HIGH PERFORMANCE COMPUTING for SCIENCE & ENGINEERING (HPCSE) I

HS 2021

EXERCISE 05: MONTE CARLO INTEGRATION & OPENMP

Pascal Weber (webepasc@ethz.ch)

Computational Science and Engineering Lab
ETH Zürich

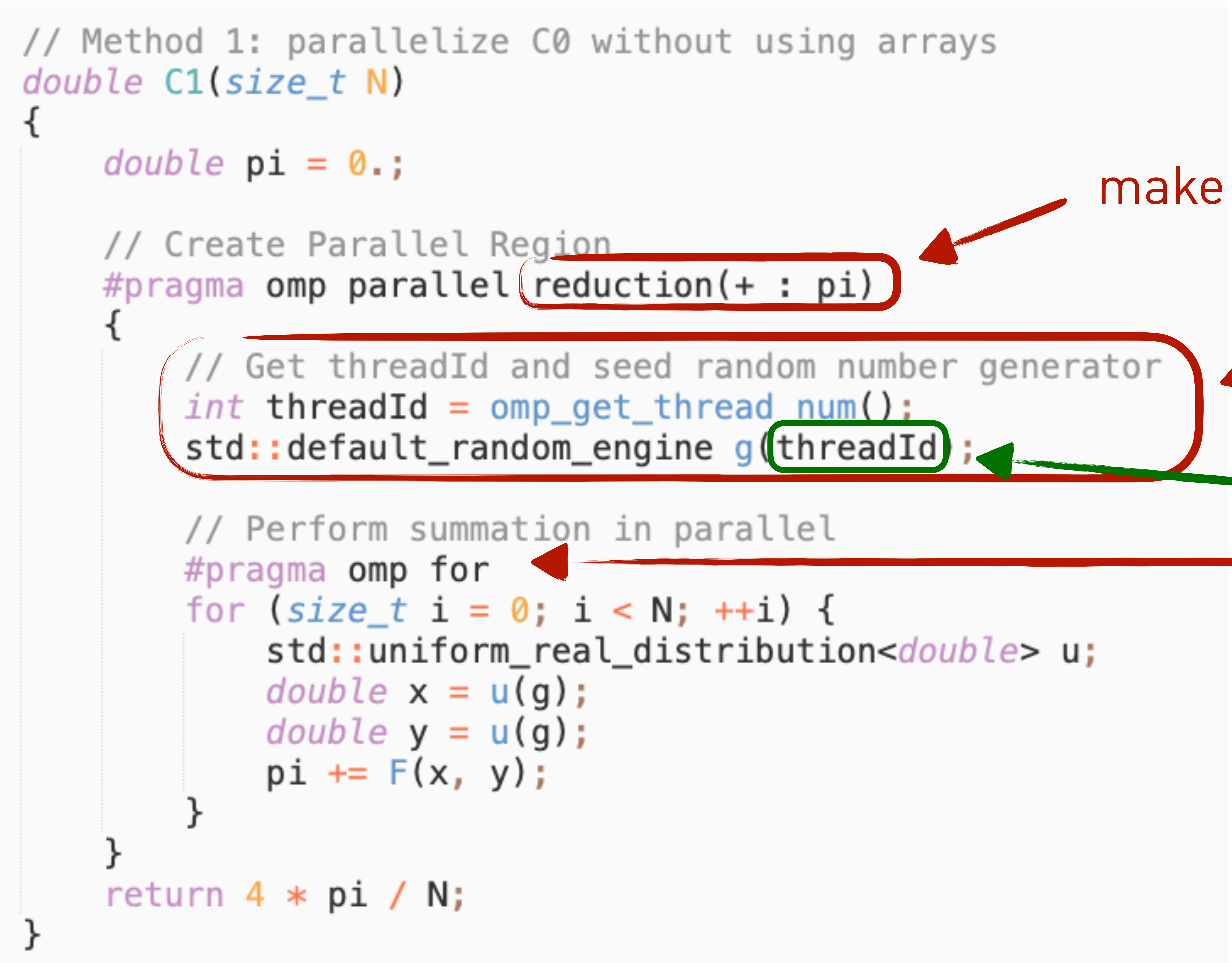
26.11.2021

Outline

I. Exercise 1 (Monte-Carlo Integration)

II. Exercise 2/3 (OpenMP Bughunt)

Exercise 1 - montecarlo.cpp (without using arrays)



```
// Method 1: parallelize C0 without using arrays
double C1(size_t N)
{
    double pi = 0.;

    // Create Parallel Region
    #pragma omp parallel reduction(+ : pi)
    {
        // Get threadId and seed random number generator
        int threadId = omp_get_thread_num();
        std::default_random_engine g(threadId);

        // Perform summation in parallel
        #pragma omp for
        for (size_t i = 0; i < N; ++i) {
            std::uniform_real_distribution<double> u;
            double x = u(g);
            double y = u(g);
            pi += F(x, y);
        }
    }
    return 4 * pi / N;
}
```

