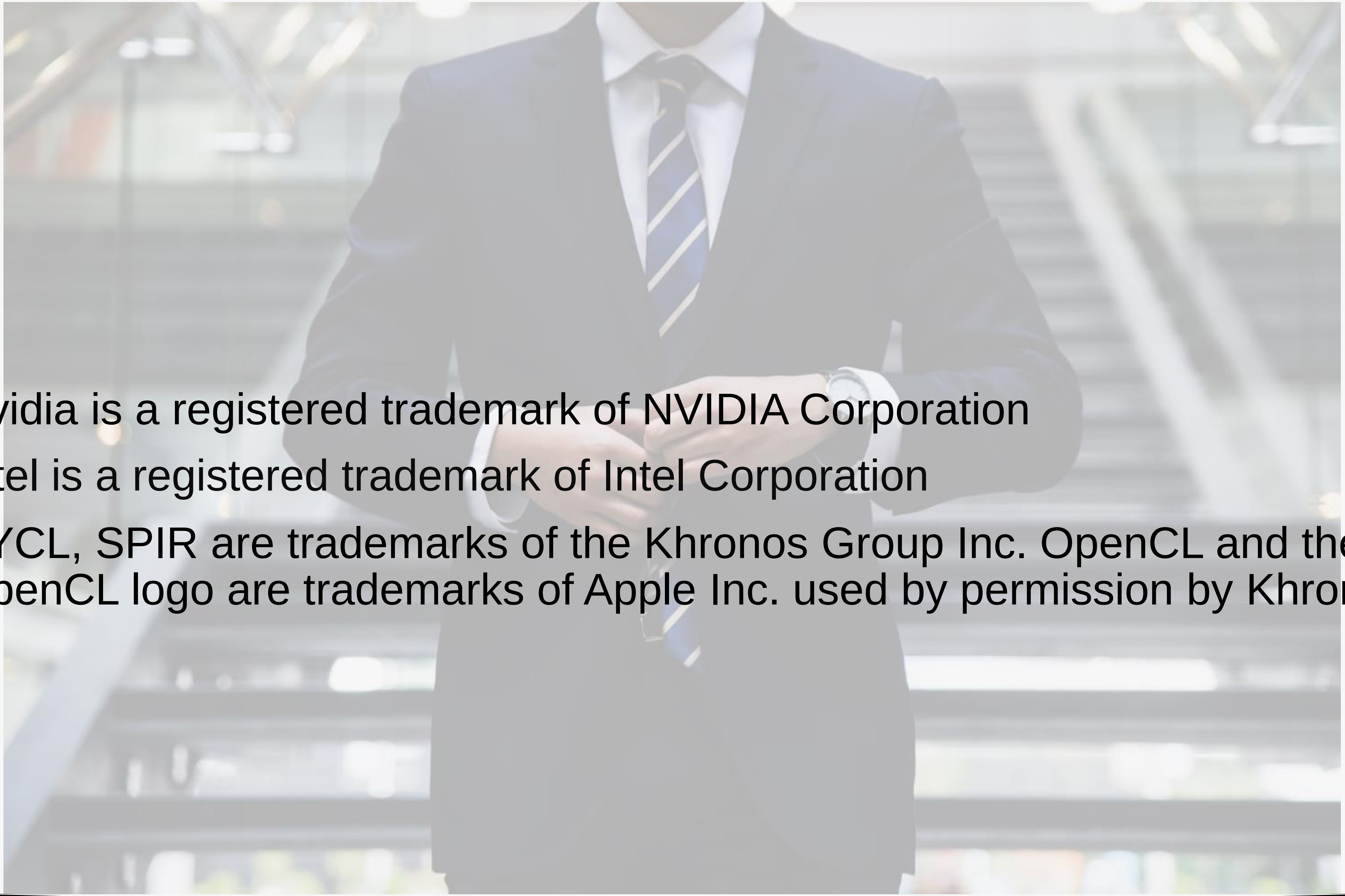
codeplay

Enable AI & HPC to be Open, Safe and Accessible to All

How to port your code from CUDA to SYCL, targeting Nvidia GPUs and more

Joe Todd – Senior Software Engineer



- Nvidia is a registered trademark of NVIDIA Corporation
- Intel is a registered trademark of Intel Corporation
- SYCL, SPIR are trademarks of the Khronos Group Inc. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos

Company

Leaders in enabling high-performance software solutions for new AI processing systems

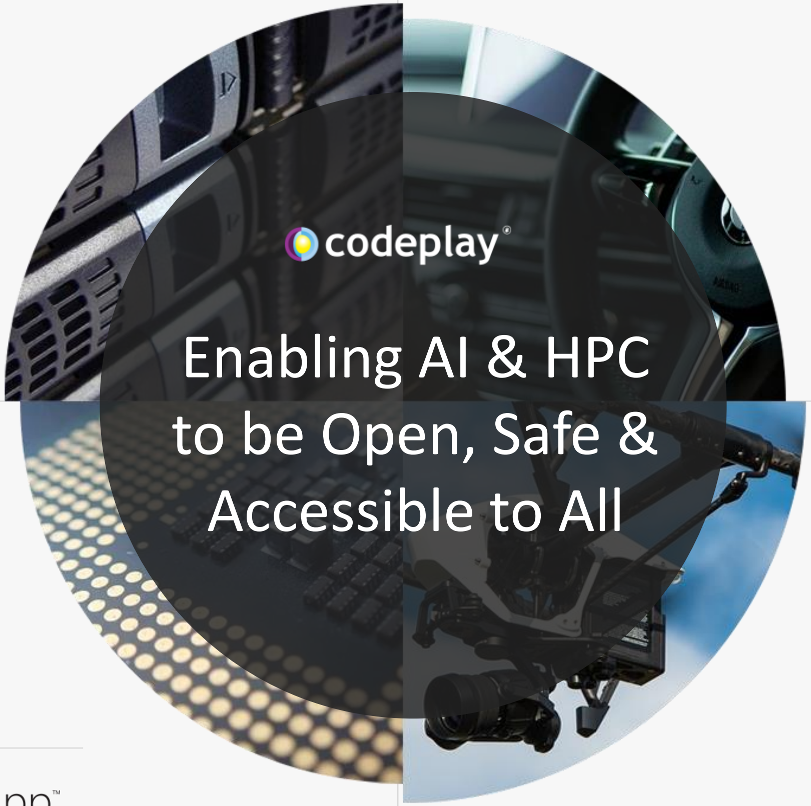
Enabling the toughest processors with tools and middleware based on open standards

