Introduction to
High Performance Computing

Performance of OpenMP

Performance

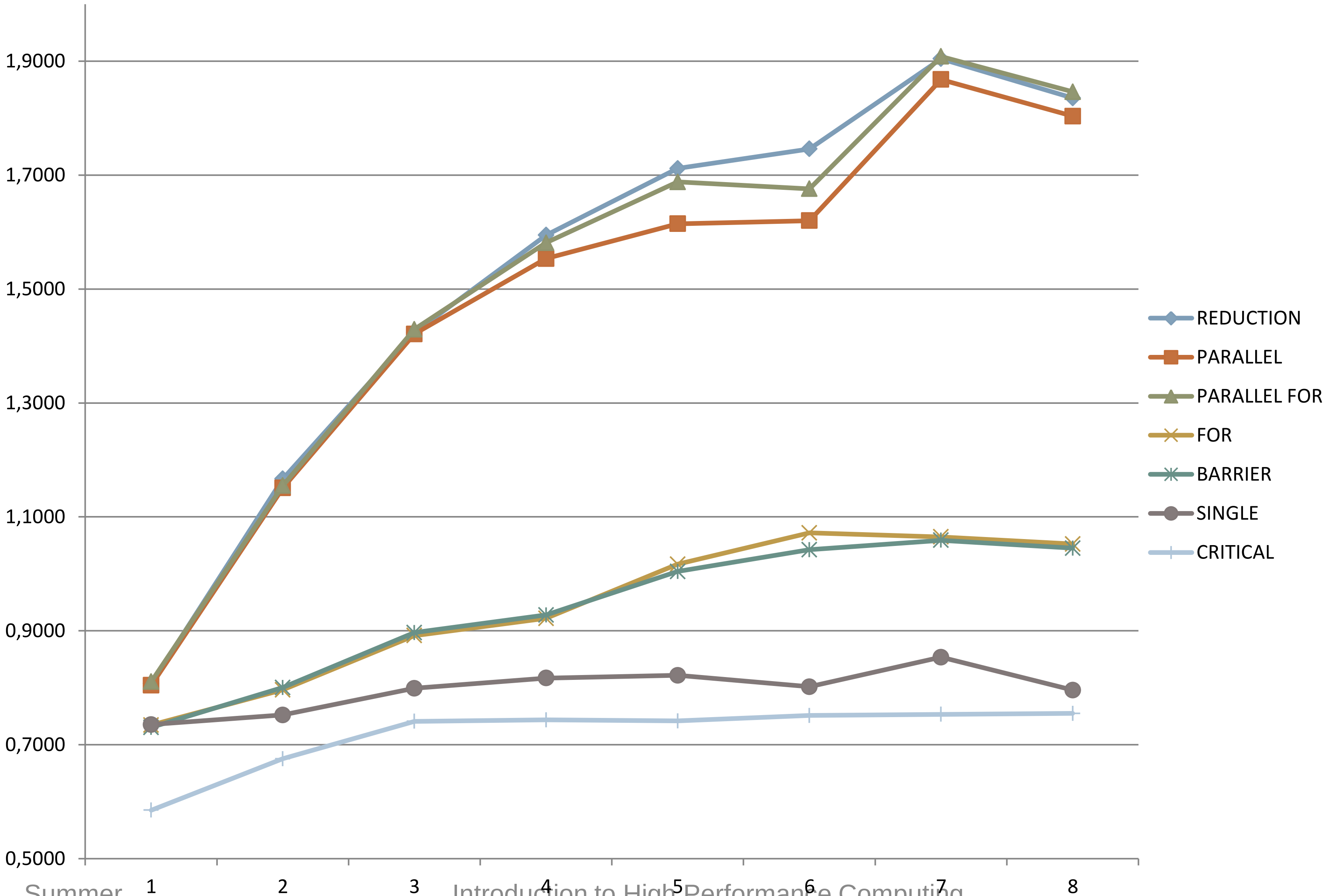
Microbenchmark - Bull, Edinburgh

https://www.epcc.ed.ac.uk/research/computing/performance-characterisation-and-benchmarking/epcc-openmp-micro-benchmark-suite

