CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Introduction to GPUs in HPC

CSCS Summer School 2024

Andreas Jocksch, Prashanth Kanduri, Radim Janalik & Ben Cumming

Diffusion MiniApp Exercise

Unit Tests

Unit testing is an essential development practice:

1. Write tests for features before implementing them;
2. Tests will pass when feature is finished;
3. If a feature is broken in the future, the test will catch it immediately;
4. Tests can also be used to catch performance regressions.

Unit Tests

Unit testing is an essential development practice:

1. Write tests for features before implementing them;
2. Tests will pass when feature is finished;
3. If a feature is broken in the future, the test will catch it immediately;
4. Tests can also be used to catch performance regressions.

The CUDA miniapp uses unit tests to validate the functions in `linalg.cu`

- Initially all of the tests fail
- The tests are run each time the project is built

Use an open source unit testing framework for serious work, e.g. Google Test.

Step 1: Host-Device Data Synchronization

The `Field` class implements storage of the 2D fields

- It is extended for CUDA to store a copy of each field in both host and device memory;
- The data is pointed to by the `Field::host_pointer_` and `Field::device_pointer_` members.

Step 1: Host-Device Data Synchronization

The `Field` class implements storage of the 2D fields

- It is extended for CUDA to store a copy of each field in both host and device memory;
- The data is pointed to by the `Field::host_pointer_` and `Field::device_pointer_` members.

The host and device copies need to be synchronised:

1. **Implement** the member functions that synchronize the host and device data fields in `data.h`
2. **Update** the initial conditions on the device after computation in `main.cu`

Two of the unit tests will pass when this task is finished.

Step 2: Linear Algebra

Implement all of the BLAS level 1 linear algebra kernels in `linalg.cu`:

- Look for the `TODO`s;
- Each kernel is covered by a unit test;
- You will have to do some research into the cublas routines

All of the unit tests will pass when this task is finished.

Step 3: Stencils

Implement the stencils in `operators.cu`

- Look for the `TODO` s.

The number of CG iterations and visualization can be used to validate results.

- **Extra** can you use shared memory for the interior stencil?
- **Extra** can you design an implementation that uses shared memory to implement the interior, boundary and corner stencils in one, simplified kernel?

How does time to solution compare with that for the serial version?

- Compare at different resolutions: 128×128, 512×512, 1024×1024

Questions?