Established 2002 in Scotland with ~80 employees

Partners

Intel, Broadcom, Synopsys, CEVA, Imagination, Renesas, KMC Kyoto Microcomputer Co., Ltd., NSI-TEXE, Berkeley Lab, Oak Ridge National Laboratory, Argonne National Laboratory

And many more!



codeplay

Enabling AI & HPC to be Open, Safe & Accessible to All

Products

Acoran

Integrates all the industry standard technologies needed to support a very wide range of AI and HPC

ComputeAorta™

The heart of Codeplay's compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

ComputeCpp™

C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

Markets

High Performance Compute (HPC)
Automotive ADAS, IoT, Cloud Compute
Smartphones & Tablets
Medical & Industrial

Technologies: Artificial Intelligence
Vision Processing
Machine Learning
Big Data Compute

Migrating from CUDA to SYCL

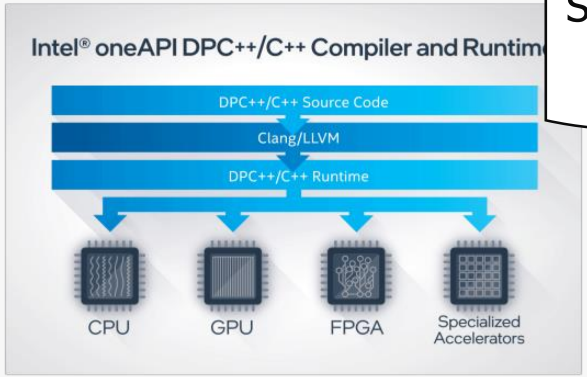
- Why migrate from CUDA to SYCL?
- How to convert CUDA code to SYCL?
- How does the code compare?
- How to achieve performance using SYCL?

oneAPI and SYCL

oneAPI

SYCL™

SYCL source code

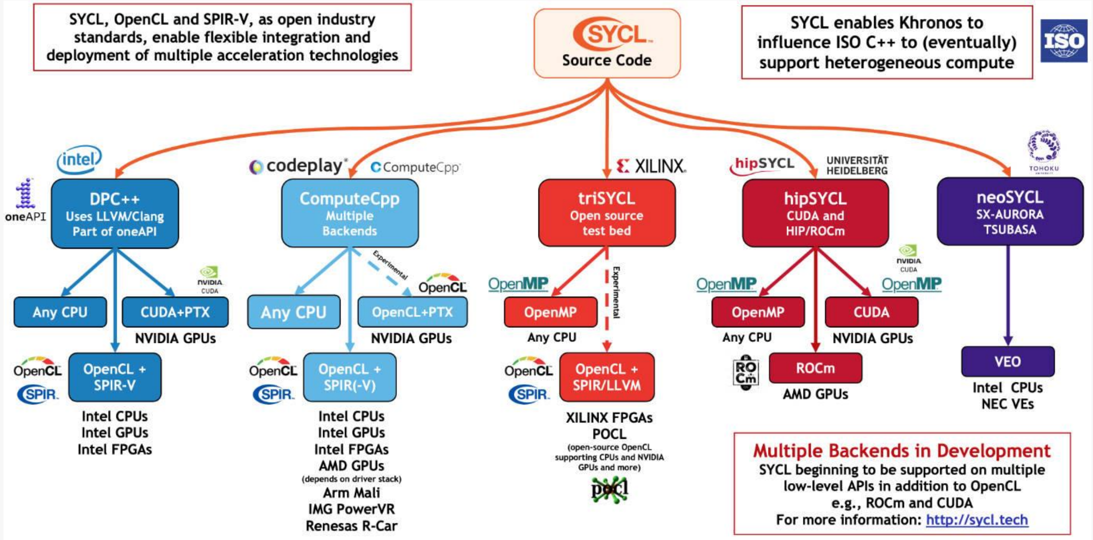


Intel® oneAPI DPC++/C++ Compiler and Runtime

- SYCL sits at the heart of oneAPI
- Provides an open standard interface for developers
- Defined by the industry

Why Migrate from CUDA to SYCL?

- CUDA is a proprietary interface
- Can only be used to target Nvidia GPUs
- SYCL is an open standard interface
- SYCL can be used to target Nvidia, Intel and AMD processors



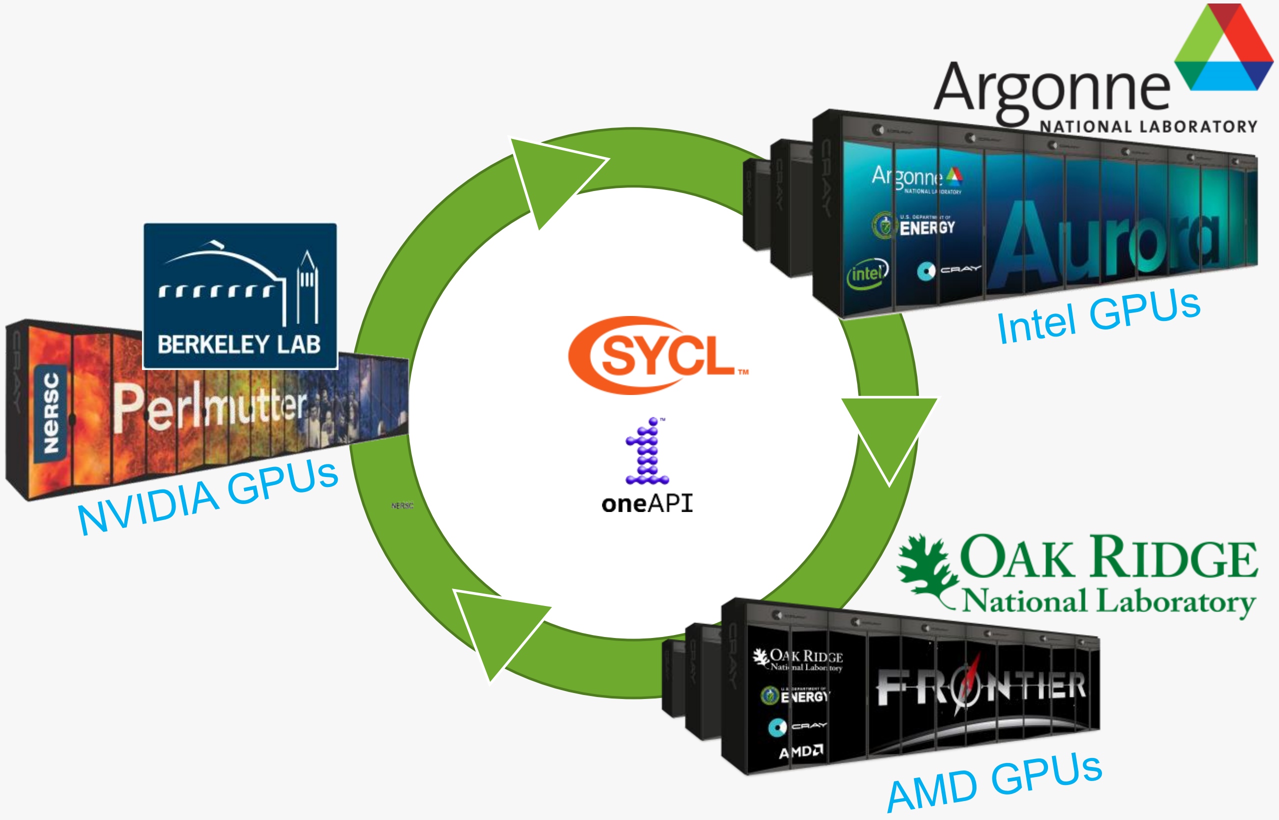
SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

