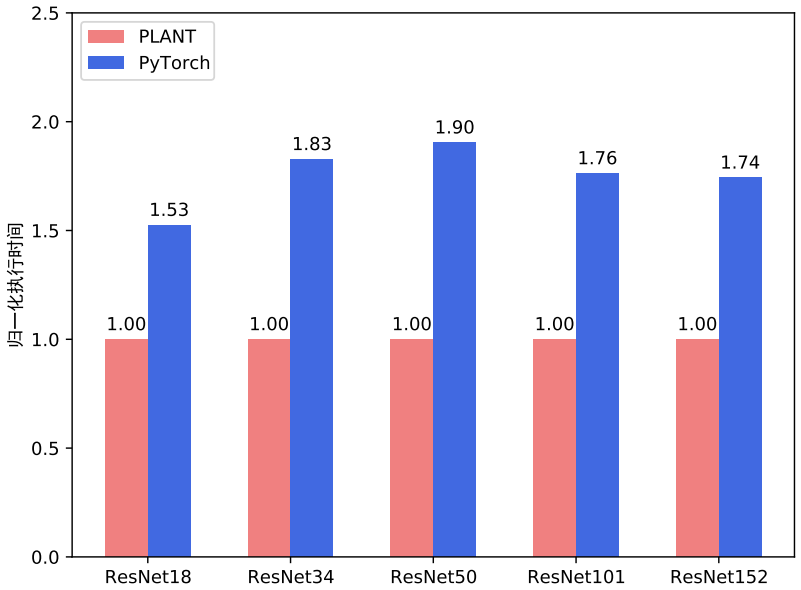
自动调度性能曲线

神经网络推理

- 利用远程执行，在嵌入式 CPU 瑞芯微 RK3399 上运行推理



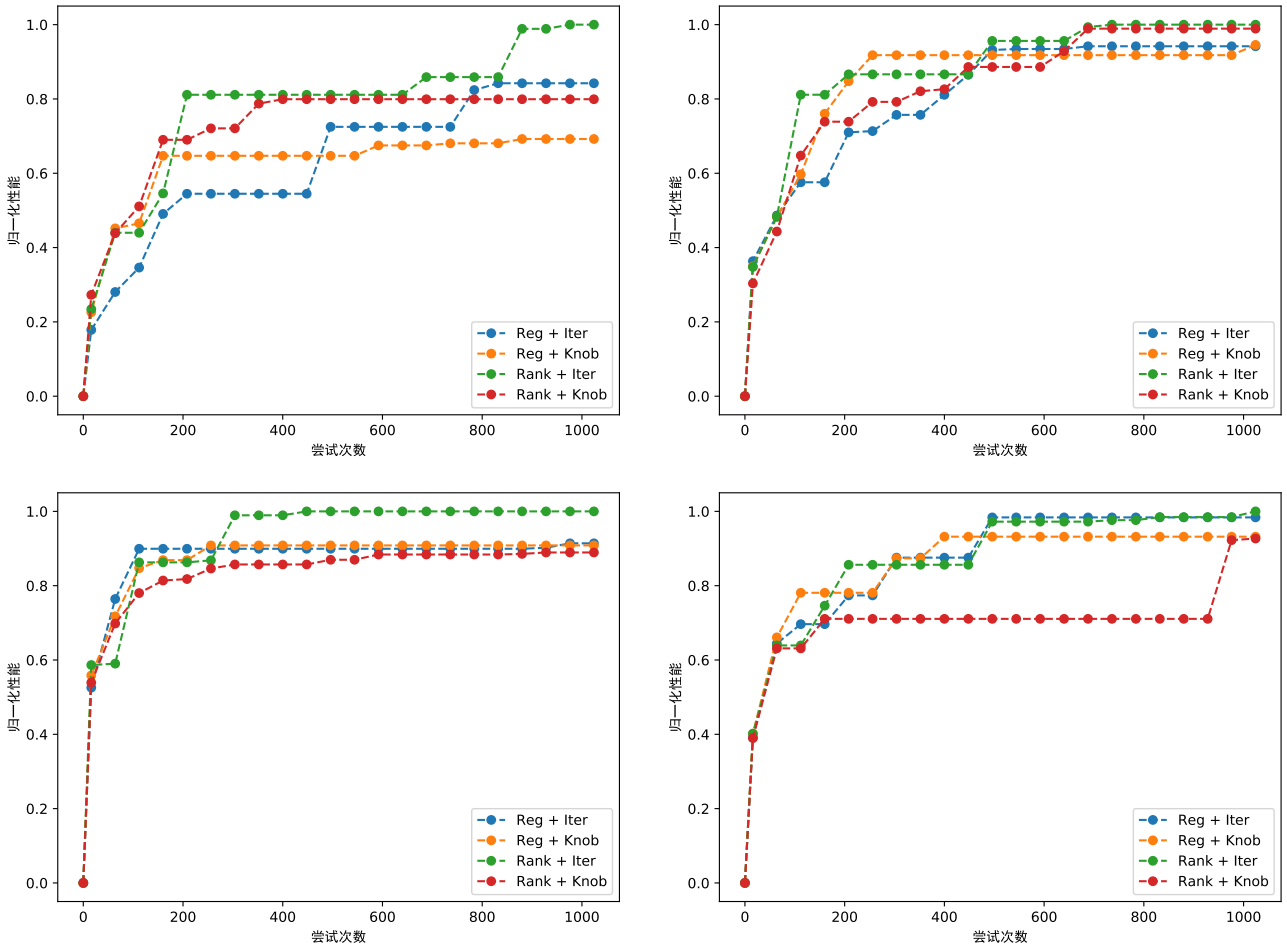
全部核心



大核心

比较自动调度不同方法

- 选择两种特征抽取方法（Knob 和 Iter）和两种损失函数（Reg 和 Rank）的四种组合



- 基于多面体模型的张量编译器 PLANT[1]
  - 丰富的调度指令
  - 支持 CPU 和 GPU 后端
  - 使用基于统计数据的性能模型实现自动调度
  - 远程执行和 Python 接口
- 未来工作
  - 提供更多调优过的张量算子代码示例
  - 实现更多调度指令，如封装 GPU 张量化接口
  - 支持更多后端，如分布式系统，FPGA
  - 为常见的深度学习前端框架设计前端，解析模型文件
  - 提供使用教程，完善代码注释和文档，方便二次开发

[1]已经在 GitHub 上开源 https://github.com/mashplant/plant

[1] R. Baghdadi, J. Ray, M. B. Romdhane, E. Del Sozzo, A. Akkas, Y. Zhang, P. Suriana, S. Kamil, and S. Amarasinghe. Tiramisu: A polyhedral compiler for expressing fast and portable code. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, CGO 2019, page 193–205. IEEE Press, 2019. ISBN 9781728114361.

[2] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, page 101–113, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781595938602. doi: 10.1145/1375581.1375595. URL https://doi.org/10.1145/1375581.1375595.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

[4] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy. Tvm: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 579–594, USA, 2018. USENIX Association. ISBN 9781931971478.

[5] T. Chen, L. Zheng, E. Yan, Z. Jiang, T. Moreau, L. Ceze, C. Guestrin, and A. Krishnamurthy. Learning to optimize tensor programs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3393–3404, Red Hook, NY, USA, 2018. Curran Associates Inc.

[6] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[7] M. Hall, J. Chame, C. Chen, J. Shin, G. Rudy, and M. Khan. Loop transformation recipes for code generation and auto-tuning. volume 5898, pages 50–64, 10 2009. ISBN 978-3-642-13373-2. doi: 10.1007/978-3-642-13374-9_4.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Intel(R). Oneapi math kernel library. Website. `https://software.intel.com/en-us/mkl`.

