Introduction to Parallel & Distributed Programming

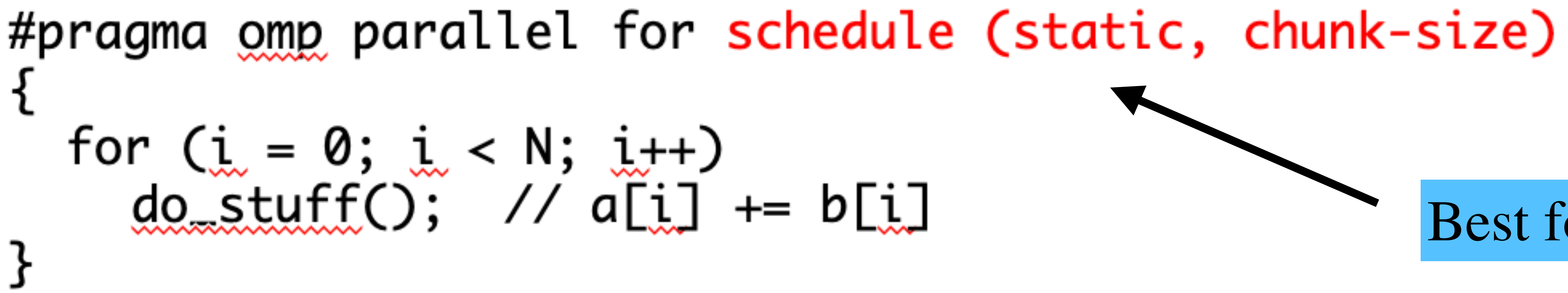
Lec 07 — OpenMP

Subodh Sharma | Jan 20, 2026

Work Sharing Constructs: FOR

Scheduling for iteration space

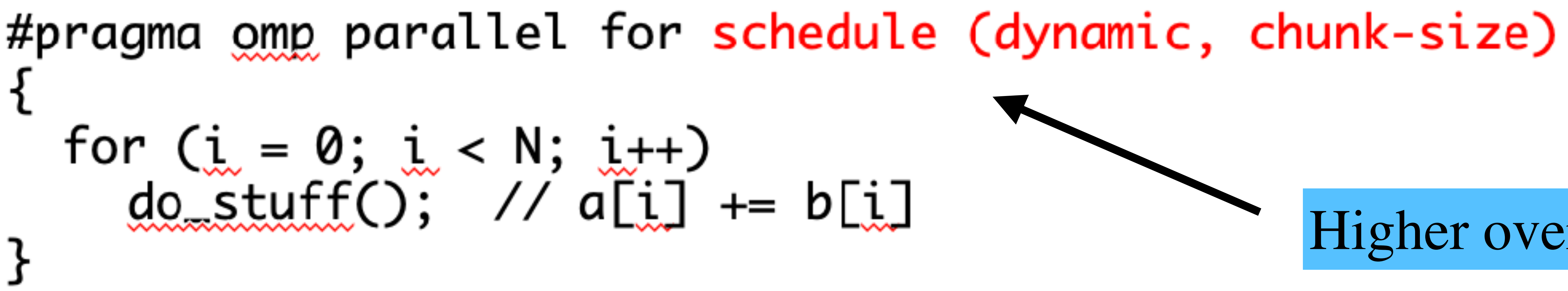
- We have already discussed the: `#pragma omp for`
- Specify the chunk-size — with static the assignment is in **round-robin** mode



```
#pragma omp parallel for schedule (static, chunk-size)
{
  for (i = 0; i < N; i++)
    do_stuff();  // a[i] += b[i]
}
```

Best for uniform work per iteration and low overhead.

