OpenMP 1

Victor Eijkhout & Cyrus Proctor

PCSE 2015

Parallel region

```
#pragma omp parallel [clauses....]
```

- `private` and `shared` control thread data
- `num_threads` use value different from default
- `if` conditional use of parallelism

Parallel region and worksharing

```
#pragma omp parallel
{
  // parallel code here
}
```

- Parallel region: create team
- Worksharing construct: distribute work

Work sharing

Work sharing constructs

construct	description
do/for	loop iterations
sections	discrete code sections
single	only one thread
workshare	(Fortran only) unit of work

Parallel loop

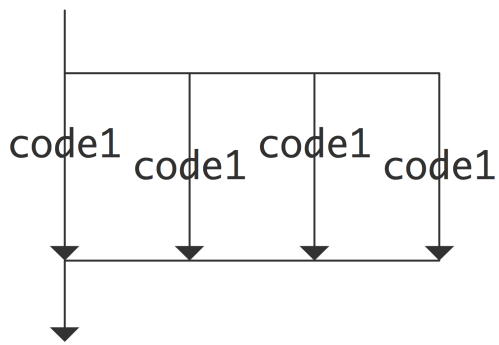
```
#pragma omp parallel
#pragma omp for
  for (i=0; i<N; i++) {
    ....
  }
```

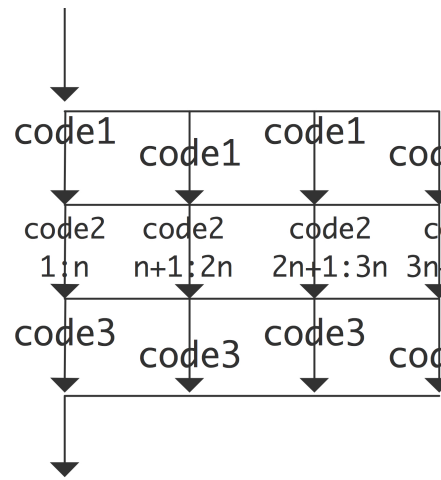
- Loop iterations are divided over the thread team
- Loop variable is automatically private
- Many ways of dividing the iterations.
- Some restrictions on loop variable.
- Special case: reductions.

Often abbreviated: `omp parallel for`

Region vs loop

```
#pragma omp parallel
{
  code1();
#pragma omp for
  for (i=1; i<=4*N; i++) {
    code2();
  }
  code3();
}
```