Multiple Backends in Development
SYCL beginning to be supported on multiple low-level APIs in addition to OpenCL
e.g., ROCm and CUDA
For more information: http://sycl.tech

SYCL on the Fastest Supercomputers

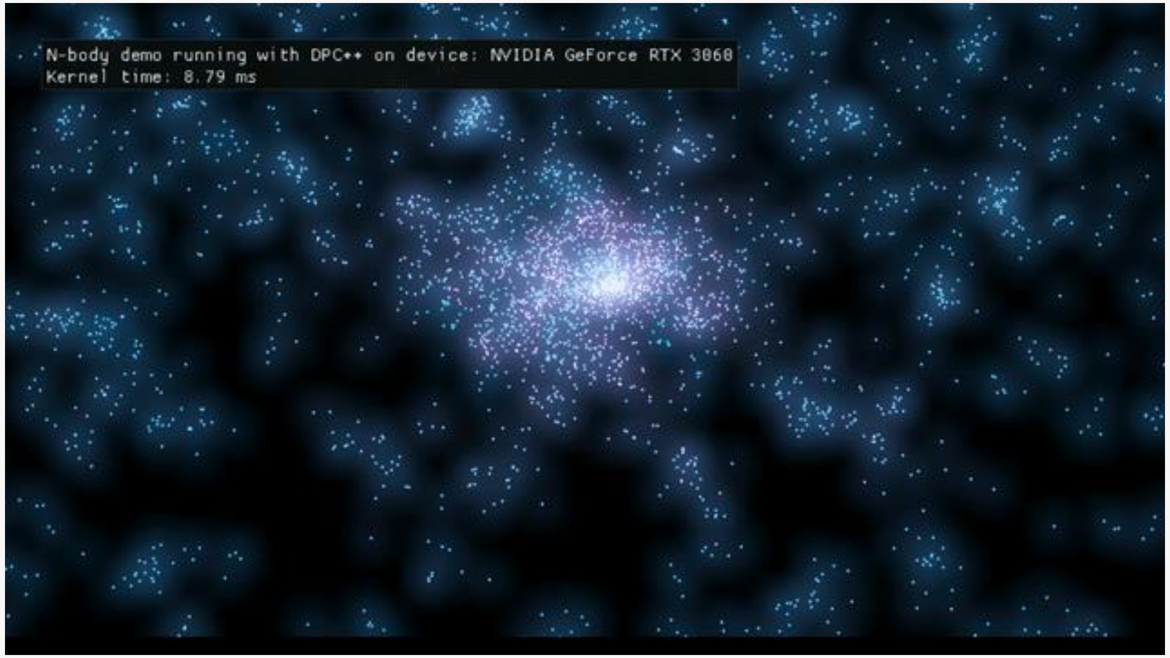
- SYCL is deployed on some of the fastest supercomputers
- Codeplay develops and maintains SYCL for Perlmutter and Frontier



Overview

Simple case study using the Intel DPC++ Compatibility Tool to convert a small CUDA project to SYCL, will cover:

- N-Body Simulation
- Using the DPCT conversion tool
- Quick look at DPCT output
- Performance
- Caveats



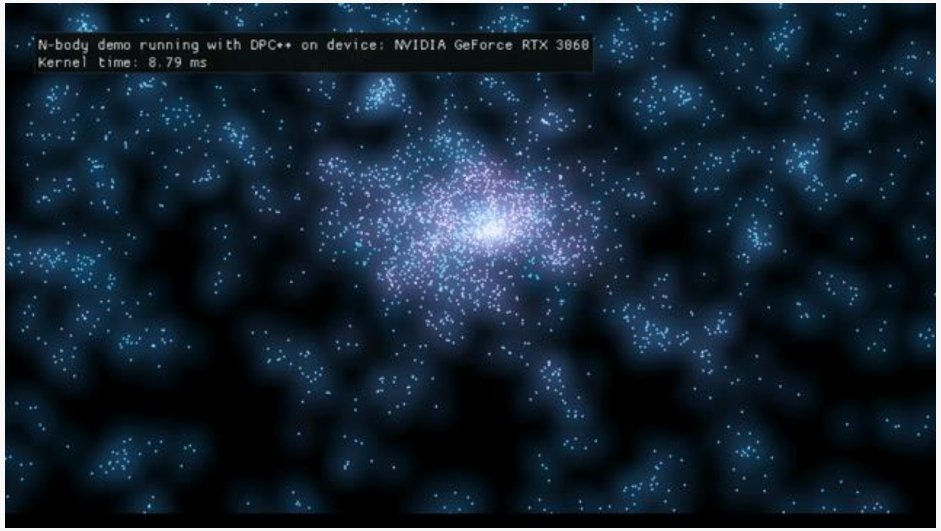
```
N-body demo running with DPC++ on device: NVIDIA GeForce RTX 3060
Kernel time: 8.79 ms
```

