Assignment 4: Qudrature and the Laplacian in openACC due: March 22, 2022– 11:59 PM

GOAL: This problem is to use opeanACC to run multithread code on processor and the NVIDI GPU. The integrtion routine are adapted from HW3 and the Laplacian will be extended to MPI in HW5. The goal is to understand the power of parallel software two classical illustrations.

I Background

Both the exercises here are modification of the two example codes on GitHUB at `OpenACC_demo!`. One is in simple integral in `OpenACC_demo/integrate!` and the second poisson code in `OpenACC_demo/Poissson2D!` The frist task is continue to run and explore these demos. The exercise here can stat with these a templates for this exercise. I am also distributing the HW4 with a larger explaination of the Laplace problem. The Laplacian exercise in here and HW5 are the same: One in openACC and one in MPI. In class the lectures will start with openACC and then MPI but you may choose to work on both together. Afterall the Laplacian problem is single problem with two parallelization method. Also HW4 does the easier case of 1D and HW5 the case of 2D.

II Coding Exercise #1. OpenACC with Uniform and Gaussian integration

Consider the large 4 dimensional integral:

$$P(m) = \int_{-1}^{1} dp_0 \int_{-1}^{1} dp_1 \int_{-1}^{1} dp_2 \int_{-1}^{1} dp_3 \frac{1}{\left(\sin^2(\pi p_0/2) + \sin^2(\pi p_1/2) + \sin^2(\pi p_2/2) + \sin^2(\pi p_3/2) + m^2\right)^2} \tag{1}$$

What is this integral anyway? Glad you asked, although there is no need to know this. It is called a one loop Feynman diagram on a 4d hyper-cubic lattice. It is related to the probability in quantum mechanics of two particles (of mass m), starting at the same point, performing a random walk in space and time, and ending up eventually together at the same place at the same time! To get this random walk into this cute integral, we first put all of the space-time on an infinite 4D lattice and let the particle hop from point to point. Using the magic of Fourier transform we convert to velocities $v_i = p_i/m$ and energy $E = p_0$. Lots of applications in engineering materials and physics need to do such integrals. Each time you add one particle you get another 4 dimensional integral. These are real computational challenges which is a very active research area.

As a first step, integrate this for $m = 1$. Now if you want an accurate result from Gaussian integration there are a lot of sums, e.g. $N = 16$ implies $16^4 = 65536$ terms. You'll want to parallelize this integral. Performing an integral is an example of a *reduction*, where we are summing a series of numbers. Reductions can be finicky in parallel code due to race conditions—in a naïve implementation,

multiple threads can try to accumulate a sum variable at the same "time". Thankfully, OpenACC can handle this. See the instruction on GitHub for running OpenACC at the SCC with a Makefile that you can copy and modify for this program.

For a four dimensional integral, you should have four nested `for` loops. Include an appropriate `pragma` to parallelize the outer loop. Try this for a range of m: for large m, the probability of them ever meeting is small, while for lighter particles, there is a much higher chance they'll meet... and the integral becomes more difficult. Scan over m from $\frac{1}{\sqrt{10}}$ to 10 in steps of $10^{-1/6}$ (so 10 different values). You'll see some inconsistency as a function of N for very small m—what's the reason for this? Don't be worried about it, ultimately. While it's not perfect, you should see a scaling $P(m) \sim -\log(m)$ at small m: make a plot showing this. Remember to set your axes properly to make it clear!

Write a main program `test_integrate_feynman.cpp` where you perform this integral in parallel, and create a plot `integral_scaling` (or whatever appropriate extension your `gnuplot` or `Mathematica` supports) showing the log behavior described above. The program should be compiles by a makefile called `makeACC` which is run with the command `make -k -f makeACC` and placed in your submission file HW4 on the SCC with the `integral_scaling`

By the way it is easy to see what the answer is as we take $m^2 \to \infty$. The leading term is

$$P(m) \simeq 16/m^4 \ .$$

Why?

Setting

$$x = (\sin^2(\pi p_0/2) + \sin^2(\pi p_1/2) + \sin^2(\pi p_2/2) + \sin^2(\pi p_3/2))/m^2 \ ,$$

you can expand the denominator as

$$(1/m^4)/(1+x)^2 = (1/m^4)(1+x)^{-2} \simeq (1/m^4)(1 - 2x + 3x^2 - 4x^3 + 5x^4 \cdots) \ .$$

The next term is a easy integral over sin squares so you can write this down as well. You will find that at large m Gaussian Quadratures is a lot better than near $m^2 = 0$. Why?

