Prof. Dr. D. Kranzlmüller, Dr. K. Fürlinger

# Parallel Computing

## WS 2017/18

## Session 8: Introduction to OpenMP

Tobias Fuchs, M.Sc.
tobias.fuchs@nm.ifi.lmu.de

# OpenMP Basics

# OpenMP Basics

```
#pragma omp parallel [clause ...]
                     if (scalar_expression)
                     private (list)
                     shared (list)
                     default (shared | none)
                     firstprivate (list)
                     reduction (operator: list)
                     copyin (list)
                     num_threads (integer-expression)
<structured block>
```

# OpenMP Basics

```
int main() {
  int x = 2;
  int t;
  #pragma omp parallel num_threads(4)
  {
    t = omp_get_thread_num();
    if (t == 0) {
      x = 5;
    }
    printf(“thread %d: x = %d\n”, t, x);
  }
}
```

Spot the errors in this example.

# OpenMP Basics

```
int main() {
  int x = 2;
  int t;
  #pragma omp parallel num_threads(4) shared(x) private(t)
  {
    t = omp_get_thread_num();
    if (t == 0) {
      x = 5;
    }
    #pragma omp flush(x)
    printf(“thread %d: x = %d\n”, t, x);
  }
}
```

## OpenMP Basics

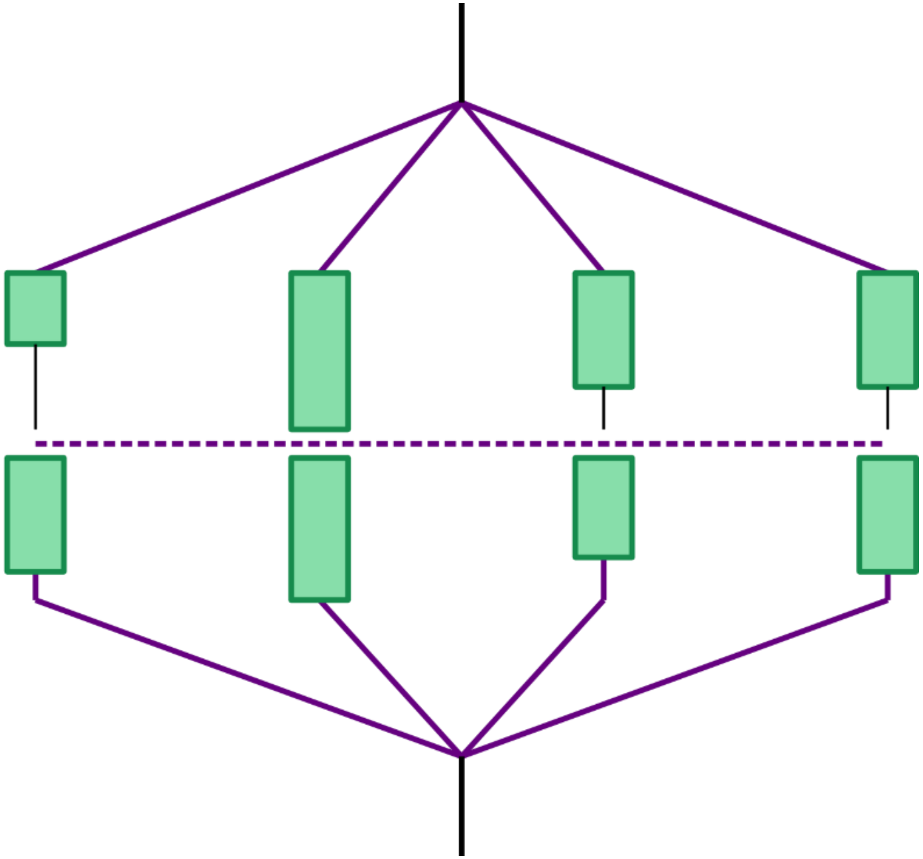
```
int main() {
  int x = 2;
  int t;
  #pragma omp parallel num_threads(4) shared(x) private(t)
  {
    t = omp_get_thread_num();
    if (t == 0) {
      x = 5;
    }
    #pragma omp barrier
    printf(“thread %d: x = %d\n”, t, x);
  }
}
```

## OpenMP Basics

```
int main() {
  int x;
  x = 2;
  #pragma omp parallel num_threads(4) shared(x) {
    do_work();
    #pragma omp barrier
    do_more_work();
  }
  return 0;
}
```

# OpenMP Basics

```
x = 2;

#pragma omp parallel {

   do_work();

   #pragma omp barrier

   do_more_work();

}

return 0;
```



