Student Name: Галактионов Кирилл
Student ID: st067889

Современные технологии программирования в научных исследованиях II
Лабораторная работа 4

---

1 Постановка задачи

Условие задачи звучит следующим образом: "Выбрать и запустить пример из Intel DevCloud". Мною был выбран пример, реализующий поэлементное умножение комплексных векторов.

2 Описание тестового стенда

Вычисления проводились на выданном вычислительном узле:

- Операционная система: Ubuntu 18.04.3 LTS (Bionic Beaver)"
- Процессор: 10 x Intel Xeon Processor (Skylake, IBRS)
- Доступная оперативная память: 15 ГБ

Время работы алгоритмов вычислялось при помощи команды **clock()** из стандартной библиотеки **time** языка **C++**. Проводилось по три серии экспериментов, затем полученные значения времен усреднялись.

Проверка правильности суммирования осуществлялась сравнением с последовательным вычислением (время не включается в время выполнения).

3 Описание алгоритма решения задачи

Условие задачи требует вычислить поэлементное произведение комплексных элементов массивов, и сохранить результаты в третьем. В последовательном режиме задача решается простым проходом по массиву. С определенной операцией комплексного умножения. Код:

```
void Scalar(std::vector<Complex2> &in_vect1,
                std::vector<Complex2> &in_vect2,
                std::vector<Complex2> &out_vect) {
    if ((in_vect2.size() != in_vect1.size()) || (out_vect.size() != in_vect1.size())){
        std::cout<<"ERROR: Vector sizes do not match"<<"\n";
        return;
    }
    for (int i = 0; i < in_vect1.size(); i++) {
        out_vect[i] = in_vect1[i].complex_mul(in_vect2[i]);
    }
}
```

Идея использования SYCL в таком алгоритме заключается в том, что вычисления каждого элемента независимы и можно выполнять параллельно. Для этого используется класс **parallel_for**. Реализация алгоритма представлена ниже:

```
// in_vect1 and in_vect2 are the vectors with num_elements complex nubers and
// are inputs to the parallel function
void SYCLParallel(queue &q, std::vector<Complex2> &in_vect1,
                  std::vector<Complex2> &in_vect2,
                  std::vector<Complex2> &out_vect) {
    auto R = range(in_vect1.size());
    if (in_vect2.size() != in_vect1.size() || out_vect.size() != in_vect1.size()){
        std::cout << "ERROR: Vector sizes do not  match"<< "\n";
        return;
    }
    // Setup input buffers
    buffer bufin_vect1(in_vect1);
    buffer bufin_vect2(in_vect2);

    // Setup Output buffers
    buffer bufout_vect(out_vect);


    // std::cout << "Target Device: " << q.get_device().get_info<info::device::name>() <<
         "\n";


    // Submit Command group function object to the queue
    q.submit([&](auto &h) {
        // Accessors set as read mode
        accessor V1(bufin_vect1,h,read_only);
        accessor V2(bufin_vect2,h,read_only);
        // Accessor set to Write mode
        accessor V3 (bufout_vect,h,write_only);
        h.parallel_for(R, [=](auto i) {
            // call the complex_mul function that computes the multiplication of the
            // complex number
            V3[i] = V1[i].complex_mul(V2[i]);
        });
    });
    // Synchronize
    q.wait_and_throw();
}
```

4 Программный код

Полный код программы представлен в Приложении. В Приложении 1 в разделе 8 представлен код класса, реализующий операции с комплексными числами. В Приложении 2 в разделе 9 представлен код алгоритма решения задачи.

5 Результаты измерений

Вывод программы (из коробки) представлен в Приложении 3 в разделе 10. Построенная таблица времен выполнения - в Приложении 4 в разделе 11.

6 Анализ и обсуждение результатов

Рассмотрим затраты времени при проверке алгоритма на векторах разного размера. На рисунке 1 представлена зависимость времени, затраченного на вычисления, от размера массива. При вычислениях в последовательном режиме зависимость является линейной ($O(N)$), что наблюдается на всём интервале размеров массива. При вычислениях в параллельном режиме (с использованием SYCL), до размера массива $3*10^5$ элементов, затраты времени практически постоянные и много больше времени последовательных вычислений, что связано с накладными затратами. При увеличении размера массива, зависимость становится так же линейной. Угол наклона полученной зависимости меньше, чем для последовательного режима, и на размере массива в $2 * 10^6$ наблюдается выигрыш по времени параллельным режимом вычислений.

