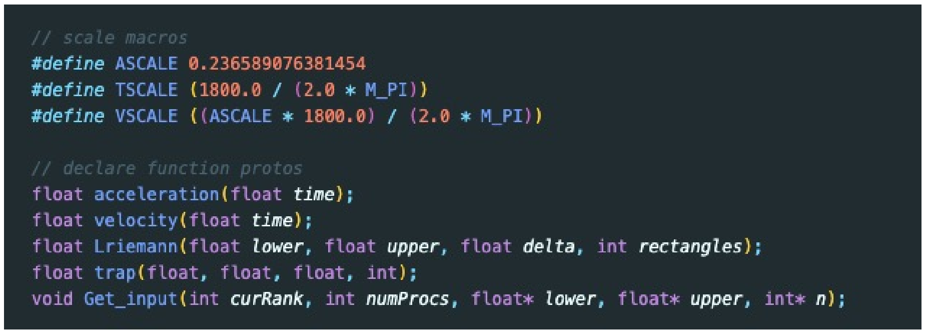
Keoni Burns
report 3
CSCI 551

Exercise 3

Q1) Left Riemann Sum

Code



```
// scale macros
#define ASCALE 0.236589076381454
#define TSCALE (1800.0 / (2.0 * M_PI))
#define VSCALE ((ASCALE * 1800.0) / (2.0 * M_PI))

// declare function protos
float acceleration(float time);
float velocity(float time);
float Lriemann(float lower, float upper, float delta, int rectangles);
float trap(float, float, float, int);
void Get_input(int curRank, int numProcs, float* lower, float* upper, int* n);
```

Figure 1: macros

above are my changes to the code for both left and trapezoidal riemann sum programs

- created macros for the functions that calaculate the scaling factors for our given circumstance
- switched from c to cpp because i felt more comfortable using the lang
- added a time struct to keep track of monotonic raw clock time
- additionally i added trap function to the left_riemann.cpp because they both use floats and so it sseemed appropriate