- Dynamic Scheduling: On demand, thread requests chunk-size one finished with its allotted task



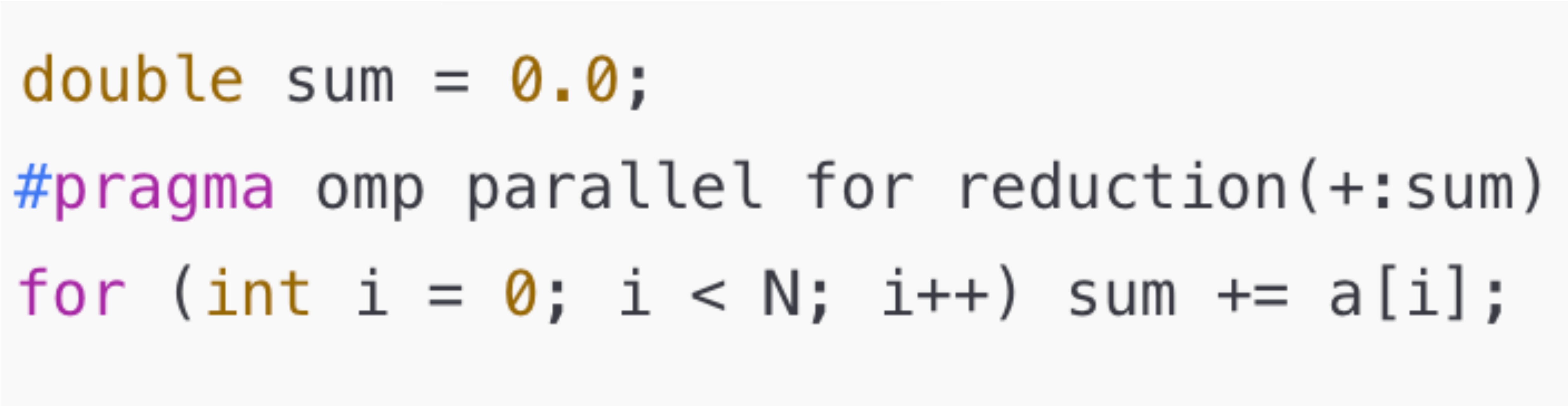
```
#pragma omp parallel for schedule (dynamic, chunk-size)
{
  for (i = 0; i < N; i++)
    do_stuff();  // a[i] += b[i]
}
```

Higher overhead but better load balance

Work Sharing Constructs: FOR

Reductions

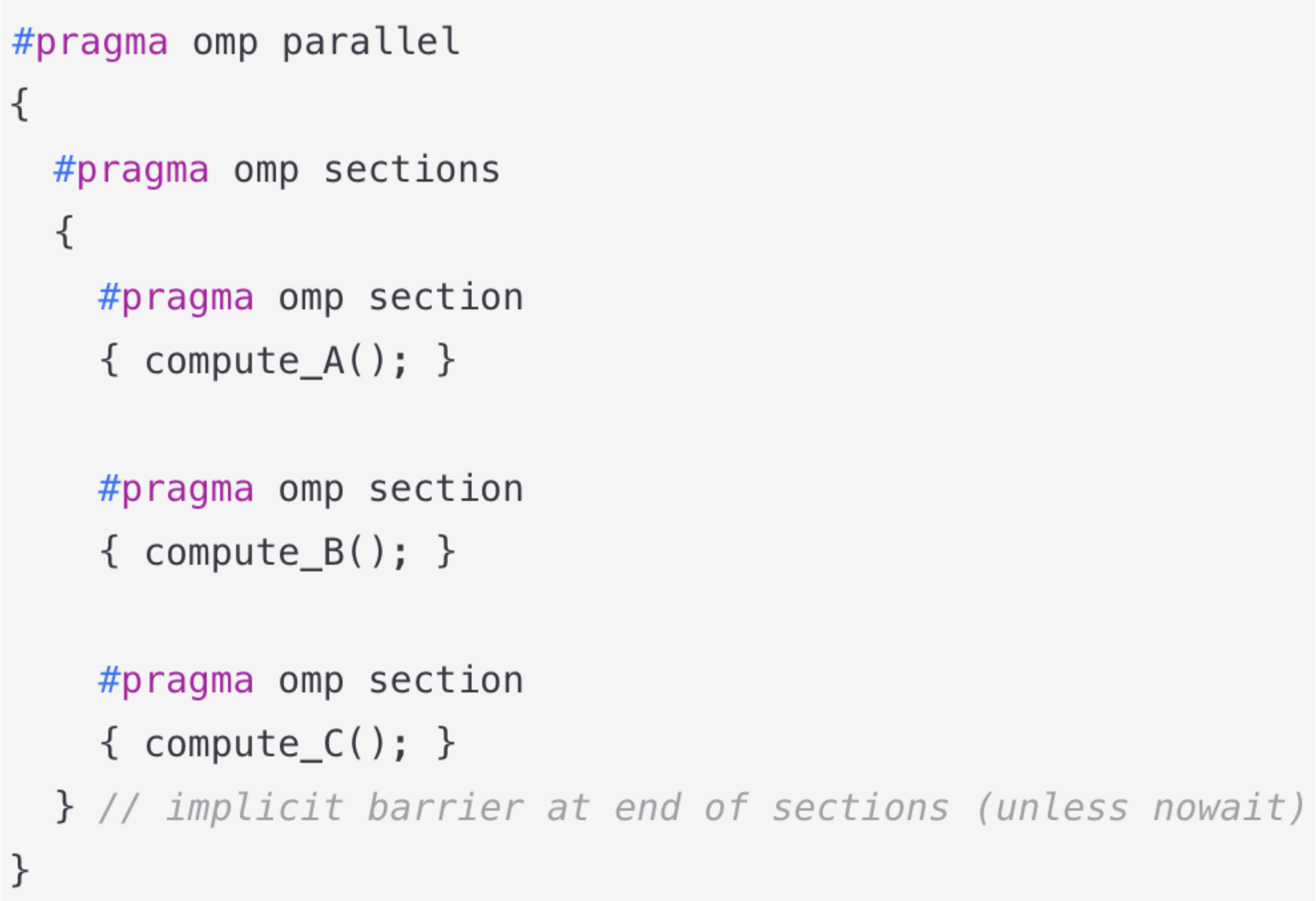
- Reductions **avoid races**
- Supported operators:
  - +, *, min, max
  - &, |, &&, ||
  - ^



```
double sum = 0.0;
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < N; i++) sum += a[i];
```