1. Measures overhead of parallel directives and primitives

2. Measures overhead of different scheduling strategies

Performance



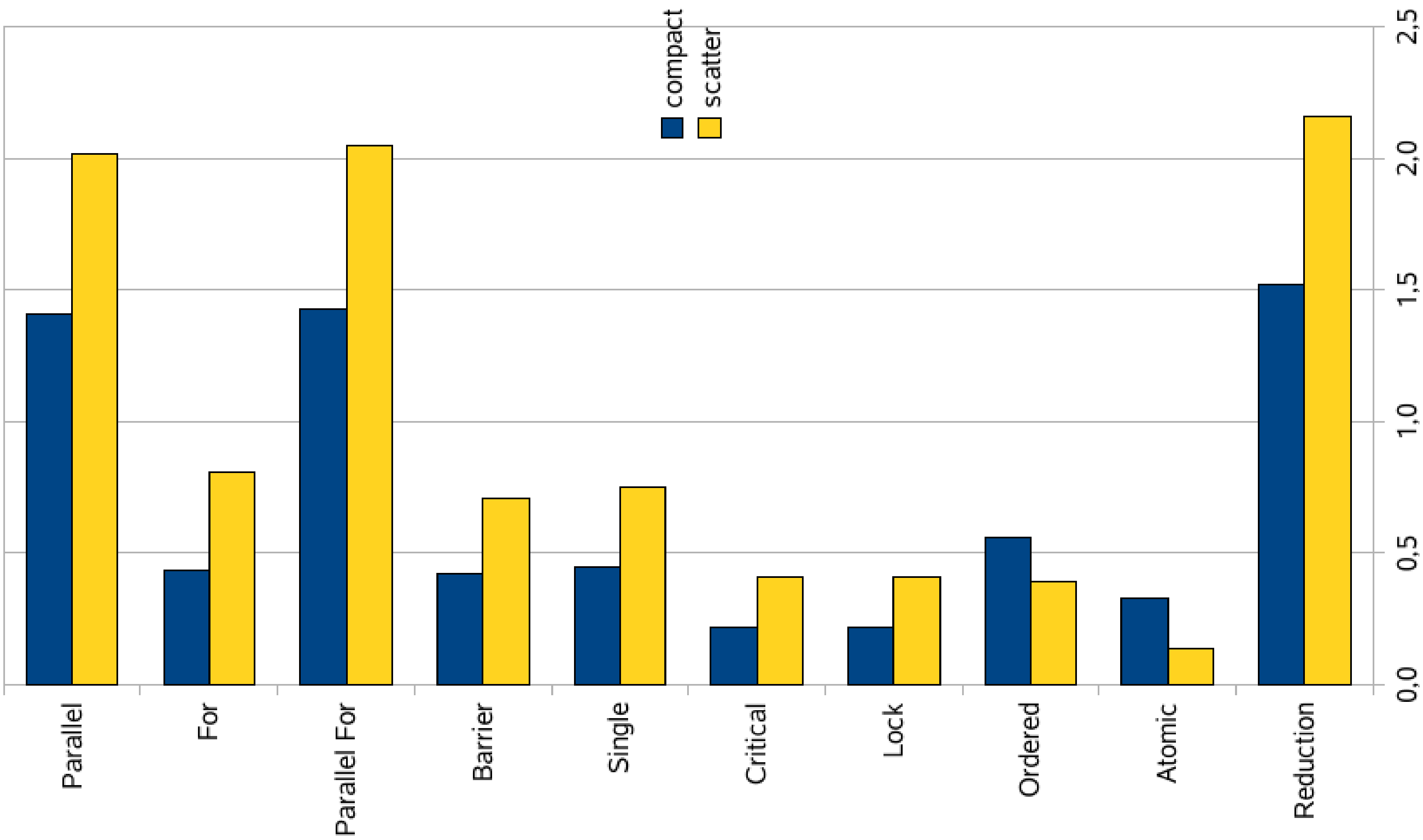
Performance

Tests were executed on 4 cores with hyperthreading (8 virtual cores)

Results:

- No extra penalty for virtual cores (exept vor barrier and for)
- The more threads, the more overhead
- Reduction is very expensive
- PARALLEL is cheaper than PARALLEL FOR, but this is cheaper than a PARALLEL and a FOR. **One loop, use PARALLEL FOR**
- **PARALLEL is expensive** – use a parallel region as long as possible
- FOR is as cheap as the barrier at its end – use NOWAIT
- SINGLE and CRITICAL are very cheap synchronizations and show only little dependency on the number of threads used

Performance



Performance

Test with 4 Threads on a 2x4 core machine.

