

Loo.py

基于循环的代码生成转换工具

何宪航 | PB14011082

范子健 | PB14209127











韦清 | PB15000027

样例演示

GitHub repo链接: [Jupyter Notebook](#)

jupyter loo.py_run Last Checkpoint: 11小时前 (autosaved) Python Logo Logout

File Edit View Insert Cell Kernel Widgets Help Trusted | loopyenv ⌵

         Code ⌵ 

```
In [1]: import numpy as np
import pyopencl as cl
import pyopencl.array
import pyopencl.clrandom
# Loopy currently requires on pyopencl
import loopy as lp
lp.set_caching_enabled(False)
from warnings import filterwarnings, catch_warnings
filterwarnings('error', category=lp.LoopyWarning)
ctx = cl.create_some_context(interactive=False)
queue = cl.CommandQueue(ctx)
# Set up pyopencl.Context & CommandQueue
```

Loopy简介

- * Loopy是帮助生成较为繁琐、复杂的代码的一种自动化工具。这些代码是实现GPU和多核CPU上的高性能计算所必须的。
- * Loopy的核心思想是用简单的方法描述一个计算，然后将这个计算转换成能够实现高性能的版本。这种转换是在用户的控制之下，以Python为宿主语言实现的。

Loopy支持的优化

- * Vector and multi-core parallelism in the OpenCL/CUDA model
- * Data layout transformations (structure of arrays to array of structures)
- * Loopy Unrolling
- * Loop tiling with efficient handling of boundary cases
- * Prefetching/copy optimizations
- * Instruction level parallelism
- * and many more

高效的实现数组上的计算：

- * dense linear algebra,
- * convolutions,
- * n-body interactions,
- * PDE solvers, such as finite element, finite difference, and Fast-Multipole-type computations

Loopy与传统编译器的区别

- * 传统编译器中，无论是否有用户注释的帮助，都等价地将用户代码重写成更高性能的变体。
- * Loo.py代码通常嵌入在高级编程语言Python的外部控制程序中。Loo.py要求用户在宿主语言中采取基于多面体模型的方式，具体描述计算模型。用户定义的计算模型被存储在宿主语言的对象中（如loopy kernel，类似于tensorflow session），而代码之间仅存在偏序关系，以便于代码生成器进行深度优化。

Loopy与传统编译器的区别

- * 代码的中间表示是有意开放给用户，并由用户进行检查和操作的。高级用户可以轻松实现自定义的转换，以此扩充已有的转换库。
- * 指令、循环边界和转换，在一起唯一地指定要生成的代码。Loopy不会试图自作主张地代表用户做出选择，而是同时保留一个足够高层的接口，以便开发人员/有一定技术能力的终端用户使用。

Loopy与传统编译器的区别

- * 传统的编译器需要证明它们对源程序进行的任何重写都不会在可观察到的层面上改变程序的运行行为，这给编译器带来了很大的负担。而Loo.py中，通过显式地调用一些转换，允许了更多的灵活性
- * 与传统的编译器指令不同，Loo.py中的转换是嵌入在宿主语言(Python)中的，这意味着代码生成可以对于目标机器的硬件或者目前的工作负载的不同或变化作出反应。

Loo.py的数据模型

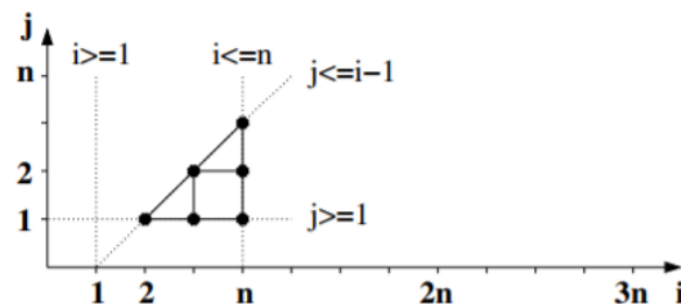
- * 类似于多面体模型
- * Loo.py内核包含两个主要的组成部分：
 - * 循环域(loop domain)
 - * 指令

多面体模型：以一段Fortran程序为例

```

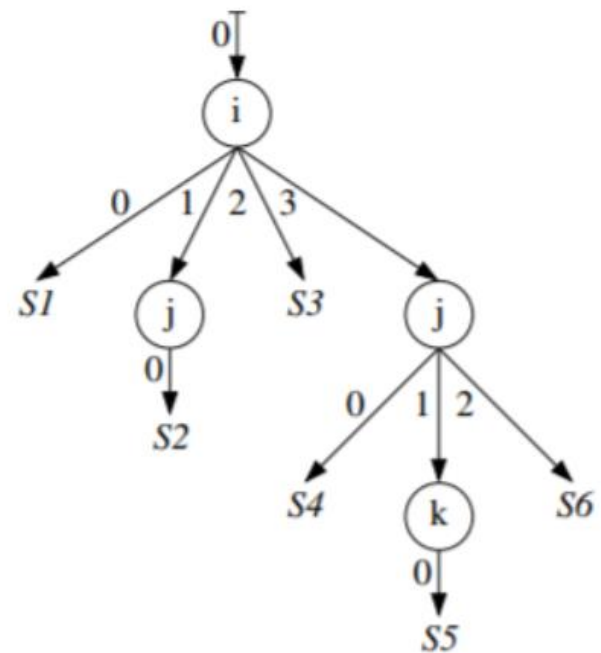
do i = 1, n
S1      x = a(i,i)
        do j = 1, i - 1
S2          x = x - a(i,j) ** 2
S3      p(i) = 1.0 / sqrt(x)
        do j = i + 1, n
S4          x = a(i,j)
            do k = 1, i - 1
S5                x = x - a(j,k) * a(i,k)
S6          a(j,i) = x * p(i)
        enddo
    enddo
enddo