## OpenMP Basics

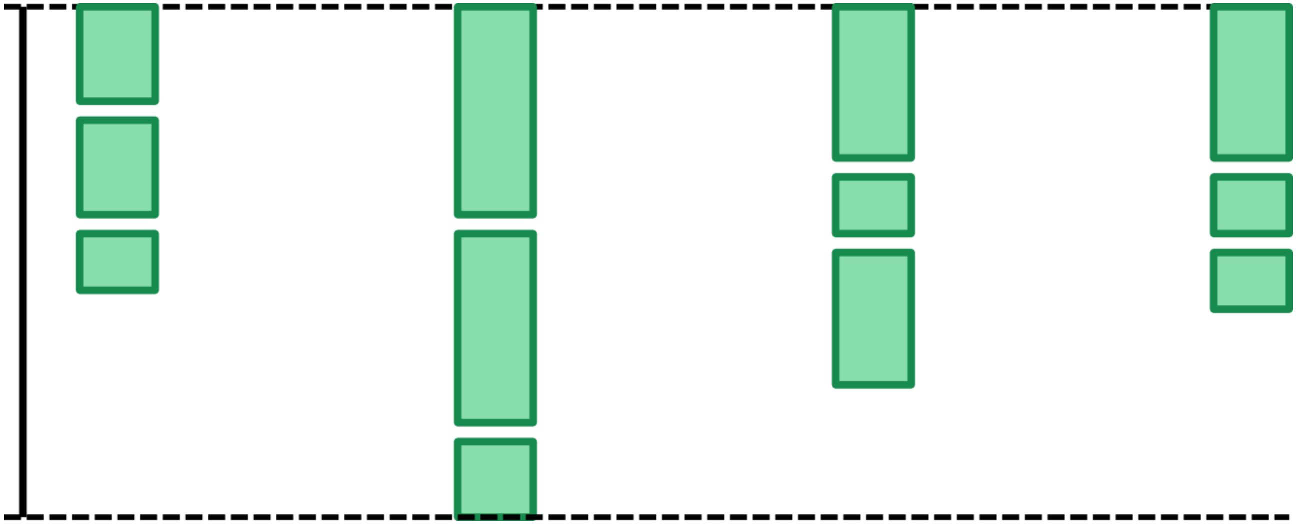
```
#pragma omp for [clause ...]
                schedule (type [,chunk])
                ordered
                private (list)
                firstprivate (list)
                lastprivate (list)
                shared (list)
                reduction (operator: list)
                collapse (n)
                nowait
<for_loop>
```

# OpenMP Basics

```
int main() {
    #pragma omp parallel for
    for (int i = 0; i < 1200; i++) {
        do_work();
    }
}
```

The end of the for-loop is an implicit barrier.

# OpenMP Basics



# OpenMP Basics

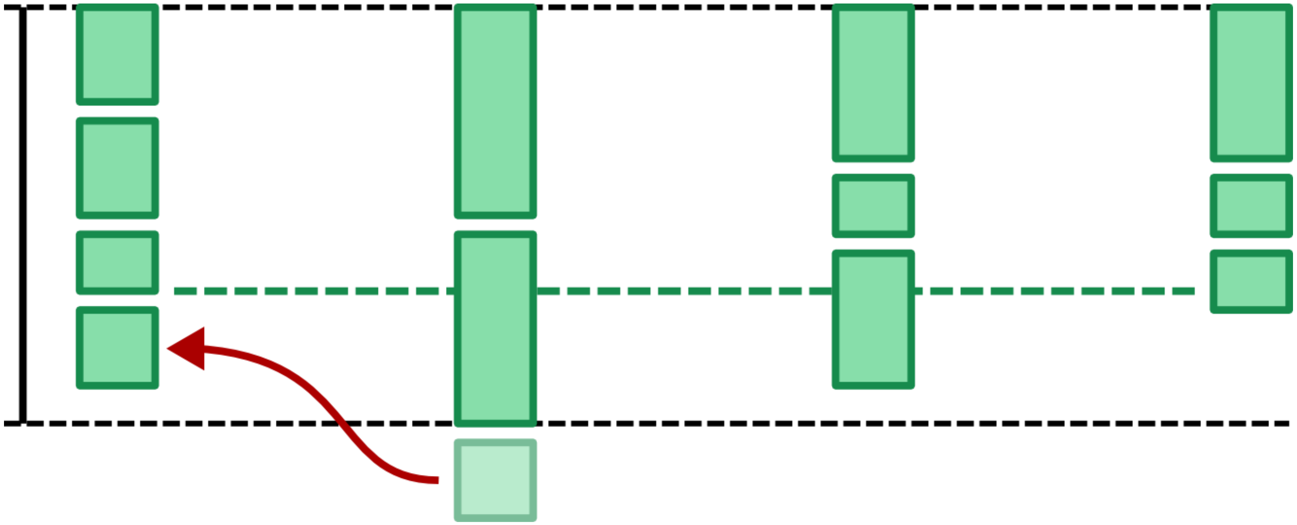
```
int main() {
    int chunk = 100;
    #pragma omp parallel for schedule(dynamic, chunk)
    for (int i = 0; i < 1200; i++) {
        do_work();
    }
}
```