N-Body

- Simulates gravitational interaction in a fictional galaxy

$$\vec{F}_i = -\sum_{i \neq j} G \frac{(\vec{r}_i - \vec{r}_j)}{|\vec{r}_i - \vec{r}_j|^3}$$

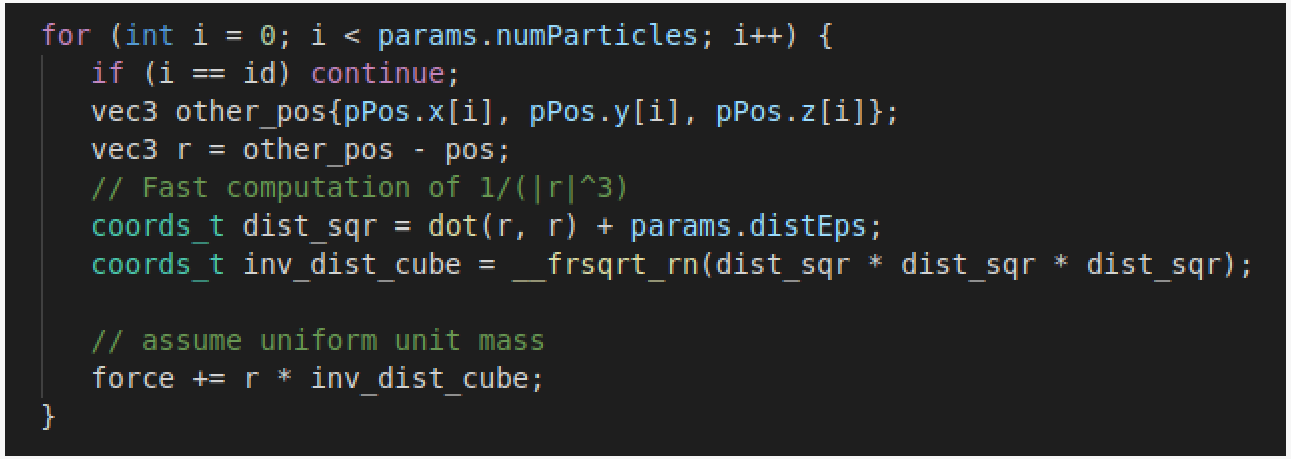
- Intentionally simple kernel
  - No use of shared memory
  - O(N^2) computation
- OpenGL for graphics (in separate TUs)



```
N-body demo running with DPC++ on device: NVIDIA GeForce RTX 3060
Kernel time: 8.79 ms
```

N-Body

$$\vec{F}_i = -\sum_{i \neq j} G \frac{(\vec{r}_i - \vec{r}_j)}{\left|\vec{r}_i - \vec{r}_j\right|^3}$$



```
for (int i = 0; i < params.numParticles; i++) {
   if (i == id) continue;
   vec3 other_pos{pPos.x[i], pPos.y[i], pPos.z[i]};
   vec3 r = other_pos - pos;
   // Fast computation of 1/(|r|^3)
   coords_t dist_sqr = dot(r, r) + params.distEps;
   coords_t inv_dist_cube = __frsqrt_rn(dist_sqr * dist_sqr * dist_sqr);

   // assume uniform unit mass
   force += r * inv_dist_cube;
}
```

N-Body

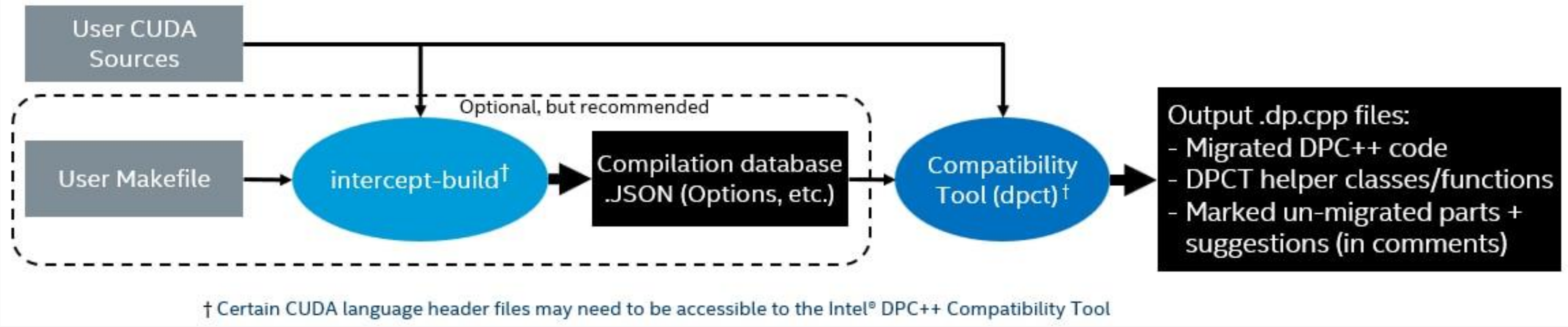
- Designed to be accessible – please try it out!
  - https://github.com/codeplaysoftware/cuda-to-sycl-nbody

- Single .cu file (simulator.cu)

- Lots of scripts provided to:
  - Convert with DPCT
  - Build the CUDA & SYCL demos
  - Run them (even without X graphical output)

Intel DPC++ Compatibility Tool (DPCT)