```



多面体模型： 以一段Fortran程序为例

```
do i = 1, n
S1    x = a(i,i)
      do j = 1, i - 1
S2        x = x - a(i,j) ** 2
S3    p(i) = 1.0 / sqrt(x)
      do j = i + 1, n
S4        x = a(i,j)
          do k = 1, i - 1
S5              x = x - a(j,k) * a(i,k)
S6          a(j,i) = x * p(i)
```



Loo.py的数据模型

- * 类似于多面体模型
- * Loo.py kernel包含两个主要的组成部分:

- * 循环域(loop domain)

```
kn1 = loopy.make_kernel(  
    "{ [i]: 0<=i<n }", #loop domain  
    "out[i] = 2*a[i]" #instructions  
)
```

- * 指令

```
kn1 = loopy.make_kernel(  
    "{ [i,j,ii,jj]: 0<=i,j,ii,jj<n }",  
    ""  
    out[j,i] = a[i,j] {id=transpose}  
    out[ii,jj] = 2*out[ii,jj] {dep=transpose}  
    ""  
)
```

The background of the slide features a faint, artistic illustration of a traditional Chinese landscape painting, possibly a 'Shan Shui' (mountain-water) scene, rendered in a style reminiscent of a fan painting. The scene depicts misty mountains, a winding river, and small figures or structures nestled in the valleys. The overall color palette is muted, with soft greens, greys, and browns. The title text is centered over this background.

Introduce to TVM

TVM 简介

- * TVM 是陈天奇团队发布的开源项目
- * 它作为神经网络和硬件后端之间的共同层，消除了为每类设备或者服务器优化各自基础架构的需要，有了 TVM，业界与学界开发者们可以快速、轻松地在各个系统（包括手机、嵌入式设备与低功耗芯片）上部署深度学习应用程序，同时无须担心资源与速度的限制。

应用

- * TVM 是一个全新的框架，它可以解决如下问题：
- * >1. 优化 CPU、GPU 和其他专业化硬件的常规深度学习的计算量
- * >2. 自动转换计算图以最小化内存利用率，优化数据布局，融合计算模式
- * > 3. 提供从现有的前端框架到裸机硬件的端到端编译，一直到浏览器可执行的 Javascript
- * 有了TVM 的帮助，只需少量额外工作我们即可轻易地在手机端、嵌入式设备甚至浏览器上运行深度学习任务

TVM的技术细节

- * TVM 堆栈的目标在于提供一个可重复使用的工具链来将高级神经网络描述从深度学习框架前端向下编译为多硬件后端的低级机器代码。
- * 挑战在于支持多个硬件后端，同时将计算、内存和能源足迹（energy footprint）保持在最低水平。TVM项目团队借鉴了编译器社区的共同理念，以弥合大量深度学习框架和硬件后端之间的差距：构建了两级由NNVM和TVM共同组成的中间层。其中前者是用于任务调度和内存管理的顶层中间代码，而后者是用于优化计算内核的、表达力强大的底层中间代码。

多语言 and 平台支持

- * TVM 的众多优势之一在于它可以支持多种语言和平台
- * 可以在包括 Android、iOS、树莓派和 Web 浏览器在内的平台上直接运行诸如 Javascript、Java、Python 和 C++ 等语言的 TVM 编译代码

远程部署和执行

- * TVM 支持使用 TVM PRC 进行交叉编译，测试嵌入式设备，这是一种轻量级界面，用于在远程嵌入式设备上部署和执行 TVM 交叉编译模块。TVM 用户提供了易用的高级 Python 界面，用于在各种低级嵌入式设备上远程编译、优化，并测试深度学习算法。

lco.py在TVM中的应用

- * 当我们做GPU 优化时，往往会着重于处理 data reuse, shared memory 以及 bank conflicts
- * 以depthwise convolution为例
- * 很自然地想到将大规模的块划分为小规模块，利用 cache 访存的局部性，减小 cache missing

```
IS = s.cache_read(PaddedInput, "shared", [DepthwiseConv2d])
FS = s.cache_read(Filter, "shared", [DepthwiseConv2d])
block_y = tvm.thread_axis("blockIdx.y")
block_x = tvm.thread_axis("blockIdx.x")
# bind the dimension of batch (N in NCHW) with block_y
s[Output].bind(Output.op.axis[0], block_y)
# bind the dimension of channel (C in NCHW) with block_x
s[Output].bind(Output.op.axis[1], block_x)
```

- * 在GTX 1080 上运行代码，并与depthwise_conv2d in tensorflow运行的结果做比较，得到如下结果：

*

Input	Filter	stride	tf-1.2 SAME pad (us)	TVM SAME pad (us)
[1, 256, 21, 21]	[256, 1, 3, 3]	[1, 1]	16.1	9.1
[1, 256, 32, 32]	[256, 1, 3, 3]	[1, 1]	34.8	14.5
[1, 256, 64, 64]	[256, 1, 3, 3]	[1, 1]	130.9	98.9
[1, 256, 96, 96]	[256, 1, 3, 3]	[1, 1]	251.6	387.4

* 在TVM中，这一优化是基于loopy实现的



谢谢!

01-loo.py

基于循环的代码生成转换工具

何宪航 | PB14011082

范子健 | PB14209127

韦清 | PB15000027