**schedule(dynamic [, chunk])**

Loop iterations are divided into pieces of size *chunk* and dynamically scheduled among the threads; when a thread finishes one chunk, it is dynamically assigned another.

The default chunk size is 1.

# OpenMP Basics



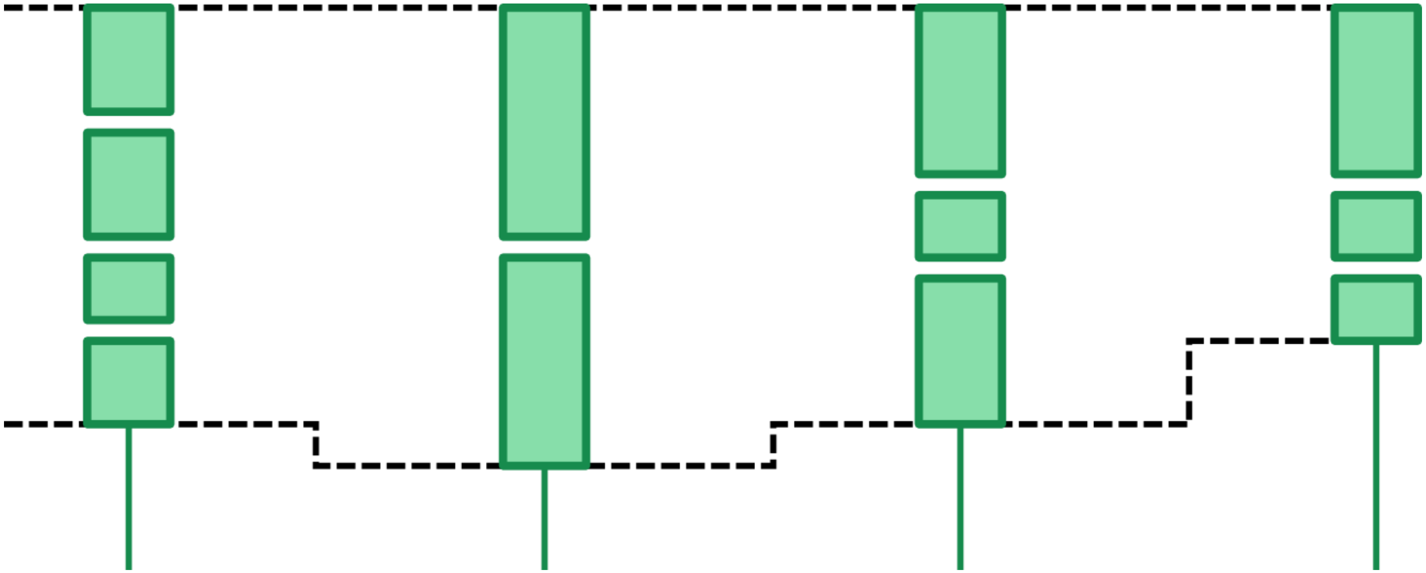
# OpenMP Basics

```
int main() {
    int chunk = 100;
    #pragma omp parallel for schedule(dynamic, chunk) nowait
    for (int i = 0; i < 1200; i++) {
        do_work();
    }
}
```

**nowait**:

Threads do not synchronize at the end of the parallel loop.

