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 2025

Assignment 0 – Task 1

- Code

```
double sum, x, delta_x, S, E;
int N = 10; // initial number of slices
do {
  S = getTimeStamp();
  delta_x = 1./N;
  sum = 0.0;
  for (int i=0; i<N; i++) {
    x = (i + 0.5) * delta_x;
    sum = sum + 1.0 / (1.0 + x));
  }
  double ln = sum * delta_x;
  E = getTimeStamp(); // T = wce-wcs
  if(E-S > 1e-7) break;
  N *= 2;
} while (1);
```

Assignment 0 – Task 1

- Code

```
int main (int argc, char**argv) {
  double f = 2.4e9;   // clock frequency in cy/s
  double sum, x, delta_x, S, E;
  int N = 10; // initial number of slices
  do {
    S = getTimeStamp();
    delta_x = 1./N;
    sum = 0.0;
    for (int i=0; i<N; i++) {
      x = (i + 0.5) * delta_x;
      sum = sum + 1.0 / (1.0 + x));
    }
    double ln = sum * delta_x;
    E = getTimeStamp(); // T = wce-wcs
    printf("ln(2)=%.15lf in %.3le s -> %.2lf cy/it (N = %d)\n",
                  ln, E-S, (E-S)/N*f, N);
    if(E-S > 1e-7) break;
    N *= 2;
  } while (1);
  return 0;
}
```

Assignment 0 – Task 1

- Number of cycles to execute one loop iteration:

$$c = \frac{T}{n} \times f \left[\frac{\text{cy}}{\text{It.}} = \frac{\text{s}}{\text{It.}} \times \frac{\text{cy}}{\text{s}}\right],$$

  where f is the clock speed, n is the number of slices, and T is the runtime of the whole loop.

- Compilation:

```
$ module load intel
$ icx -O3 -xHost integral.c timing.c
```

- Important: Fix clock speed when running binary:

```
$ srun --cpu-freq=2400000-2400000:performance ./a.out
```

Assignment 0 – Task 2

```
ln(2)=0.692835360409960 in 1.470e-07 s -> 35.28 cy/it (N = 10)
ln(2)=0.693069098225587 in 1.040e-07 s -> 12.48 cy/it (N = 20)
ln(2)=0.693127651979310 in 1.020e-07 s -> 6.12 cy/it (N = 40)
ln(2)=0.693142297914324 in 1.100e-07 s -> 3.30 cy/it (N = 80)
ln(2)=0.693145959867251 in 1.760e-07 s -> 2.64 cy/it (N = 160)
ln(2)=0.693146875384816 in 3.100e-07 s -> 2.33 cy/it (N = 320)
ln(2)=0.693147104266041 in 5.780e-07 s -> 2.17 cy/it (N = 640)
ln(2)=0.693147161486461 in 1.110e-06 s -> 2.08 cy/it (N = 1280)
ln(2)=0.693147175791574 in 2.181e-06 s -> 2.04 cy/it (N = 2560)
ln(2)=0.693147179367853 in 4.318e-06 s -> 2.02 cy/it (N = 5120)
ln(2)=0.693147180261922 in 8.596e-06 s -> 2.01 cy/it (N = 10240)
ln(2)=0.693147180485440 in 1.715e-05 s -> 2.01 cy/it (N = 20480)
ln(2)=0.693147180541319 in 3.426e-05 s -> 2.01 cy/it (N = 40960)
ln(2)=0.693147180555288 in 6.847e-05 s -> 2.01 cy/it (N = 81920)
ln(2)=0.693147180558782 in 1.369e-04 s -> 2.01 cy/it (N = 163840)
ln(2)=0.693147180559654 in 2.738e-04 s -> 2.01 cy/it (N = 327680)
ln(2)=0.693147180559873 in 5.511e-04 s -> 2.02 cy/it (N = 655360)
```

Assignment 0 – Task 3

- Loop body:

```
x = (i+0.5)*delta_x;
sum = sum + 1.0 / (1.0 + x));
```

→ 5 flops (3 ADD, 1 MULT, 1 DIV). Or not???

- Possible performance metrics
  - Flop/s → not portable
    - compiler might transform code
    - int→float conversion might count or not
    - DIV might not even be an instruction but comprise several flops (depends if hardware unit available)
  - $1/T$ → Usually OK but varies with loop length in a trivial way
  - n/T → probably best metric overall in this case (it/s, it/cy)
  - Code performance: $P \approx 0.5$ it/cy

Assignment 0 – Task 4

- Not fixing the clock frequency but using the performance governor:

```
$ srun --cpu-freq=performance ./a.out
0.0962s instead of 0.1404s at N=167772160
```

→ so the actual clock frequency is

$$\frac{0.1404}{0.0962} \times 2.4\ \text{GHz} \approx 3.5\ \text{GHz}$$