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

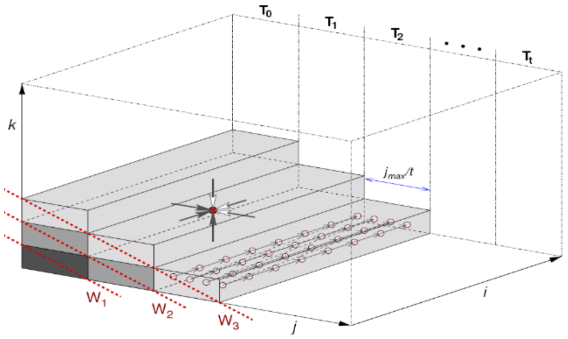
Dr. Georg Hager
Erlangen Regional Computing Center (RRZE) at Friedrich-Alexander-Universität Erlangen-Nürnberg
Institute of Physics, Universität Greifswald

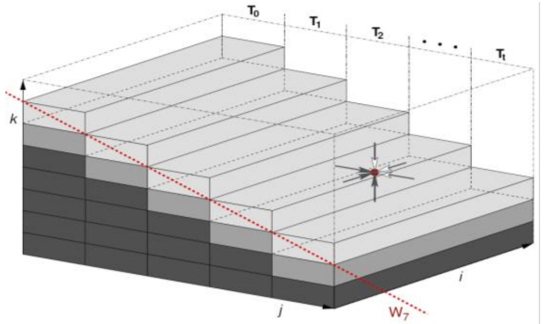
Assignment 6 discussion

High Performance Computing

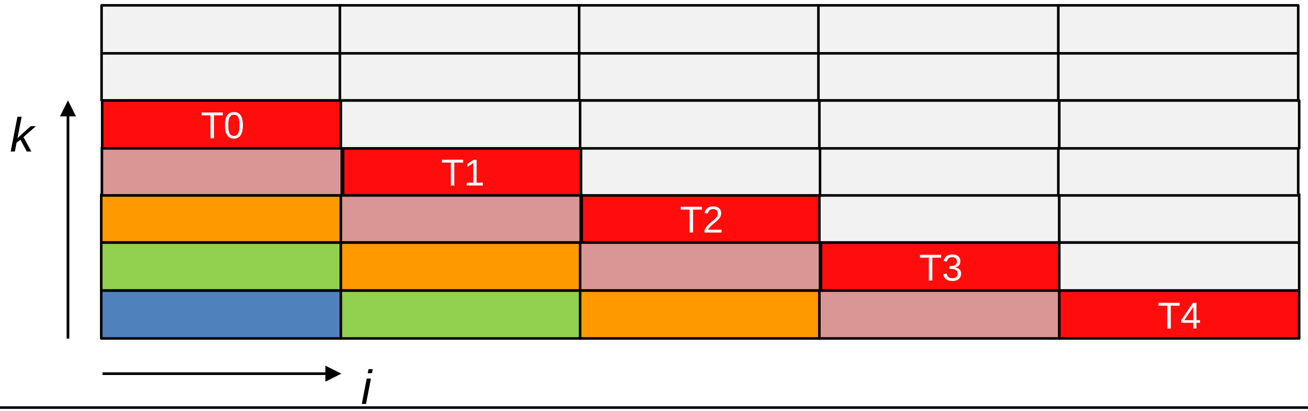
Assignment 6, Task 1: wavefront-parallel Gauss-Seidel

Principle in 3D (“pipeline parallel processing”):





