OpenMP

Hello OpenMP

Ensure that you have the OpenMP libraries installed on your system. You can check this by constructing the following OpenMP HelloWorld program.

```c
#include <stdio.h>
#include <omp.h>

int main(void) {
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

You should observe your application outputting "Hello, world." to the terminal based on the number of cores your device has.

Parallel Sum

By now, you should be completely capable to write a parallel sum program using pthreads.

In this exercise, we want you to use the code below and the correct OpenMP preprocessor directives to transform this single threaded program into a threaded version.

```c
int main() {
    //data here
    int64_t sum = 0;
    for(size_t i = 0; i < end; i++) {
        sum += data[i];
    }
    printf("%ld\n", sum);
    return 0;
}
```

- Try to observe what OpenMP is doing by running the preprocessor only, what do you observe?
- What are the advantages and disadvantages to this approach? When would we prefer this approach over pthreads

Parallel Intersection

Often in computer simulations and video games we want to check if two objects collide into each other. We are able to simply achieve this by constructing axis-aligned bounding boxes and comparing their intersections with each other.

```
intersect(box1, box2):
 return (box1.x < (box2.x + box2.width)) &&
 ((box1.x + box1.width) > box2.x &&
 ((box1.y < (box2.y + box2.height) &&
 ((box1.y + box1.height) > box2.y)
```

Construct a simple application that will construct a large number of objects that will move around in a large box and check if they collide.

Translating Matrix Multiplication

Time to revisit matrix multiplication again! This time, you will need to modify the following code to utilise OpenMP. Consider how you could refactor this code to make it work with OpenMP.

```
void multiply(const float* mata, size_t mata_width, size_t mata_height,
    const float* matb, size_t matb_width, size_t matb_height,
    float** result_mat, size_t* res_width, size_t* res_height) {

    if(result_mat) {
        *res_width = matb_width;
        *res_height = mata_height);
        *result_mat = calloc(mata_height * matb_width
        * sizeof(float));
        for(size_t y = 0; y < mata_height; y++) {
            for(size_t x = 0; x < matb_width; x++) {
                for(size_t k = 0; k < mata_width; k++) {
                    (*result_mat)[(y * matb_width) + x] +=
                        (mata[(y * mata_width) + k] *
                        matb[(k * matb_width) + x]);
                }
            }
        }
    }
}
```

- Refactor the code to use multiple threads with OpenMP
- Construct a set of test cases of different sizes (try 32x32, 64x64, 128x128 ... 2048x2048 and larger)