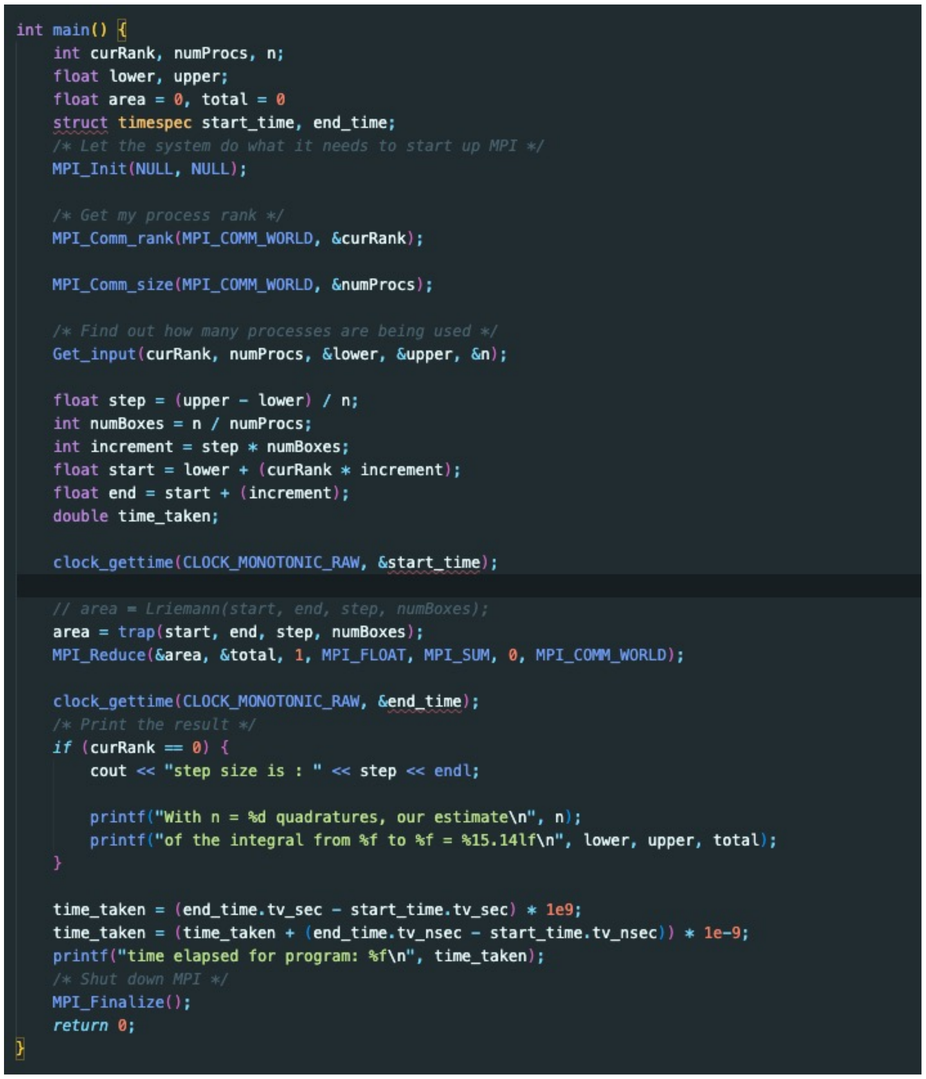
Speed Up

single threaded

mpi implementation

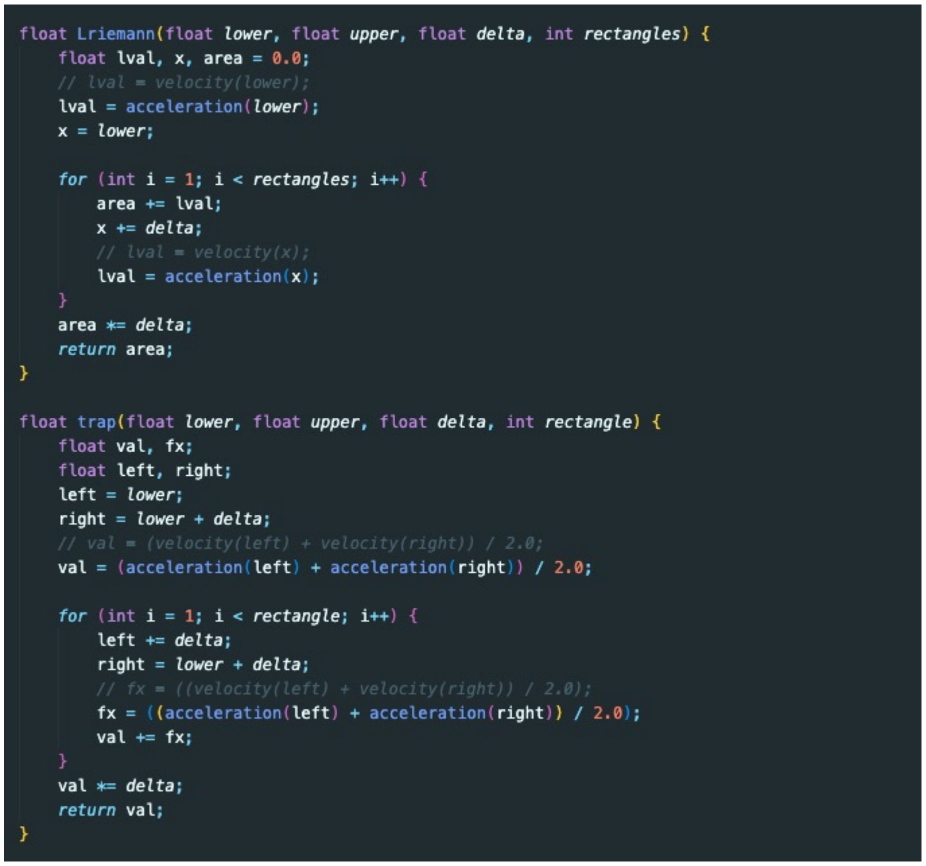
$$\boxed{\textit{speed up} = \frac{0.002259s}{0.003557s} \approx 0.63}$$

- there is minor speed up but not significant



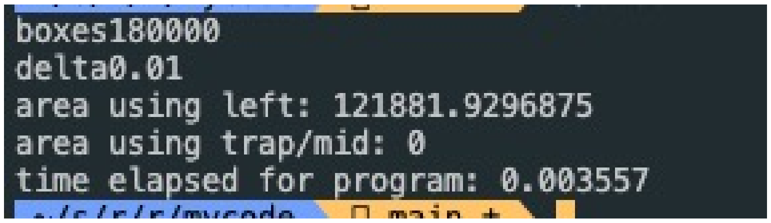
```
int main() {
    int curRank, numProcs, n;
    float lower, upper;
    float area = 0, total = 0
    struct timespec start_time, end_time;
    /* Let the system do what it needs to start up MPI */
    MPI_Init(NULL, NULL);

    /* Get my process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &curRank);

    MPI_Comm_size(MPI_COMM_WORLD, &numProcs);

    /* Find out how many processes are being used */
    Get_input(curRank, numProcs, &lower, &upper, &n);

    float step = (upper - lower) / n;
    int numBoxes = n / numProcs;
    int increment = step * numBoxes;
    float start = lower + (curRank * increment);
    float end = start + (increment);
    double time_taken;

    clock_gettime(CLOCK_MONOTONIC_RAW, &start_time);

    // area = Lriemann(start, end, step, numBoxes);
    area = trap(start, end, step, numBoxes);
    MPI_Reduce(&area, &total, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);

    clock_gettime(CLOCK_MONOTONIC_RAW, &end_time);
    /* Print the result */
    if (curRank == 0) {
        cout << "step size is : " << step << endl;

        printf("With n = %d quadratures, our estimate\n", n);
        printf("of the integral from %f to %f = %15.14lf\n", lower, upper, total);
    }

    time_taken = (end_time.tv_sec - start_time.tv_sec) * 1e9;
    time_taken = (time_taken + (end_time.tv_nsec - start_time.tv_nsec)) * 1e-9;
    printf("time elapsed for program: %f\n", time_taken);
    /* Shut down MPI */
    MPI_Finalize();
    return 0;
}
```