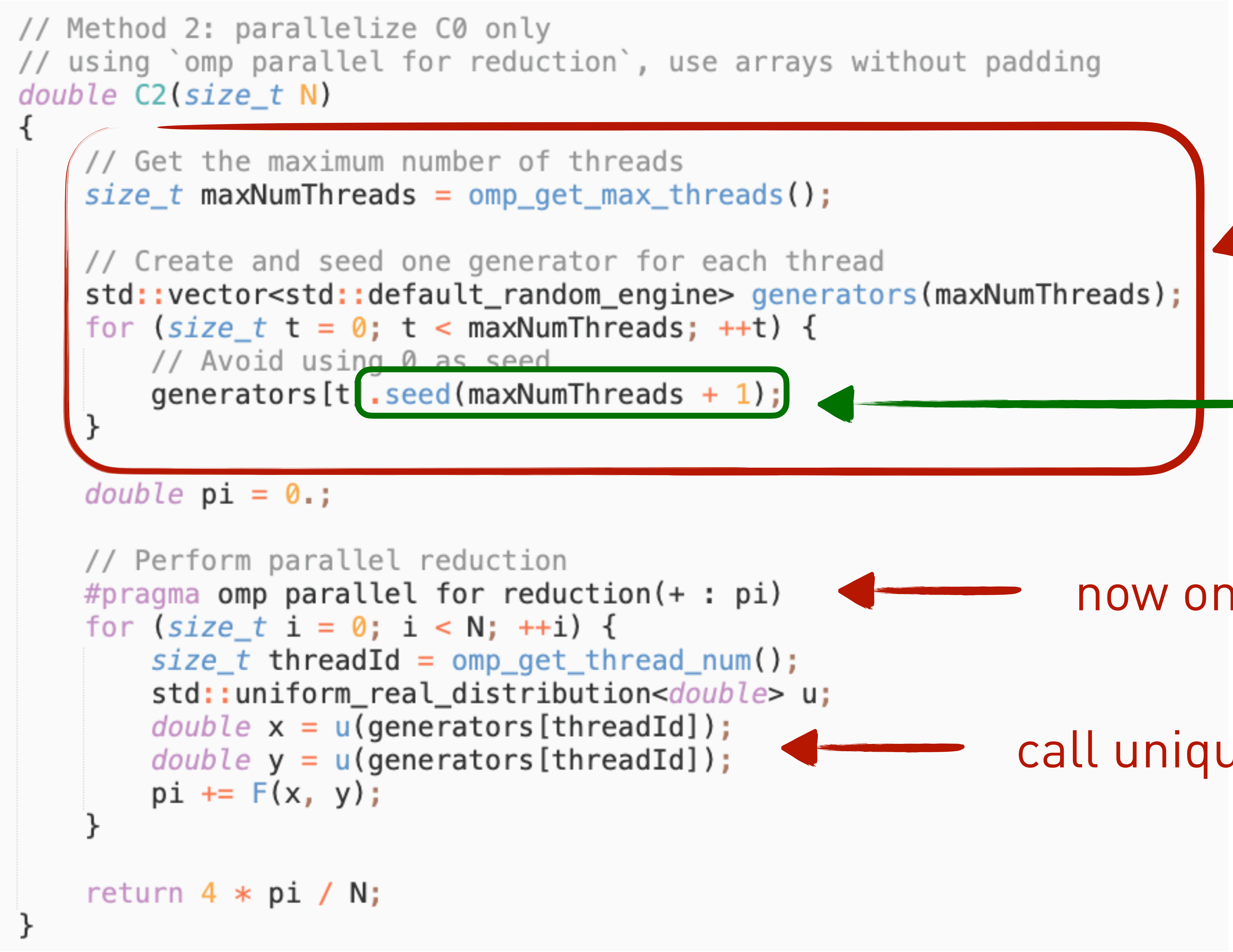
make sure you avoid race conditions in pi

create one generator per thread

seed generator by threadId via constructor

.. at the cost of another pragma

Exercise 1 - montecarlo.cpp (only parallel for reduction)



```cpp
// Method 2: parallelize C0 only
// using `omp parallel for reduction`, use arrays without padding
double C2(size_t N)
{
    // Get the maximum number of threads
    size_t maxNumThreads = omp_get_max_threads();

    // Create and seed one generator for each thread
    std::vector<std::default_random_engine> generators(maxNumThreads);
    for (size_t t = 0; t < maxNumThreads; ++t) {
        // Avoid using 0 as seed
        generators[t].seed(maxNumThreads + 1);
    }
    double pi = 0.;

    // Perform parallel reduction
    #pragma omp parallel for reduction(+ : pi)
    for (size_t i = 0; i < N; ++i) {
        size_t threadId = omp_get_thread_num();
        std::uniform_real_distribution<double> u;
        double x = u(generators[threadId]);
        double y = u(generators[threadId]);
        pi += F(x, y);
    }

    return 4 * pi / N;
}
```