Work Sharing Constructs: Sections

- Each thread executes the region within the section
- Each section is executed only **ONCE**
- **Good use cases:**
  - Pipelined tasks
  - Overlapping I/O and compute
  - Multimodal streaming tasks
  - Divide-and-conquer



```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    { compute_A(); }

    #pragma omp section
    { compute_B(); }

    #pragma omp section
    { compute_C(); }
  } // implicit barrier at end of sections (unless nowait)
}
```

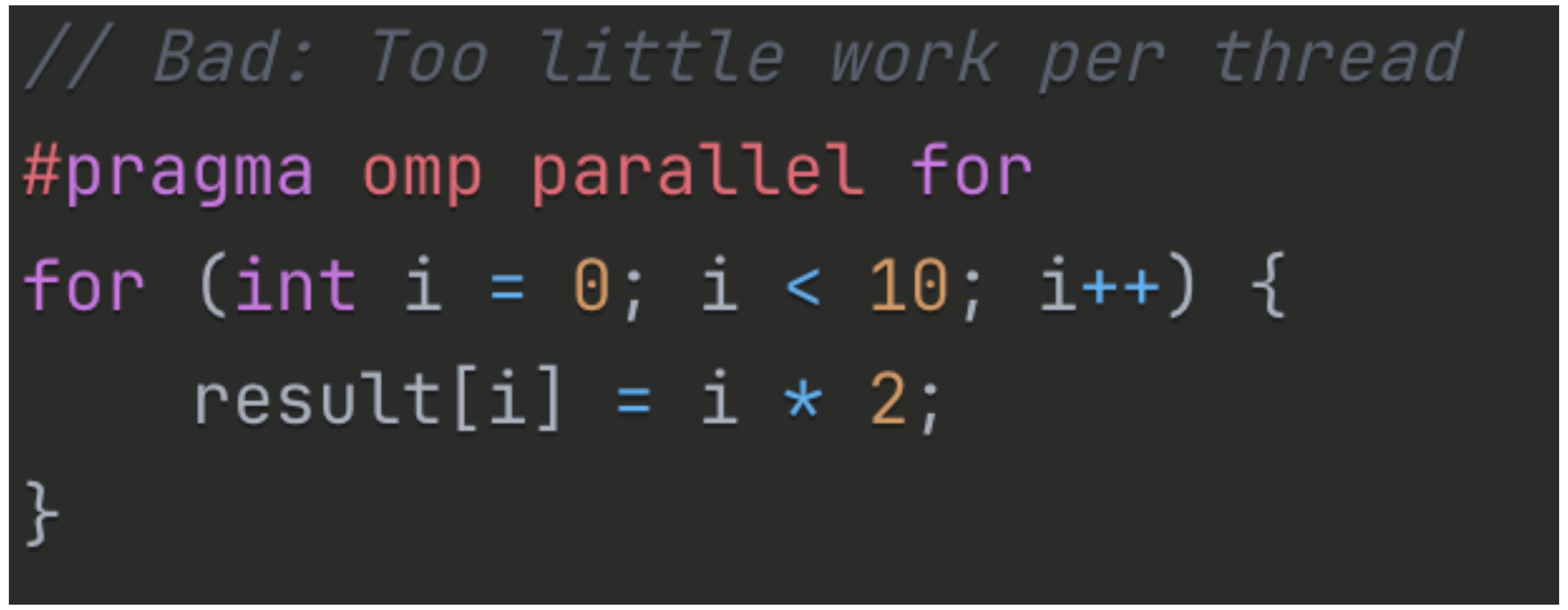
Performance Optimisation Tips

- Must have **sufficient parallelism** to mask the parallelism overhead

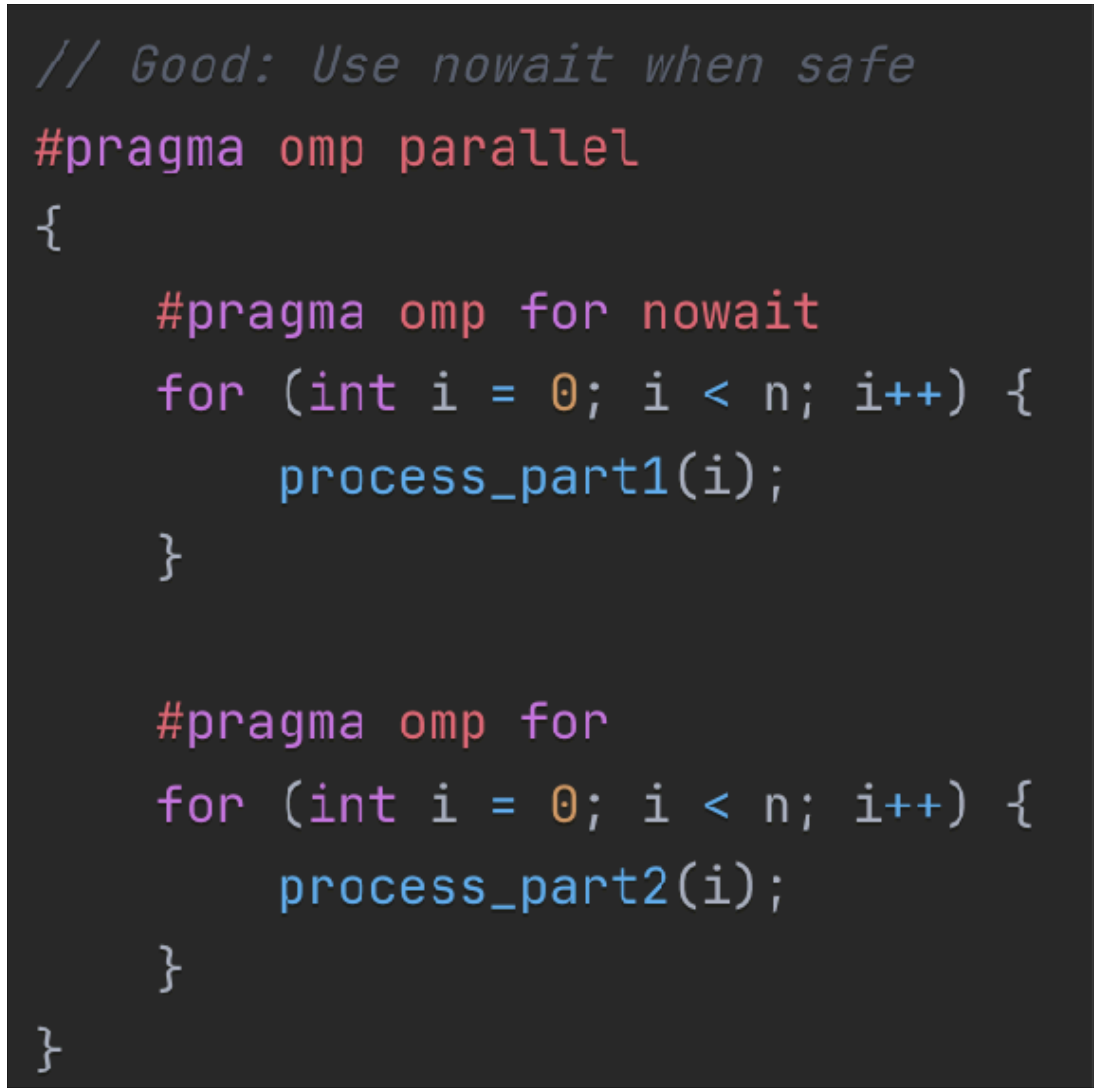
- Reduce False Sharing — pad enough bits to separate the cache lines

