Lab2

HW/SW Co-design

Νικόλαος Γιαννόπουλος 9629 | Ανδρονίκου Δημήτριος 9836| 10/12/2022

Αρχείο host.cpp

Κρατήσαμε τον βασικό κώδικα ίδιο με αυτόν από το παράδειγμα, κάνοντας κάποιες μικρές αλλαγές και προσθήκες

Αλλάξαμε το `size_t vector_size_bytes = sizeof(int) * n*m;`

Επίσης, φτιάξαμε τους πίνακες

```
std::vector<int, aligned_allocator<int>> A(n*m);
std::vector<int, aligned_allocator<int>> B(m*p);
std::vector<int, aligned_allocator<int>> C_hw(n*p);
std::vector<int, aligned_allocator<int>> C_sw(n*p)
```

Και τους γεμίζουμε

```
// Create the test data
std::generate(A.begin(), A.end(), std::rand );
std::generate(B.begin(), B.end(), std::rand);
```

```
-        for(int i = 0; i<n; i++){

-              for(int j = 0; j < p; j++){

-                    C_hw[i*n+j] = 0;

-                    C_sw[i*n+j] =0;

              }
```

Υπολογίζουμε στον host το αποτέλεσμα

```
        for(int i = 0; i<n; i++){
                for(int j = 0; j < p; j++){
                        for(int k = 0; k < m; k++){
                                C_sw[i*n+j] += A[i *n + k] * B[k * m + j];

                        }

                }
        }
```

Έπειτα συνδέουμε τους πίνακες με τον kernel

```
// Allocate Buffer in Global Memory
// Buffers are allocated using CL_MEM_USE_HOST_PTR for efficient memory and
// Device-to-host communication
OCL_CHECK(err, cl::Buffer buffer_in1(
                    context, CL_MEM_USE_HOST_PTR | CL_MEM_READ_ONLY,
                    vector_size_bytes, A.data(), &err));
OCL_CHECK(err, cl::Buffer buffer_in2(
                    context, CL_MEM_USE_HOST_PTR | CL_MEM_READ_ONLY,
                    vector_size_bytes, B.data(), &err));
OCL_CHECK(err, cl::Buffer buffer_output(
                    context, CL_MEM_USE_HOST_PTR | CL_MEM_WRITE_ONLY,
                    vector_size_bytes, C_hw.data(), &err))
OCL_CHECK(err, err = krnl_vector_mult.setArg(0, buffer_in1));
OCL_CHECK(err, err = krnl_vector_mult.setArg(1, buffer_in2));
OCL_CHECK(err, err = krnl_vector_mult.setArg(2, buffer_output));
```

Στη συνέχεια αντιγράφουμε τα δεδομένα στην global memory

```
OCL_CHECK(err, err = q.enqueueMigrateMemObjects({buffer_in1, buffer_in2},
                                                0 /* 0 means from host*/));
```

Και ξεκινάμε τον kernel και αντιγράφουμε πίσω στον host το αποτέλεσμα

```
  OCL_CHECK(err, err = q.enqueueTask(krnl_vector_mult));

  // Copy Result from Device Global Memory to Host Local Memory
  OCL_CHECK(err, err = q.enqueueMigrateMemObjects({buffer_output},
                                                  CL_MIGRATE_MEM_OBJECT_HOST));
```

Τέλος συγκρίνουμε το αποτέλεσμα του host με το αποτέλεσμα του kernel και εκτυπώνουμε αντίστοιχο κείμενο

```
  bool match = true;

        for(int i = 0; i<n; ++i){
                for(int j = 0; j < p; ++j){
                        if (C_hw[i *n +j] != C_sw[i *n +j]) {
                                std::cout << "Error: Result mismatch" << std::endl;
                                std::cout << "i = " << i << " CPU result = " << C_sw[i
*n + j]
                                                        << " Device result = " << C_hw[i * n
+j ] << std::endl;
                                match = false;
                                break;


                }
        }
        }

  std::cout << "TEST " << (match ? "PASSED" : "FAILED") << std::endl;
  return (match ? EXIT_SUCCESS : EXIT_FAILURE)
```

Αρχείο matrix_mul.cpp

Αρχικά έχουμε τα ορίσματα

void matrix_mul (const int * A, //read only A NxM

const int * B, //read only B MxP

int *C){// write C NxP matrix

Δημιουργούμε buffers ώστε τα δεδομένα να αποθηκεύονται τοπικά

int A_buffer[n*m]; // Local memory to store A

int B_buffer[m*p]; // Local memory to store B

int C_buffer[n*p]; // Local Memory to store result

Αντιγρλαφουμε τους δύο πίνακες από τα ορίσματα

readA: for(int i = 0; i < n; i++){

for (int j = 0; j < m; j++) {

```
#pragma HLS LOOP_TRIPCOUNT min = SIZE max = SIZE
#pragma HLS PIPELINE II = 1
            A_buffer[i*n+j] = A[i*n+j];
   }
 }

readB: for(int i = 0; i < m; i++){
  for (int j = 0; j < p; j++) {
#pragma HLS LOOP_TRIPCOUNT min = SIZE max = SIZE
#pragma HLS PIPELINE II = 1
            B_buffer[i*m+j] = B[i*m+j];
   }
 }
```

Και στη συνέχεια μηδενίζουμε τον πίνακα που θα υπολογίσουμε το αποτέλεσμα

```
fill_C_with_zeros:for(int i = 0; i < n; i++){
            for (int j = 0; j < p; j++) {
                  C_buffer[i*n +j] =0;
            }
}
```

Τέλος υπολογίζουμε το τελικό αποτέλεσμα και το γράφουμε πίσω στο όρισμα

```
mult:for(int i = 0; i< n; i++){
          for(int j = 0; j < p; j++){
#pragma HLS PIPELINE II=1
                  int results =0;
                  product: for(int k = 0; k < m; k++){
                        results += A_buffer[i *n + k] * B_buffer[k * m + j];
                  }
                   C[i*n+j] = results;
          }
}
```