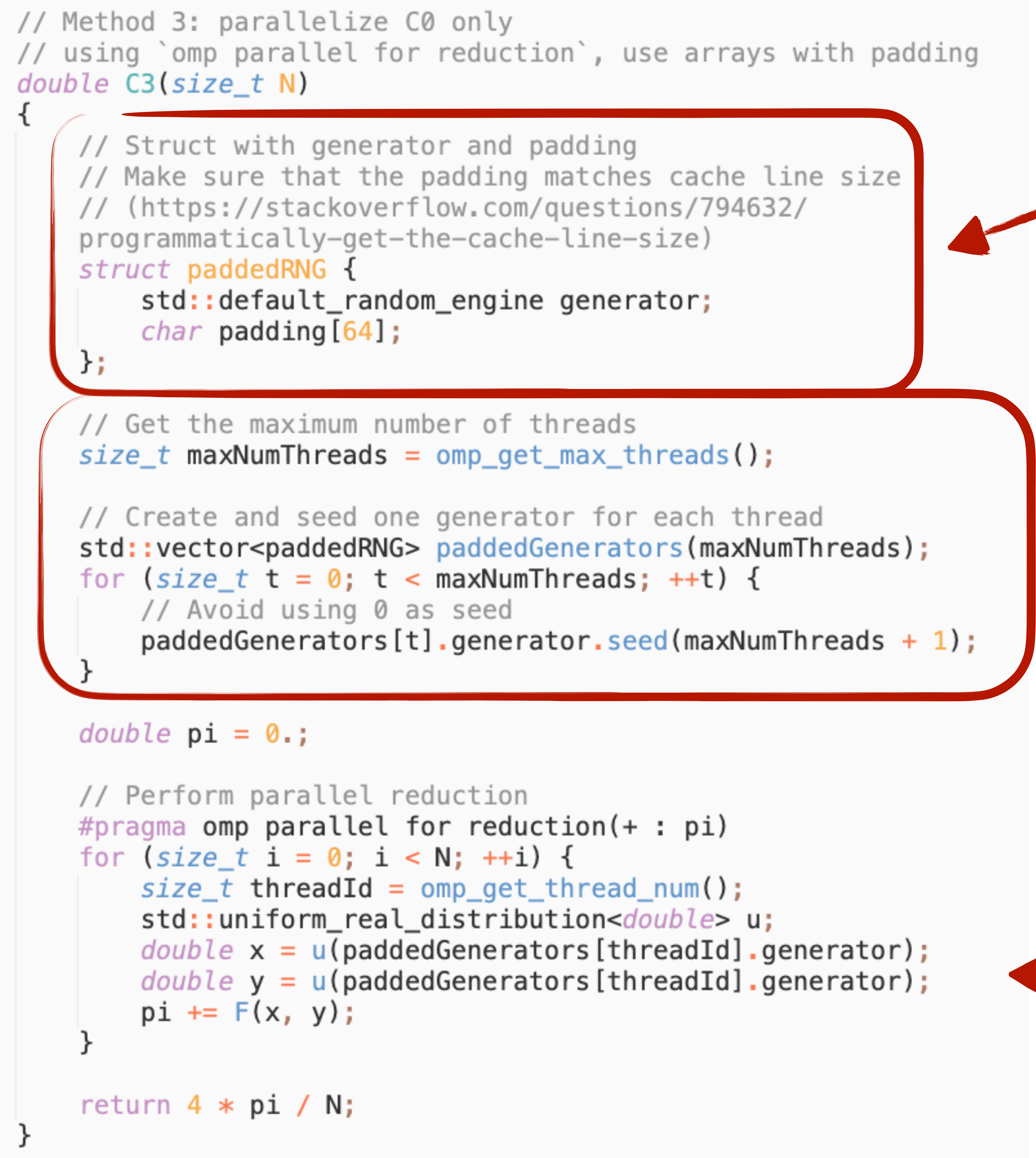
create and seed one generator per thread

seed generator by threadId via seed function (constructor was already called by std::vector)

now only one pragma

call unique generator

Exercise 1 - montecarlo.cpp (parallel for reduction + padding)



```cpp
// Method 3: parallelize C0 only
// using `omp parallel for reduction`, use arrays with padding
double C3(size_t N)
{
    // Struct with generator and padding
    // Make sure that the padding matches cache line size
    // (https://stackoverflow.com/questions/794632/
    programmatically-get-the-cache-line-size)
    struct paddedRNG {
        std::default_random_engine generator;
        char padding[64];
    };

    // Get the maximum number of threads
    size_t maxNumThreads = omp_get_max_threads();

    // Create and seed one generator for each thread
    std::vector<paddedRNG> paddedGenerators(maxNumThreads);
    for (size_t t = 0; t < maxNumThreads; ++t) {
        // Avoid using 0 as seed
        paddedGenerators[t].generator.seed(maxNumThreads + 1);
    }

    double pi = 0.;

    // Perform parallel reduction
    #pragma omp parallel for reduction(+ : pi)
    for (size_t i = 0; i < N; ++i) {
        size_t threadId = omp_get_thread_num();
        std::uniform_real_distribution<double> u;
        double x = u(paddedGenerators[threadId].generator);
        double y = u(paddedGenerators[threadId].generator);
        pi += F(x, y);
    }

    return 4 * pi / N;
}
```

