Erlangen Regional Computing Center

UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Winter term 2020/2021

Parallel Programming with OpenMP and MPI

Dr. Georg Hager
Erlangen Regional Computing Center (RRZE) at Friedrich-Alexander-Universität Erlangen-Nürnberg
Institute of Physics, Universität Greifswald

Assignment 4 discussion

High Performance Computing

Assignment 4, Task 1

- Recurrence is only “fake” – arrays can be computed from index alone

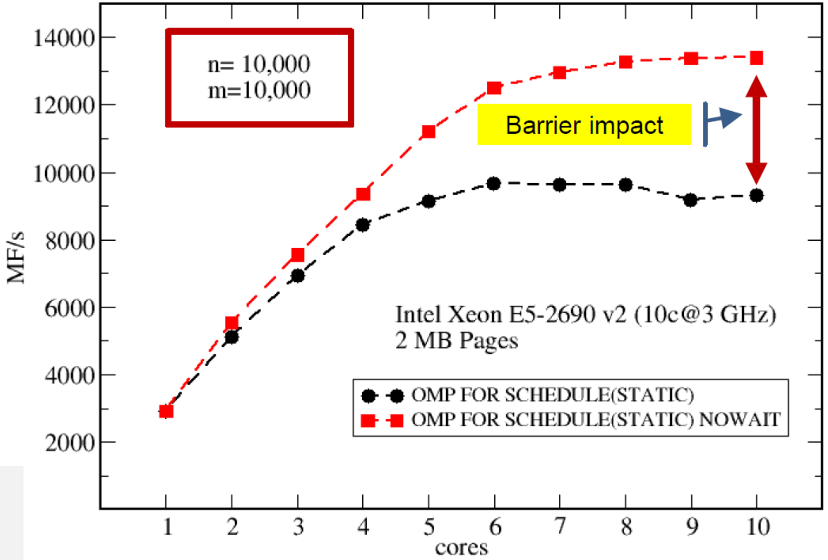
```
for(i=1;i<N;i++) {
  b[i]=1+i;
  c[i]=b[i-1]+i;
  d[i]=c[i-1]+i;
}
```

```
b[1]=2
c[1]=b[0]+1
d[1]=c[0]+1
b[2]=3;
c[2]=b[1]+2;
d[2]=c[1]+2;
#pragma omp parallel for
for(i=3;i<N;i++) {
  b[i]=1+i;
  c[i]=2*i;
  d[i]=3*i-2;
}
```

Assignment 4, Task 2

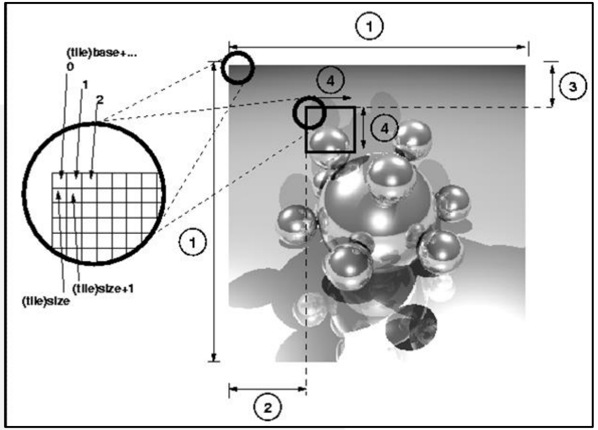
- Dense MVM has 2 flops per iteration
  - One barrier per inner loop traversal ($2m$ flops)
- Time per inner loop = $\frac{2m}{P}$
- Barrier time = $\frac{2m}{P_{bad}} - \frac{2m}{P_{good}} \approx 1900\ \mathrm{cy}$

```
#pragma omp parallel
{
  for(int c=0; c<n; ++c)
    int offset = m * c;
    #pragma omp for schedule(static)
    for(int r=0; r<m; ++r)
      lhs[r] += mat[r + offset] * rhs[c];
}
```