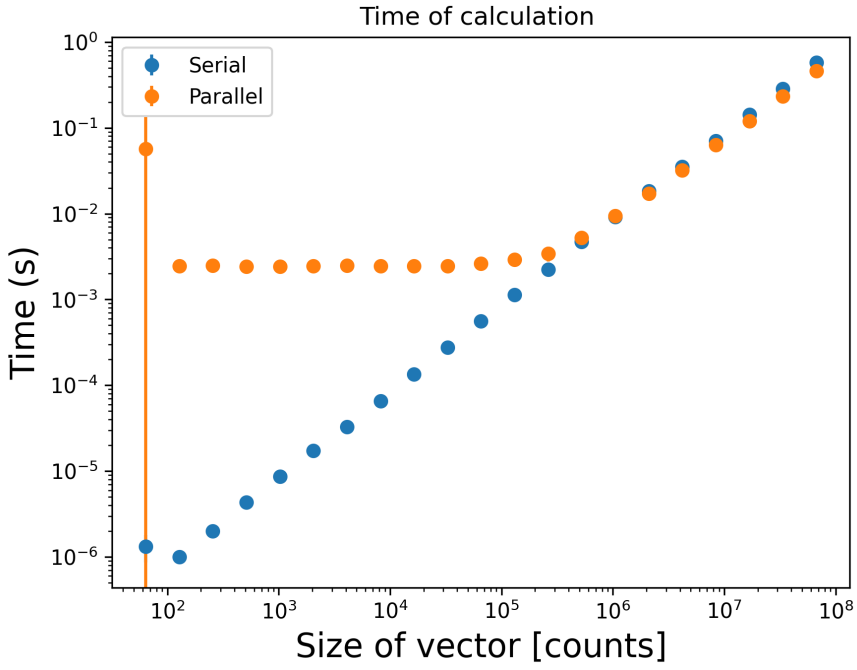

Рис. 1: Зависимость времени вычисления от размера массива.

7 Выводы

Поставленные задачи были выполнены. Был изучен инструмент Intel DevCloud, а также проведено сравнение скорости работы последовательного и параллельного выполнения программы.

8 Приложение 1. Код класса комплексных чисел

```
1 //==============================================================
2 // Copyright    2020 Intel Corporation
3 //
4 // SPDX-License-Identifier: MIT
5 // =============================================================
6
```