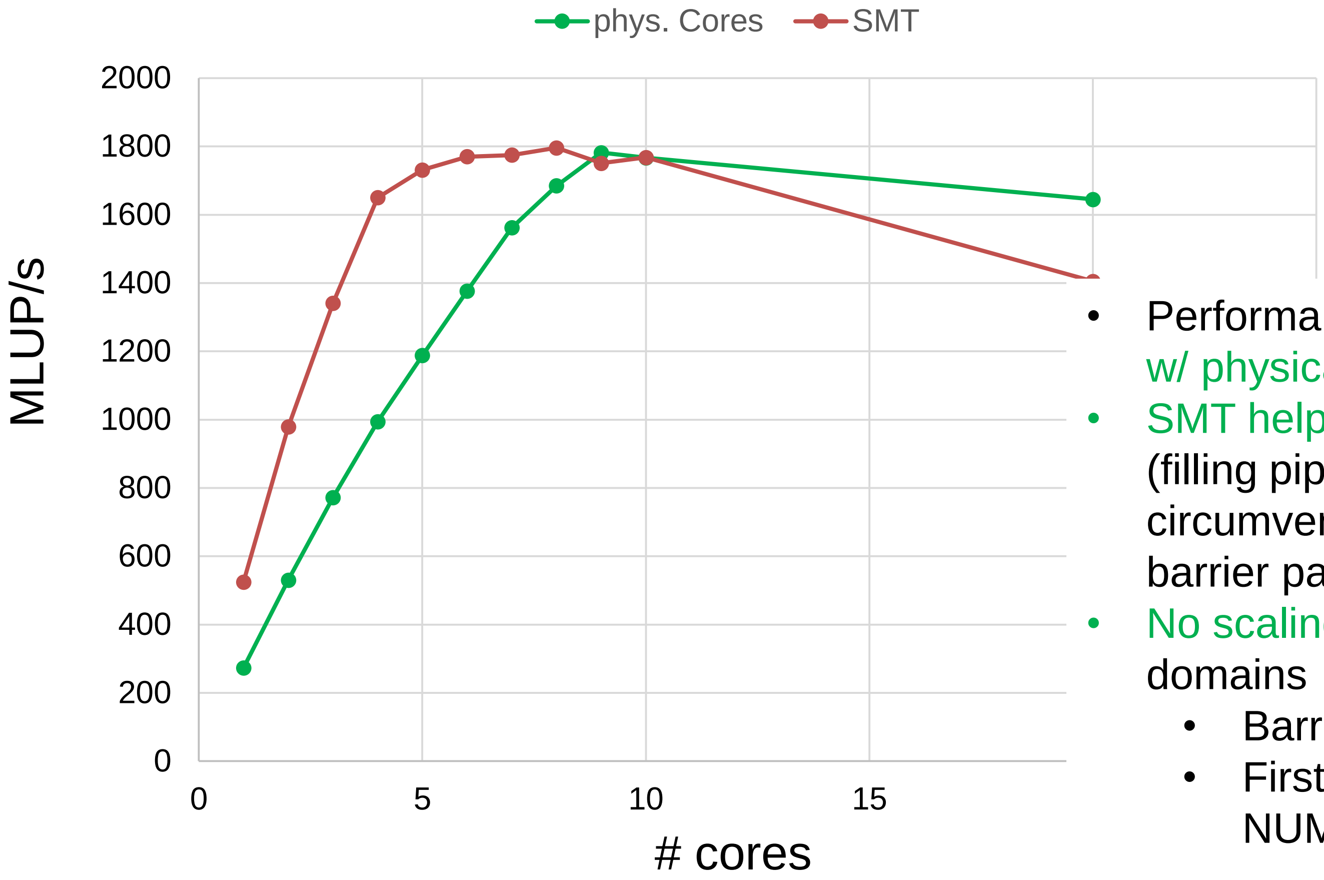
2D:



Assignment 6, Task 1: wavefront-parallel Gauss-Seidel

```
!$OMP parallel private(nthreads,istart,iend,tid,kk,it,k,i)
  nthreads = omp_get_num_threads()
  tid = omp_get_thread_num()
  istart= (imax-1)/nthreads * tid +1
  iend  = istart+(imax-1)/nthreads-1
  do it=1,itmax
     do k=1,kmax-1+nthreads-1
        kk = k - tid
        if(kk .ge. 1 .and. kk .le. kmax-1) then
           do i=istart, iend
              phi(i,kk) = 0.25d0 * ( phi(i,kk-1)+
                phi(i+1,kk) +  phi(i,kk+1) + phi(i-1,kk))
           enddo
        endif
!$OMP barrier
     enddo ! k
  enddo !it
!$OMP end parallel
```