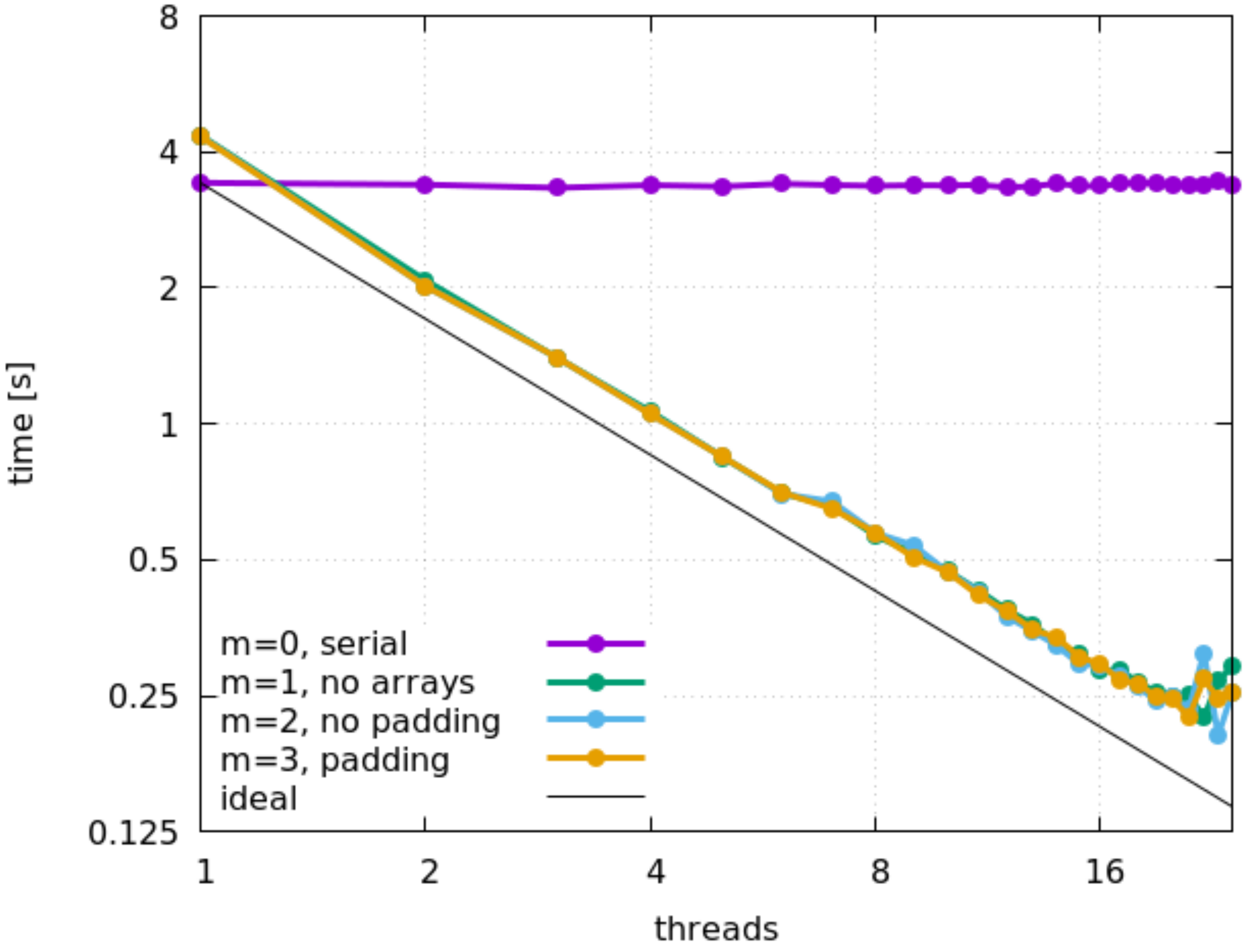
create struct that contains random number generator and padding