Figure 2: main



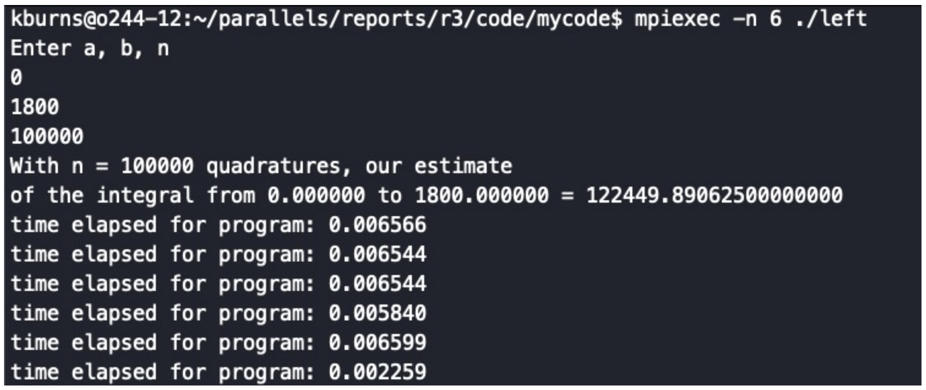
```c
float Lriemann(float lower, float upper, float delta, int rectangles) {
    float lval, x, area = 0.0;
    // lval = velocity(lower);
    lval = acceleration(lower);
    x = lower;

    for (int i = 1; i < rectangles; i++) {
        area += lval;
        x += delta;
        // lval = velocity(x);
        lval = acceleration(x);
    }
    area *= delta;
    return area;
}

float trap(float lower, float upper, float delta, int rectangle) {
    float val, fx;
    float left, right;
    left = lower;
    right = lower + delta;
    // val = (velocity(left) + velocity(right)) / 2.0;
    val = (acceleration(left) + acceleration(right)) / 2.0;

    for (int i = 1; i < rectangle; i++) {
        left += delta;
        right = lower + delta;
        // fx = ((velocity(left) + velocity(right)) / 2.0);
        fx = ((acceleration(left) + acceleration(right)) / 2.0);
        val += fx;
    }
    val *= delta;
    return val;
}
```

Figure 3: left_riemann



```
boxes180000
delta0.01
area using left: 121881.9296875
area using trap/mid: 0
time elapsed for program: 0.003557
```

Figure 4: single_left



```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./left
Enter a, b, n
0
1800
100000
With n = 100000 quadratures, our estimate
of the integral from 0.000000 to 1800.000000 = 122449.89062500000000
time elapsed for program: 0.006566
time elapsed for program: 0.006544
time elapsed for program: 0.006544
time elapsed for program: 0.005840
time elapsed for program: 0.006599
time elapsed for program: 0.002259
```

Figure 5: mpi_left

- this is due to the problem scaling and our questions not explicitly pertaining to speed up but rather accuracy

Accuracy and Precision

the formula for percent error is as follows:

$$\boxed{\frac{(measured\ value - theoretical)}{true\ value}}$$

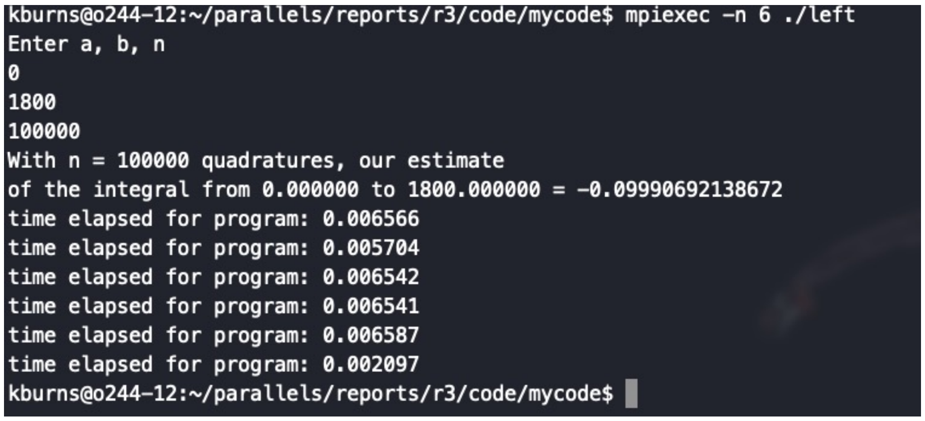
- from the image above we can see that our value at t = 1800 is 122449.8906250

substuting values from the given excel sheet as our theoretical we get

$$\boxed{\frac{(122449.8906250 - 122000)}{122000} \times 100 \approx 0.36829\ percent\ error}$$

- as we can see there is a 0.36% margin of error with this result for position approximation

unfortunately the velocity my program produced was not correct and should we try to plug it into the formula we would get infinity because we would be dividing by 0



```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./left
Enter a, b, n
0
1800
100000
With n = 100000 quadratures, our estimate
of the integral from 0.000000 to 1800.000000 = -0.09990692138672
time elapsed for program: 0.006566
time elapsed for program: 0.005704
time elapsed for program: 0.006542
time elapsed for program: 0.006541
time elapsed for program: 0.006587
time elapsed for program: 0.002097
kburns@o244-12:~/parallels/reports/r3/code/mycode$
```