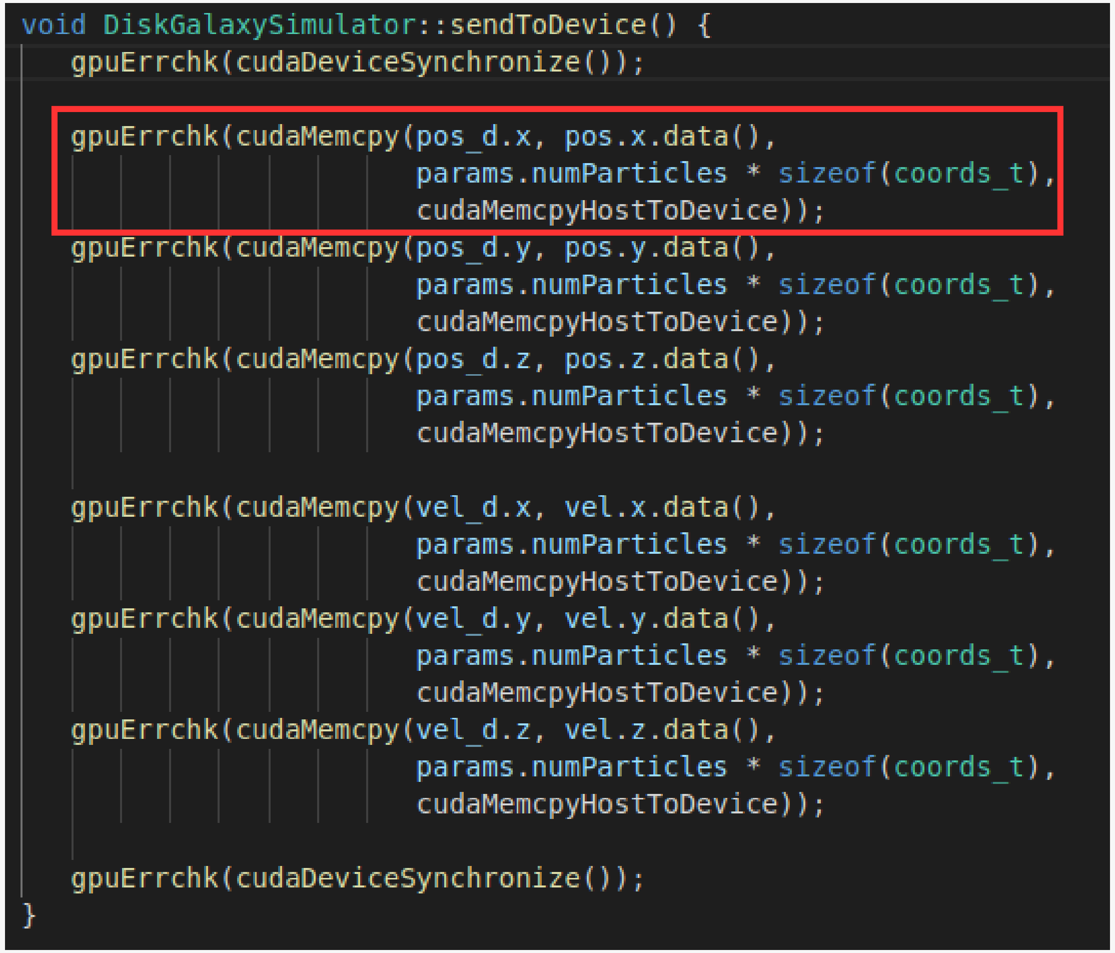
- Converts CUDA code to SYCL
- Operates on individual .cu files
  - But can `intercept-build make` to generate list of DPCT conversions
- Promises ~90% code conversion
  - Managed 100% for N-Body!
- Verbose warnings in SYCL output


User CUDA Sources
Optional, but recommended
User Makefile
intercept-build†
Compilation database .JSON (Options, etc.)
Compatibility Tool (dpct)†
Output .dp.cpp files:
- Migrated DPC++ code
- DPCT helper classes/functions
- Marked un-migrated parts + suggestions (in comments)

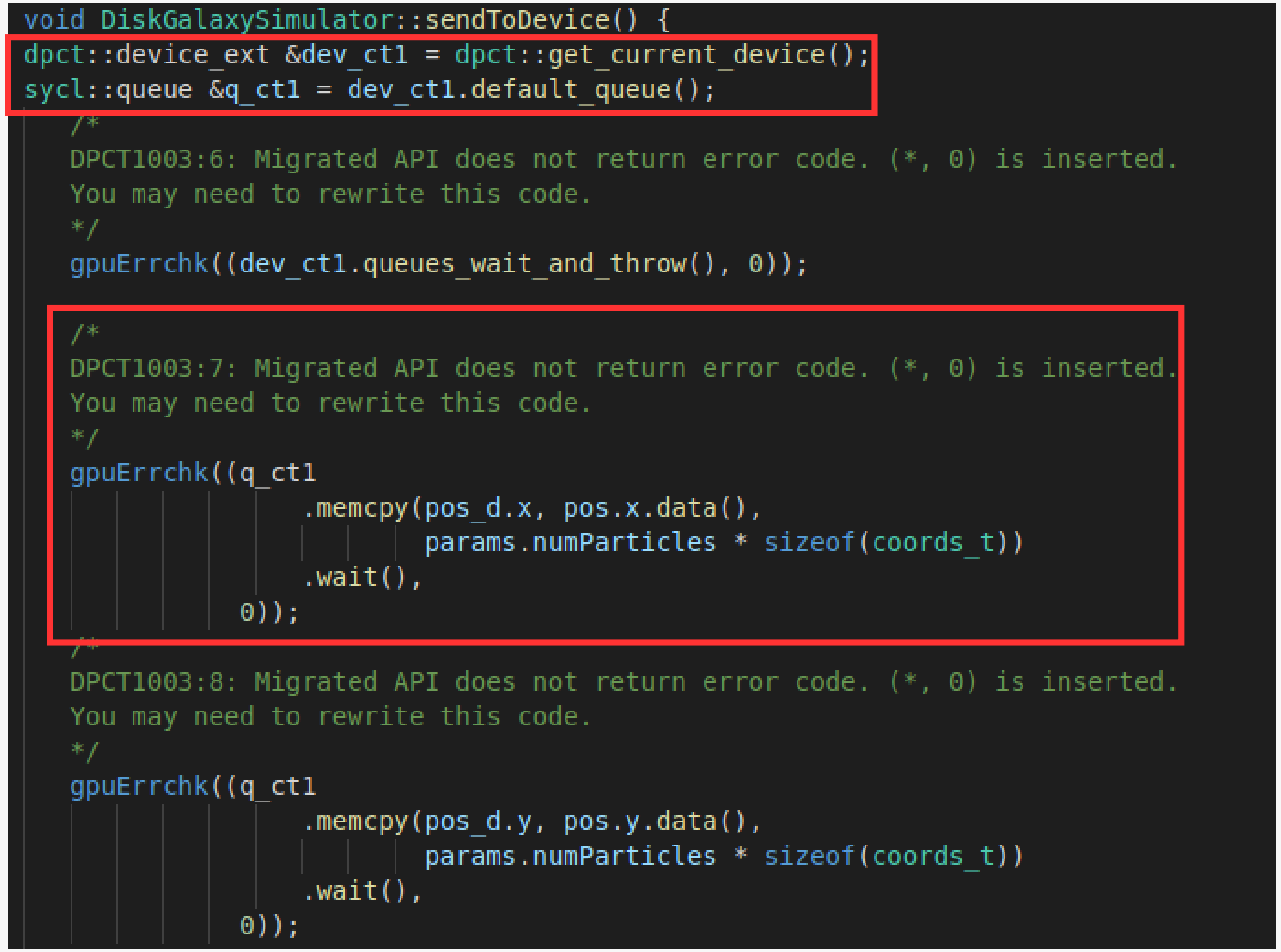
† Certain CUDA language header files may need to be accessible to the Intel® DPC++ Compatibility Tool

