Programming Techniques for Supercomputers Tutorial

Erlangen National High Performance Computing Center

Department of Computer Science

FAU Erlangen-Nürnberg

Sommersemester 2024

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 div.c timing.c
```

- Important: Fix clock speed when running binary:

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

Assignment 0 – Task 1

- Code

```
#include <stdio.h>
#include “timing.h”

int main (int argc, char**argv) {
  double f = 2.4e9;   // clock frequency in cy/s
  int n = 1000000000; // # of slices
  double delta_x = 1./n,x,sum=0.,wcs,wce,Pi;
  wcs = getTimeStamp();
  for (int i=0; i < n; i++) {
    x = (i+0.5)*delta_x;
    sum += 4.0 * sqrt(1.0 - x * x));
  }
  wce = getTimeStamp(); // T = wce-wcs
  Pi = sum * delta_x;
  printf(“Pi=%.15lf in %.3lf s -> %.2lf cy/it\n”,Pi,wce-wcs,(wce-wcs)/n*f);
  return 0;
}
```

- Run:

```
$ srun --cpu-freq=2400000-2400000 ./a.out
Pi=3.141592653589493 in 1.255 s -> 3.01 cy/it
```

Assignment 0 – Task 2

- Loop body:

```
x = (i+0.5)*delta_x;
sum += 4.0 * sqrt(1.0 - x * x));
```

→ 7 flops (2 ADD, 1 SUB, 3 MULT, 1 SQRT). Or not???

- Possible performance metrics
  - Flop/s → not portable
    - compiler might transform code
    - int→float conversion might count or not
    - SQRT might not even be an instruction but comprise several flops
  - $1/T$ → 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: $P \approx 0.33\ \mathrm{it/cy}$

Assignment 0 – Task 3

- Single precision code:

```
double f = 2.4e9;   // clock frequency in cy/s
int n = 1000000000; // # of slices
float delta_x = 1.f/n,x,sum=0.f,wcs,wce,Pi;
wcs = getTimeStamp();
for (int i=0; i < n; i++) {
  x = (i+0.5f)*delta_x;
  sum += 4.0f * sqrtf(1.0f - x * x));
}
wce = getTimeStamp(); // T = wce-wcs
Pi = sum * delta_x;
printf("Pi=%.7f in %.3lf s -> %.2lf cy/it\n",Pi,wce-wcs,(wce-wcs)/n*f);
```

- Take care to use "**f**" qualifier for all float constants
  - Language has strict rules about type conversion
  - Compiler may be forced to generate code with runtime conversions for inconsistent types

Assignment 0 – Task 3

- Result

```
$ srun --cpu-freq=2400000-2400000 ./a.out
Pi=2.1474836 in 0.318 s -> 0.76 cy/it
```

- 4 times faster than DP code
  → SP SQRT appears to be much faster than DP

- Result is not at all π
  - Summing 10^9 numbers, all between 1 and 0
  - Float type has only ~7 mantissa digits
    - → Massive loss of accuracy as soon as the sum gets > 10^7
    - → Sum is much too small

Assignment 0 – Task 4

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

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

`0.860 s` instead of `1.255 s`

→ so the actual clock frequency is

$$\frac{1.255}{0.860} \times 2.4\ \text{GHz} \approx 3.5\ \text{GHz}$$