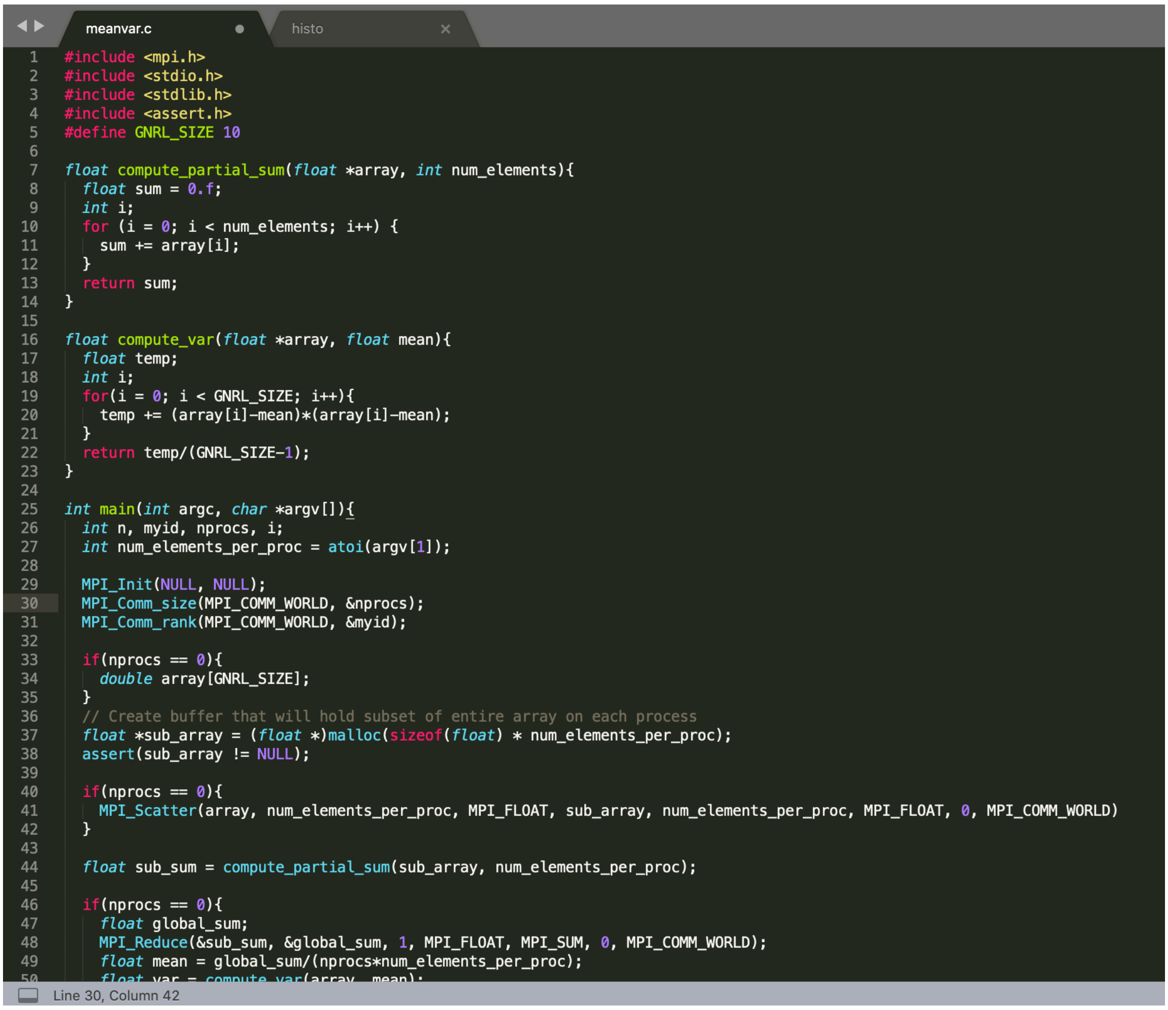
Matt Taylor
ECE 6105
Assignment 5

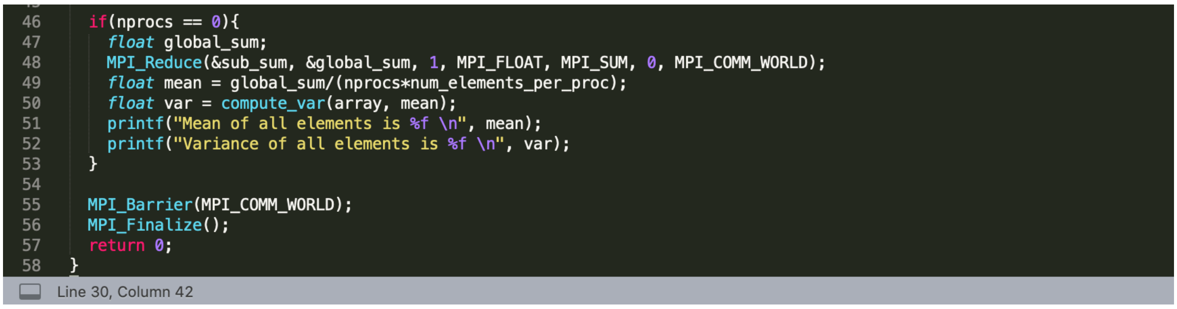
Question 1

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#define GNRL_SIZE 10

float compute_partial_sum(float *array, int num_elements){
        float sum = 0.f;
        int i;
        for (i = 0; i < num_elements; i++) {
        sum += array[i];
        }
        return sum;
}

float compute_var(float *array, float mean){
        float temp;
        int i;
        for(i = 0; i < GNRL_SIZE; i++){
                temp += (array[i]-mean)*(array[i]-mean);
        }
        return temp/(GNRL_SIZE-1);
}

int main(int argc, char *argv[]){
        int n, myid, nprocs, i;
        int num_elements_per_proc = atoi(argv[1]);

        MPI_Init(NULL, NULL);
        MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
        MPI_Comm_rank(MPI_COMM_WORLD, &myid);

        if(nprocs == 0){
                double array[GNRL_SIZE];
        }
        // Create buffer that will hold subset of entire array on each process
        float *sub_array = (float *)malloc(sizeof(float) * num_elements_per_proc);
        assert(sub_array != NULL);

  if(nprocs == 0){