Assignment 4, Task 3

- OpenMP-parallel raytracer: Parallelize outer loop



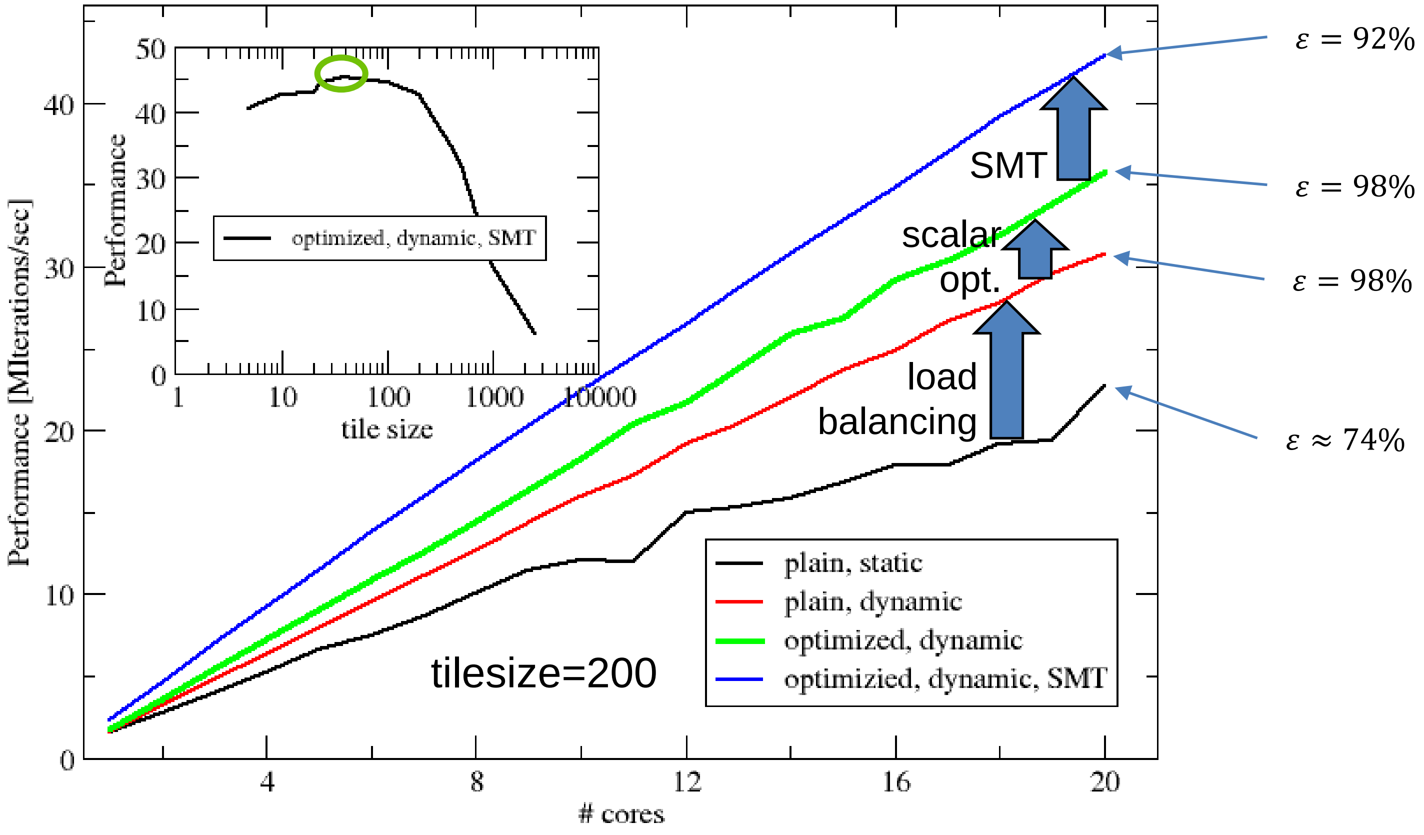
```
#pragma omp parallel private(tile)
{
  tile=(char*)malloc(tilesize*tilesize*sizeof(char))

#pragma omp for schedule(runtime) collapse(2)
  for(int yc=0; yc<ytiles; yc++)
    for(int xc=0; xc<xtiles; xc++) {
        /* calc one tile */
        calc_tile(size, xc*tilesize, yc*tilesize, tilesize, tile);
        /* copy to picture buffer */
        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));
} } }
```

Assignment 4, Task 3

- Code optimizations
- Avoiding FP divides in **`shade()`** and **`calc_tile()`** yields about 15% speedup

```
r = 1./sqrt(nx * nx + ny * ny + nz * nz);
nx *= r; ny *= r; nz *= r;
...
r = sqrt(ldx * ldx + ldy * ldy + ldz * ldz);
rr = 1./r;
ldx *= rr; ldy *= rr; ldz *= rr;
...
r = 1./sqrt(dx * dx + dy * dy + dz * dz);
c = 100 * shade(2.1, 1.3, 1.7, dx * r, dy * r, dz * r, 0);
```

Assignment 4, Task 3



Assignment 4, Task 3

- Measured performance: ~20-40 MPixels/s
- Every pixel is one byte that has to be read from memory and written back
- The memory bandwidth caused by the code is thus about 40-80 Mbyte/s

- The available bandwidth on an Emmy socket is 40 Gbyte/s, so we are far away from saturation → this code is not limited by memory bandwidth.