Programming Techniques for Supercomputers Tutorial

Erlangen National High Performance Computing Center

Department of Computer Science

FAU Erlangen-Nürnberg

Sommersemester 2024

Assignment 10 – Task 1: tasking for the ray tracer

```
#pragma omp parallel
{
  #pragma omp single
  {
    for(yc=0; yc<xtiles; yc++)
      for(xc=0; xc<ytiles; xc++) {
        #pragma omp task firstprivate(xc,yc)
        {
          unsigned char tile[tilesize*tilesize];
          calc_tile(size, xc*tilesize, yc*tilesize, tilesize, tile);
          for(int i=0; i<tilesize; i++) {
            tilebase=yc*tilesize*tilesize*xtiles+xc*tilesize;
            memcpy((void*)(picture+tilebase+i*tilesize*xtiles),
                   (void*)(tile+i*tilesize),
                   tilesize*sizeof(char));
          }
        } // end task
      }
  } // end single
} // end parallel
```

Other option: one tile per thread on heap:

`tile[tID][ts*ts]`

Same performance as loop-parallel version at 15k x 15x and tilesize=100:
≈ 105 Mpx/s @ 36c

Assignment 10 – Task 1: loop-based reference

```
// ...
#pragma omp parallel private(tile)
{
tile=(char*)malloc(tilesize*tilesize*sizeof(char));

//..

count = 0;
#pragma omp for schedule(runtime) collapse(2) private(xc,i)
for(yc=0; yc<ytiles; yc++) {
  for(xc=0; xc<xtiles; xc++) {
     /* calc one tile */
     calc_tile(size, xc*tilesize, yc*tilesize, tilesize, tile);
     /* copy to picture buffer */
     for(i=0; i<tilesize; i++) {
         int tilebase = yc*tilesize*tilesize*xtiles+xc*tilesize;
         memcpy((void*)(picture+tilebase+i*tilesize*xtiles),
                (void*)(tile+i*tilesize),
                tilesize*sizeof(char));
     }
   }
}

}
```

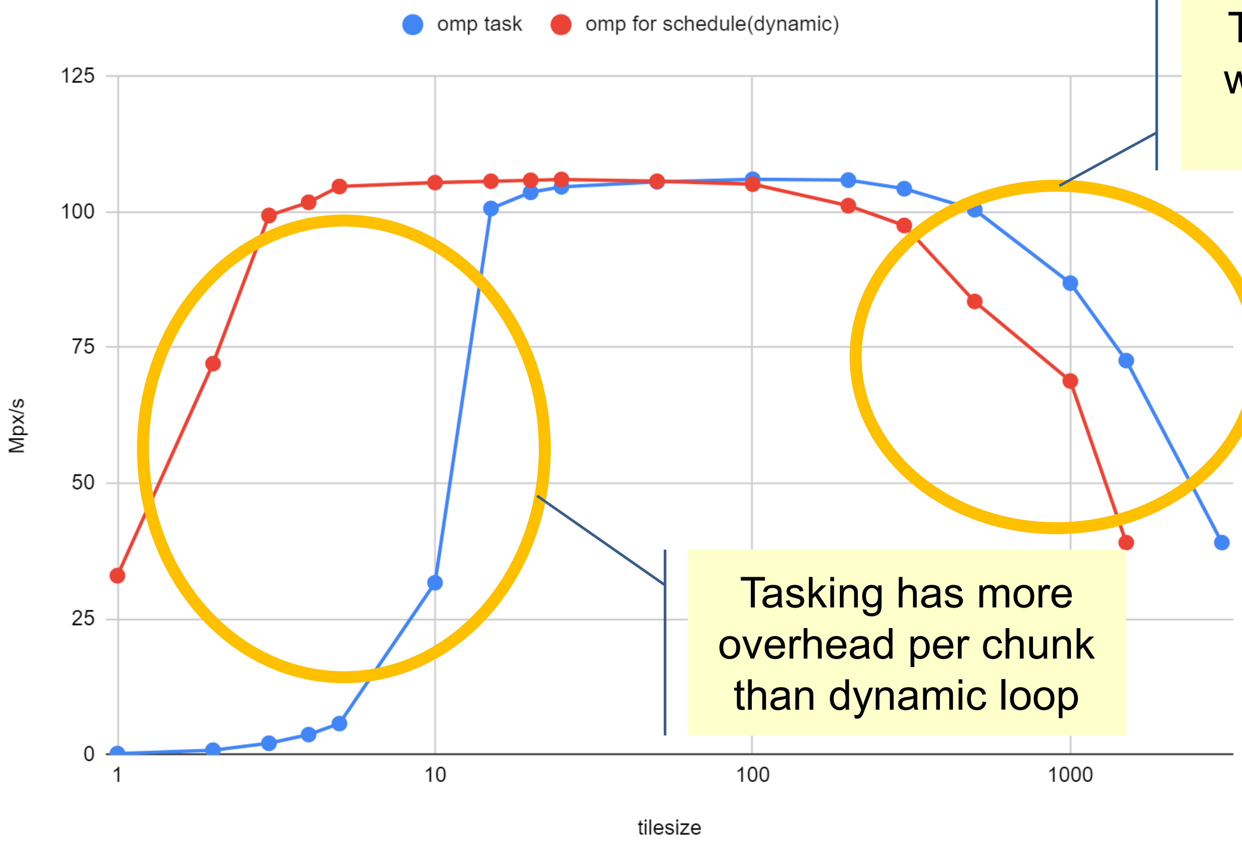
Assignment 10 – Task 1

Shell loop for measurement

```
$ for t in 1 2 3 4 5 10 15 20 25 50 100 200 300 500 1000 1500 3000; do \
    echo -n $t " "; OMP_SCHEDULE=dynamic OMP_NUM_THREADS=36 \
    OMP_PROC_BIND=close OMP_PLACES=cores \
    srun --cpu-freq=2000000-2000000 ./a.out 15000 $t; \
  done
```

Assignment 10 – Task 1

Results



Tasking has better work distribution in “starved” case

Tasking has more overhead per chunk than dynamic loop

Assignment 10 – Task 2

- The SPR nodes in Fritz have 4 ccNUMA domains per socket (13 cores each)
- Shell script for measurement:

```
$ for c in `seq 0 7`; do
    for m in `seq 0 7`; do \
      echo -n $c " " $m " "; \
      srun --cpu-freq=2000000-2000000 \
        numactl -m $m \
           likwid-bench -t daxpy_avx -w M${c}:2GB:13 2>&1 | \
           grep MByte/s ; \
    done; \
  done
```

Assignment 11 – Task 3

Numbers in Gbyte/s

c (down)/m (across)	0	1	2	3	4	5	6	7
0	**62.1**	61.8	61.9	62.0	43.7	43.8	43.6	43.7
1	61.8	**62.4**	61.9	62.1	43.7	43.9	43.5	43.6
2	61.8	61.9	**62.2**	62.0	43.6	43.9	43.5	43.6
3	61.8	62.0	61.8	**62.3**	43.7	43.8	43.6	43.6
4	43.7	43.8	43.6	43.8	**62.2**	62.1	61.9	61.9
5	43.8	43.8	43.8	43.8	61.9	**62.5**	61.8	61.9
6	43.8	43.8	43.8	43.8	61.9	62.0	**62.2**	61.9
7	43.8	43.9	43.9	43.9	61.7	62.1	61.7	**62.3**

≈ 30% loss for inter-socket access

Hardly any loss for intra-socket inter-domain access