Figure 6: left_accel

Analysis

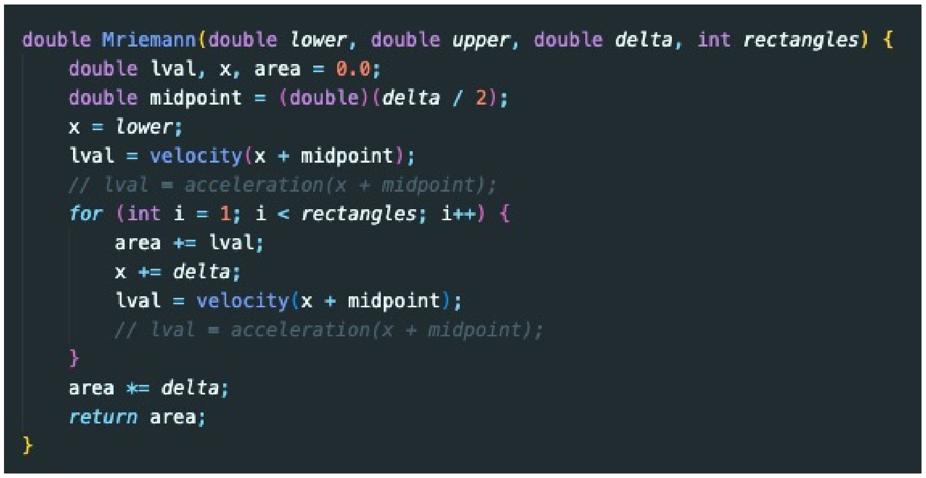
- My results for position have a 0.36% error when tested against the theoretical value given to us
  - this error percent is most likely due to floating point being unable to hold that much data given that it is only 4 bytes, 32 bits. Ergo precision can only be guaranteed up to 7 decimal places. This should not matter though because we were off by about 450 meters (this would not be good in real life)
  - the results from acceleration are most likely due to inefficient programming on my part. Were we to get accurate results we should have outputted something along the lines of 3x10ˆ-14.
    - it should be noted that running on a single thread my acceleration results were accurate and more correct. i'm not too sure what happened along the way.

Q2) Mid-Riemann sum

Code

the image shows my implementation of mid-point riemann

- it should be noted that program is nearly identical to my left_riemann.cpp and the only differences are this function and the typecasting of the variables from float to double

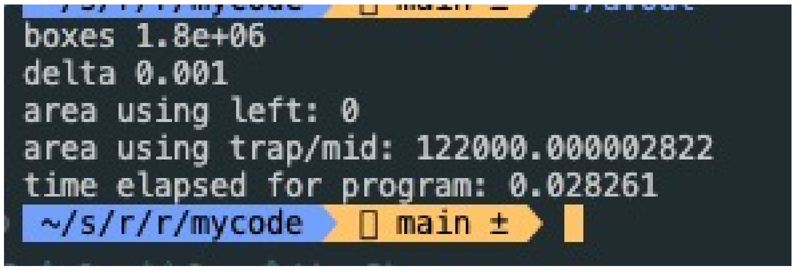


```
double Mriemann(double lower, double upper, double delta, int rectangles) {
    double lval, x, area = 0.0;
    double midpoint = (double)(delta / 2);
    x = lower;
    lval = velocity(x + midpoint);
    // lval = acceleration(x + midpoint);
    for (int i = 1; i < rectangles; i++) {
        area += lval;
        x += delta;
        lval = velocity(x + midpoint);
        // lval = acceleration(x + midpoint);
    }
    area *= delta;
    return area;
}
```

Figure 7: mid_riemann

Speed Up

single threaded



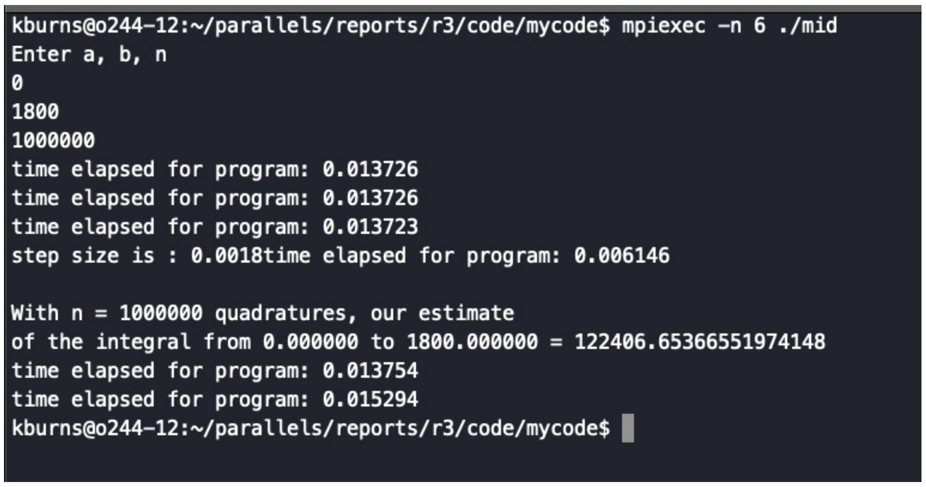
```
boxes 1.8e+06
delta 0.001
area using left: 0
area using trap/mid: 122000.000002822
time elapsed for program: 0.028261
~/s/r/r/mycode  main ±
```

