NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Programming Techniques for Supercomputers Tutorial

Erlangen National High Performance Computing Center

Department of Computer Science

FAU Erlangen-Nürnberg

Sommersemester 2024

Assignment 9 – Task 1

Claim: “Slow code scales better”

$$T(1) = T_s + T_p,\ \ T(N) = T_s + \frac{T_p}{N} + T_c(N) \ \ \text{(and } T_c(1) = 0\text{)}$$

a) $$S(N) = \frac{T(1)}{T(N)} = \frac{T_s + T_p}{T_s + \frac{T_p}{N} + T_c(N)} = \frac{1}{s + \frac{p}{N} + c(N)}$$

with the dimensionless quantities

$$s = \frac{T_s}{T_s+T_p}, \quad p = \frac{T_p}{T_s+T_p}, \quad c(N) = \frac{T_c(N)}{T_s+T_p}$$

Assignment 9 – Task 1

b) Slowdown factor $\sigma < 1$: $s \rightarrow \frac{s}{\sigma},\ p \rightarrow \frac{p}{\sigma}$, $\mathrm{c}(N)$ unchanged

$$S(N,\sigma) = \frac{1/\sigma}{s/\sigma + \frac{p}{N\sigma} + c(N)} = \frac{1}{s + \frac{p}{N} + \sigma c(N)}$$

→ communication overhead gets damped (reduced) by a factor of $\sigma < 1$

→ $S(N, \sigma < 1) > S(N, \sigma = 1)$ as long as $\mathrm{c}(N) > 0$

Prerequisite: There must be a non-negligible communication overhead for this "stunt" to work!

Assignment 9 – Task 1

c) Assume no serial fraction: $s = 0$
Assume linear overhead (e.g., via unoptimized barrier) : $c(N) = \alpha N$

Performance $P(N,\sigma) = \sigma P(1) S(N,\sigma) = \frac{\sigma P(1)}{\frac{p}{N} + \sigma\alpha N}$,

1. Large N in denominator → 0
2. The σ term cancels out

so $\lim_{N \gg 1} P(N,\sigma) = \frac{P(1)}{\alpha N}$.

This does not depend on σ, because the runtime is dominated by the barrier overhead for very large N: Execution takes no time (p/N) and the overhead grows without bounds, albeit slowly (αN)

Assignment 9 – Task 2

```
double x[100],y[100],tmp;
int i;
double work1(int);
// initialization code etc. omitted
#pragma omp parallel
{
  #pragma omp for
  for(i=0; i<100; i++) {
    tmp=work1(i);
    x[i]=x[i]+tmp;
  }
  #pragma omp for
  for(i=1; i<100; i++) {
    y[i]=x[i-1]*y[i];
  }
}
```

`tmp` declared outside of parallel region

Yet it is used by each thread without a **`private`** clause, which leads to a race condition

Assignment 9 – Task 2

```
double x[100],y[100],tmp;
int i;
double work1(int);
// initialization code etc. omitted
#pragma omp parallel private(tmp)
{
  #pragma omp for
  for(i=0; i<100; i++) {
    tmp=work1(i);
    x[i]=x[i]+tmp;
  }
  #pragma omp for
  for(i=1; i<100; i++) {
    y[i]=x[i-1]*y[i];
  }
}
```

```
double x[100],y[100];
int i;
double work1(int);
// initialization code etc. omitted
#pragma omp parallel
{
  #pragma omp for
  for(i=0; i<100; i++) {
    double tmp=work1(i);
    x[i]=x[i]+tmp;
  }
  #pragma omp for
  for(i=1; i<100; i++) {
    y[i]=x[i-1]*y[i];
  }
}
```

Assignment 9 – Task 3

```
unsigned int seed = 1;
long hist[16];
for(int i=0; i<16; ++i)
  hist[i]=0;
wcstart = getTimeStamp();
for(long i=0; i<2000000000; ++i)
{
  hist[rand_r(&seed) & 15]++;
}
wcend = getTimeStamp();
for(int i=0; i<16; ++i) {
  cout << "hist[" << i << "]=" <<
hist[i] << endl;
}
cout << "Time: " << wcend-wcstart
<< " sec" << endl;
```