DPCT output

DPCT output: Error Handling

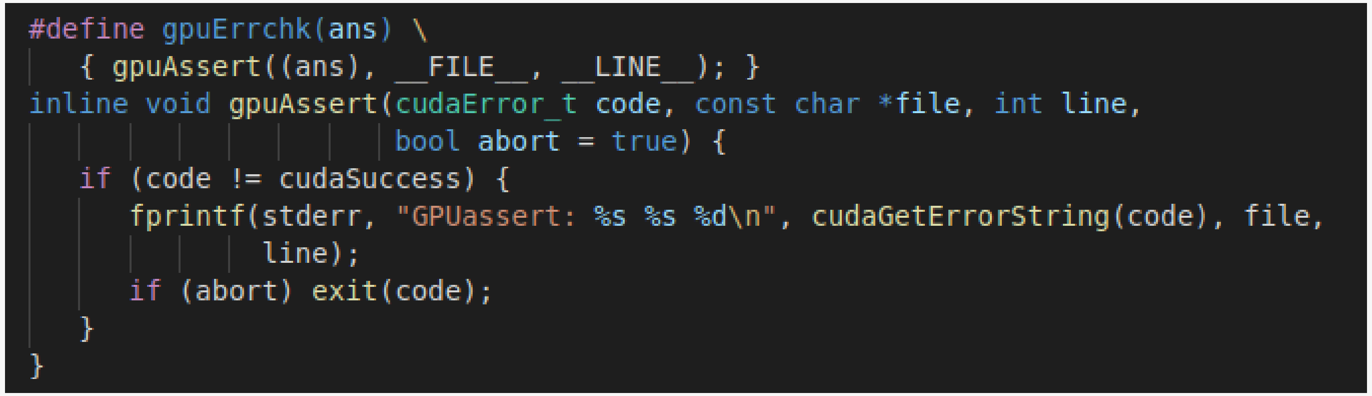


```
void DiskGalaxySimulator::sendToDevice() {
    gpuErrchk(cudaDeviceSynchronize());

    gpuErrchk(cudaMemcpy(pos_d.x, pos.x.data(),
                         params.numParticles * sizeof(coords_t),
                         cudaMemcpyHostToDevice));
    gpuErrchk(cudaMemcpy(pos_d.y, pos.y.data(),
                         params.numParticles * sizeof(coords_t),
                         cudaMemcpyHostToDevice));
    gpuErrchk(cudaMemcpy(pos_d.z, pos.z.data(),
                         params.numParticles * sizeof(coords_t),
                         cudaMemcpyHostToDevice));

    gpuErrchk(cudaMemcpy(vel_d.x, vel.x.data(),
                         params.numParticles * sizeof(coords_t),
                         cudaMemcpyHostToDevice));
    gpuErrchk(cudaMemcpy(vel_d.y, vel.y.data(),
                         params.numParticles * sizeof(coords_t),
                         cudaMemcpyHostToDevice));
    gpuErrchk(cudaMemcpy(vel_d.z, vel.z.data(),
                         params.numParticles * sizeof(coords_t),
                         cudaMemcpyHostToDevice));

    gpuErrchk(cudaDeviceSynchronize());
}
```

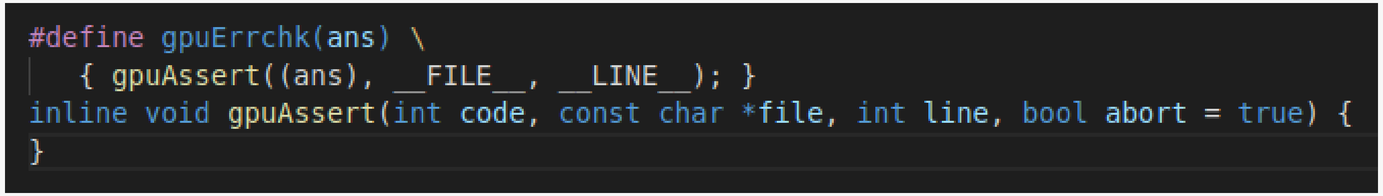


```
void DiskGalaxySimulator::sendToDevice() {
dpct::device_ext &dev_ct1 = dpct::get_current_device();
sycl::queue &q_ct1 = dev_ct1.default_queue();
    /*
    DPCT1003:6: Migrated API does not return error code. (*, 0) is inserted.
    You may need to rewrite this code.
    */
    gpuErrchk((dev_ct1.queues_wait_and_throw(), 0));

    /*
    DPCT1003:7: Migrated API does not return error code. (*, 0) is inserted.
    You may need to rewrite this code.
    */
    gpuErrchk((q_ct1
                   .memcpy(pos_d.x, pos.x.data(),
                           params.numParticles * sizeof(coords_t))
                   .wait(),
               0));
    /*
    DPCT1003:8: Migrated API does not return error code. (*, 0) is inserted.
    You may need to rewrite this code.
    */
    gpuErrchk((q_ct1
                   .memcpy(pos_d.y, pos.y.data(),
                           params.numParticles * sizeof(coords_t))
                   .wait(),
               0));
```