[10] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM (JACM)*, 14(3):563–590, 1967.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[12] L. Lamport. The parallel execution of do loops. *Communications of the ACM*, 17(2):83–93, 1974.

[13] NVIDIA(R). Cublas library. Website. https://developer.download.nvidia.cn/compute/DevZone/docs/html/CUDALibraries/doc/CUBLAS_Library.pdf.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[15] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *SIGPLAN Not.*, 48(6):519–530, June 2013. ISSN 0362-1340. doi: 10.1145/2499370.2462176. URL https://doi.org/10.1145/2499370.2462176.

[16] S. Verdoolaege. isl: An integer set library for the polyhedral model. In K. Fukuda, J. v. d. Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software – ICMS 2010*, pages 299–302, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15582-6.

[17] S. Verdoolaege, J. Carlos Juega, A. Cohen, J. Ignacio Gómez, C. Tenllado, and F. Catthoor. Polyhedral parallel code generation for cuda. *ACM Trans. Archit. Code Optim.*, 9(4), Jan. 2013. ISSN 1544-3566. doi: 10.1145/2400682.2400713. URL https://doi.org/10.1145/2400682.2400713.

[18] T. Yuki, G. Gupta, K. Daegon, T. Pathan, and S. Rajopadhye. Alphaz: A system for design space exploration in the polyhedral model. pages 17–31, 01 2013. doi: 10.1007/978-3-642-37658-0_2.

[19] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, et al. Ansor: Generating high-performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 863–879, 2020.

[20] S. Zheng, Y. Liang, S. Wang, R. Chen, and K. Sheng. Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 859–873, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378508. URL https://doi.org/10.1145/3373376.3378508.

Thanks!