

# Lab Course: Efficient Programming of Multicore Processors and Supercomputers

Jonas Mayer, Paul Preißner, Konrad Pröll

May 28, 2017

## 3.3 OpenMP

### 3.3.1 Parallelize the Jacobi algorithm with OpenMP

```
1 #pragma omp parallel
2 {
3 #pragma omp for schedule (guided) private (unew,
4   diff) reduction (+:sum) nowait
5   for( i=1; i<sizey-1; i++ ) {
6     int ii=i*sizey;
7     int iim1=(i-1)*sizey;
8     int iip1=(i+1)*sizey;
9 #pragma ivdep
10    for( j=1; j<sizey-1; j++ ){
11      unew = 0.25 * (u[ ii+(j-1) ]+
12                    u[ ii+(j+1) ]+
13                    u[ iim1+j ]+
14                    u[ iip1+j ]);
15      diff = unew - u[ii + j];
16      utmp[ii+j] = unew;
17      sum += diff * diff;
18    }
19  }
```

#Pragma omp parallel creates a parallel region. #Pragma omp for shares the work among the threads. Each thread writes into diff and unew and reads from them, so we have to declare them as private data. Every thread adds some value into sum (which is our residuum), so we have to use reduction for it. We specify nowait because #pragma omp parallel already introduces a barrier and there is no additional value in having a barrier after #pragma omp for.

### 3.3.2 Optimize for NUMA according to first touch allocation policy

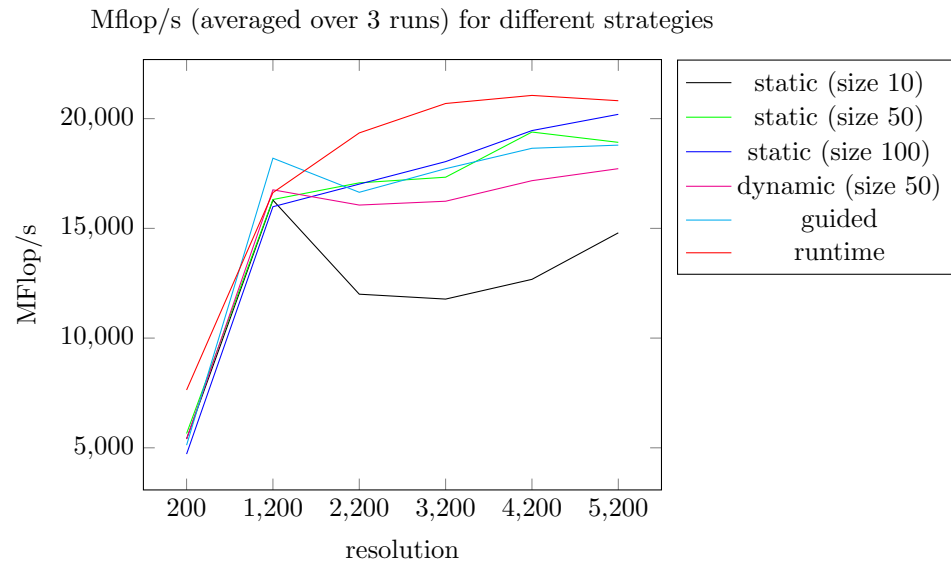
First touch allocation policy means that the operating system allocates memory upon initialization in the memory that is nearest to the allocating cpu. Thus, in a NUMA system, if we just use one cpu to initialize the arrays, the other cpu will have a higher latency when accessing its data. We can avoid this by using the very same OpenMP directives for allocating the memory as we used to calculate the new values.

```
1 int initialize( algoparam_t *param )
2 {
3     //...
4     #pragma omp parallel
5     {
6         #pragma omp for schedule (guided) nowait
7         for (i=0; i<np; i++){
8             for (j=0; j<np; j++){
9                 param->u[i*np+j]=0;
10                param->uhelp[i*np+j]=0;
11            }
12        }
13    }
14    //...
15 }
```

Source: <http://www.nersc.gov/users/computational-systems/cori/application-porting-and-performing-improving-openmp-scaling/#toc-anchor-3>

### 3.3.3 Any other optimization possibilities?

One idea was to check the performance change for the various scheduling strategies. We tested static at sizes 10, 50 and 100, dynamic at size 50, guided and runtime.



**3.3.4 Run problem size 5200 with 1, 2, 4, 8, 16, 32 threads.  
Give the speedup diagram and explain your findings.**