Библиотека параллельных алгоритмов для C++

Лекция 6

Thrust – это

– STL – подобная библиотека обработки данных на GPU;
– Реализация как для GPU, так и для CPU;
– Открытый проект, поддерживаемый NVIDIA;
– Разработчики: Nathan Bell И др.

Основные возможности

– унифицированный интерфейс для выполнения типичных задач обработки данных;
– уделяется внимание производительности;
– по сравнению с CUDA C, возможности тонкого контроля (например, разделяемая память) не так богаты;
– дизайн, схожий с STL: контейнеры, итераторы, алгоритмы.

Компоненты

- Контейнеры
  - Управление памятью на host и device
  - Упрощенный обмен данными
- Итераторы
  - Подобны указателям
  - «Следят» за областью памяти (host или device)
- Алгоритмы
  - Применяются к контейнерам и итераторам

Распространение

Входит в состав CUDA 4+

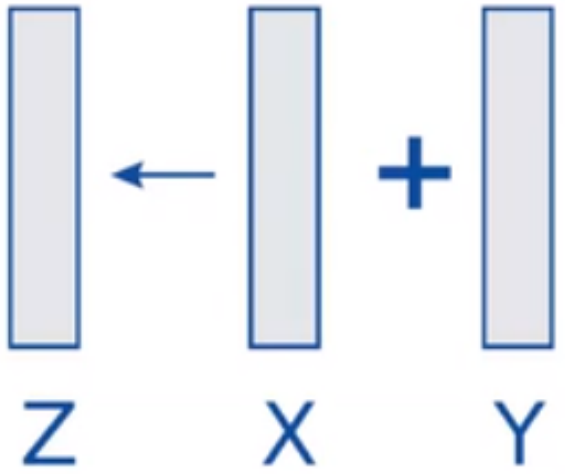
Хостинг кода: GitHub

git clone https://github.com/thrust/thrust.git

Пример 1. Сложение векторов

Простое сложение векторов в Thrust:

```
for (int i = 0; i < N; ++i)
  Z[i] = X[i] + Y[i];
```



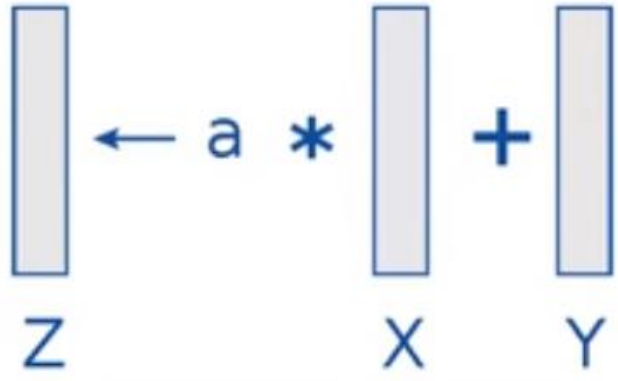
Пример 1. Сложение векторов

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
 #include <thrust/functional.h>
#include <iostream>

int main(void) {

   thrust::device_vector<float> X(3);
  thrust::device_vector<float> Y(3);
  thrust::device_vector<float> Z(3);

   X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;

  thrust::transform(X.begin(), X.end(),
    Y.begin(),
    Z.begin(),
     thrust::plus<float>());

   for (size_t i = 0; i < Z.size(); i++)
    std::cout << "Z[" << i << "] = " << Z[i] << "\n";
}

return 0;
```

Пример 2. SAXPY

SAXPY: z <- a*x + y

```
for (int i = 0; i < N; ++i)
  Z[i] = a * X[i] + Y[i];
```



Функторы в C++

```
#include <iostream>
#include <string>

class SimpleFunctor {
    std::string name_;
public:
    SimpleFunctor(const char *name) : name_(name) {}
    void operator()() { std::cout << "Oh, hello, " << name_ << endl; }
};

int main() {
    SimpleFunctor sf("catonmat");
    sf();  // выводит "Oh, hello, catonmat"
}
```

Пример 2. SAXPY – класс-функтор

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <iostream>

struct saxpy {

  float a;

  saxpy(float a) : a(a) {}

  __host__ __device__ float operator()(float x, float y) {
    return a * x + y;
   }

};
```

Пример 2. SAXPY – main

```
int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;

  float a = 2.0f;

  thrust::transform(X.begin(), X.end(),
          Y.begin(),
          Z.begin(),
           saxpy(a));

  for (size_t i = 0; i < Z.size(); i++)
    std::cout << "Z[" << i << "] = " << Z[i] << "\n";

  return 0;

}
```

Пример 3. SAXPY , λ-выражения (Thrust 1.5+)

```
using namespace thrust::placeholders;

int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;

  float a = 2.0f;

  thrust::transform(X.begin(), X.end(),
          Y.begin(),
          Z.begin(),
           a * _1 + _2);

  for (size_t i = 0; i < Z.size(); i++)
    std::cout << "Z[" << i << "] = " << Z[i] << "\n";

}
```