Error codes vs Exceptions



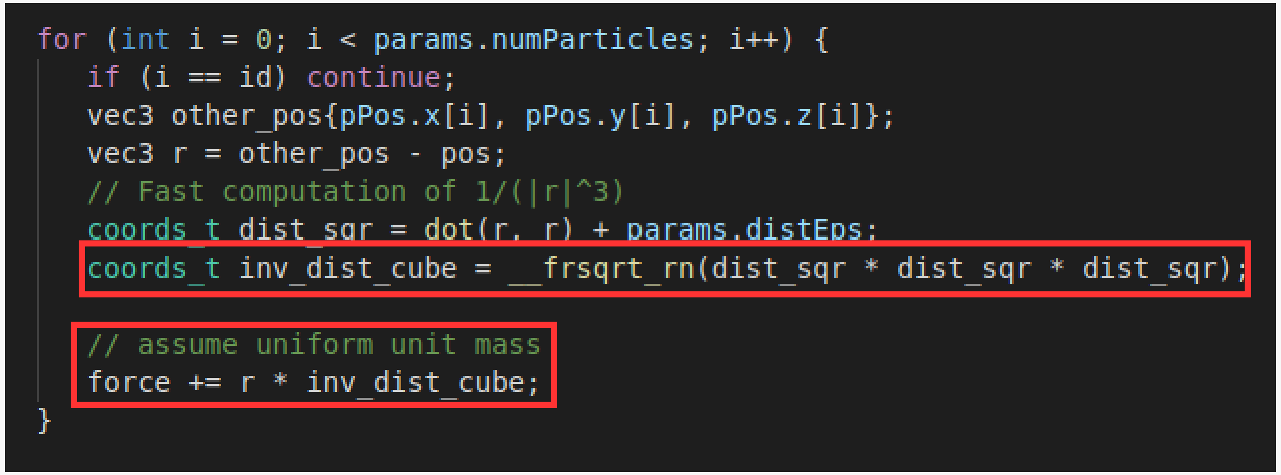
```
#define gpuErrchk(ans) \
    { gpuAssert((ans), __FILE__, __LINE__); }
inline void gpuAssert(cudaError_t code, const char *file, int line,
                      bool abort = true) {
    if (code != cudaSuccess) {
        fprintf(stderr, "GPUassert: %s %s %d\n", cudaGetErrorString(code), file,
                line);
        if (abort) exit(code);
    }
}
```



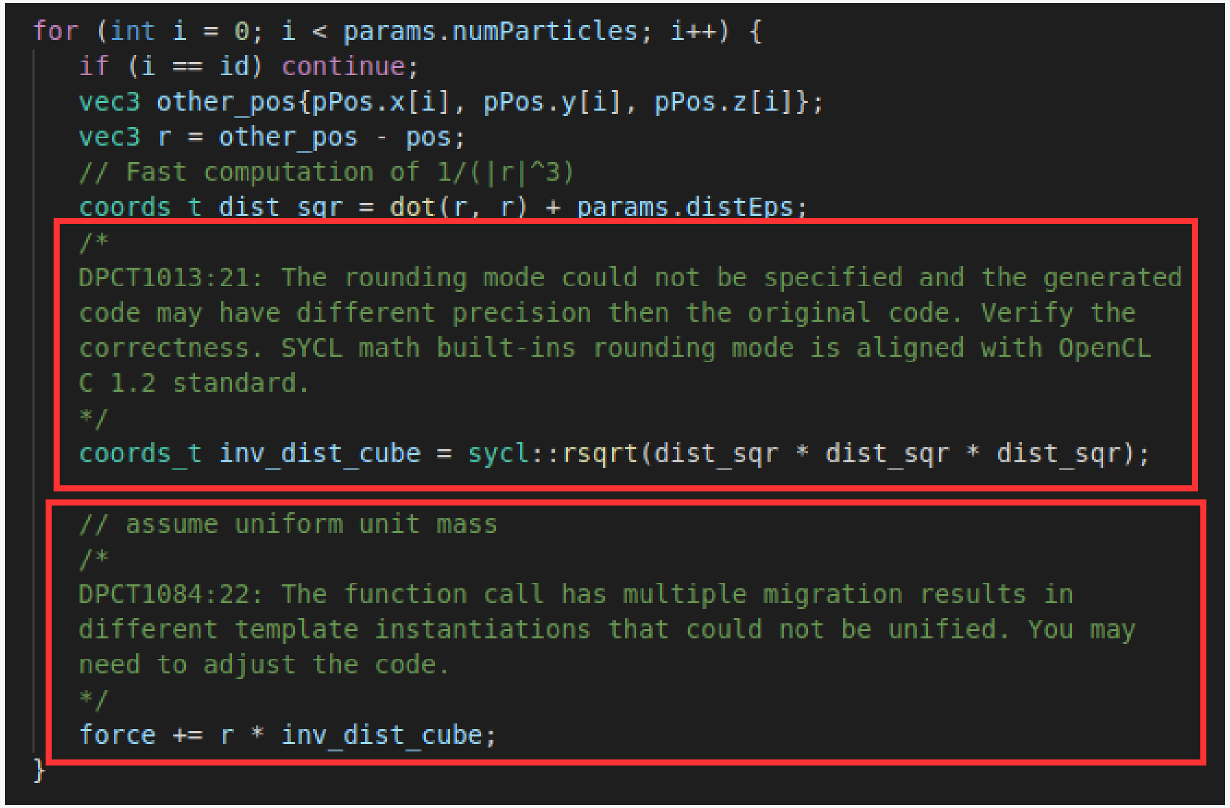
```
#define gpuErrchk(ans) \
    { gpuAssert((ans), __FILE__, __LINE__); }
inline void gpuAssert(int code, const char *file, int line, bool abort = true) {
}
```