    MPI_Scatter(array, num_elements_per_proc, MPI_FLOAT, sub_array,
num_elements_per_proc, MPI_FLOAT, 0, MPI_COMM_WORLD)
```

```
        }

        float sub_sum = compute_partial_sum(sub_array, num_elements_per_proc);

  if(nprocs == 0){
   float global_sum;
   MPI_Reduce(&sub_sum, &global_sum, 1, MPI_FLOAT, MPI_SUM, 0,
MPI_COMM_WORLD);
   float mean = global_sum/(nprocs*num_elements_per_proc);
   float var = compute_var(array, mean);
   printf("Mean of all elements is %f \n", mean);
   printf("Variance of all elements is %f \n", var);
  }

        MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();
        return 0;
}
```



```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#define GNRL_SIZE 10

float compute_partial_sum(float *array, int num_elements){
  float sum = 0.f;
  int i;
  for (i = 0; i < num_elements; i++) {
    sum += array[i];
  }
  return sum;
}

float compute_var(float *array, float mean){
  float temp;
  int i;
  for(i = 0; i < GNRL_SIZE; i++){
    temp += (array[i]-mean)*(array[i]-mean);
  }
  return temp/(GNRL_SIZE-1);
}

int main(int argc, char *argv[]){
  int n, myid, nprocs, i;
  int num_elements_per_proc = atoi(argv[1]);

