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 3 discussion

High Performance
Computing

Assignment 3, Task 1

“π by integration”

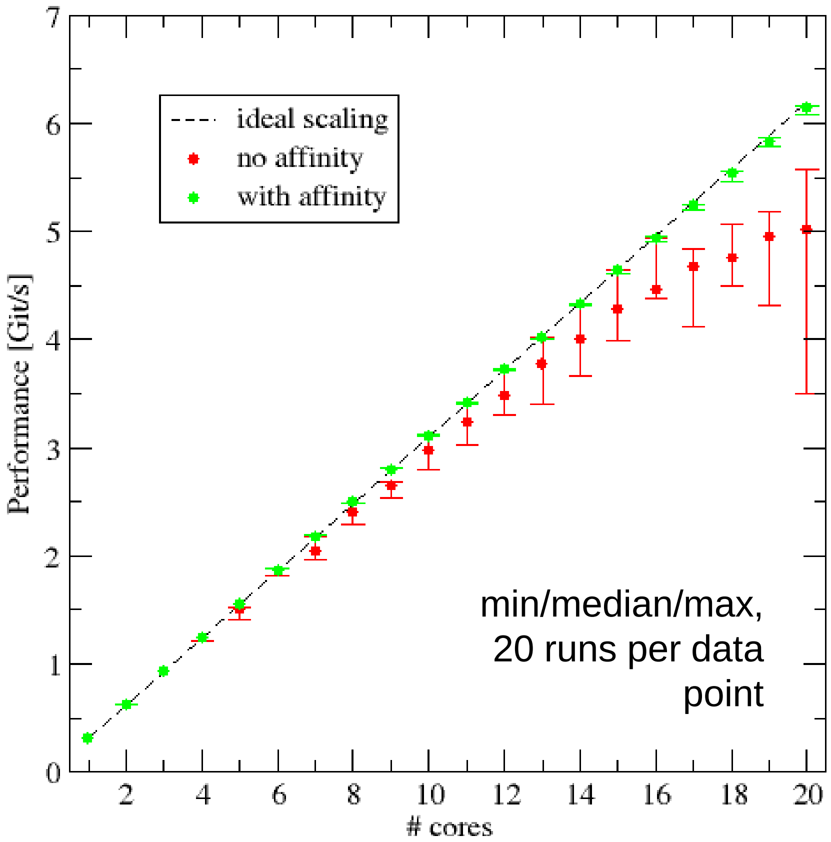
```
#include "timing.h"

int main() {
 double wcs,wce;
 int i,n = 2000000000;

 double  delta_x,x,sum,Pi;

 wcs = getTimeStamp();
 sum = 0.;
#pragma omp parallel for private(x) reduction(+:sum)
 for(i=0; i<n; ++i) {
    x = (i+0.5)*delta_x;
    sum = sum + 4.0 / ( 1.0 + x * x);
 }
 Pi = sum * delta_x;
 wce = getTimeStamp();
 printf("Pi=%.15lf in %.3lf s -> perf = %.3lf Git/s\n",
         Pi,wce-wcs,n/(wce-wcs)*1.e-9);
 return 0;
}
```

Assignment 3, Task 1

- Performance fluctuates if affinity is not used
  - $\varepsilon \lesssim 0.9$, median $\varepsilon \approx 0.81$ @ 20 cores
- Proper binding:

```
OMP_NUM_THREADS=$t \
OMP_PLACES=cores \
OMP_PROC_BIND=close \
./a.out
```

→ fluctuations practically gone, close to perfect scaling ($\varepsilon \approx 0.99$)

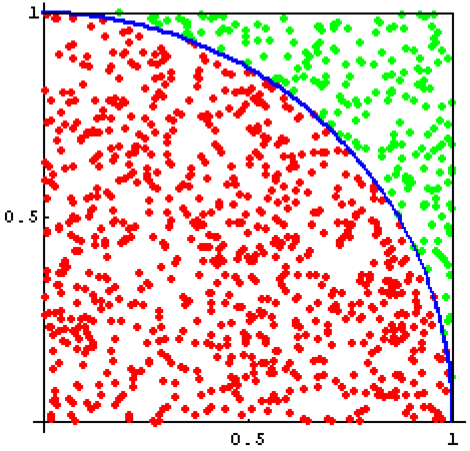


min/median/max, 20 runs per data point

Assignment 3, Task 2

"π by Monte Carlo"

Naïve serial version

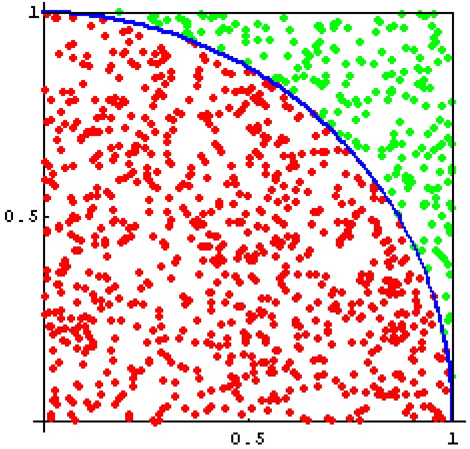


```
wcstart = getTimeStamp();
seed = 2; // arbitrary
for(i=0; i<nn; ++i) {
  double x = rand_r(&seed)/(double)RAND_MAX;
  double y = rand_r(&seed)/(double)RAND_MAX;
  if(sqrt(x*x+y*y) <1.0) ++count;
}
}
pi = 4.0*(double)count/nn;
wcend = getTimeStamp();
printf("Time: %.3lf sec, accuracy: %.12lf\n",
        wcend-wcstart,fabs(M_PI-pi)/M_PI);
```

Assignment 3, Task 2

“π by Monte Carlo”

Optimized parallel version



$\Delta\pi = 1.2 \times 10^{-5}$

@ 20 cores, runtime = 1sec,

nn = 10^9

```
 wcstart = getTimeStamp();
 seed = 2;
 double rmp = 1./RAND_MAX;
#pragma omp parallel private(seed)
{
#ifdef _OPENMP
 seed = omp_get_thread_num()+2;
#endif
 #pragma omp for reduction(+:count)
 for(i=0; i<nn; ++i) {
   double x = rand_r(&seed)*rmp;
   double y = rand_r(&seed)*rmp;
   if(x*x+y*y <1.0) ++count;
 }
}
 pi = 4.0*(double)count/nn;
 wcend = getTimeStamp();
 printf("Time: %.3lf sec, accuracy: %.12lf\n",
         wcend-wcstart,fabs(M_PI-pi)/M_PI);
```

Avoid expensive FP divides

Per-thread seed

Avoid expensive sqrt