Figure 8: single_mid

using mpi

$$\boxed{\textit{speed up} = \frac{0.013723s}{0.028261s} \approx 0.485}$$

- once again we can see that our speed up isn't by any means impressive and it is due to the scale of our problem and what our objectives are in this exercise.



```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./mid
Enter a, b, n
0
1800
1000000
time elapsed for program: 0.013726
time elapsed for program: 0.013726
time elapsed for program: 0.013723
step size is : 0.0018time elapsed for program: 0.006146

With n = 1000000 quadratures, our estimate
of the integral from 0.000000 to 1800.000000 = 122406.65366551974148
time elapsed for program: 0.013754
time elapsed for program: 0.015294
kburns@o244-12:~/parallels/reports/r3/code/mycode$ 
```

Figure 9: mpi_mid

Accuracy and Precision

if we use our percent error formula on the area we measured against the theoretical we get the following:

$$\boxed{\frac{(122406.65366551974148 - 122000)}{122000} \times 100 \approx 0.33308 \textit{ percent error}}$$

- it can be seen that once again our percent error, eventhough below a single percent, is still significant enough to be dangerous in critical situations. 400 meters is a large distance, its about 3.5 football fields.

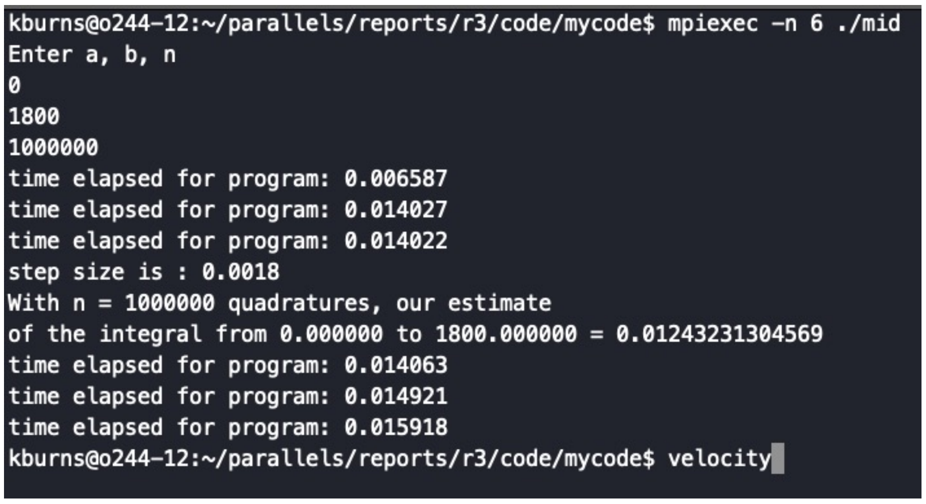
- once again the velocity at t=1800 is not nearly small enough to be considered correct and were it to be an accurate value it would be smaller by about 10~12 orders of magnitude

Analysis

sources of error

- poor implementation of code
- accuracy was simply not there which may have been due to how i was dividing up the work between the processes

notes



```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./mid
Enter a, b, n
0
1800
1000000
time elapsed for program: 0.006587
time elapsed for program: 0.014027
time elapsed for program: 0.014022
step size is : 0.0018
With n = 1000000 quadratures, our estimate
of the integral from 0.000000 to 1800.000000 = 0.01243231304569
time elapsed for program: 0.014063
time elapsed for program: 0.014921
time elapsed for program: 0.015918
kburns@o244-12:~/parallels/reports/r3/code/mycode$ velocity
```

Figure 10: mid_vel

- precision didn't seem to be nearly as much of an error simply because the data i got was not accurate enough to give a precision error.

Q3 & Q4) Trapezoidal Riemann Sum

Code

- the image above showss the logic of my trap program which took inspiration from some starter code given

