I Extra Credit on Parallel Integration

GOAL: Here is nice set of problems to extend integration to multi-dimensional integrals and keep up your new aquired openMP skills. Don't want to have you be bored over the spring beak. It is REALLY OPTIONAL. You may have better things to to do so you can also wait till after Spring Break of course and we will discuss this in class. Multidimension integration is simple generalization that rapidly ad common engineering and science problem that really requires big data set and high performance parallel software.

As a quick refresher, a single dimensional integral of a function $f(x)$ from -1 to 1 is written as:

$$\int_{-1}^{1} f(x)\ dx \tag{1}$$

On the other hand the two dimensional integral of a function $g(x, y)$ inside the square defined by $-1 < x < 1, -1 < y < 1$ can be written as:

$$\int_{x\in[-1,1]y\in[-1,1]} g(x, y)\ dA \tag{2}$$

I'm sure this is different from how you may have seen it in a multivariable calculus class. It's actually a non-trivial proof that this is equivalent, but a much clearer (and likely familiar) formulation is:

$$\int_{-1}^{1} dy \int_{-1}^{1} dx\ g(x, y) \tag{3}$$

Now that the function is written in this form, it becomes a bit easier to now see how we'll numerically approximate it:

$$\int_{-1}^{1} dy \int_{-1}^{1} dx\ g(x, y) \approx \int_{-1}^{1} dy \sum_{i=1}^{N} w_i g(x_i, y) \tag{4}$$

$$\approx \sum_{j=1}^{N} \sum_{i=1}^{N} w_i w_j g(x_i, y_j) \tag{5}$$

This approximation generalizes trivially to even higher dimensions.

I.1 Integrating a few more functions

This problem should seem familiar: we're going to integrate a few problems on the interval $[-1, 1]$, except in multiple dimensions. Write a main program `test_integrate_2d.c` that performs the following three integrals using the Trapezoidal rule and Gaussian integration for $N = 2$ to 24 in the sum:

$$\int_{-1}^{1} dy \int_{-1}^{1} dx \, [x^8 + y^8 + (y-1)^3(x-3)^5] = ? \tag{6}$$

$$\int_{-1}^{1} dy \int_{-1}^{1} dx \, \sqrt{x^2 - y^2 + 2} = ? \tag{7}$$

$$\int_{-1}^{1} dy \int_{-1}^{1} dx \, e^{-x^2 - \frac{y^2}{8}} \cos(\pi x) \sin(\frac{\pi}{8} y) = ? \tag{8}$$

Next, let's try a 4 dimensional integral:

$$P(m) = \int_{-1}^{1} dp_0 \int_{-1}^{1} dp_1 \int_{-1}^{1} dp_2 \int_{-1}^{1} dp_3 \frac{1}{\left(\sin^2(\pi p_0/2) + \sin^2(\pi p_1/2) + \sin^2(\pi p_2/2) + \sin^2(\pi p_3/2) + m^2\right)^2} \tag{9}$$

What is this integral anyway? Glad you asked! It is called a one loop Feynman diagram. It is related to the probability[1] 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 $v_i = p_i/m$b and energy $E = p_0$. Lots of applications in engineering materials and high energy physics need to do such integrals. Each time you add one particle you get another 4 dimensional integral. These are real computational challenges.

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, OpenMP can handle this. Here's an example of summing a one dimensional vector in OpenMP:

```
double sum = 0.0; #pragma omp parallel for shared(v1) reduction (+:sum)
for(int i = 0; i < SIZE; i++)
sum = sum + v1[i];
```

In this example, `sum` is the variable we accumulate into and `v1` is the array we're summing over. All of the threads access `v1`, which is why it's noted as shared. We also need to tell OpenMP what variable we're reducing into, thus the part `reduction (+:sum)`, where `sum` is the variable name (which could

[1] Emphasis on *related*, so don't be worried if it's greater than 1.

be anything, such as `wheelbarrel`, and the `+` indicates what operation we're performing (it could be `-`, `*`, etc).

For a four dimensional integral, you should have four nested `for` loops (unless you want to try to solve this by recursion, in which case you'll still have an outer `for` loop). 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).[2] 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.c` where you perform this integral in parallel, and create a plot `integral_scaling.pdf` (or whatever appropriate extension your version of `gnuplot` supports) showing the log behavior described above.

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?

II Submitting This Extra Credit

Any time over the Spring Break or just bring it into the Lecture on March 11 or March 13. If you want to email it as a **tarball** send it to bualghpc@gmail.com. Include your name in the tarball filename.

If you're not familiar with tar, here's a sample instruction that perhaps I would use:

```
tar -cvf evan_weinberg_extra4.tar [directory with files]
```

You may want to include other files. **Do NOT include a compiled executable!** That's a dangerous, unsafe practice to get into.

[2] In the original version of the assignment, we had you scan over m from 0.0001 to 100, steps of $\sqrt{10}$, which gave poor results for small m.