HIP Training Day 3 Exercises

1. Explore using `rocgdb` on your "vector add" code from day 1.
    a. Be sure to compile with the flags `-g -ggdb`
    b. You could, for example, remove a host-to-device copy and to try and trigger a failure to test backtracing and switching threads, using `i th` and `t #` where "#" is the thread number to switch to.
    c. Be sure to try and print values of device variables from host code before and after data transfers to see that printing device variables from host code "just works".

2. Construct an example using rocBLAS. The rocBLAS documentation can be found at https://rocblas.readthedocs.io/en/latest/ A rough outline of your program would be:
    a. Create dense matrices on the host.
    b. Copy them to the device
    c. Call rocblas_initialize(). This only needs to be called once.
    d. Perform a dgemm.
    e. Copy the result back to the host and check your answer.
    f. **Bonus**: Set a non-default stream to the rocblas handle so that you can compute on a non-null stream and overlap computation with other data transfers (which would also need to be on a non-null stream)

3. This problem will explore using rocFFT. The rocFFT documentation can be found at https://rocfft.readthedocs.io/en/latest/ A rough outline of your program would be:
    a. Initialize an array on the host, can be random
    b. Copy data to the device
    c. Create an FFT plan
    d. Do a forward FFT
    e. Do an inverse FFT
    f. Copy to the host
    g. Confirm you get back your original array (to within reasonable floating point tolerances)
    h. Cleanup any work buffers, FFT plans

4. **Challenge "hipify" problem**: Use the steps below to clone a CUDA-based version of CHOLLA. Hipify CHOLLA and compile it on the AAC. As a test of your hipification, run the test in "./cholla tests/3D/sound_wave.txt"**. Hint**: We expect the build system modifications to take the most time.
    a. Git clone https://github.com/cholla-hydro/cholla.git cholla-hipify
    b. cd cholla-hipify
    c. git checkout HIP
    d. git reset --hard 04022d50888674a31f53269859e759f05ac21fd0
    e. cd src

5. **Challenge rocFFT problem**: Consider the following 1D poisson equation on a unit domain:

$$\frac{d^2u}{dx^2} = f$$

Suppose we have the following RHS vectors:

$$f_1 = -4\pi^2 \sin(2\pi x)$$

and

$$f_2 = \left(\frac{(x-b)^2 - c}{c^2}\right) \exp\left(-\frac{(x-b)^2}{2c}\right)$$

Their corresponding exact solutions would be:

$$u_1 = \sin(2\pi x)$$

and

$$u_2 = \exp\left(-\frac{(x-b)^2}{2c}\right)$$

Your objective is to obtain the approximate solutions u_1 and u_2 using FFTs. The steps are as follows:

a. Perform forward FFT on the RHS vector
b. Scale the resulting vector by wave numbers (see the `solve` kernel below)
c. Perform inverse FFT

See https://en.wikipedia.org/wiki/Spectral_method for more details.

Below is an incomplete code for computing the 1D Poisson equation using the Fourier Spectral Method. Complete the steps outlined by the comments and verify that your resulting solution vector somewhat matches the exact solutions.

Hint: Use real forward and inverse transforms. See the following documentation on data layout https://rocfft.readthedocs.io/en/latest/real.html

Bonus: Compute the error norm on the GPU. It should return the same error as the host function `error_norm` in step 9.

```
#include <vector>
#include <cmath>
#include "hip/hip_runtime.h"

__global__ void solve(float2 *f, int N) {
    size_t index = blockIdx.x * blockDim.x + threadIdx.x;
    size_t stride = blockDim.x * gridDim.x;
    float k2inv;
    for (size_t i = index; i < N; i += stride) {
        k2inv = (i == 0) ? 1.0f : -1.0f / (i * i * 4 * M_PI * M_PI);
        f[i].x *= k2inv;
        f[i].y *= k2inv;
    }
}

void init1(float *h_f, float *ans, int N) {
    float hx = 1.0f / ((float)N);
    for (size_t i = 0; i < N; i++) {
        ans[i] = sin(2*M_PI*hx*i); // exact solution
        h_f[i] = -4 * M_PI * M_PI * ans[i]; // RHS vector
    }
}
void init2(float *h_f, float *ans, int N) {
    float c2 = 0.01, r2;
    float hx = 1.0f / ((float)N);
    for (size_t i = 0; i < N; i++) {
        r2 = pow(hx * i - 0.5, 2);
        ans[i] = expf(-r2 / (2 * c2)); // exact solution
        h_f[i] = (r2 - c2) / (c2 * c2) * ans[i]; // RHS vector
    }
}

double error_norm(float *h_f, float *ans, int N) {
  double error = 0;
  float relative, invN = 1.0f / N;
  for (size_t i = 0; i < N; i++) {
    relative = (h_f[i] - h_f[0])*invN;
    error += pow(ans[i] - relative,2);
  }
  return error;
}

int main()
{
  // Step 0: Initialize Host data
  size_t N = 1024;
  std::vector<float> h_f(N);
  std::vector<float> ans(N);
  init1(h_f.data(), ans.data(), (int)N);
  //init2(h_f.data(), ans.data(), (int)N);

  // Step 1: Create HIP device buffer(s)

  // Step 2: Copy h_f.data() to device

  // Step 3: Create rocFFT plans

  // Step 3a: Optional work buffers

  // Step 4: Execute forward plan

  // Step 5: Launch solve kernel

  // Step 6: Execute inverse scaling

  // Step 6a: Optional clean up work buffer

  // Step 7: Destroy plan

  // Step 8: Copy data back to h_f.data() and view solution
  for (size_t i = 0; i < N; i++)
    printf("ans[%d] = %f, h_f[%d] = %f\n",(int)i, ans[i], (int)i, (h_f[i] - h_f[0])/N);

  // Step 9: Compute error norm on host
  printf("error norm = %1.3e\n", sqrt(error_norm(h_f.data(),ans.data(),N)));

  // Step 10: Free device buffer and cleanup

  return 0;
}
```