Heterogenous AD with Clad - CUDA kernels differentiation support - Project Roadmap

Ioana Ifrim

Agenda

- ACAT proceedings
- Differentiating CPU programs
- Differentiating GPU programs
- Heterogeneous AD
- Improve test cases and demonstrator
- Q1 remaining goals

12. ACAT proceedings – Q1/II

11. Differentiate CUDA kernels – Q1/II

10. Improve test cases and demonstrators – Q1/II

12. ACAT proceedings – Q1/II

11. Differentiate CUDA kernels – Q1/II

10. Improve test cases and demonstrators – Q1/II

Tasks

ACAT proceedings: http://arxiv.org/abs/2203.06139

12. ACAT proceedings – Q1/II

11. Differentiate CUDA kernels – Q1/II

10. Improve test cases and demonstrators – Q1/II

Tasks

Differentiate CUDA kernels

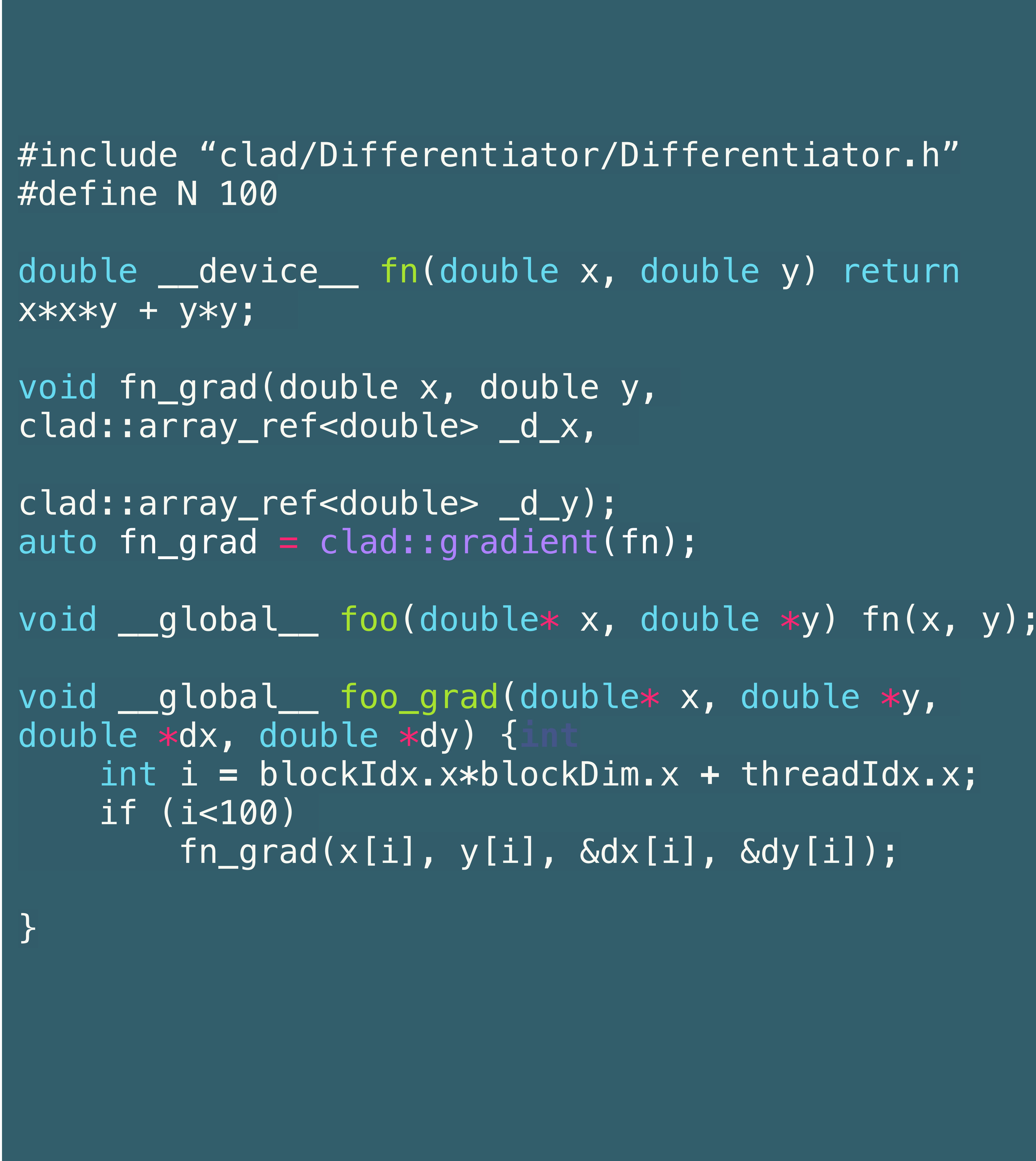
Differentiating CPU programs

- The RHS code is a typical CPU computation of a function’s gradient wrt given values
- Data iteration speeds result from both number of data points and available computational power
- We care deeply about iteration speeds for converging gradient descent procedures
- Increase iteration speeds:
    - accelerate computation by porting CPU C++ code to CUDA
    - distribute computations onto multiple GPUs (compute objective function wrt outputs on GPU:0; return results to the different GPUs.; calculate gradients on each GPU; sum up gradients on GPU:0; use the optimiser on GPU:0)

```
#include “clad/Differentiator/
Differentiator.h”
#define N 100

double fn(double x, double y) {
    return x*x*y + y*y;
}

auto fn_dx = clad::gradient(fn, “x”);

//void fn_grad_0(double x, double y,
//       clad::array_ref<double> _d_x) {..}


double x = 5.0, y = 4.0, d_x = 0;
fn_dx.execute(x, y, &d_x);
```

Differentiating GPU programs

- Clad could be invoked on __device__ functions
- We can compute the derivative of GPU programs by calling a Clad function in a GPU kernel
- __device__ functions can then be called from CUDA kernels => Clad has GPU support (can be called from the device with no conflicts)



```
#include “clad/Differentiator/Differentiator.h”
#define N 100

double __device__ fn(double x, double y) return
x*x*y + y*y;

void fn_grad(double x, double y,
clad::array_ref<double> _d_x,

clad::array_ref<double> _d_y);
auto fn_grad = clad::gradient(fn);

void __global__ foo(double* x, double *y) fn(x, y);

void __global__ foo_grad(double* x, double *y,
double *dx, double *dy) {int
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i<100)
        fn_grad(x[i], y[i], &dx[i], &dy[i]);

}
```

Differentiating GPU programs