# OpenMP Basics



# OpenMP Basics

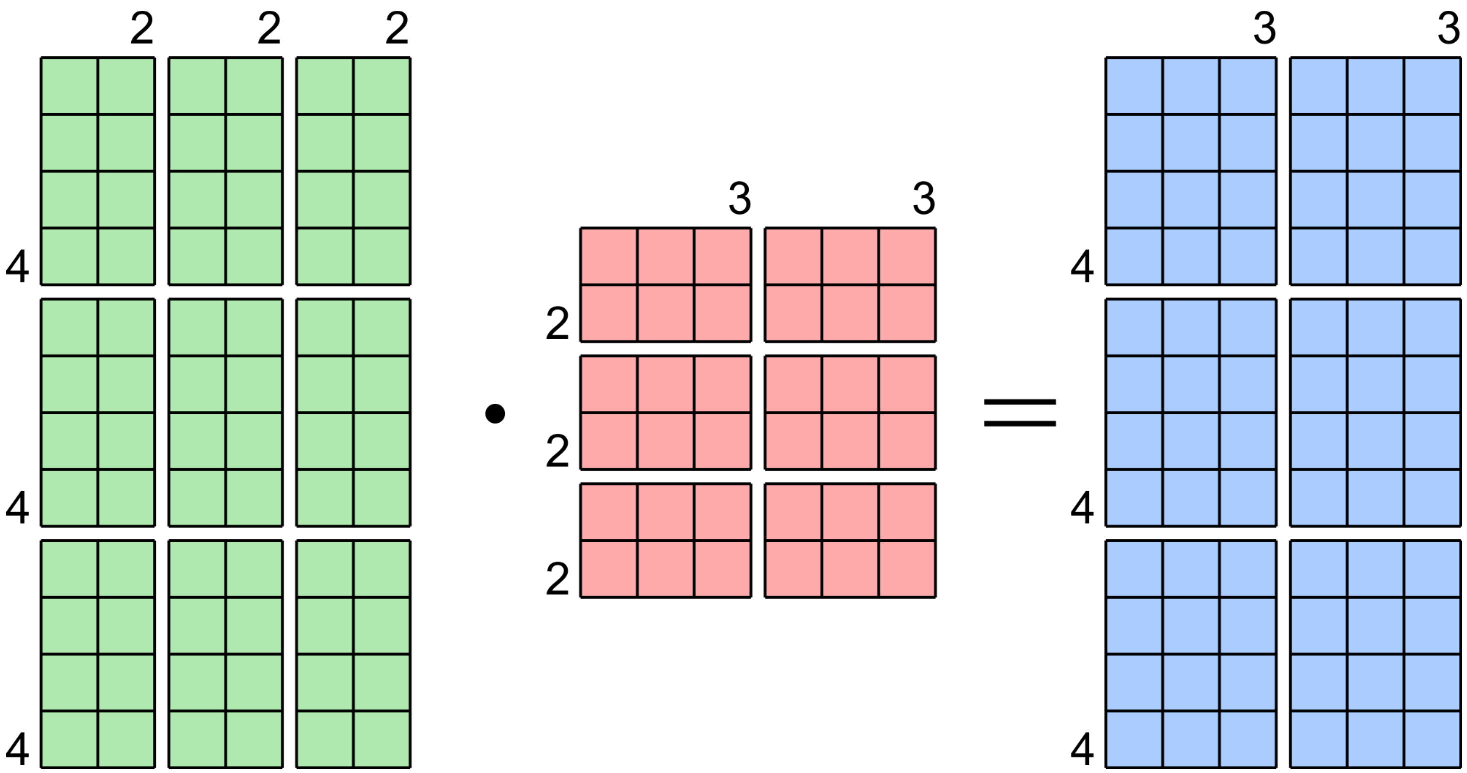
```
#pragma omp sections [clause ...]
                     private (list)
                     firstprivate (list)
                     lastprivate (list)
                     reduction (operator: list)
                     nowait
{
  #pragma omp section
  <structured_block>

  #pragma omp section
  <structured_block>


  ...
}
```

# OpenMP Basics

```
int main() {
  #pragma omp parallel
  {
    #pragma omp sections nowait
    {
      #pragma omp section
      for (int a = 0; a < 1024; a++) { compute_foo(a); }
      #pragma omp section
      for (int b = 0; b < 1024; b++) { compute_bar(b); }
    }
  }
}
```

# Discussion: DGEMM Optimization

Block matrix multiplication:



# Discussion: DGEMM Optimization

```
// Outer loops: Iterate in b x b block steps
for (i0 = 0; i0 < n; i0 += b):
    for (j0 = 0; j0 < n; j0 += b):
        for (k0 = 0; k0 < n; k0 += b):
            // Inner loop: Matrix product of single block
            // -> 2b^3 operations on 3b^2 elements
            for (i = i0; i < min(i0+b, n); i++):
                for (j = j0; j < min(j0+b, n); j++):
                    double sum = C[i][j];
                    for (k = k0; k < min(k0+b, n); k++):
                        sum += A[i][k] * B[k][j];
                    C[i][j] = sum;
```

**How to parallelize this using OpenMP? (→ next homework assignment)**

Tobias Fuchs

tobias.fuchs@nm.ifi.lmu.de

www.mnm-team.org/~fuchst