- Use **appropriate scheduling**

- **Minimise synchronisation** wherever unnecessary



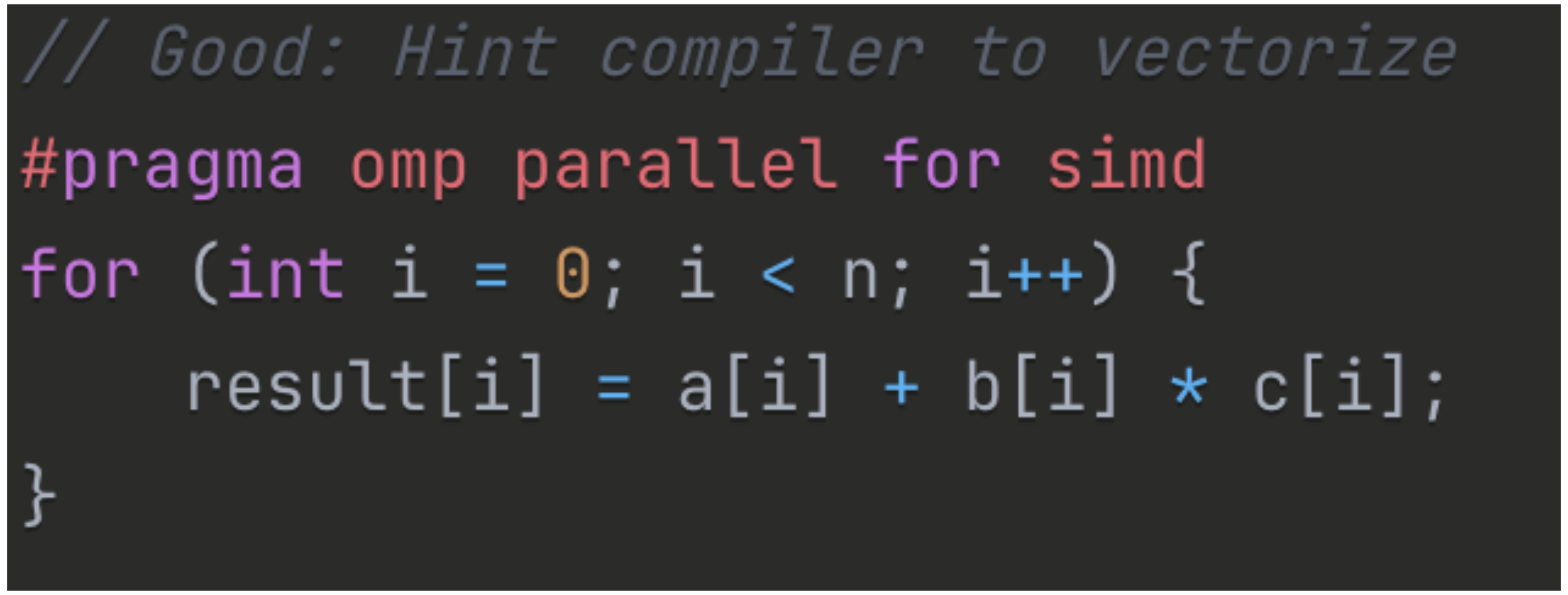
```
// Bad: Too little work per thread
#pragma omp parallel for
for (int i = 0; i < 10; i++) {
    result[i] = i * 2;
}
```