  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);

  if(nprocs == 0){
    double array[GNRL_SIZE];
  }
  // Create buffer that will hold subset of entire array on each process
  float *sub_array = (float *)malloc(sizeof(float) * num_elements_per_proc);
  assert(sub_array != NULL);

  if(nprocs == 0){
    MPI_Scatter(array, num_elements_per_proc, MPI_FLOAT, sub_array, num_elements_per_proc, MPI_FLOAT, 0, MPI_COMM_WORLD)
  }

  float sub_sum = compute_partial_sum(sub_array, num_elements_per_proc);

  if(nprocs == 0){
    float global_sum;
    MPI_Reduce(&sub_sum, &global_sum, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);
    float mean = global_sum/(nprocs*num_elements_per_proc);
    float var = compute_var(array, mean);
```

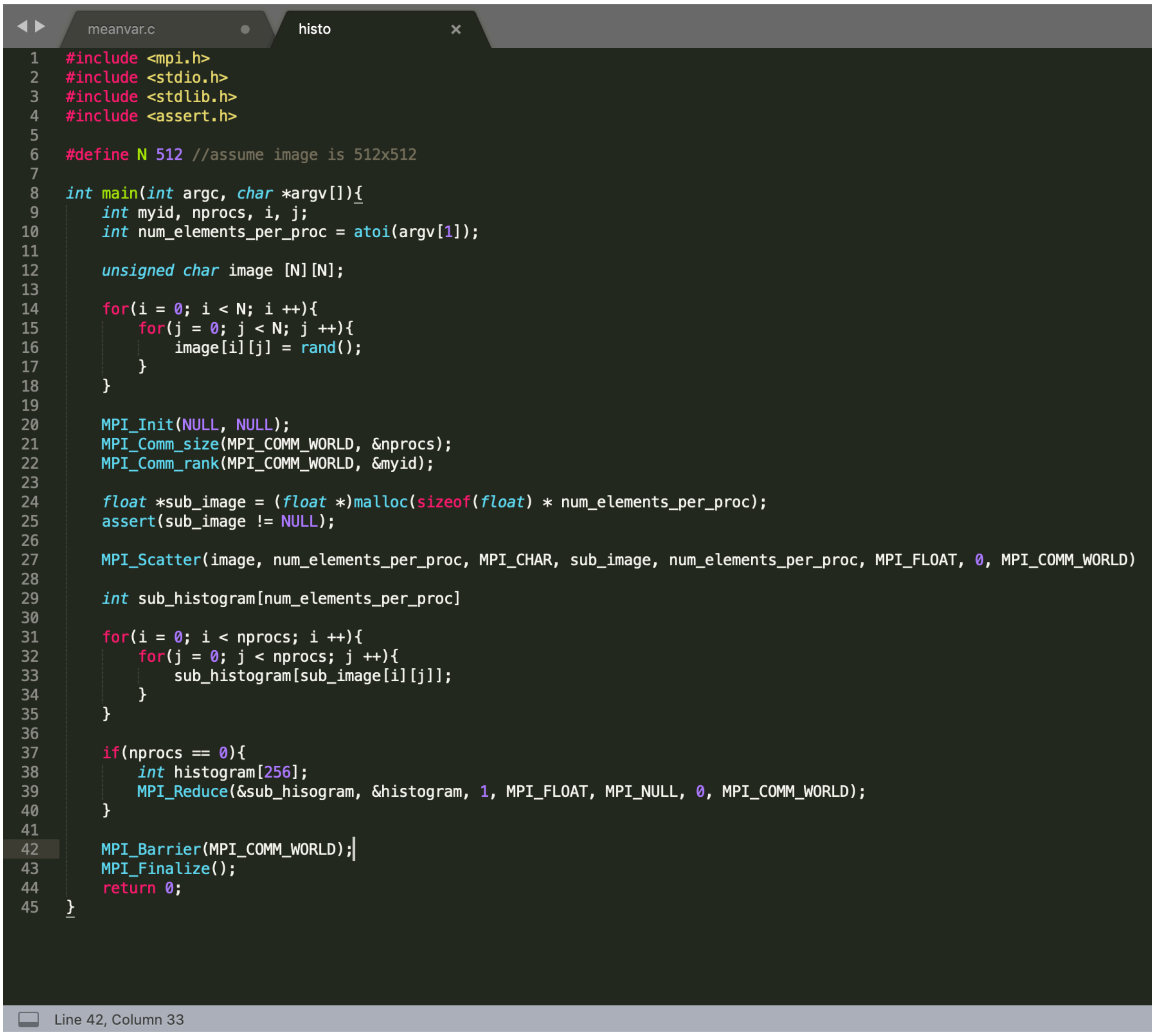