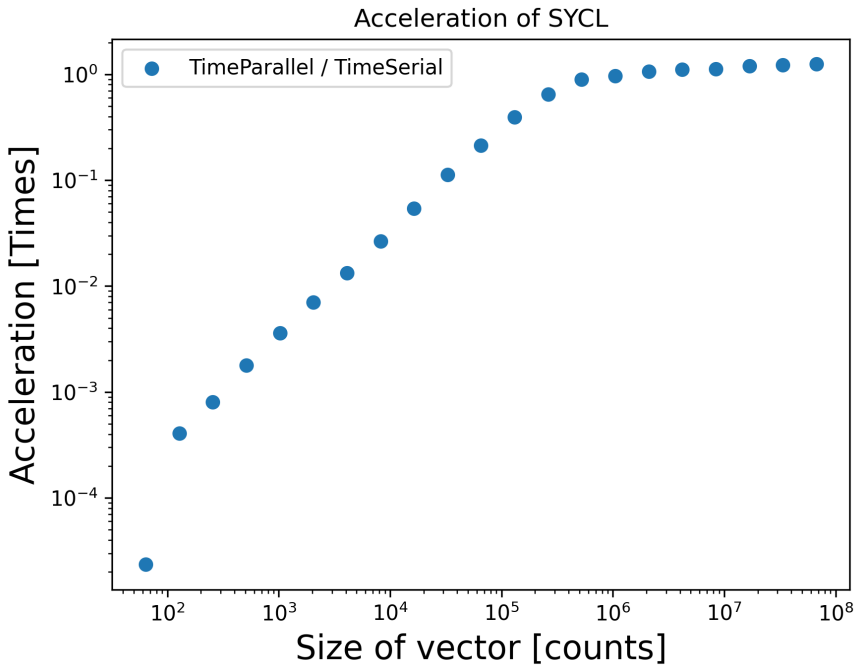

Рис. 2: Зависимость ускорения вычисления от размера массива.

```
#include <iostream>
#include <vector>
using namespace std;
class Complex2 {
 private:
  int m_real_, m_imag_;

 public:
  Complex2() {
    m_real_ = 0;
    m_imag_ = 0;
  }
  Complex2(int x, int y) {
    m_real_ = x;
    m_imag_ = y;
  }

  // Overloading the  != operator
  friend bool operator!=(const Complex2& a, const Complex2& b) {
    return (a.m_real_ != b.m_real_) || (a.m_imag_ != b.m_imag_);
  }

  // The function performs Complex number multiplication and returns a Complex2
  // object.
  Complex2 complex_mul(const Complex2& obj) const{
    return Complex2(((m_real_ * obj.m_real_) - (m_imag_ * obj.m_imag_)),
                    ((m_real_ * obj.m_imag_) + (m_imag_ * obj.m_real_)));
  }
```

```
35
36    // Overloading the ostream operator to print the objects of the Complex2
37    // object
38    friend ostream& operator<<(ostream& out, const Complex2& obj) {
39      out << "(" << obj.m_real_ << " : " << obj.m_imag_ << "i)";
40      return out;
41    }
42  };
```

