

# Исследование и развитие технологий автогенерации кода для GPU в применении к задачам тензорной аппроксимации. Алогоритм ALS

Кузнецов М.А.

20 мая 2013

# Утверждения

- Тензорные алгоритмы сложные и требуют больших вычислительных затрат
- GPU обладают большой мощностью в плане вычислений

Отсюда логичный вывод — попытаться писать код алгоритма для GPU

# Код для GPU — особенности

У программирования для GPU есть свои плюсы и минусы

“+”

- Возможность использования множества процессоров
- Возможность серьезного ускорения программы

“-”

- Очень длительный процесс написания кода
- Сложность написания кода
- Платформенно-зависим

## Пример кода для GPU

```
__kernel void __attribute__((reqd_work_group_size(16, 1, 1)))  
{  
    float acc_j_outer_j_inner_k_outer_k_inner;  
  
    if ((-1 + -16 * gid(0) + -1 * lid(0) + n) >= 0)  
    {  
        acc_j_outer_j_inner_k_outer_k_inner = 0.0f;  
        for (int j_inner = 0; j_inner <= 15; ++j_inner)  
            for (int k_inner = 0; k_inner <= 15; ++k_inner)  
                for (int k_outer = 0; k_outer <= (-1 + -1 * k_inner)  
                    for (int j_outer = 0; j_outer <= (-1 + -1 * j_inner)  
                        acc_j_outer_j_inner_k_outer_k_inner = acc_j_outer_j_inner_k_outer_k_inner + f[n * (lid(1) + gid(1)) + lid(0) + gid(0) * 16] = acc_j_outer_j_inner_k_outer_k_inner;  
    }  
}
```

# Идея автоматической генерации

Код на OpenCL сложный, нужно учитывать барьеры, разбиение и вычислительные сетки. Поэтому логично использовать пакеты автоматической генерации кода. Использован новый, активно разрабатывающийся пакет loo.py

# Почему Python

Вообще говоря Python медленный язык. Почему же его можно использовать для ускорения работы программ? Однако:

- Python обладает большим количеством стандартных модулей
- Использование модулей дает возможность получить скорость С-кода
- Простой синтаксис языка позволяет быстро писать программы даже для сложных алгоритмов
- Экономит время программиста

# Пакет Loo.py

Пакет loo.py разрабатывается Андреасом Клекнером (Andreas Kloeckner). <http://gitlab.tiker.net/inducer/loopy> Вышеуказанный репозиторий — закрытый, однако автор включен в список “разработчиков”. Стабильную версию можно найти здесь <http://git.tiker.net/loopy.git>

Пакет предназначен для “развертки” циклов и последующей генерации OpenCL кода. Для этого нужно сформулировать ядро в специальном синтаксисе.

# Как правильно сформулировать алгоритм

Для применения пакета loo.py алгоритм должен быть формализован и записан в виде вложенных циклов:

```
for i in xrange(...):  
    for j in xrange(...):  
        .....  
        for k n xrange(...):  
            .....
```



# Пример ядра

```
def Prav_U(ctx):
    order='C'
    dtype = np.float32
    knl = lp.make_kernel(ctx.devices[0],
    [
        "{[i,j,k,alpha]: 0<=alpha<r and 0<=i,j,k<n}",
    ],
    [
        "f[alpha,i]=sum((j,k), a[i,j,k]*v[alpha,j]*w[alpha,k])"
    ],
    [
        lp.GlobalArg("a", dtype, shape="n,n,n", order=order),
```

```
lp.GlobalArg("v", dtype, shape="r, n", order=order),
    lp.GlobalArg("w", dtype, shape="r, n", order=order),
    lp.GlobalArg("f", dtype, shape="r, n", order=order),
    lp.ValueArg("n", np.int64),
    lp.ValueArg("r", np.int64),
],
assumptions="n>=1")
knl = lp.split_iname(knl, "i", 16, outer_tag="g.0", inner_tag="g.1")
knl = lp.split_iname(knl, "alpha", 1, outer_tag="g.1", inner_tag="g.2")
knl = lp.split_iname(knl, "j", 16)
knl = lp.split_iname(knl, "k", 16)
print lp.CompiledKernel(ctx, knl).get_highlighted_code()
return knl
```

# Актуальность исследования

Привлекательность исследования обусловлена несколькими факторами:

- 1 Тензорные алгоритмы начали активно разрабатываться в последнее время
- 2 Написание GPU-кода — сложная задача, существует необходимость исследовать возможности автогенерации GPU-кода
- 3 Вычислительная мощность GPU превосходит многоядерные CPU, использование GPU эффективней

Ввиду того, что процесс написания GPU-кода вручную длительный и трудоемкий, хоть и эффективный, в вычислительных задачах хотелось бы использовать следующий “идеальный” способ его написания:

- 1 Использование в динамических языках (Python)
- 2 Автоматическое распараллеливание стандартных задач (циклов), генерация OpenCL/CUDA-кода
- 3 Высокая эффективность

# Цель курсовой работы

- Научится использовать пакет `loo.py`
- С помощью пакета получить эффективную параллельную реализацию алгоритма ALS
- Научится “параллелить” тензорные алгоритмы

## Метод ALS: идея

Основная идея алгоритма, состоит в том, чтобы фиксировать все факторы, кроме одного, канонического разложения и искать минимум функционала

$$F = \sum_{i,j,k=1} (A_{ijk} - \sum_{\alpha=1}^r U_{i\alpha} V_{j\alpha} W_{k\alpha})^2.$$