create and seed one generator per thread

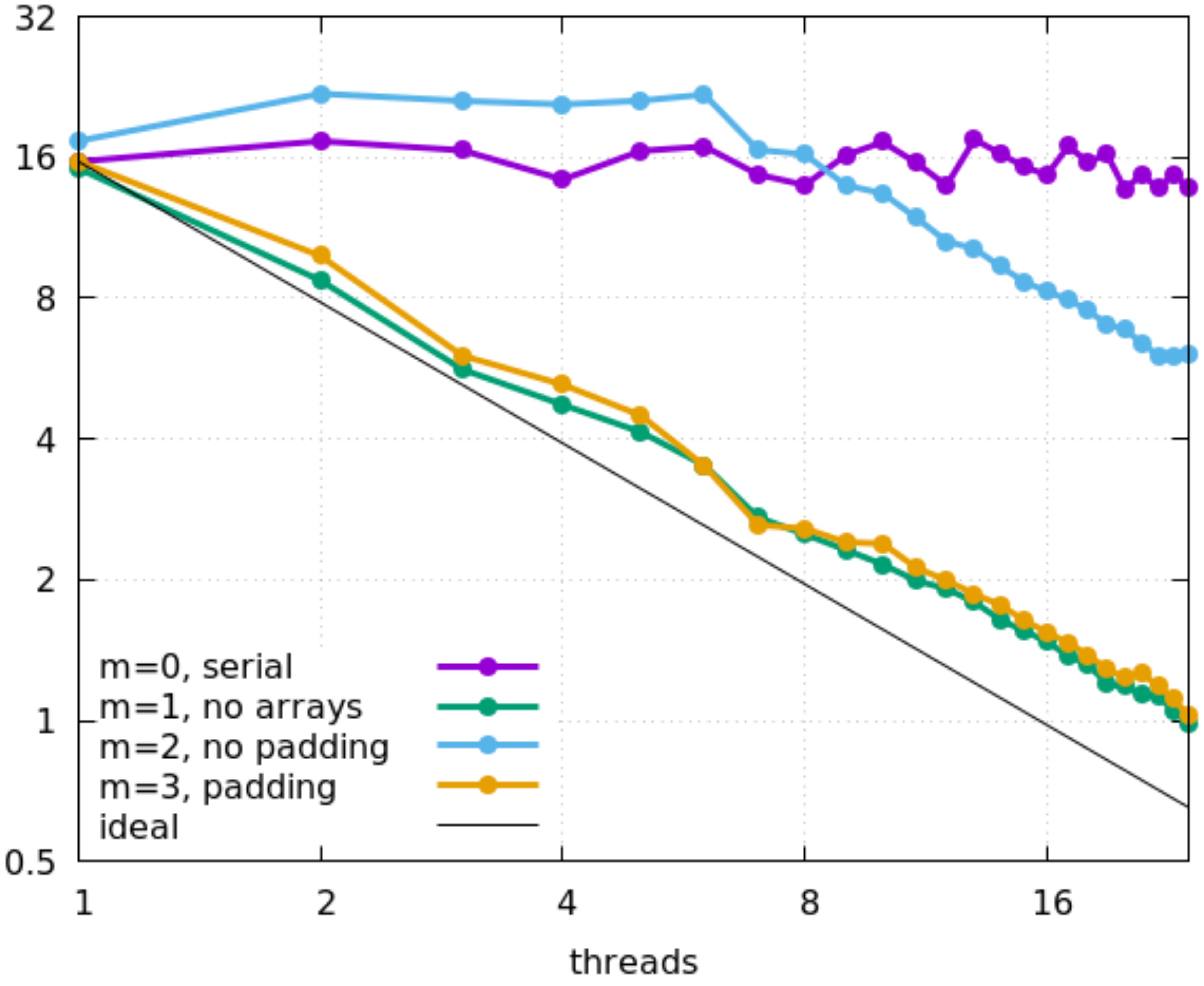
call unique generator

Exercise 1 - Results (On Euler bsub -W 02:00 -n 24 -R fullnode -Is bash)