- Each parameter’s gradient is computed by a thread

- The gradient matrices are copied back to the host for further computations

```
int main() {

    double *host_x, *x, *host_y, *y, dx, host_dx, dy, host_dy;

    host_x  = (double*)malloc(N*sizeof(double));
    host_y  = (double*)malloc(N*sizeof(double));
    host_dx = (double*)malloc(N*sizeof(double));
    host_dy = (double*)malloc(N*sizeof(double));

    for (int i = 0; i < N; i++) {
         host_x[i] = rand()%100;
         host_y[i] = rand()%100;
    }

    cudaMalloc(&x, N*sizeof(double));
    cudaMalloc(&y, N*sizeof(double));
    cudaMalloc(&dx, N*sizeof(double));
    cudaMalloc(&dy, N*sizeof(double));

    cudaMemcpy(x, host_x, N*sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(y, host_y, N*sizeof(double), cudaMemcpyHostToDevice);

    foo_grad<<<N/256+1, 256>>>(x, y, dx, dy);
    cudaDeviceSynchronize();

    cudaMemcpy(&host_dx, dx, N*sizeof(double),
cudaMemcpyDeviceToHost);
    cudaMemcpy(&host_dy, dy, sizeof(double),
cudaMemcpyDeviceToHost);
}
```

```
#include “clad/Differentiator/Differentiator.h”

// GPU Kernel
__global__
void collide(float* x_in, float* x_out) {
    size_t i = threadIdx.x;
    if (i < 100) {
        x_out[i] += 7 * x_in[i] * x_in[i] - 5 * x_in[i];
    }
}
```

```
// Wrapper CPU function which calls kernel
void kern(float* x_in, float* x_out) {
    collide<<<1, 100>>>(x_in, x_out);
}
```

```
// Main CPU code that calls wrapper function
void compute(int nTimeSteps, float* x_in, float* x_out) {
    for (unsigned int i=0; i<nTimeSteps/2; i++) {
        kern(x_in, x_out);
        kern(x_out, x_in);
    }
}

auto compute_grad = clad::gradient(compute);
```

Heterogeneous AD

Goal: take derivatives of programs running both on CPU and GPU

```
__device__
void collide_body(float* x_in, float* x_out) {
    size_t i = threadIdx.x;
    if (i < 100) {
        x_out[i] += 7 * x_in[i] * x_in[i] - 5 * x_in[i];
    }
}
```

```
// GPU Kernel
__global__
void collide(float* x_in, float* x_out) {
    collide_body(x_in, x_out);
}
```

```
auto grad_collide = clad::gradient(collide_body);


// A wrapper GPU kernel for calling the reverse mode on collide.

_global__
void grad_collide_kern(float* x_in, float* x_out, float* d_x_in,
float* d_x_out) {
    grad_collide(x_in, x_out, d_x_in, d_x_out);
}


// CPU kern call; calls the gradient collide GPU kernel.
void grad_kern(float* x_in, float* x_out,
              float* d_x_in, float* d_x_out) {
    grad_collide_kern<<<1, 100>>>(x_in, x_out, d_x_in, d_x_out);
}
```

