OpenMP 3

1 Rotation

The below code takes a vector and rotates it: It shifts all elements to the left, and appends the first element.

```
void rot(int N, double* a) {
  for (int i = 0; i < N - 1; i++) {
    double tmp = a[i];
    a[i] = a[i+1];
    a[i+1] = tmp;
  }
}
```

- Parallelize the code using OpenMP.
- If possible, do not replicate the vector.
- What kind of dependencies are present in the code?

2 Questions

- Code that is marked using `pragma omp critical` or `pragma omp single` is only executed by one thread at a time. How do they differ?
- Some loop-carried dependencies can, in principle, always be resolved. Which ones can not? Can you think of an example?
- Both `pragma omp single nowait` and `pragma omp task` let a piece of code execute asynchronously with respect to other threads. How do they differ?

3 Find the bugs

```
int foo(int N, int *a, int *b, int *c, int *d) {
  double tmp, total = 0;
  #pragma omp parallel
  {
    #pragma omp sections
```

```
    {
      #pragma omp section
      {
        #pragma omp for
        for (int i = 0; i < N; i++) {
          a[b[i]] += b[i];
          total += b[i];
        }
      }
      #pragma omp section
      {
        #pragma omp for
        for (int i = 0; i < N; i++) {
          tmp = c[i];
          c[i] = d[i];
          d[i] = tmp;
          total += c[i];
        }
        #pragma omp for
        for (int i = 0; i < N; i++) {
          total += d[i];
        }
      }
    }
  }
  return total;
}
```

- Try to find the issues with this code, explain why they are issues, and suggest a way to fix them.

4 Tasks

```
void annotate(tree_t* tree) {
  if (tree->left) annotate(tree->left);
  if (tree->right) annotate(tree->right);
  tree->annotation = sqrt(tree->data);
}
double compute(tree_t* tree) {
  double l = 0, r = 0;
  if (tree->left) l = compute(tree->left);
  if (tree->right) r = compute(tree->right);
  return l + r + tree->annotation;
}
double execute(tree_t* tree) {
```

```
  annotate(tree);
  return compute(tree);
}
```

- Parallelize the above calculation using tasks.
- Try to express the dependencies using the different constructs we covered in class.
- Do you achieve a speedup? What needs to change to achieve a speedup?