Assignment 6, Task 1: wavefront-parallel Gauss-Seidel

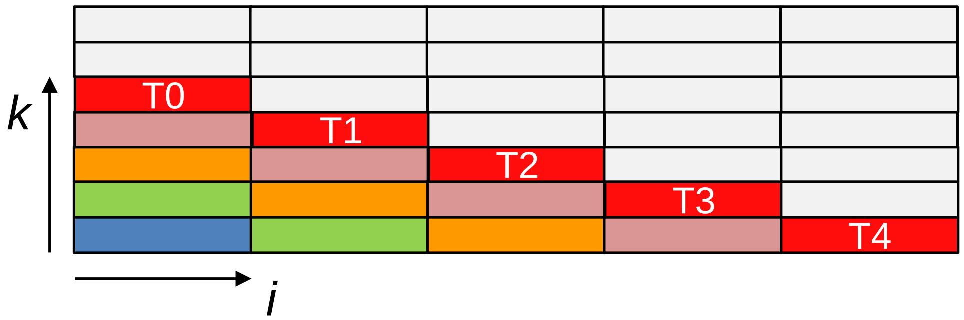


Emmy (IVB) @ 2.2 GHz

- Performance saturates just barely w/ physical cores only
- SMT helps a lot below saturation (filling pipeline bubbles) but cannot circumvent BW saturation or barrier pain
- No scaling across ccNUMA domains
  - Barrier is **much** more costly
  - First-touch placement? NUMA balancing?

Assignment 6, Task 1: wavefront-parallel Gauss-Seidel

- Which loop to parallelize upon initialization for proper ccNUMA placement?
  - Steady state (after wind-up) has fixed i-block-to-thread mapping → parallelize i loop upon init!



- But does it work for imax=8000 on Emmy?
  - No! ← 2 MiB page size on Emmy nodes (it would work with 4 KiB, sort of)
  - Last resort: static,1 parallelization of outer loop

```
#pragma omp parallel for schedule(static,1)
for(k=1; k<kmax-1; ++k)
  for(i=1; i<imax-1; ++i)
    phi[k][i]=0.;
```

Assignment 6, Task 1: wavefront-parallel Gauss-Seidel

- Bandwidth-bound performance limit on one Emmy socket:

  B_c = 16 B/LUP (best case – read & write each lattice node)
  b_S = 41 GB/s
  → b_S / B_c = 2560 MLUP/s

- Impact of 2000 cy OpenMP barrier @ 8000x8000
  Duration of one barrier-free update sweep (7998 LUPs):