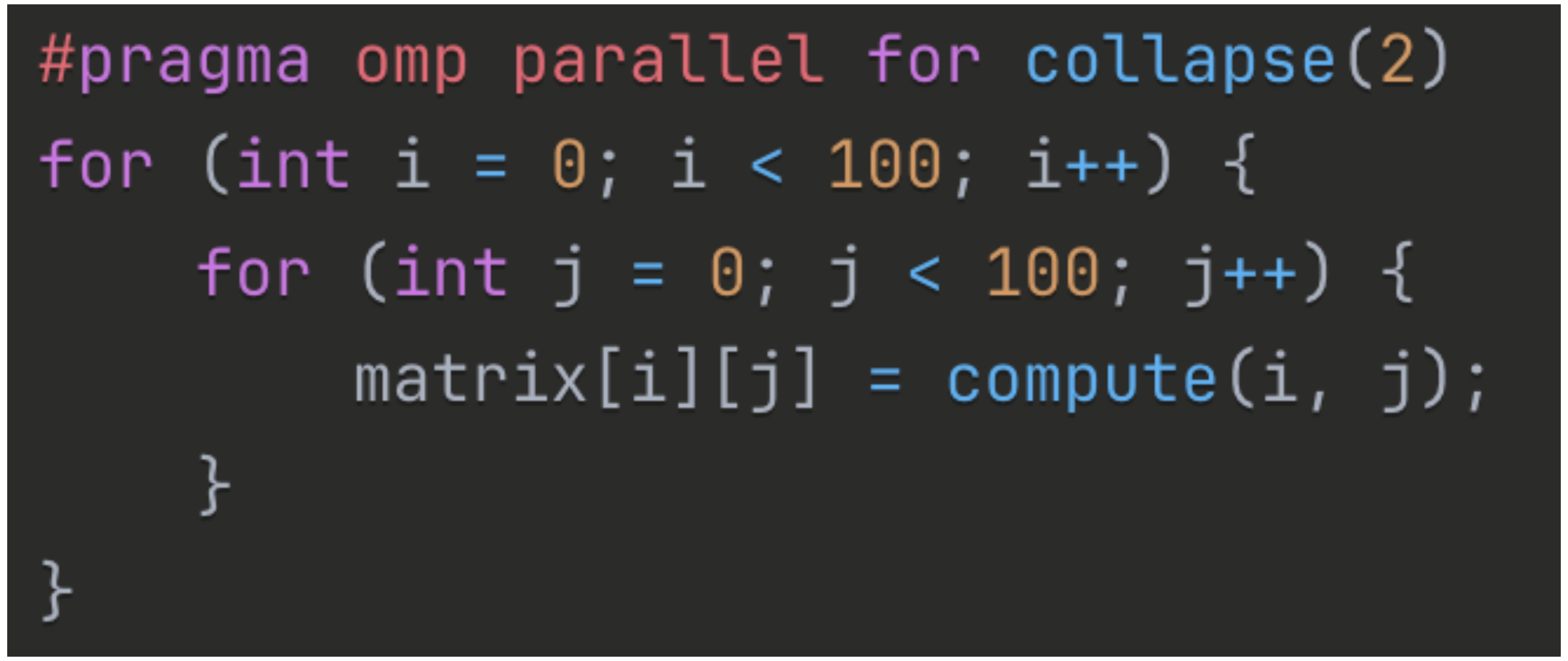
```
// Good: Use nowait when safe
#pragma omp parallel
{
    #pragma omp for nowait
    for (int i = 0; i < n; i++) {
        process_part1(i);
    }

    #pragma omp for
    for (int i = 0; i < n; i++) {
        process_part2(i);
    }
}
```

Performance Optimisation Tips

- Use SIMD Vectorisation
  - **Unsafe when:**
    - **Loop-carried dependency**
    - **Aliasing between a,b,c**
    - **Complex branches — prediction**
- Optimise for loops with **`collapse`**