```
  if(nprocs == 0){
    float global_sum;
    MPI_Reduce(&sub_sum, &global_sum, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);
    float mean = global_sum/(nprocs*num_elements_per_proc);
    float var = compute_var(array, mean);
    printf("Mean of all elements is %f \n", mean);
    printf("Variance of all elements is %f \n", var);
  }

  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();
  return 0;
}
```

Question 2

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#define N 512 //assume image is 512x512

int main(int argc, char *argv[]){
    int myid, nprocs, i, j;
    int num_elements_per_proc = atoi(argv[1]);

    unsigned char image [N][N];

    for(i = 0; i < N; i ++){
        for(j = 0; j < N; j ++){
            image[i][j] = rand();
        }
    }

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    float *sub_image = (float *)malloc(sizeof(float) * num_elements_per_proc);
    assert(sub_image != NULL);

    MPI_Scatter(image, num_elements_per_proc, MPI_CHAR, sub_image,
num_elements_per_proc, MPI_FLOAT, 0, MPI_COMM_WORLD)

    int sub_histogram[num_elements_per_proc]

    for(i = 0; i < nprocs; i ++){
        for(j = 0; j < nprocs; j ++){
            sub_histogram[sub_image[i][j]];
        }
    }
```

```
	if(nprocs == 0){
		int histogram[256];
		MPI_Reduce(&sub_hisogram, &histogram, 1, MPI_FLOAT, MPI_NULL, 0, 
MPI_COMM_WORLD);
	}

	MPI_Barrier(MPI_COMM_WORLD);
	MPI_Finalize();
	return 0;
}
```



```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#define N 512 //assume image is 512x512

int main(int argc, char *argv[]){
    int myid, nprocs, i, j;
    int num_elements_per_proc = atoi(argv[1]);

    unsigned char image [N][N];

    for(i = 0; i < N; i ++){
        for(j = 0; j < N; j ++){
            image[i][j] = rand();
        }
    }

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    float *sub_image = (float *)malloc(sizeof(float) * num_elements_per_proc);
    assert(sub_image != NULL);

    MPI_Scatter(image, num_elements_per_proc, MPI_CHAR, sub_image, num_elements_per_proc, MPI_FLOAT, 0, MPI_COMM_WORLD)

    int sub_histogram[num_elements_per_proc]

    for(i = 0; i < nprocs; i ++){
        for(j = 0; j < nprocs; j ++){
            sub_histogram[sub_image[i][j]];
        }
    }

    if(nprocs == 0){
        int histogram[256];
        MPI_Reduce(&sub_hisogram, &histogram, 1, MPI_FLOAT, MPI_NULL, 0, MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
}
```

Question 3

a. **Source lines of code**
   Mean/Variance Code:
      UPC = 28; MPI = 58
   Histogram Code:
      UPC = 31; MPI = 45

b. **Number of keywords used**
   Mean/Variance Code:
      UPC = 6; MPI = 7
   Histogram Code:
      UPC = 11; MPI = 7

c. **Number of parameters passed to function calls**
   Mean/Variance Code:
      UPC = 4 (largest), 1 (smallest); MPI = 8 (largest), 1 (smallest)
   Histogram Code:
      UPC = 4 (largest), 1(smallest); MPI = 8 (largest), 1 (smallest)

d. **Size of binary file**
   Mean/Variance Code:
      UPC = 948 bytes; MPI = 1,481 bytes
   Histogram Code:
      UPC = 567 bytes; MPI = 1,003 bytes

e. **PGAS vs message passing**
   In my experience, programming using the PGAS model is much easier than using the message passing model. MPI required more lines of code and more parameters in the function calls. UPC allows for access to different parts of a shared memory without having to explicitly find the processor that owned that piece. Both models require vastly different mindsets while coding. UPC, for example, gathers data by simply taking it from the right location. MPI on the other hand requires synchronization of the nodes involved. UPC is easier to debug and read, but more difficult to optimize.