  T_{bf} = (7998 LUPs / 2560 MLUP/s) * 2.2 Gcy/s = 6870 cy

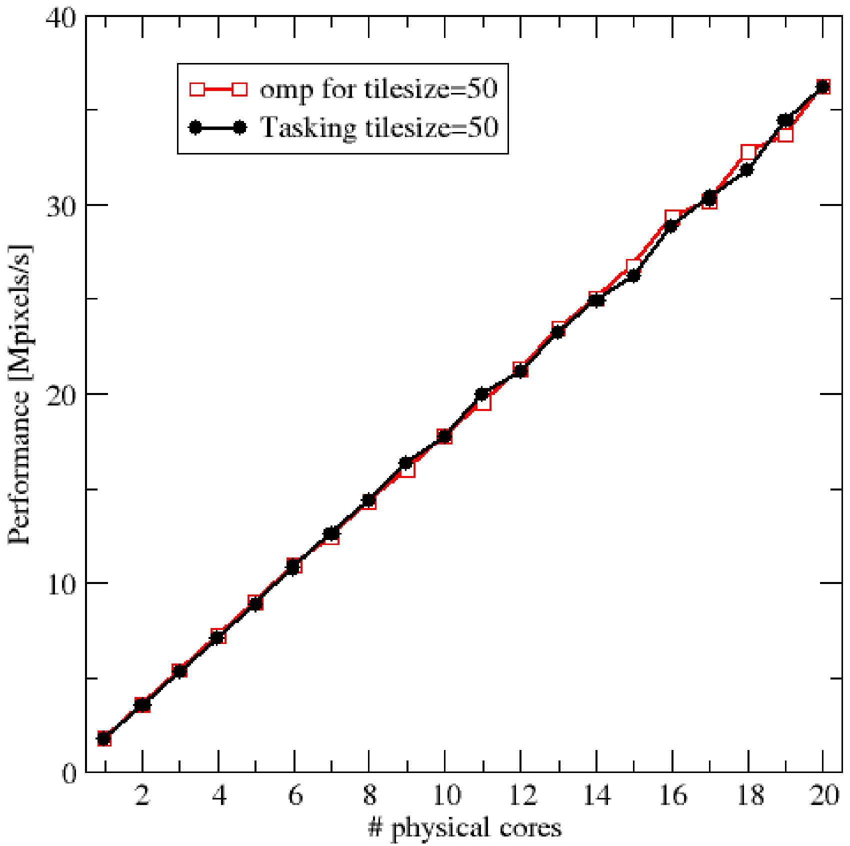
  → performance reduction by factor of
  6870/(2000+6870) = 0.77 → expected performance 1980 MLUP/s

Assignment 6, Task 2: 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)
        {
          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
```

Assignment 6, Task 2: tasking for the ray tracer

- Performance and scaling identical to for-loop parallel case (dynamic,1)



Assignment 6, Task 3: polynomial evaluation

- SIMD function **`poly_eval`** (separate compilation unit)

```
#pragma omp declare simd uniform(deg,coeff) simdlen(2)
#pragma omp declare simd uniform(deg,coeff) simdlen(4)
#pragma omp declare simd uniform(deg,coeff) simdlen(8)
#pragma omp declare simd uniform(deg,coeff) simdlen(16)
#pragma omp declare simd uniform(deg,coeff) simdlen(32)
double poly_eval(double x, int deg, const double *coeff) {
  double f=0.;
  for(int i=0; i<deg+1; ++i) {
    f = x*(f+coeff[i]);
  }
  return f;
}
```

One version for each SIMD width

Assignment 6, Task 3: polynomial evaluation

```
#pragma omp declare simd uniform(deg,coeff) simdlen(2)
//… and the other declare simd directives
double poly_eval(double, int, const double *);
...
x = new double[N]; f = new double[N];
for(int i=0; i<N; ++i) { x[i] = double(rand())/RAND_MAX/100.; f[i] = 0.; }
int iter=1;
do {
  ws = getTimeStamp();
  for(int k=0; k<iter; k++) {
    #pragma omp simd simdlen(4)
    for(int i=0; i<N; ++i) {
      f[i] = poly_eval(x[i], 10, coeff);
    }
    if(f[N/2]<-10000000.) std::cout << f << std::endl;
  }
  we = getTimeStamp();
  iter *= 2;
} while(we-ws<0.2 || iter<=2);
iter /= 2;
```

SIMD function declaration

SIMD loop

Benchmarking harness

Assignment 6, Task 3: polynomial evaluation

Work unit: one polynomial evaluation (22 flops)

Observations

- Performance is largely independent of problem size
  - Best case (simdlen=16): 6.8 cy/eval
  - → 54 cy per cache line
- CPU has SIMD width of 4, but higher speedups possible with ≤16
  - Call overhead mitigated by doing several evaluations at once
  - simdlen=32 → overhead for arg passing starts dominating