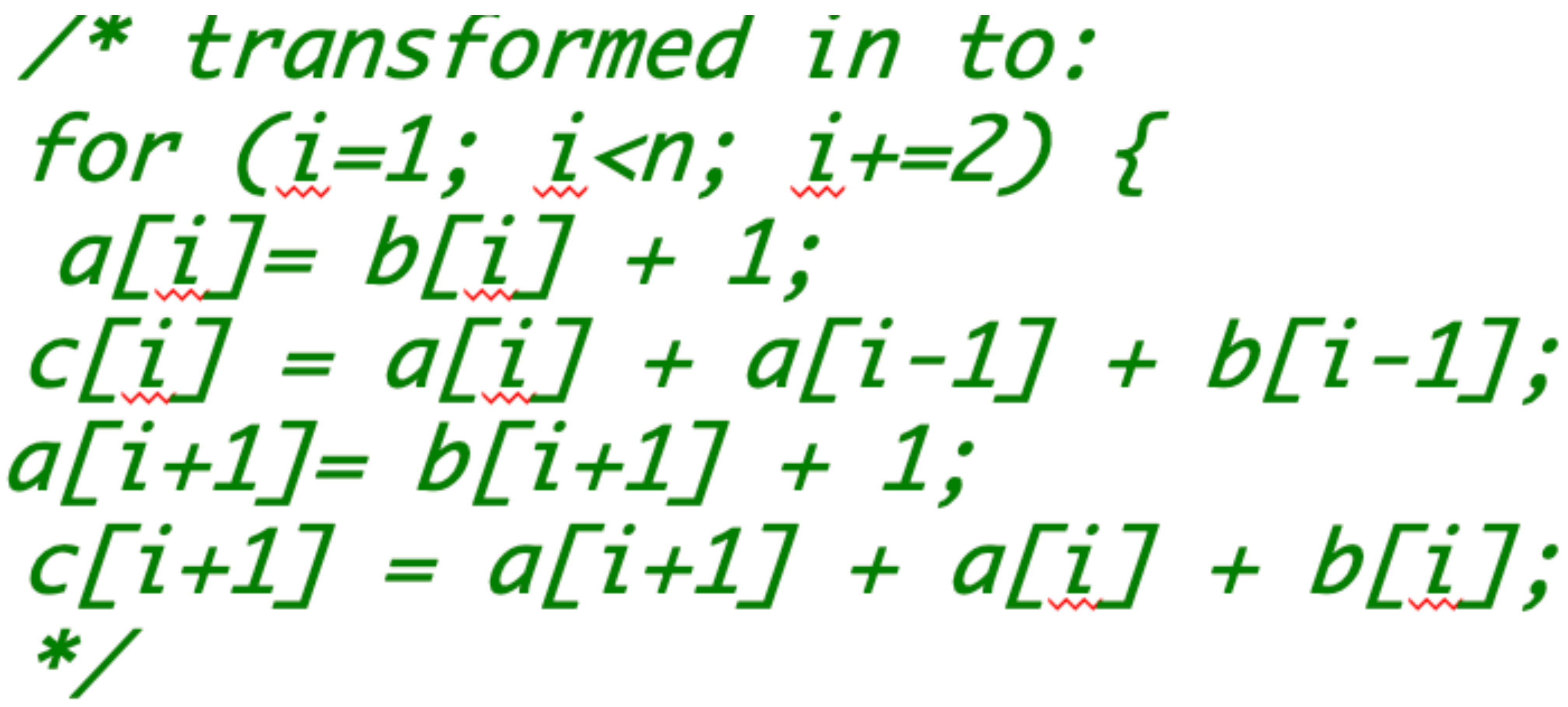
```
// Good: Hint compiler to vectorize
#pragma omp parallel for simd
for (int i = 0; i < n; i++) {
    result[i] = a[i] + b[i] * c[i];
}
```



```
#pragma omp parallel for collapse(2)
for (int i = 0; i < 100; i++) {
    for (int j = 0; j < 100; j++) {
        matrix[i][j] = compute(i, j);
    }
}
```

Performance Optimisation Tips

- Optimise thread count based on problem size
- Avoid the use of locks in loops
- Unroll loops to avoid loop overheads
- Avoid memory allocation in a parallel region
- Use Task Parallel for irregular applications



```
/* transformed in to:
for (i=1; i<n; i+=2) {
 a[i]= b[i] + 1;
c[i]  = a[i] + a[i-1] + b[i-1];
a[i+1]= b[i+1] + 1;
c[i+1] = a[i+1] + a[i] + b[i];
*/
```