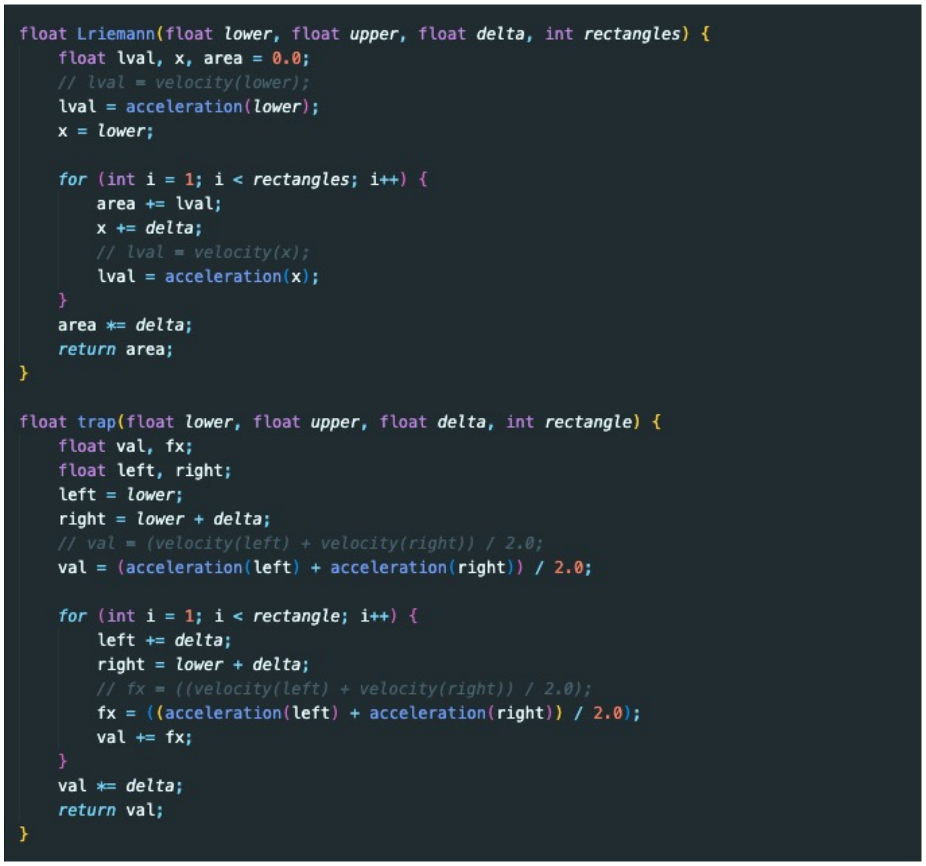
Speed Up

single threaded

mpi trap

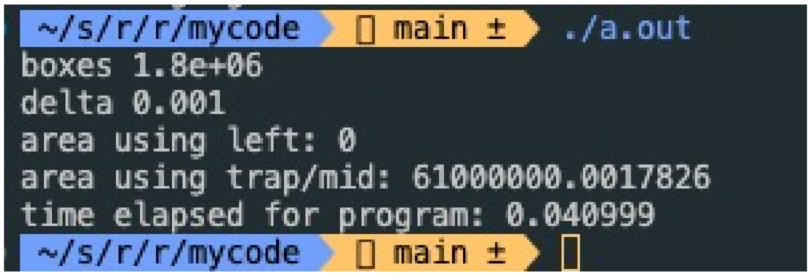
$$\boxed{speed\ up = \frac{0.025884s}{0.040999s} \approx 0.631}$$

- not significant enough to make any remarks about but it seemed relevant to include



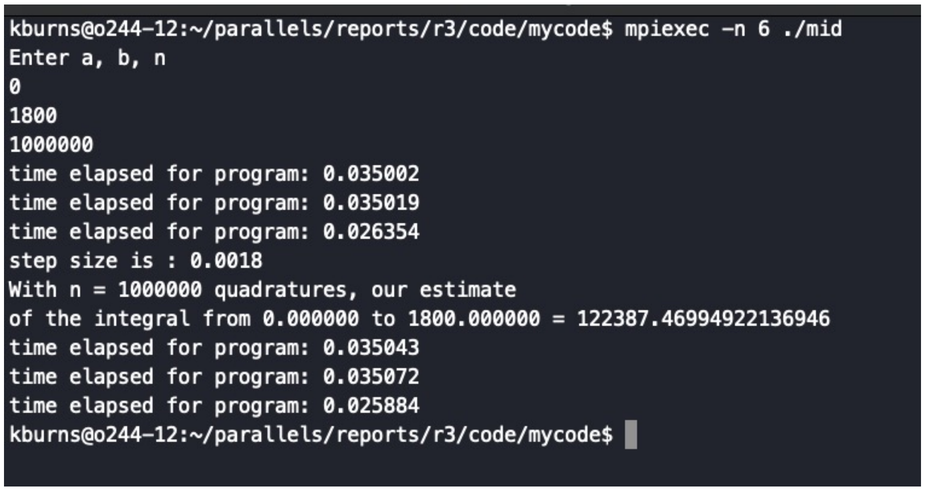
```c
float Lriemann(float lower, float upper, float delta, int rectangles) {
    float lval, x, area = 0.0;
    // lval = velocity(lower);
    lval = acceleration(lower);
    x = lower;

    for (int i = 1; i < rectangles; i++) {
        area += lval;
        x += delta;
        // lval = velocity(x);
        lval = acceleration(x);
    }
    area *= delta;
    return area;
}

float trap(float lower, float upper, float delta, int rectangle) {
    float val, fx;
    float left, right;
    left = lower;
    right = lower + delta;
    // val = (velocity(left) + velocity(right)) / 2.0;
    val = (acceleration(left) + acceleration(right)) / 2.0;

    for (int i = 1; i < rectangle; i++) {
        left += delta;
        right = lower + delta;
        // fx = ((velocity(left) + velocity(right)) / 2.0);
        fx = ((acceleration(left) + acceleration(right)) / 2.0);
        val += fx;
    }
    val *= delta;
    return val;
}
```