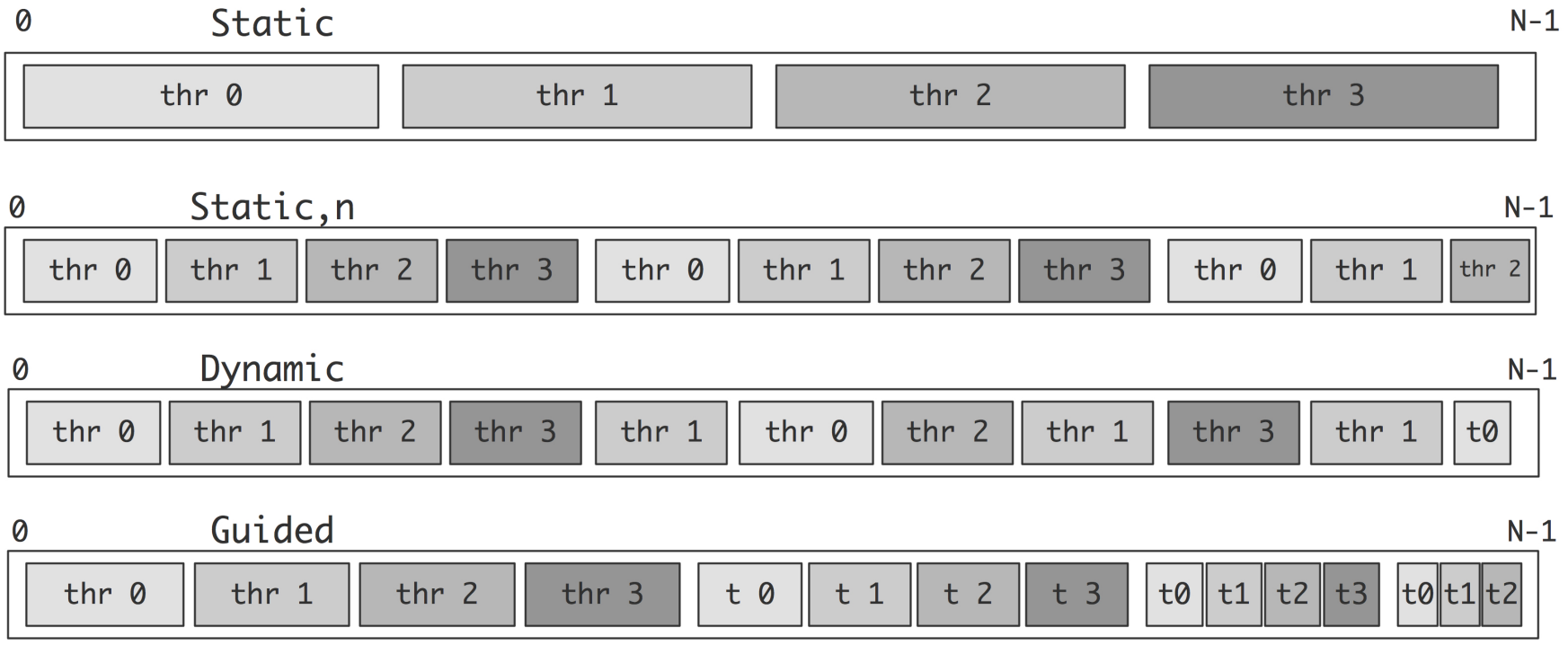


Iteration scheduling

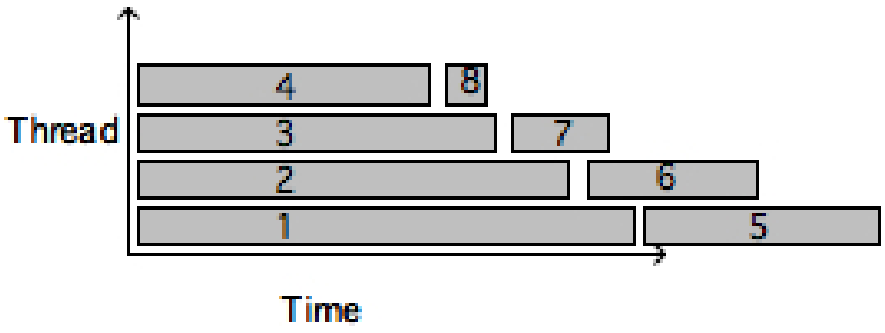
Add a scheduling clause:

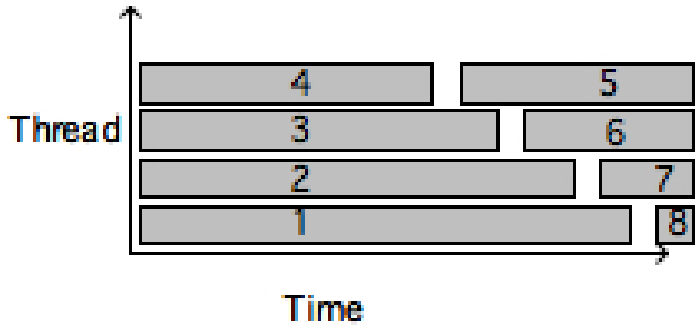
- `schedule(static)` n iterations divided in blocks of n/p.
- `schedule(static,m )` iterations divided in blocks of m ('chunk size'), assigned cyclically.
- `schedule(dynamic)` single iterations, assigned whenever a thread is idle
- `schedule(dynamic,m)` blocks of m iterations, assigned whenever a thread is idle
- `schedule(guided)` decreasing size blocks
- `schedule(auto)` leave it up to compiler/runtime
- `schedule(runtime)` using environment variable `OMP_SCHEDULE`

schedules


0
Static
N-1
thr 0
thr 1
thr 2
thr 3
0
Static,n
N-1
thr 0
thr 1
thr 2
thr 3
thr 0
thr 1
thr 2
thr 3
thr 0
thr 1
thr 2
0
Dynamic
N-1
thr 0
thr 1
thr 2
thr 3
thr 1
thr 0
thr 2
thr 1
thr 3
thr 1
t0
0
Guided
N-1
thr 0
thr 1
thr 2
thr 3
t 0
t 1
t 2
t 3
t0
t1
t2
t3
t0
t1
t2

Static vs dynamic





Reduction

```
#pragma omp parallel for \
        reduction(+:s)
  for (i=0; i<N; i++)
    s += f(i);
```

- Reductions are atomic operations
- Can be solved by private variable per thread
- `reduction` clause is shorthand for all that

Subtlety

The partial sums are initialized to the unit:

```
  int v = 256, i;
#pragma omp parallel for reduction(+:v)
  for (i=0; i<16; i++) {
    v += 2*i;
    if (v>450) printf("hitting %d\n",v);
  }
```

Final result: 496