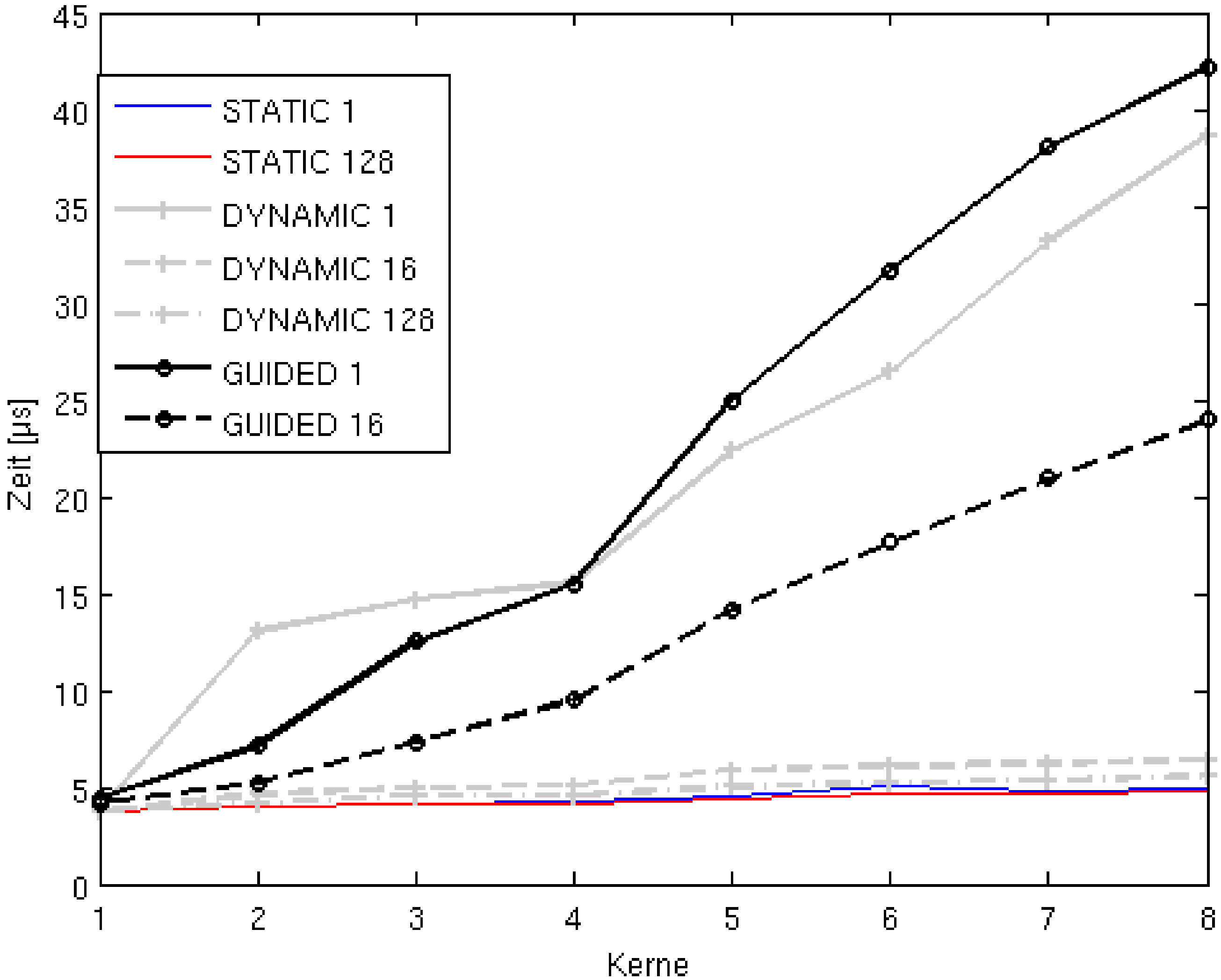
Blue: Threads are located on one socket (occupy one CPU)
Yellow: Threads are spread – two on each socket

Collocating threads is very important (LSF does this automatically if possible)

Possibilities to bind threads to cores:

- KMP_AFFINITY (Intel)
- Environment variable OMP_PROC_BIND={true, false} (a thread should not be moved by OS
- GOMP_CPU_AFFINITY in conjunction with GNU-compilers

Performance



Performance

Different scheduling strategies

Results:
•STATIC shows least overhead irrespective of chunk size
•DYNAMIC with very small chunks (1) should be avoided if possible
•GUIDED generates considerable overhead – only to be used if really necessary

False Sharing

Considering multi-core applications caches and cache coherency gain increasing importance.
The following examples shows 2 version of an alternative to a reduction. The first one with automatic variables allocated on the stack of each thread, the second one with a shared array there each thread access the corresponding array element.

Examples

```
#define Intervalle 10000
void main(int argc, char **argv) {
 int i;
 double step,x,pi,sum=0.,psum;
 step=1./Intervalle;
 /* Fork and work sharing combined */
#pragma omp parallel shared(sum)
{
#pragma omp for private(x,i,psum) nowait
for(i=0,psum=0.;i<Intervalle;i++) {
     x=(i+0.5)*step;
     psum+=4.0/(1.0+x*x);
   }
#pragma omp critical
   sum+=psum;
} /* end of parallel region */
 pi=step*sum;
}
```

Examples

```
#define Intervalle 10000
void main(int argc, char **argv) {
 int i;
 double step,x,pi,sum=0.,psum[max_threads];
 step=1./Intervalle;
 /* Fork and work sharing combined */
#pragma omp parallel shared(sum)
{
me=omp_get_thread_num();
#pragma omp for private(x,i)
for(i=0,psum[me]=0.;i<Intervalle;i++) {
     x=(i+0.5)*step;
     psum[me]+=4.0/(1.0+x*x);
   }
#pragma omp single
   for(i=0;i<omp_get_team_size();i++) sum+=psum[i];
} /* end of parallel region */
 pi=step*sum;
}
```

False Sharing

- Caches are organized in cache lines – usually one line 2 words.

- psum[4] may occupy two cache lines when loaded to a core.
- Is psum[1] changed, the cache coherency protocol of the system invalidates the cache line in all other caches.
- Thread 0 has to reload psum[0] (and the complete cache line).
- Is one of the values changed, the cache line is invalidated ……

Don't use small arrays for writing by different threads.

Einführung in das Hochleistungsrechnen

Introduction to High Performance Computing

VIELEN DANK
THANK YOU