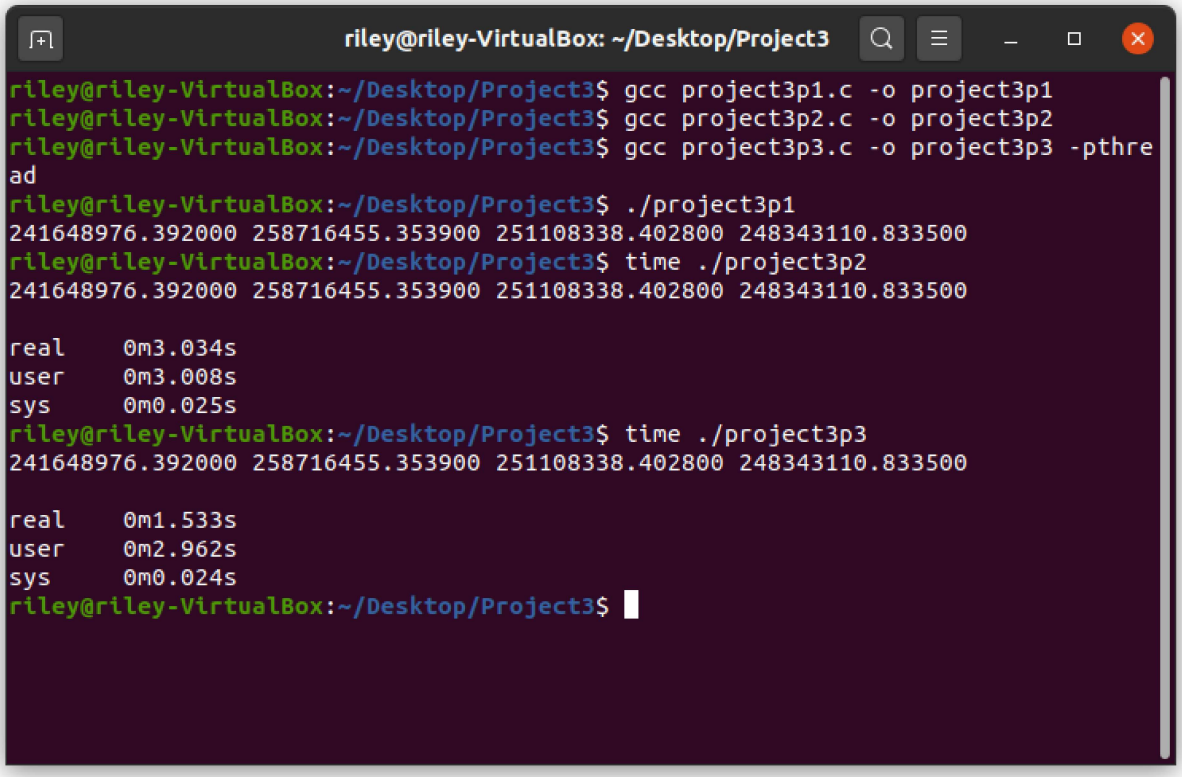
Riley Ruckman
TCES420, Au20
Project 3

Runtime of all parts (except Part1; expecting runtime to be similar to Part 2):



Part 4: The difference between parts 2 and 3 is the use of two parallel threads to perform two matmul() functions at the same time in part 3 instead of running them concurrently in part 2. Since part 3 has two parallel threads to do the calculations in matmul(), it is able to complete all matmul() calls in half the time of part 2, since it's using twice the number of threads as part 2. This can be seen in the "real" times for parts 2 and 3, where part 2 has 3.034s and part 3 has 1.533s.

Part 1 Code:

```
// Riley Ruckman
// TCES420, Au20
// Project 3

#include <stdio.h>
#include <stdlib.h>

double first[1000][1000], second[1000][1000], mult[1000][1000];
int m=1000, n=1000, p=1000; /* matric dimensions mxn and nxp */

void* matmul(void *);

int main(void) {

	FILE *f;

	// Read contents of matrix1.data into first
	f = fopen("matrix1.data", "r");
	fread(first, sizeof(double), sizeof(first)/sizeof(double), f);
	fclose(f);

	// Read contents of matrix2.data into second
	f = fopen("matrix2.data", "r");
	fread(second, sizeof(double), sizeof(second)/sizeof(double), f);
	fclose(f);

	matmul(NULL);

	// Read contents of matrix2.data into second
	f = fopen("matrix3.data", "w");
	fwrite(mult, sizeof(double), sizeof(mult)/sizeof(double), f);
	fclose(f);

	printf("%lf %lf %lf %lf\n", mult[6][0], mult[5][3], mult[5][4], mult[901][7]);

	return 0;
}

// Multiplies two global 1000-by-1000 matrices into 1000-by-1000 matrix
void* matmul (void *arg) {

	int i, j, k;
```

```
        int s = 0, e = m;
        double sum = 0;
        for (i = s; i < e; i++) {
                for (j = 0; j < p; j++) {
                        for (k = 0; k < n; k++) {
                                sum = sum + first[i][k]*second[k][j];
                        }
                        mult[i][j] = sum;
                        sum = 0;
                }
        }
}
```