WIP - Design Changes

Moved kernel body into a device function

```
__device__
void collide_body(float* x_in, float*
x_out) {
    size_t i = threadIdx.x;
    if (i < 100) {
        x_out[i] += 7 * x_in[i] *
x_in[i] - 5 * x_in[i];
    }
}
```

currently

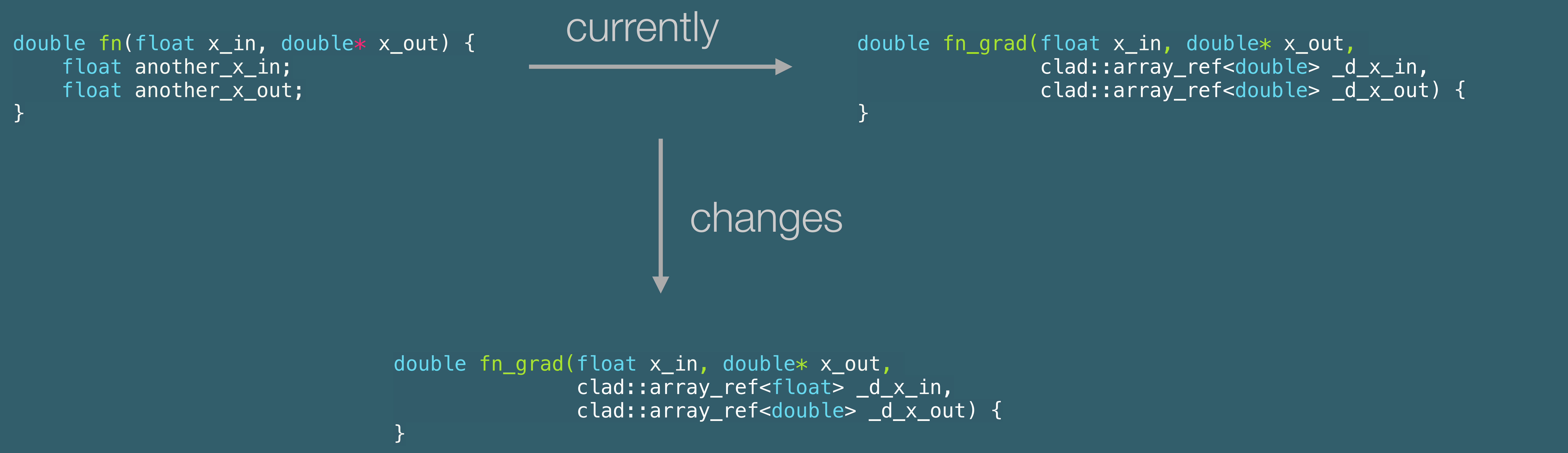
changes

```
// Clad supports differentiating calls to functions that
// Return void like so:

void collide_body(float* x_in, float* x_out) {}

float fn(float* x_in, float* x_out) {
    collide_body(float* x_in, float* x_out);
    return x_in[99]+ x_out[99];
}

auto compute_grad = clad::gradient(fn);
```

```
auto compute_grad = clad::gradient(collide_body);
```

WIP - Design Changes

Modified Clad to accept `void` type functions



```
double fn(float x_in, double* x_out) {
    float another_x_in;
    float another_x_out;
}
```

currently

```
double fn_grad(float x_in, double* x_out,
               clad::array_ref<double> _d_x_in,
               clad::array_ref<double> _d_x_out) {
}
```

changes

```
double fn_grad(float x_in, double* x_out,
               clad::array_ref<float> _d_x_in,
               clad::array_ref<double> _d_x_out) {
}
```

https://github.com/vgvassilev/clad/issues/385

WIP - Design Changes

Modified Clad arrays to take parameter type not function type

```
void __device__ foo(double* x, double *y, double *dx,
double *dy) {int
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i<100)
       x[i] += x[i] * x[i] * 3 + y[i] * y[I];
}
```

WIP - Design Changes

Support CUDA constructs (blockIdx / threadIdx)

12. ACAT proceedings – Q1/II

11. Differentiate CUDA kernels – Q1/II

10. Improve test cases and demonstrators – Q1/II

Tasks

Improve test cases and demonstrators - Binder

Q1 Remaining Goals

- **WIP** Refactoring all the code for the changes and prepare a PR
- **WIP** FIX test case for GPU differentiation
- **WIP** ADD test cases for heterogenous differentiation
- **WIP** Reiterate on the ACAT started PR - Introduce tape usage optimisation in loops

Thank you!

- questions -