9 Приложение 2. Код программы

```
1   //==============================================================
2   // Copyright    2020 Intel Corporation
3   //
4   // SPDX-License-Identifier: MIT
5   // =============================================================
6   #include <sycl/sycl.hpp>
7   #include <iomanip>
8   #include <vector>
9   #include <time.h>
10  // dpc_common.hpp can be found in the dev-utilities include folder.
11  // e.g., $ONEAPI_ROOT/dev-utilities/<version>/include/dpc_common.hpp
12  #include "dpc_common.hpp"
13  #include "Complex.hpp"
14
15  using namespace sycl;
16  using namespace std;
17
18  // Number of complex numbers passing to the SYCL code
19  //static const int num_elements = 10000;
20
21  class CustomDeviceSelector {
22      public:
23          CustomDeviceSelector(std::string vendorName) : vendorName_(vendorName){};
24              int operator()(const device &dev) const {
25                  int device_rating = 0;
26                  // In the below code we are querying for the custom device specific to a
27                  // Vendor and if it is a GPU device we are giving the highest rating. The
28                  // second preference is given to any GPU device and the third preference
                        is
29                  // given to CPU device.
30                  if (dev.is_gpu() & (dev.get_info<info::device::name>().find(vendorName_)
                        !=
31                                          std::string::npos))
32                      device_rating = 3;
33                  else if (dev.is_gpu())
34                      device_rating = 2;
35                  else if (dev.is_cpu())
36                      device_rating = 1;
37
38                  return device_rating;
39          };
```

```
40
41      private:
42          std::string vendorName_;
43  };
44
45  // in_vect1 and in_vect2 are the vectors with num_elements complex nubers and
46  // are inputs to the parallel function
47  void SYCLParallel(queue &q, std::vector<Complex2> &in_vect1,
48                    std::vector<Complex2> &in_vect2,
49                    std::vector<Complex2> &out_vect) {
50      auto R = range(in_vect1.size());
51      if (in_vect2.size() != in_vect1.size() || out_vect.size() != in_vect1.size()){
52          std::cout << "ERROR: Vector sizes do not  match"<< "\n";
53          return;
54      }
55      // Setup input buffers
56      buffer bufin_vect1(in_vect1);
57      buffer bufin_vect2(in_vect2);
58
59      // Setup Output buffers
60      buffer bufout_vect(out_vect);
61
62
63      // std::cout << "Target Device: " << q.get_device().get_info<info::device::name>() <<
            "\n";
64
65
66      // Submit Command group function object to the queue
67      q.submit([&](auto &h) {
68          // Accessors set as read mode
69          accessor V1(bufin_vect1,h,read_only);
70          accessor V2(bufin_vect2,h,read_only);
71          // Accessor set to Write mode
72          accessor V3 (bufout_vect,h,write_only);
73          h.parallel_for(R, [=](auto i) {
74              // call the complex_mul function that computes the multiplication of the
75              // complex number
76              V3[i] = V1[i].complex_mul(V2[i]);
77          });
78      });
79      // Synchronize
80      q.wait_and_throw();
81  }
82
83  void Scalar(std::vector<Complex2> &in_vect1,
84                  std::vector<Complex2> &in_vect2,
85                  std::vector<Complex2> &out_vect) {
86      if ((in_vect2.size() != in_vect1.size()) || (out_vect.size() != in_vect1.size())){
87          std::cout<<"ERROR: Vector sizes do not match"<<"\n";
88          return;
89      }
90      for (int i = 0; i < in_vect1.size(); i++) {
91          out_vect[i] = in_vect1[i].complex_mul(in_vect2[i]);
92      }
```

```
}

// Compare the results of the two output vectors from parallel and scalar. They
// should be equal
int Compare(std::vector<Complex2> &v1, std::vector<Complex2> &v2) {
    int ret_code = 1;
    if(v1.size() != v2.size()){
        ret_code = -1;
    }
    for (int i = 0; i < v1.size(); i++) {
        if (v1[i] != v2[i]) {
            ret_code = -1;
            break;
        }
    }
    return ret_code;
}

int main() {
    std::cout<<"Intel" << std::endl;
    std::cout << "N        Time1     Time2      Time3" << std::endl;
    for (int num_elements = 64; num_elements < 100000000; num_elements*=2){
        std::cout<<  num_elements << "   ";
        for (int doesnt_matter = 0; doesnt_matter < 3; doesnt_matter++){
            // Declare your Input and Output vectors of the Complex2 class
            vector<Complex2> input_vect1;
            vector<Complex2> input_vect2;
            vector<Complex2> out_vect_parallel;
            vector<Complex2> out_vect_scalar;
            // Declare time variables
            clock_t start_time, end_time;
            // Fill vectors
            for (int i = 0; i < num_elements; i++) {
                input_vect1.push_back(Complex2(2 * (i % 2), 3* (i % 2 + 1)));
                input_vect2.push_back(Complex2( 4,  6));
                out_vect_parallel.push_back(Complex2(0, 0));
                out_vect_scalar.push_back(Complex2(0, 0));
            }

            // Initialize your Input and Output Vectors. Inputs are initialized as below.
            // Outputs are initialized with 0
            try {

                // Pass in the name of the vendor for which the device you want to query
                std::string vendor_name = "Intel";
                // std::string vendor_name = "AMD";
                // std::string vendor_name = "Nvidia";

                start_time = clock();
                // queue constructor passed exception handler
                CustomDeviceSelector selector(vendor_name);
                queue q(selector);
                // Call the SYCLParallel with the required inputs and outputs
                SYCLParallel(q, input_vect1, input_vect2, out_vect_parallel);
```

```
147                 end_time = clock();
148             } catch (...) {
149                 // some other exception detected
150                 std::cout << "Failure" << std::endl;
151                 std::terminate();
152             }
153 
154             // std::cout
155             //     << "****************************************Multiplying Complex
                    numbers "
156             //         "in Parallel
                    ******************************************************"
157             //     << std::endl;
158 
159             // Print the outputs of the Parallel function
160             int indices[]{0, 1, 2, 3, 4, (num_elements - 1)};
161             constexpr size_t indices_size = sizeof(indices) / sizeof(int);
162 
163             // for (int i = 0; i < indices_size; i++) {
164             //     int j = indices[i];
165             //     if (i == indices_size - 1) std::cout << "...\n";
166             //     std::cout << "[" << j << "] " << input_vect1[j] << " * " <<
                    input_vect2[j]
167             //                 << " = " << out_vect_parallel[j] << "\n";
168             // }
169             // Call the Scalar function with the required input and outputs
170             Scalar(input_vect1, input_vect2, out_vect_scalar);
171 
172             // Compare the outputs from the parallel and the scalar functions. They
                    should
173             // be equal
174             int ret_code = Compare(out_vect_parallel, out_vect_scalar);
175             if (ret_code == 1) {
176                 // std::cout << "Complex multiplication successfully run on the device"
177                 //             << "\n";
178                 std::cout << (float)(end_time - start_time) / CLOCKS_PER_SEC << "   ";
179             } else {
180                 std::cout
181                     << "*********************************************Verification Failed.
                            Results are "
182                        "not matched**************************"
183                     << "\n";
184             }
185         }
186         std::cout<<std::endl;
187     }
188 
189     std::cout<<"Scalar" << std::endl;
190     std::cout << "N        Time1     Time2      Time3" << std::endl;
191     for (int num_elements = 64; num_elements < 100000000; num_elements*=2){
192         std::cout<<  num_elements << "    ";
193         for (int doesnt_matter = 0; doesnt_matter < 3; doesnt_matter++){
194             // Declare your Input and Output vectors of the Complex2 class
195             vector<Complex2> input_vect1;
```

```
            vector<Complex2> input_vect2;
            vector<Complex2> out_vect_parallel;
            vector<Complex2> out_vect_scalar;
            // Declare time variables
            clock_t start_time, end_time;
            // Fill vectors
            for (int i = 0; i < num_elements; i++) {
                input_vect1.push_back(Complex2(2 * (i % 2), 3* (i % 2 + 1)));
                input_vect2.push_back(Complex2( 4,  6));
                out_vect_parallel.push_back(Complex2(0, 0));
                out_vect_scalar.push_back(Complex2(0, 0));
            }

            start_time = clock();
            // Call the Scalar function with the required input and outputs
            Scalar(input_vect1, input_vect2, out_vect_scalar);
            end_time = clock();

            std::cout << (float)(end_time - start_time) / CLOCKS_PER_SEC << "  ";
        }
        std::cout<<std::endl;
    }

    return 0;
}
```