Part 2 Code:

```
// Riley Ruckman
// TCES420, Au20
// Project 3

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

double first[1000][1000], second[1000][1000], mult[1000][1000];
int m=1000, n=1000, p=1000; /* matric dimensions mxn and nxp */

void* matmul(void *);

int main(void) {

	FILE *f;

	// Read contents of matrix1.data into first
	f = fopen("matrix1.data", "r");
	fread(first, sizeof(double), sizeof(first)/sizeof(double), f);
	fclose(f);

	// Read contents of matrix2.data into second
	f = fopen("matrix2.data", "r");
	fread(second, sizeof(double), sizeof(second)/sizeof(double), f);
	fclose(f);

	/*
	// Creates two threads for two calls of matmul(). Each thread is executed consecutively
to
	// simulate single-threaded performance.
	pthread_t th1, th2;
	// mult = firstLower * second + firstHigher * second
	int sec = 1;
	pthread_create(&th1, NULL, matmul, (void*)&sec);
	pthread_join(th1, NULL);

	sec = 2;
	pthread_create(&th2, NULL, matmul, (void*)&sec);
	pthread_join(th2, NULL);
	*/

	int sec = 1;
```

```
	matmul((void*)&sec);
	sec = 2;
	matmul((void*)&sec);


	// Read contents of matrix2.data into second
	f = fopen("matrix3.data", "w");
	fwrite(mult, sizeof(double), sizeof(mult)/sizeof(double), f);
	fclose(f);

	printf("%lf %lf %lf %lf\n", mult[6][0], mult[5][3], mult[5][4], mult[901][7]);

	return 0;
}

// Multiplies two global 1000-by-1000 matrices into 1000-by-1000 matrix
void* matmul (void *arg) {

	int i, j, k;
	int s, e;

	if ((int *)arg == NULL) {

		s = 0;
		e = m;
	} else if (*(int *)arg == 1) {

		s = 0;
		e = m/2;
	} else if(*(int *)arg == 2) {

		s = m/2;
		e = m;
	}
	double sum = 0;
	for (i = s; i < e; i++) {
		for (j = 0; j < p; j++) {
			for (k = 0; k < n; k++) {
				sum = sum + first[i][k]*second[k][j];
			}
			mult[i][j] = sum;
			sum = 0;
		}
	}
```

```
}
```

Part 3 Code:

```
// Riley Ruckman
// TCES420, Au20
// Project 3

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

double first[1000][1000], second[1000][1000], mult[1000][1000];
int m=1000, n=1000, p=1000; /* matric dimensions mxn and nxp */

void* matmul(void *);

int main(void) {

	FILE *f;

	// Read contents of matrix1.data into first
	f = fopen("matrix1.data", "r");
	fread(first, sizeof(double), sizeof(first)/sizeof(double), f);
	fclose(f);

	// Read contents of matrix2.data into second
	f = fopen("matrix2.data", "r");
	fread(second, sizeof(double), sizeof(second)/sizeof(double), f);
	fclose(f);

	// Creates two threads for two calls of matmul(). Each thread is executed in parallel for
multi-threaded
	// performance.
	pthread_t th1, th2;

	// mult = firstLower * second + firstHigher * second
	int sec1 = 1;
	pthread_create(&th1, NULL, matmul, (void*)&sec1);

	int sec2 = 2;
	pthread_create(&th2, NULL, matmul, (void*)&sec2);

	// Waits for each thread to finish
	pthread_join(th1, NULL);
	pthread_join(th2, NULL);
```

```
	// Read contents of matrix2.data into second
	f = fopen("matrix3.data", "w");
	fwrite(mult, sizeof(double), sizeof(mult)/sizeof(double), f);
	fclose(f);

	printf("%lf %lf %lf %lf\n", mult[6][0], mult[5][3], mult[5][4], mult[901][7]);

	return 0;
}

// Multiplies two global 1000-by-1000 matrices into 1000-by-1000 matrix
void* matmul (void *arg) {

	int i, j, k;
	int s, e;

	if ((int*)arg == NULL) {

		s = 0;
		e = m;
	} else if (*(int*)arg == 1) {

		s = 0;
		e = m/2;
	} else if(*(int*)arg == 2) {

		s = m/2;
		e = m;
	}
	double sum = 0;
	for (i = s; i < e; i++) {
		for (j = 0; j < p; j++) {
			for (k = 0; k < n; k++) {
				sum = sum + first[i][k]*second[k][j];
			}
			mult[i][j] = sum;
			sum = 0;
		}
	}
	pthread_exit(NULL);
}
```