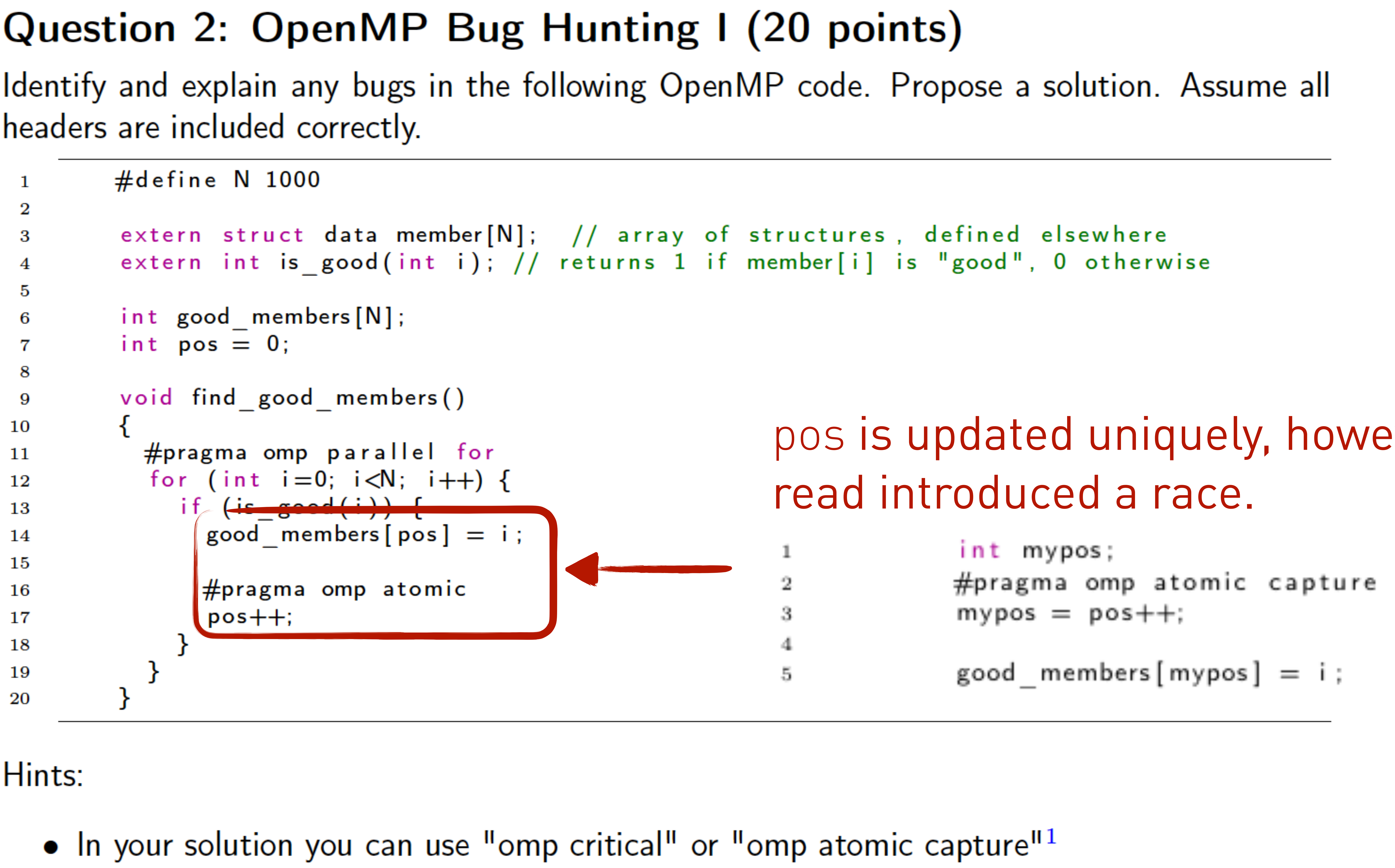
On Euler with -O3



On Euler with -O0



Exercise 2



Question 2: OpenMP Bug Hunting I (20 points)

Identify and explain any bugs in the following OpenMP code. Propose a solution. Assume all headers are included correctly.

```
#define N 1000

extern struct data member[N];  // array of structures, defined elsewhere
extern int is_good(int i); // returns 1 if member[i] is "good", 0 otherwise

int good_members[N];
int pos = 0;

void find_good_members()
{
  #pragma omp parallel for
  for (int i=0; i<N; i++) {
    if (is_good(i)) {
      good_members[pos] = i;

      #pragma omp atomic
      pos++;
    }
  }
}
```

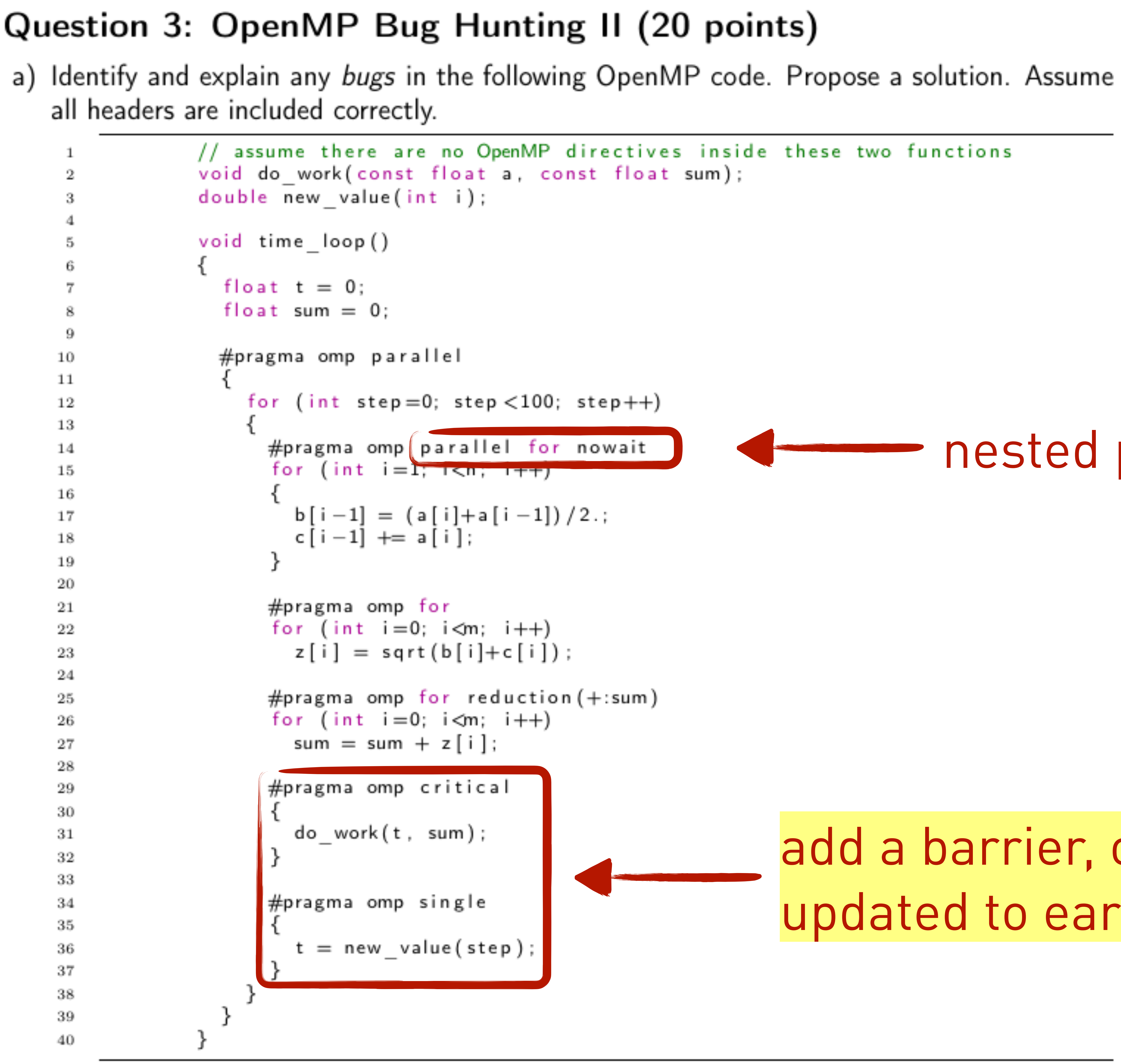
pos is updated uniquely, however the read introduced a race.

```
int mypos;
#pragma omp atomic capture
mypos = pos++;

good_members[mypos] = i;
```