DPCT output

- Relies on helper headers for migration
- Verbose comments
- No-op error handling macros



```
for (int i = 0; i < params.numParticles; i++) {
    if (i == id) continue;
    vec3 other_pos{pPos.x[i], pPos.y[i], pPos.z[i]};
    vec3 r = other_pos - pos;
    // Fast computation of 1/(|r|^3)
    coords_t dist_sqr = dot(r, r) + params.distEps;
    coords_t inv_dist_cube = __frsqrt_rn(dist_sqr * dist_sqr * dist_sqr);

    // assume uniform unit mass
    force += r * inv_dist_cube;
}
```



```
for (int i = 0; i < params.numParticles; i++) {
    if (i == id) continue;
    vec3 other_pos{pPos.x[i], pPos.y[i], pPos.z[i]};
    vec3 r = other_pos - pos;
    // Fast computation of 1/(|r|^3)
    coords_t dist_sqr = dot(r, r) + params.distEps;
    /*
    DPCT1013:21: The rounding mode could not be specified and the generated
    code may have different precision then the original code. Verify the
    correctness. SYCL math built-ins rounding mode is aligned with OpenCL
    C 1.2 standard.
    */
    coords_t inv_dist_cube = sycl::rsqrt(dist_sqr * dist_sqr * dist_sqr);

    // assume uniform unit mass
    /*
    DPCT1084:22: The function call has multiple migration results in
    different template instantiations that could not be unified. You may
    need to adjust the code.
    */
    force += r * inv_dist_cube;
}
```

- Informative, slightly pedantic warnings
- Occasionally spurious warnings

DPCT output

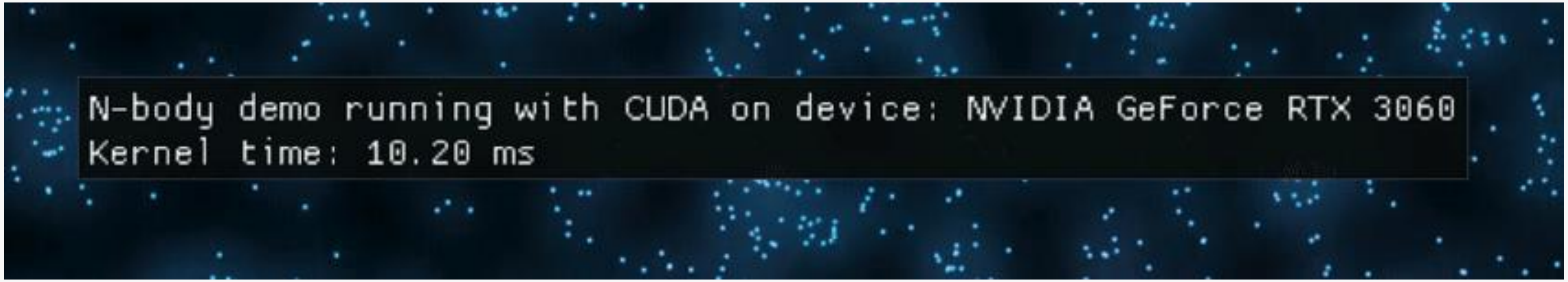
- Relies on helper headers for migration
- Verbose comments
- No-op error handling macros
- Informative, slightly pedantic warnings
- Occasionally spurious warnings

Helper Headers

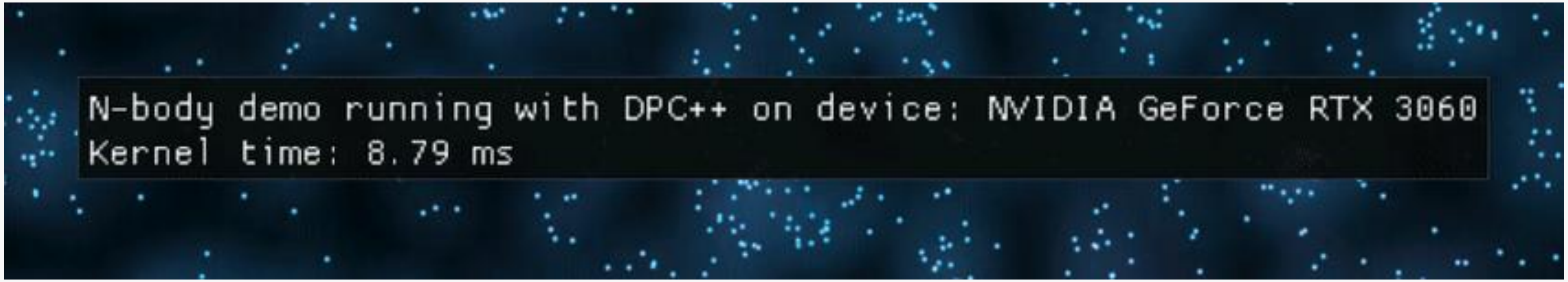
- Variety of helper functions:
  - Device info
  - Software atomics (compare and swap)
  - Memory transfer & info
  - etc...
- All headers generated by default
  - Possibly unneeded, lots of 'dead' code
  - Consider what you need/don't need
- Good for initial rapid porting but advise to remove dependencies later
  - Produce portable code for all SYCL compilers

Performance

- Should match performance on given Nvidia platform



```
N-body demo running with CUDA on device: NVIDIA GeForce RTX 3060
Kernel time: 10.20 ms
```



```
N-body demo running with DPC++ on device: NVIDIA GeForce RTX 3060
Kernel time: 8.79 ms
```

Performance

- Should match performance on given Nvidia platform
- N-body is actually faster!
- You can test this yourself

- What if your code isn't as fast...

Performance Tips

- Profile with Nvidia tools (Nsight Systems/Compute)
- Avoid shared USM when possible
- Experiment with work group size
- Ensure you're inlining as much as possible:
  - -fgpu-inline-threshold=100000
- Ensure you're using hardware atomics if needed:
  - -DSYCL_USE_NATIVE_FP_ATOMICS

DPCT caveats

- Doesn't quite track latest CUDA version
- Only ~90% code translation
- Can't quite handle e.g. cuRAND on device
- Relies on 'helper' headers
- Struggles with kernel range dimensions (1D, 3D?)

But:

- Rapid initial porting to get working code
- Clear comments on required manual coding
- Possible to remove need for helper headers later

SYCLomatic

- Open-source version of DPC++ Compatibility Tool
- May be slightly different – difficult to say
- Now we can:
  - Submit issues
  - Propose solutions
  - Submit PRs

https://github.com/oneapi-src/SYCLomatic

Summary

- DPCT converted our simple n-body code entirely automatically
- Performance is *better* than CUDA!
- The tool is very helpful to rapidly get working code, but...
  - It leaves muddy footprints
  - It doesn't really touch the architecture
  - Result relies on DPCT helper headers

We're Hiring!

codeplay.com/careers/

codeplay®

Enable AI & HPC to be Open, Safe and Accessible to All

@codeplaysoft

info@codeplay.com

www

codeplay.com