Figure 11: trap



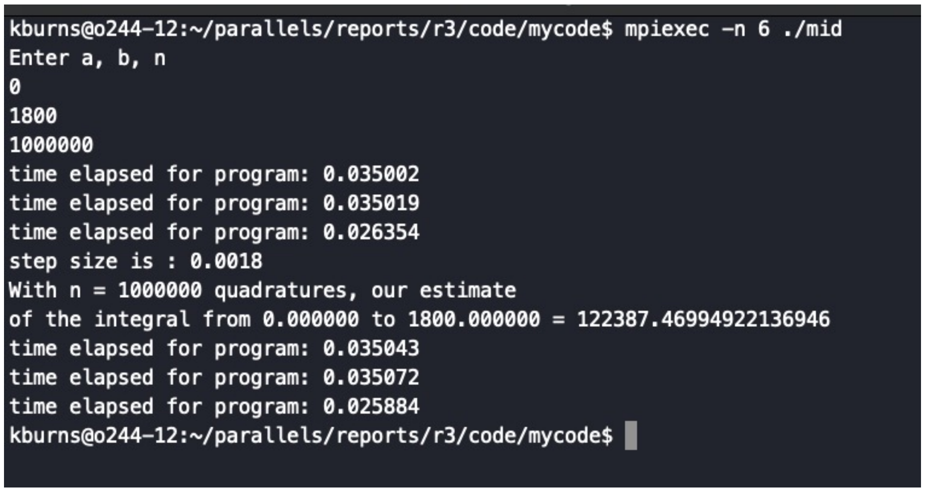
```
~/s/r/r/mycode  main ±  ./a.out
boxes 1.8e+06
delta 0.001
area using left: 0
area using trap/mid: 61000000.0017826
time elapsed for program: 0.040999
~/s/r/r/mycode  main ±
```

Figure 12: single_trap



```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./mid
Enter a, b, n
0
1800
1000000
time elapsed for program: 0.035002
time elapsed for program: 0.035019
time elapsed for program: 0.026354
step size is : 0.0018
With n = 1000000 quadratures, our estimate
of the integral from 0.000000 to 1800.000000 = 122387.46994922136946
time elapsed for program: 0.035043
time elapsed for program: 0.035072
time elapsed for program: 0.025884
kburns@o244-12:~/parallels/reports/r3/code/mycode$
```

Figure 13: mpi__trap

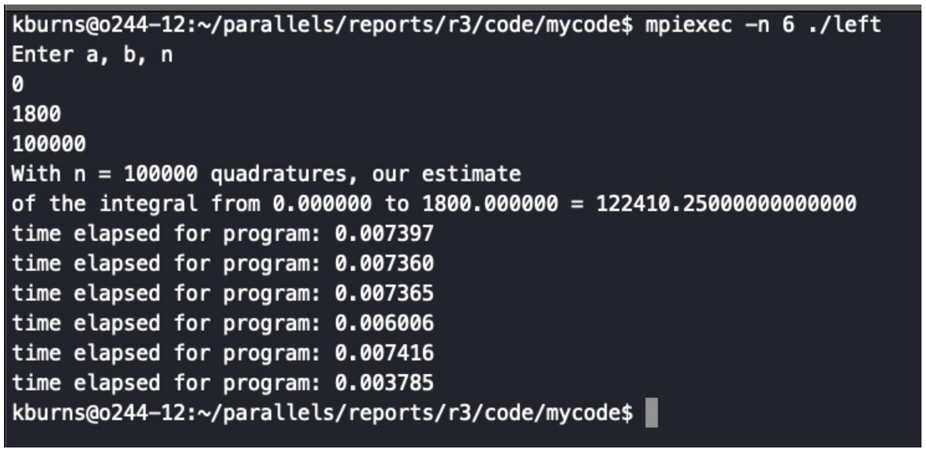


```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./mid
Enter a, b, n
0
1800
1000000
time elapsed for program: 0.035002
time elapsed for program: 0.035019
time elapsed for program: 0.026354
step size is : 0.0018
With n = 1000000 quadratures, our estimate
of the integral from 0.000000 to 1800.000000 = 122387.46994922136946
time elapsed for program: 0.035043
time elapsed for program: 0.035072
time elapsed for program: 0.025884
kburns@o244-12:~/parallels/reports/r3/code/mycode$
```

Figure 14: trap__pos__double

Accuracy and Precision

$$\boxed{\frac{(122387.46994922136946 - 122000)}{122000} \times 100 \approx 0.3192 \; \textit{percent error}}$$



```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./left
Enter a, b, n
0
1800
100000
With n = 100000 quadratures, our estimate
of the integral from 0.000000 to 1800.000000 = 122410.25000000000000
time elapsed for program: 0.007397
time elapsed for program: 0.007360
time elapsed for program: 0.007365
time elapsed for program: 0.006006
time elapsed for program: 0.007416
time elapsed for program: 0.003785
kburns@o244-12:~/parallels/reports/r3/code/mycode$ 
```

Figure 15: trap__pos_float

$$\boxed{\frac{(122410.250000000 - 122000)}{122000} \times 100 \approx 0.3368 \; \textit{percent error}}$$