Hints:

- In your solution you can use "omp critical" or "omp atomic capture"[1]

Exercise 3



Question 3: OpenMP Bug Hunting II (20 points)

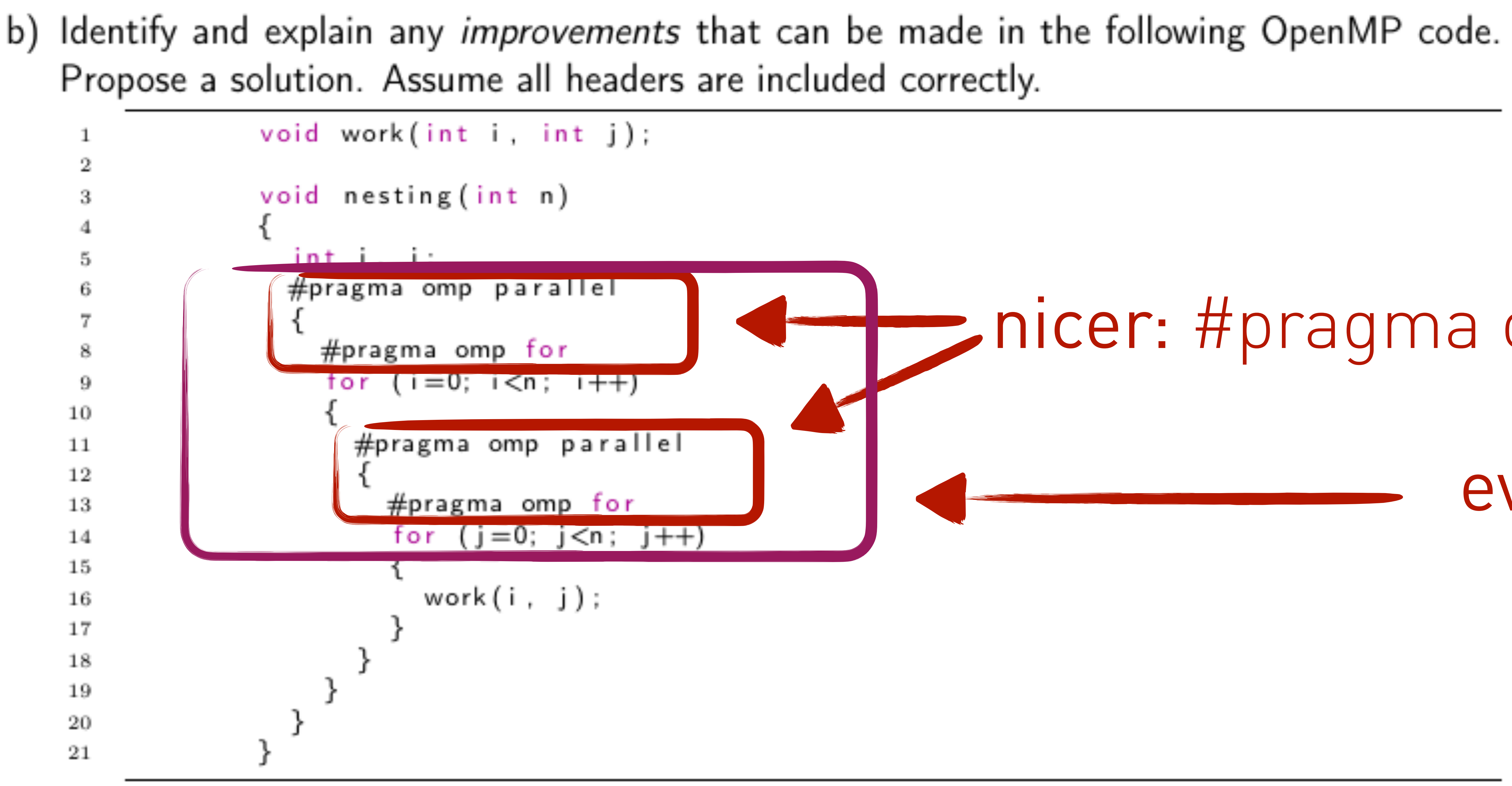
a) Identify and explain any *bugs* in the following OpenMP code. Propose a solution. Assume all headers are included correctly.

```
// assume there are no OpenMP directives inside these two functions
void do_work(const float a, const float sum);
double new_value(int i);

void time_loop()
{
  float t = 0;
  float sum = 0;

  #pragma omp parallel
  {
    for (int step=0; step<100; step++)
    {
      #pragma omp parallel for nowait
      for (int i=1; i<n; i++)
      {
        b[i-1] = (a[i]+a[i-1])/2.;
        c[i-1] += a[i];
      }

      #pragma omp for
      for (int i=0; i<m; i++)
        z[i] = sqrt(b[i]+c[i]);

      #pragma omp for reduction(+:sum)
      for (int i=0; i<m; i++)
        sum = sum + z[i];

      #pragma omp critical
      {
        do_work(t, sum);
      }

      #pragma omp single
      {
        t = new_value(step);
      }
    }
  }
}
```