III Coding Exercise #2. OpenACC with 1D Laplacian Problem

Write a program to solve the one-dimensional finite difference heat equation with fixed boundaries using the methods of Gauss-Seidel and Red-Black iterations. These algorithms are described in depth, albeit in the two-dimensional case, in the class notes. For your convenience, we've provided a base code which solves this problem with Jacobi iterations on the class GitHub in the file `MPI_OPM_examples/n05Jacobi/jacobi_scal.cpp`. Remember, each algorithm is nearly identical, with only a few changes in the iterative loop!

In the reference code, we accumulate a solution in the array `double T[N +1]` with values $T[i]$ spanning $i = 0, ..., N$. The fixed boundary values live in the sites $T[0] = T[N] = 0$. The source now

lives at the mid-point, $i = N/2$. The code tracks the iterative solution by outputting the residual $r = b - AT$ every 1000 iterations, that is, printing the norm of the vector

$$r[i] = b[i] - AT[i] = b[i] - (T[i] - \frac{1}{2}[T[i-1] + T[i+1]]). \tag{2}$$

(Note we have chosen to define A by delving by $1/2$ relative to the discrete one-dimensional Laplace matrix – a very simple "preconditioner".)

The program stops iterating when the relative residual reaches a target $\epsilon = 10^{-6}$, that is, the constraint

$$\frac{\sqrt{\sum_{i=1}^{N-1} r[i] * r[i]}}{\sqrt{\sum_{i=1}^{N-1} b[i] * b[i]}} < 10^{-6} \tag{3}$$

Since the input is at a single point when you increase N a better metric for convergence might the root means square (RMS) error in the residuals: $\sqrt{N^{-1} \sum_{i=1}^{N-1} r[i] * r[i]} < 10^{-6}$.

The goal of this assignment is to compare how each method converges by tracking the relative residual as a function of the iteration count at large N. You should produce a new source file for each method: `gauss_seidel_scal.cpp` and `red_black_scal.cpp`. For this part of the assignment, keep N fixed at the default value of 512. You should also make a plot which shows the relative residual as a function of the iteration count, one curve per method, all overlaid on the same figure.

As a next step, parallelize the code using MPI! As we discussed, Gauss-Seidel can't be parallelized well, so we'll skip that. Instead, just parallelize the Jacobi iterations using the appropriate MPI instructions. We start by explain the full solved 1d example using mpi at : `MPI_OPM_examples/n05Jacobi/jacobi_`

Here you are asked to modify the poisson code in `OpenACC_demo/Poissson2D!`Since you will be asked to the 2D example in HW5 you might want to do this one here at the same time (See HW5 for details.

The 2D geometry has boundaries are lines just off the edge of the square with $x = -1, L$ for all y and $y = -1, L$ for all x. These are sometime called *ghost zones*. Including the ghost zones, this means you need $L + 2$ sites in each dimension—for this reason, it's convenient in C to layout the arrays to include these buffer zones: `double T[L+2][L+2]`.

With that memory layout convention, the zero boundaries lead to `T[0][y] = T[L+1][y] = 0` for all y, and likewise `T[X][0] = T[X][L+1] = 0` for x. The relaxation routine has a very simple iterative scheme:

$$T_c[x][y] = \frac{1}{4}(T_c[x+1][y] + T_c[x-1][y] + T_c[x][y+1] + T_c[x][y-1]) + b_0[x][y] \tag{4}$$

Now put in a hot source in the midle of the plate $b_0 = 0$ except $b_0[L/2][b[L/2] = 100$.

What values of x and y should you loop over?

A cute trick is to use an off-set index $T[x+1][y+1]$ so with interior values are still given by $x, y = 0, 1, \cdots L-1$ and the boundaries are at $x, y = -1$ and $x, y = L$. Defining off-set array,

$T_c[x][y] = T[x+1][y+1]$, the boundaries are now stored in $T_c[-1][y], T_c[x][-1]$ and $T_c[L][y], T_c[x][L]$ as they should. This allows one to have a very simple iterative scheme for the interior,

$$T_c[x][y] = \frac{1}{4}(T_c[x+1][y] + T_c[x-1][y] + T_c[x][y+1] + T_c[x][y-1]) + b_0[x][y] \tag{5}$$

with loops over $x, y = 0, 1, \cdots, L-1$.

With this in mind, the first part of the assignment is to write a serial code in 2D to find the solution to the temperature profile. This is a small modification of the one-dimensional code using Jacobi. Sweep over a range of problem sizes, $L = 256$ to $L = 2048^2$, in multiplicative steps of 2 and time it.

NOTE HW5 compare the 2D Laplacian MPI code and the 2D openACC