Parallelize first for loop to pin your team of threads

Parallelize loop, where each thread gets it's own random seed

Do a sum reduction of some kind (either by hand or with a **`reduction`** clause)

Assignment 9 – Task 3

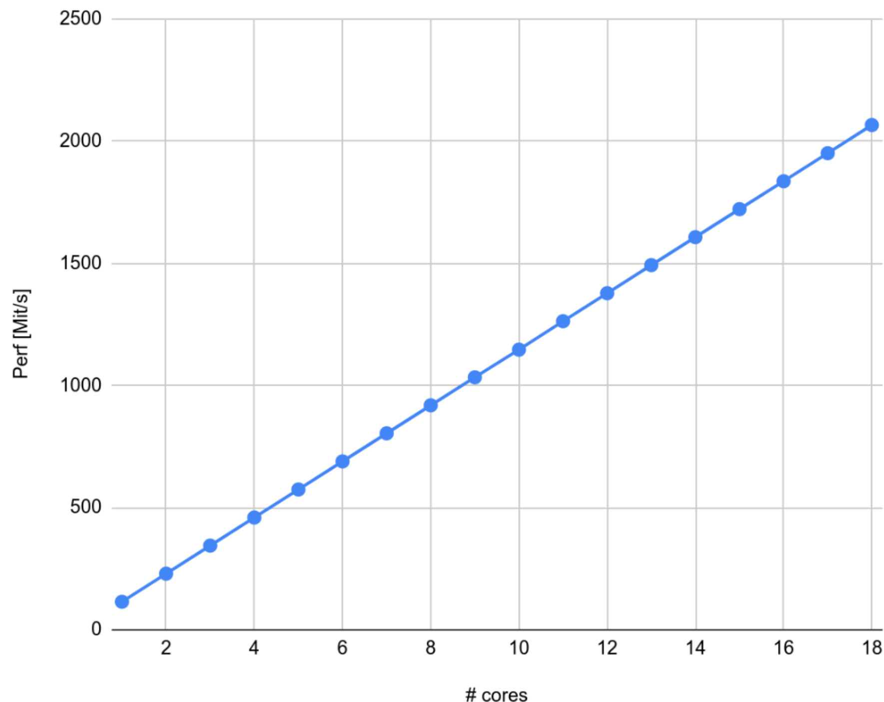
```
int i,len=16;
long hist[16],j;
double wcstart,wcend,ct;

// This loop is parallel to reduce the overhead for the next
#pragma omp parallel for
for(i=0; i<16; ++i)
  hist[i]=0;

wcstart = getTimeStamp();
#pragma omp parallel
{
  unsigned int seed = omp_get_thread_num();
#pragma omp for reduction(+:hist[0:len])
  for(j=0; j<2000000000; ++j) {
    hist[rand_r(&seed) & 15]++;
  }
}
wcend = getTimeStamp();
```

Assignment 9 – Task 3

- Should be near-perfect scaling (99.8% efficiency with 18 cores)
- Typical blunders
  - Not using separate seeds
    - Variables or values!
  - Using a critical region to protect the update for the histogram values
  - Not using a warmup loop to pin the thread team
- General Rule: "Privatizing is better than synchronizing!"



Assignment 9 – Task 3

```
do {
  wcstart = getTimeStamp();
  for(int k=0; k<NITER; ++k) {
  #pragma omp parallel
  {
      unsigned int seed = omp_get_thread_num();
      #pragma omp for reduction(+:hist[0:len])
      for(j=0; j<n; ++j) {
          hist[rand_r(&seed) & 0xf]++;
      }
  }
  } wcend = getTimeStamp();
  NITER *= 2;
 } while(wcend-wcstart < 0.2);
 NITER /= 2;
```

Assignment 9 – Task 3

- With $n = 2 \cdot 10^9$, you should get $17.4\ cy/it$ with a single thread
  - The code scales perfectly with a speedup of almost exactly 18 on 18 cores.

- With $n = 2 \cdot 10^4$, you should only get a speedup of about 10 on 18 cores
  - The computation itself takes $2 \times 10^4 \times 17.4\ cy\ /\ 18\ =\ 19300\ cy$
  - The runtime, however is 17.5 µs, which is 35000 cy

- Therefore, the OpenMP overhead amounts to

$$35000 - 19300\ =\ 15700\ cy \text{ at 18 threads}$$