nested parallelism, remove nowait

add a barrier, otherwise it might be updated to early!

Exercise 3

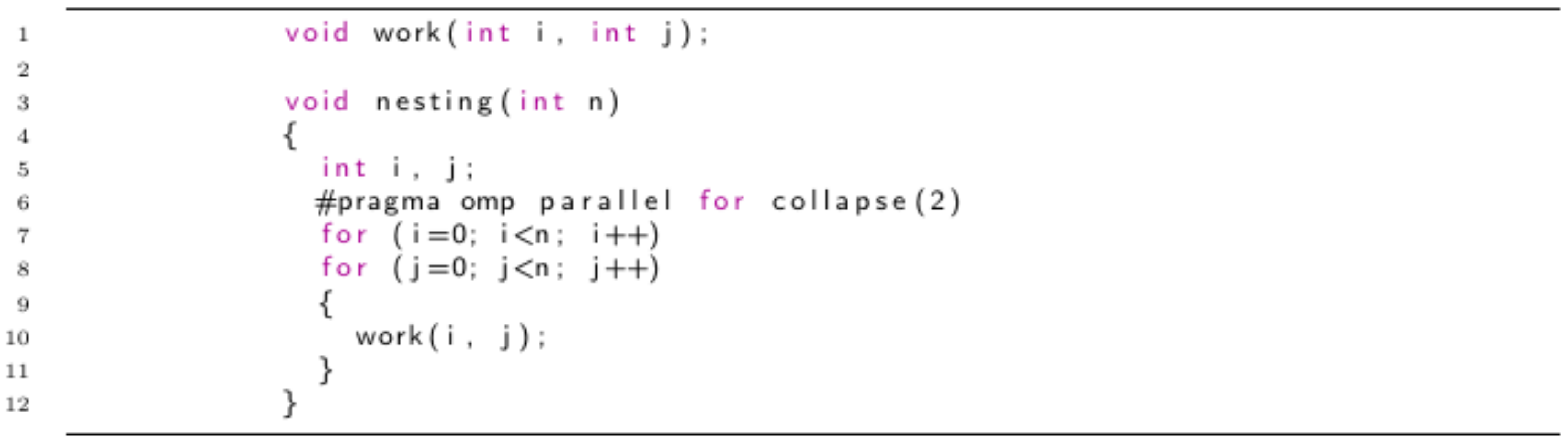


b) Identify and explain any *improvements* that can be made in the following OpenMP code. Propose a solution. Assume all headers are included correctly.

```
void work(int i, int j);

void nesting(int n)
{
  int i, j;
  #pragma omp parallel
  {
    #pragma omp for
    for (i=0; i<n; i++)
    {
      #pragma omp parallel
      {
        #pragma omp for
        for (j=0; j<n; j++)
        {
          work(i, j);
        }
      }
    }
  }
}
```

nicer: #pragma omp parallel for

even nicer, #pragma omp parallel for collapse(2)



```
void work(int i, int j);

void nesting(int n)
{
  int i, j;
  #pragma omp parallel for collapse(2)
  for (i=0; i<n; i++)
  for (j=0; j<n; j++)
  {
    work(i, j);
  }
}
```