10 Приложение 3. Вывод программы

```
################################################################################
#      Date:           Tue 06 Jun 2023 02:31:39 PM PDT
#    Job ID:           2313583.v-qsvr-1.aidevcloud
#      User:           u194687
# Resources:           cput=75:00:00,neednodes=1:gpu:ppn=2,nodes=1:gpu:ppn=2,walltime
    =06:00:00
################################################################################


start: 23/06/06 14:31:41.827

./complex_mult.exe
Target Device: Intel(R) UHD Graphics [0x9a60]
****************************************Multiplying Complex numbers in Parallel
    ************************************************************
[0] (2 : 4i) * (4 : 6i) = (-16 : 28i)
[1] (3 : 5i) * (5 : 7i) = (-20 : 46i)
[2] (4 : 6i) * (6 : 8i) = (-24 : 68i)
[3] (5 : 7i) * (7 : 9i) = (-28 : 94i)
[4] (6 : 8i) * (8 : 10i) = (-32 : 124i)
...
[9999] (10001 : 10003i) * (10003 : 10005i) = (-40012 : 200120014i)
Complex multiplication successfully run on the device

```

```
end: 23/06/06 14:31:45.604


################################################################################
# End of output for job 2313583.v-qsvr-1.aidevcloud
# Date: Tue 06 Jun 2023 02:31:45 PM PDT
################################################################################
```

11 Приложение 4. Результаты оценки времени выполнения

```
Serial
       N      Time1     Time2     Time3
      64      2e-06     1e-06     1e-06
     128      1e-06     1e-06     1e-06
     256      2e-06     2e-06     2e-06
     512      4e-06     5e-06     4e-06
    1024      9e-06     8e-06     9e-06
    2048    1.8e-05   1.7e-05   1.7e-05
    4096    3.3e-05   3.3e-05   3.3e-05
    8192    6.6e-05   6.6e-05   6.5e-05
   16384   0.000133  0.000132  0.000137
   32768   0.000281  0.000270  0.000282
   65536   0.000570  0.000572  0.000535
  131072   0.001144  0.001141  0.001131
  262144   0.002275  0.002195  0.002223
  524288   0.004726  0.004669  0.004695
 1048576   0.009198  0.008957  0.009194
 2097152   0.018535  0.018044  0.018202
 4194304   0.035498  0.035419  0.035395
 8388608   0.071511  0.069997  0.070812
16777216   0.143298  0.141777  0.142613
33554432   0.286227  0.286585  0.284476
67108864   0.572891  0.585602  0.572626

Parallel
       N      Time1     Time2     Time3
      64   0.164994  0.002748  0.002540
     128   0.002540  0.002466  0.002361
     256   0.002600  0.002396  0.002447
     512   0.002504  0.002365  0.002380
    1024   0.002358  0.002475  0.002409
    2048   0.002498  0.002500  0.002414
    4096   0.002488  0.002458  0.002495
    8192   0.002527  0.002454  0.002427
   16384   0.002487  0.002448  0.002470
   32768   0.002520  0.002385  0.002496
   65536   0.002765  0.002534  0.002577
  131072   0.003359  0.002695  0.002650
  262144   0.004410  0.002966  0.002944
  524288   0.006695  0.004504  0.004483
 1048576   0.009494  0.009378  0.009378
 2097152   0.017007  0.017253  0.017302
```

```
 4194304   0.032177  0.032058  0.031927
 8388608   0.061105  0.061174  0.066771
16777216   0.119367  0.119701  0.119522
33554432   0.234918  0.233657  0.231074
67108864   0.464906  0.460783  0.461675
```