только по нему. Путем циклических перестановок, используя уже полученные факторы, строятся последующие, до тех пор, пока не будет достигнута требуемая точность аппроксимации или, пока не сработают другие критерии остановки алгоритма

# Формулы метода ALS

Найдем частную производную функционала  $F$  по  $U_{i\hat{\alpha}}$  и приравняем ее к 0:

$$\frac{\partial F}{\partial U_{i\hat{\alpha}}} = 2 \left( \sum_{i,j,k} (A_{ijk} - \sum_{\alpha} U_{i\alpha} V_{j\alpha} W_{k\alpha}) \right) \left( - \sum_{\check{\alpha}} (V_{j\check{\alpha}} W_{k\check{\alpha}}) \frac{\partial U_{i\check{\alpha}}}{\partial U_{i\hat{\alpha}}} \right) = 0;$$

$$\frac{\partial U_{i\check{\alpha}}}{\partial U_{i\hat{\alpha}}} = \delta_{i,\hat{i}} \delta_{\check{\alpha}\hat{\alpha}};$$

Окончательно, получаем следующие соотношения:

$$\sum_{j,k} A_{ijk} V_{j\hat{\alpha}} W_{k\hat{\alpha}} = \sum_{j,k,\alpha} U_{i\alpha} V_{j\alpha} W_{k\alpha} V_{j\hat{\alpha}} W_{k,\hat{\alpha}},$$

# Формулы ALS

Обозначим через  $M_{\alpha\hat{\alpha}}$  матрицу с элементами

$$M_{\alpha,\hat{\alpha}} = \left( \sum_j V_{j,\alpha} V_{j,\hat{\alpha}} \right) \left( \sum_k W_{k\alpha} W_{k\hat{\alpha}} \right); \quad (1)$$

тогда

$$\sum_{\alpha} U_{i,\alpha} M_{\alpha,\hat{\alpha}} = \sum_{j,k} A_{i,j,k} V_{j,\hat{\alpha}} W_{k,\hat{\alpha}}; \quad (2)$$

Через  $F_{i,\hat{\alpha}}$  обозначим правую часть. Тогда, имеем

$$\sum_{\alpha} U_{i\alpha} M_{\alpha\hat{\alpha}} = F_{i\hat{\alpha}}. \quad (3)$$



# В виде системы

или в виде системы линейных уравнений

$$UM = F. \quad (4)$$

где  $M \in \mathbb{R}^{r \times r}$ .

В ходе экспериментов использовались следующие платформы:

- Мобильная видеокарта NVIDIA
- Мобильный процессор Intel Core i5
- Кластер Tesla IBM РАН

## Характеристики Tesla

Device Tesla C2070

NAME:	Tesla C2070
VENDOR:	NVIDIA Corporation
VERSION:	304.54
VERSION:	OpenCL 1.1 CUDA
OPENCL_ C_ VERSION:	OpenCL C 1.1
MAX_ WORK_ GROUP_ SIZE:	1024
ADDRESS_ BITS:	32
MAX_ MEM_ ALLOC_ SIZE:	1343 MByte
GLOBAL_ MEM_ SIZE:	5375 MByte

## Таблицы времен

Для фиксированного ранга  $r = 3$  и размерности тензора  $n$  исследована скорость выполнения как отдельных ядер, так и всего алгоритма ALS. Однако ALS алгоритм не гарантирует сходимость, только убывание невязки, поэтому будем указывать только время выполнения одной итерации.

Приведем таблицу с временем выполнения.

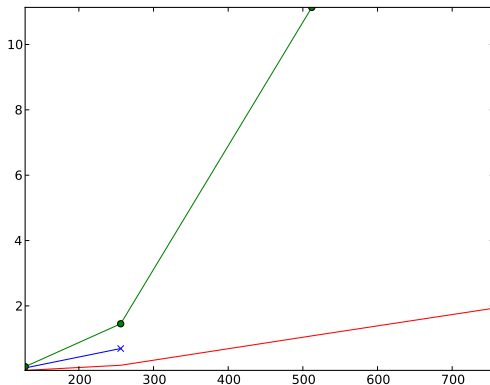
размер $n$	128	256	512	756
$t_r$	0.013803	0.08674	0.65225	0.92513
$t_l$	0.00035595	0.0004210	0.000552	0.000673
$t_{\text{solve}}$	0.00025391	0.00025510	0.000256	0.000256
LU	0.00024890	0.0002851	0.00035	0.000391
$T_i$	0.026740	0.1834	1.08289	1.92985

Приведем также таблицу с временем выполнения одной итерации программы, вычисления правой части в зависимости от ранга  $r$  и фиксированной размерности тензора  $n = 128$

ранг $r$	3	6	10	20
$t_r$	0.01380	0.0152	0.0162	0.0184
$T_i$	0.04326	0.0437	0.0468	0.0556

## График

Для наглядности также построим графики поведения времени вычисления правой части на CPU, мобильном GPU и Tesla:



В ходе выполнения курсовой работы были получены следующие результаты:

- ❶ Изучен пакет автоматической генерации OpenCL-кода
- ❷ Реализованы алгоритмы:
  - LU-разложения, решения систем в стандартном виде LU
  - подсчета правой части алгоритма ALS
  - ALS-алгоритм

## Важные выводы

Ключевые выводы:

- 1 Генерировать OpenCL код можно автоматически
- 2 Сильно экономится время, а качество реализации не страдает
- 3 Можно избежать ошибок “технического” характера
- 4 Можно параллелить произвольный алгоритм, записанный в нужном формате



# Вопросы

Спасибо за внимание! Ваши вопросы?