Типы трансформации

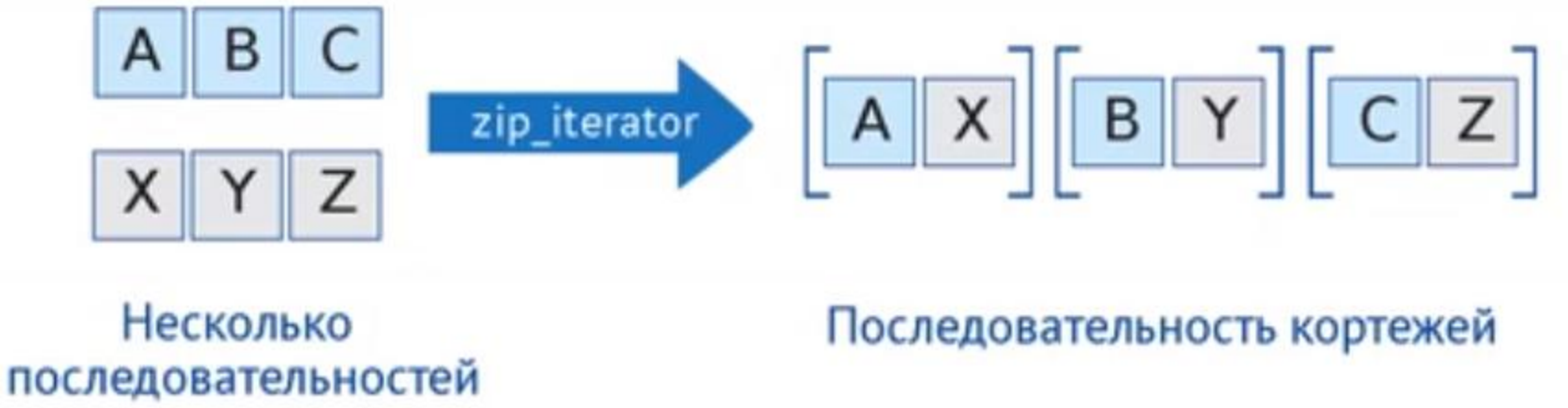
Унарная: $X[i] = f(A[i])$

Бинарная: $X[i] = f(A[i], B[i])$

Тернарная: $X[i] = f(A[i], B[i], C[i])$

Обобщённая: $X[i] = f(A[i], B[i], C[i], \ldots)$

zip - итераторы



Пример 4. Тернарная трансформация

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

struct linear_combo {

   __host__ __device__ float operator()(thrust::tuple<float,float,float> t) {

    float x, y, z;

     thrust::tie(x,y,z) = t;

     return 2.0f * x + 3.0f * y + 4.0f * z;

  }

};
```

Пример 4. Тернарная трансформация

```
int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);
  thrust::device_vector<float> U(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;
  Z[0] = 20; Z[1] = 30; Z[2] = 25;

  thrust::transform(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin(), Z.begin())),
     thrust::make_zip_iterator(thrust::make_tuple(X.end(),
    Y.end(),
    Z.end())),
    U.begin(),
    linear_combo());

  for (size_t i = 0; i < Z.size(); i++)
  std::cout << "U[" << i << "] = " << U[i] << "\n";

  return 0;

}
```

Пример 5. Сумма

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <iostream>

int main(void) {

  thrust::device_vector<float> X(3);

  X[0] = 10; X[1] = 30; X[2] = 20;

   float result = thrust::reduce(X.begin(), X.end());

  std::cout << "sum is " << result << "\n";

  return 0;

}
```

Пример 6. Поиск максимума

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <iostream>

int main(void) {

  thrust::device_vector<float> X(3);

  X[0] = 10; X[1] = 30; X[2] = 20;

  float init = 0.0f;

  float result = thrust::reduce(X.begin(), X.end(),
                  init,
                 thrust::maximum<float>());

  std::cout << "maximum is " << result << "\n";

  return 0;

}
```

Пример 7. Индекс максимума

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

typedef thrust::tuple<int,int> Tuple;

struct max_index {

  __host__ __device__ Tuple operator()(Tuple a, Tuple b) {

    if (thrust::get<0>(a) > thrust::get<0>(b))
      return a;
    else
      return b;
   }

};
```

Пример 7. Индекс максимума

```
int main(void) {

  thrust::device_vector<int> X(3), Y(3);

  X[0] = 10; X[1] = 30; X[2] = 20; // values
   Y[0] = 0; Y[1] = 1; Y[2] = 2;    // indices

   Tuple init(X[0],Y[0]);

  Tuple result = thrust::reduce(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(), Y.end())),
     init,
    max_index());

  int value, index;

  thrust::tie(value,index) = result;

  std::cout << "maximum value is " << value << " at index " << index << "\n";

  return 0;

}
```

Взаимодействие Thrust и CUDA

Преобразовать контейнер к обычному указателю:

```
thrust::device_vector<int> d_vec(4);

int* ptr = thrust::raw_pointer_cast(&d_vec[0]);

my_kernel<<< N / 256, 256 >>>(N, ptr);

cudaMemcpyAsync(ptr, ... );
```

Взаимодействие Thrust и CUDA

«Обернуть» обычный массив в спец. Контейнер Thrust:

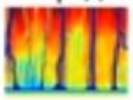
```
int *raw_ptr;

cudaMalloc((void**) &raw_ptr, N * sizeof(int));

thrust::device_ptr<int> dev_ptr(raw_ptr);

thrust::fill(dev_ptr, dev_ptr + N, (int) 0);

dev_ptr[0] = 1;

cudaFree(raw_ptr);
```

Дополнительные примеры на GitHub

- Monte Carlo Integration
- Run-Length Encoding
- Summed Area Table
- Moving Average
- Word Count
- VoronoiDiagram
- Graphics Interop
- Stream Compaction
- Lexicographical Sort
- Summary Statistics
- Histogram
- ...

Приложения Thrust

CUSP – операции с разреженными матрицами и вычисления на графах с поддержкой CUDA
http://code.google.com/p/cusp-library/



PETSc – параллельное решение научных задач, моделируемых дифференциальными
http://www.mcs.anl.gov/petsc/

Trilinos – объектно-ориентированная библиотека для решения сложных научных и инженерных задач
http://trilinos.sandia.gov/