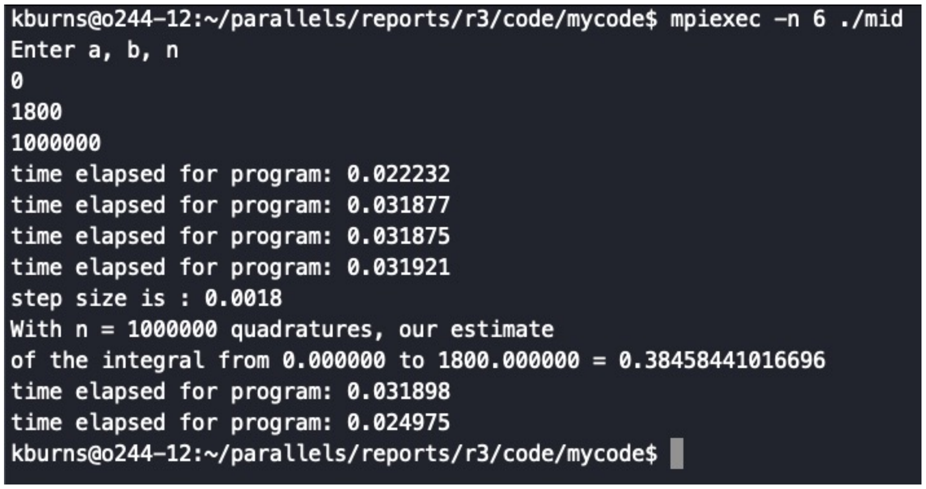
- from the two trap results for position at t=1800, we can see that neither are very accurate(a common theme throughout this report). This is due to my implementation. Both are below 1 percent however neither are nearly accurate enough to be deemed practically correct.
- interestingly we see the lack of precision on the float typecasted implementation maxing out at 0.25

- these results are not accurate enough to be reliable.

Analysis

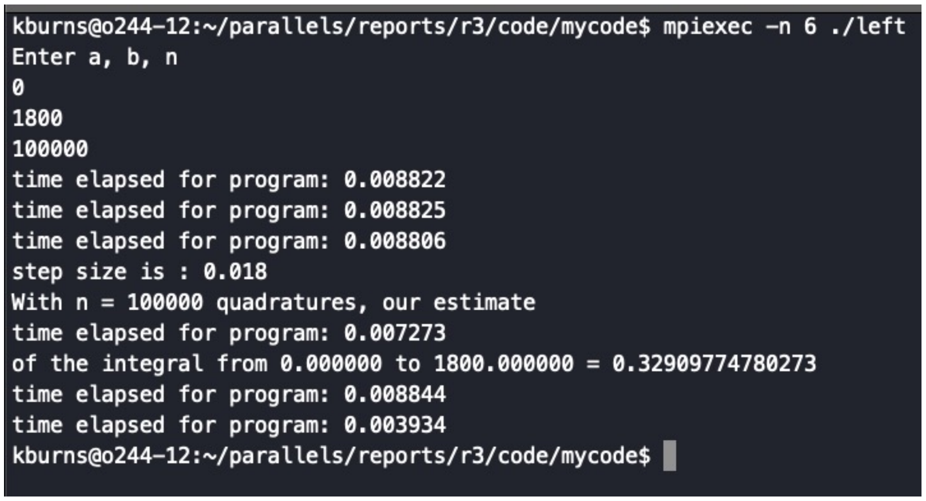
sources of error:

- I believe the source of error for these are mainly due to implementation and i will go over what i would do to improve these results and my thoughts on how my implementation is flawed
- typecasting for float didn't hold and had the approximation been accurate there definitely would have been precision errors.



```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./mid
Enter a, b, n
0
1800
1000000
time elapsed for program: 0.022232
time elapsed for program: 0.031877
time elapsed for program: 0.031875
time elapsed for program: 0.031921
step size is : 0.0018
With n = 1000000 quadratures, our estimate
of the integral from 0.000000 to 1800.000000 = 0.38458441016696
time elapsed for program: 0.031898
time elapsed for program: 0.024975
kburns@o244-12:~/parallels/reports/r3/code/mycode$
```

Figure 16: trap_vel_double



```
kburns@o244-12:~/parallels/reports/r3/code/mycode$ mpiexec -n 6 ./left
Enter a, b, n
0
1800
100000
time elapsed for program: 0.008822
time elapsed for program: 0.008825
time elapsed for program: 0.008806
step size is : 0.018
With n = 100000 quadratures, our estimate
time elapsed for program: 0.007273
of the integral from 0.000000 to 1800.000000 = 0.32909774780273
time elapsed for program: 0.008844
time elapsed for program: 0.003934
kburns@o244-12:~/parallels/reports/r3/code/mycode$
```

Figure 17: trap_vel_float

Summary

- faulty code
  - I do believe that these could have been better had i managed my time more effectively
  - one possible downfall in my code may have been that i was double counting at the edges of each section and so i may have been double counting values and adding them to the final result.
    - i would have to look more into it but i believe this is the reason for my positions having such a large margin of error
  - another possible error is that I was using the same implementationss and switching which antiderivative being used for pos i used velocity(x) and for velocity i used acceleration
    - there may have been a human error where i mightve had a combination of the two insted of one or the other because i forgot to comment out and uncomment the respective function calls.
  - finally it may have been because i was unknowingly using cmath library instead of math.h, however, im unsure how much this would have changed the results from the program.