Nested Parallelism

Collapse Construct, Nested Parallel Constructs

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

How Will You Parallelize This Loop?

```
N=10;
for (i=0; i< N; i++)
   for (j=0; j< N; j++)
    for (k=0; k< N; k++)
       A[i][j][k] = f( i, j, k);
          // assume f is expensive but
          // variable in execution time
```

Assume that you have a machine with 30 cores

All the loops are parallel, but none is adequately large enough to employ all the threads
Further, the variability in execution time means the need to use dynamic schedule

How Will You Parallelize This Loop?

Assume that you have a machine with 30 cores

```
N=10;
#pragma omp parallel for schedule(dynamic)
for (m=0; i< N*N*N; i++)   {
       i = m /(N*N);
       j = (m % (N*N))/N;
       k  =m % N;
       A[i][j][k] = f( i, j, k);
           // assume f is expensive but
           // variable in execution time
}
```

All the loops are parallel, but none is adequately large enough to employ all the threads
Further, the variability in execution time means the need to use dynamic schedule

Collapse Clause

```
N=10;
#pragma omp parallel for collapse(3) schedule(dynamic)
for (i=0; i< N; i++)
   for (j=0; j< N; j++)
    for (k=0; k< N; k++)
       A[i][j][k] = f( I, j, k);
          // assume f is expensive but
          // variable in execution time
```

3 nested loops are combined to make a single loop with 1000 iterations. I, j, k calculated automatically. Also, no need to declare j and k private. They are implicitly private

Of course, the collapse clause is useful even when we don't have an expensive yet variable function f, requiring dynamic balancing

Still useful to automatically assign iterations to threads

`collapse` Clause and **`lastprivate`**

- What happens to **`lastprivate`** variables in a loop nest controlled by the collapse clause?
- They get the value the sequentially last iteration would have gotten

Nested Parallelism

- Parallel region within parallel region

```
 int p;
 omp_set_nested(1);
 omp_set_dynamic(0); // make thread number adjustment explicit
 #pragma omp parallel num_threads(8)
 {
   #pragma omp single
   printf("outer total number of omp threads = %d\n",
           omp_get_num_threads());

   printf("thread number: %d\n", omp_get_thread_num());
   #pragma omp parallel num_threads(2)
   printf("inner parallel region thread number: %d\n",
           omp_get_thread_num());

  }
}
```

Nested Parallelism

- Output

```
total number of procs = 8
outer total number of omp threads = 8
thread number: 0
thread number: 1
thread number: 3
thread number: 7
thread number: 2
inner parallel region thread number: 1
thread number: 6
thread number: 4
thread number: 5
inner parallel region thread number: 0
inner parallel region thread number: 1
inner parallel region thread number: 0
inner parallel region thread number: 1
inner parallel region thread number: 0
inner parallel region thread number: 1
inner parallel region thread number: 0
inner parallel region thread number: 0
inner parallel region thread number: 1
inner parallel region thread number: 1
inner parallel region thread number: 1
inner parallel region thread number: 0
inner parallel region thread number: 0
inner parallel region thread number: 0
inner parallel region thread number: 1
```

Nested Levels of Parallelism

- You can also control the amount of parallelism at each level using an environment variable
  - Set in your shell
  - Csh: setenv OMP_NUM_THREADS 3,5
  - First level of parallel creates a team of 3 threads, and the next level nested "parallel" gets 5 threads in each of the 3 teams
  - The num_threads clause in parallel construct overrides this