OpenMP / DPC++ for accelerators

Porting code on GPUs using Intel OneAPI Toolkits

Student:

Alessandro Masini

Supervisors:

Antonio Sciarappa (Leonardo Labs)

Irina Davydenkova (MHPC)

Master in
High Performance Computing

LEONARDO

Goals of this work

(~ timeline)

1. Test Intel OneAPI Toolkits (~ Jun - Jul)
    - Test OpenMP offload for simple codes
    - Try to port simple codes from CUDA to DPC++
2. Port existing company CUDA codes to Intel GPUs (~ Aug - Dec)
3. Analysis of the ported codes (~ as above)
    - Performance analysis
    - Programmer-side benefits of DPC++ / OpenMP

OpenMP offload

- Open standard
- Cross-platform (compatible with different kinds of accelerators)
- Multi-vendor (not specific for Intel, *)
- Single source (single language C/C++/Fortran)
- Less hard to maintain
- Allows incremental porting

* Intel compilers specific: tight integration with MKL

Testing OpenMP offload

Basic functionalities:

- Target regions
- Data mapping
- Reductions

```
t_gpu = omp_get_wtime()
write(*,*) " - dgetrf"
!$omp target variant dispatch use_device_ptr(A_dev, ipiv_dev, info_dev)
call dgetrf(N, N, A_dev, lda, ipiv_dev, info_dev)
!$omp end target variant dispatch
write(*,*) " - dgetrs"
!$omp target variant dispatch use_device_ptr(A_dev, b_dev, ipiv_dev, info_dev)
call dgetrs(transa, N, nrhs, A_dev, lda, ipiv_dev, b_dev, ldb, info_dev)
!$omp end target variant dispatch
t_gpu = omp_get_wtime() - t_gpu
```

MKL routines:

- Lapack dgetrf / dgetrs (1)
- Random number generators (2)
- Blas sgemm (3)
- FFTW

```
!$omp target data map(tofrom: rn)
time_gpu = omp_get_wtime()
!$omp target variant dispatch use_device_ptr(rn)
err = vdrnguniform(method, stream, N, rn, a, b)
!$omp end target variant dispatch
time_gpu = omp_get_wtime() - time_gpu
!$omp end target data
```

```
// start timer
double etime = omp_get_wtime();

// perform cblas on device
#pragma omp target variant dispatch use_device_ptr(devA, devB, devC)
cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            SIZE, SIZE, SIZE, 1.0, devA, SIZE, devB, SIZE, 0.0, devC, SIZE);

// stop timer
etime = omp_get_wtime() - etime;
```

DPC++ / SYCL

- Based on open standard (Khronos' SYCL)
- Cross-platform (compatible with different kinds of accelerators)
- Multi-vendor (not specific for Intel, *)
- High-level language (extension of C++17)
- Single source for host and devices

* support for NVIDIA GPUs requires compilation from source

Porting nbody DPC++

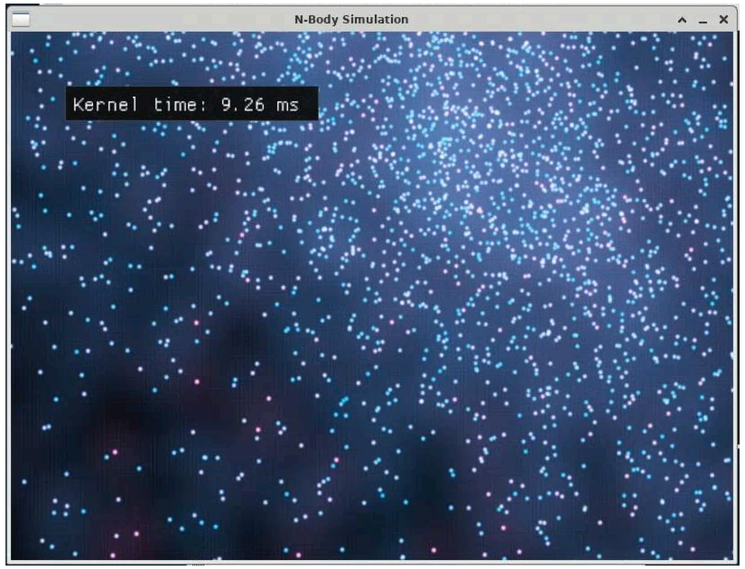
Goal: replicating test porting for nbody simulation code

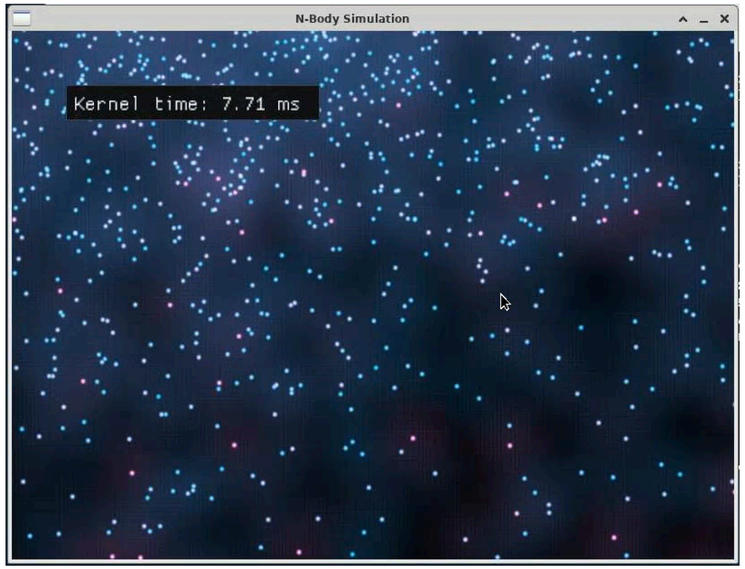
Workflow:

1. Build support graphics library (for visualization)
2. Build dpcpp/clang with support for NVIDIA GPUs
3. Convert CUDA to SYCL using Intel's Compatibility Tool
4. Compile source
5. Run code on Leonardo's cluster (davinci-1)

Results -> ported SYCL code is faster (~15%)

Source: https://github.com/codeplaysoftware/cuda-to-sycl-nbody





Current project status

Codes to be ported are mainly Fortran, with external CUDA C routines

-> Crucial to test compatibility between DPC++ and Fortran

Main steps:

- Convert CUDA to SYCL with Intel DPC++ Compatibility Tool (and fix issues)
- Compile sources separately (ifort for Fortran & clang/dpcpp for DPC++)
- Link objects into working executable

Next -> starting work on real company codes

Example of converted code

Same function in CUDA (below) and SYCL (right) generated via Intel Compatibility Tool

```cpp
void test_gpu(void * __restrict__ A_cpu, void * __restrict__ B_cpu,
              void * __restrict__ PP_cpu, void * __restrict__ g_cpu,
              void * __restrict__ rv_cpu, void * __restrict__ V_cpu,
              int Vidx, int iSize)
{
    // ...

    dim3 grid, block;

    HANDLE_ERROR(cudaGetDeviceCount(&N_GPUS));

    device = 0;
    HANDLE_ERROR(cudaSetDevice(device));

    HANDLE_ERROR(cudaDeviceGetAttribute(&warp_size,cudaDevAttrWarpSize,device));
    HANDLE_ERROR(cudaDeviceGetAttribute(&n_sm,cudaDevAttrMultiProcessorCount,device));

    block.x = 2* warp_size;
    grid.x = n_sm;
    grid.y = n_sm;

    // ...

    HANDLE_ERROR( cudaMalloc( (void **) &  A_gpu, 3 *  Vidx * sizeof(double) ) );

    // ...

    HANDLE_ERROR( cudaMemcpy( A_gpu,  A, 3 *  Vidx * sizeof(double) , cudaMemcpyHostToDevice) );

    // ...

    test_gpu_inner<<<grid,block>>>(A_gpu, B_gpu, PP_gpu, g_gpu, rv_gpu, V_gpu, Vidx, iSize);

    HANDLE_ERROR( cudaGetLastError() );

    HANDLE_ERROR( cudaMemcpy( V,  V_gpu, 3 * iSize * sizeof(double) , cudaMemcpyDeviceToHost) );

    HANDLE_ERROR( cudaFree( A_gpu) );

    // ...
}
```

```cpp
void test_gpu(void * __restrict__ A_cpu, void * __restrict__ B_cpu,
              void * __restrict__ PP_cpu, void * __restrict__ g_cpu,
              void * __restrict__ rv_cpu, void * __restrict__ V_cpu,
              int Vidx, int iSize)
{
    // ...

    sycl::range<3> grid(1, 1, 1), block(1, 1, 1);

    HANDLE_ERROR((N_GPUS = dpct::dev_mgr::instance().device_count(), 0));

    device = 0;
    HANDLE_ERROR((dpct::dev_mgr::instance().select_device(device), 0));

    HANDLE_ERROR(cudaDeviceGetAttribute(&warp_size,cudaDevAttrWarpSize,device));
    HANDLE_ERROR((n_sm = dpct::dev_mgr::instance()
                             .get_device(device)
                             .get_max_compute_units(),
                  0));

    block[2] = 2 * warp_size;
    grid[2] = n_sm;
    grid[1] = n_sm;

    // ...

    HANDLE_ERROR((A_gpu = sycl::malloc_device<double>(
                      3 * Vidx, dpct::get_default_queue()),
                  0));

    // ...

    HANDLE_ERROR((dpct::get_default_queue()
                      .memcpy(A_gpu, A, 3 * Vidx * sizeof(double))
                      .wait(),
                  0));

    // ...

    dpct::get_default_queue()
        .parallel_for<dpct_kernel_name<class test_gpu_inner_f56bea>>(
            sycl::nd_range<3>(grid * block, block),
            [=](sycl::nd_item<3> item_ct1) {
                test_gpu_inner(A_gpu, B_gpu, PP_gpu, g_gpu, rv_gpu, V_gpu,
                               Vidx, iSize, item_ct1);
            });

    HANDLE_ERROR(0);

    HANDLE_ERROR((dpct::get_default_queue()
                      .memcpy(V, V_gpu, 3 * iSize * sizeof(double))
                      .wait(),
                  0));

    HANDLE_ERROR((sycl::free(A_gpu, dpct::get_default_queue()), 0));

    // ...
}
```

problem