Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

Practical 1

- Some facts and figures:
  - no. of submissions by the deadline: 116 (86% of class size (135) according to SITS)
  - no. that get full marks (provisionally): 99 (85% approx of submissions)
  - no. that may fall short (provisionally): 17 (15% approx of submissions)

Integration once more

- Consider integrating f(x) = 16 - x^2 between 0 and 4
- The analytic solution is 128/3 or 42.666…(optional exercise: check my math)
- We can produce a sequential numerical solution by splitting the range into 1000 pieces. (**`seq-integrate.c`**)
- Running this code gives the answer: 42.666656
  - 4Ghz Intel Core i7, 32GB, macOS Mojave, clang compiler

integration preamble

```c
#include <stdio.h>

#define NUM_SLICES 1000
#define H 4.0/NUM_SLICES

double answer;

double f(double x) { return (16.0 - x*x) ; }

double trapezoid(double a, double b) { return H*(f(a)+f(b))/2.0; }
```

seq-integrate.c body

```
int main (int argc, const char * argv[]) {
  answer    = 0.0;
  double a ;
  double b ;
  int i ;
  for (i=0;i<NUM_SLICES;i++) {
    a = (int)i * H ;
    b = a + H;
    answer += trapezoid(a,b);
   }
  printf("\nAnswer is %f\n",answer);
    return 0;
}
```

Integration with threads (attempt 1)

- We might use 1000 threads, but gamble on not needing a mutex (`nom-integrate.c`)

nom-integrate.c thread

```
void *IntegratePart(void *i) {

  double a,b,area;

  a = (int)i * H ;
  b = a + H;
  area = trapezoid(a,b);

  // critical section with no mutex !!!!!
  answer=answer+area;

    pthread_exit(NULL);
}
```

nom-integrate body

```
int main (int argc, const char * argv[]) {
    pthread_t threads[NUM_SLICES];
    long rc,t;

  answer = 0.0;

    for (t=0;t<NUM_SLICES;t++) {
        rc = pthread_create(&threads[t],NULL,IntegratePart,(void *)t);
        if (rc) {
            printf("ERROR return code from pthread_create(): %ld\n",rc);
            exit(-1);
        }
    }
    for(t=0;t<NUM_SLICES;t++) {
        pthread_join( threads[t], NULL);
    }
  printf("%f\n",answer);
    return 0;
}
```

Integration with threads (attempt 1)

- When I ran this I got 42.666656, most of the time
  - (e.g. 46 times out of 52)
- However sometimes I got an answer in the range 42.6026 .. 42.6038 !
  - (e.g. 6 times of of 52)
  - Approximately 11% of the time I got an answer that underestimates by -0.15%
    - on my office iMac, at least.
- Hard to explain this result. May differ on different machines.
- Let's give it a go!

Integration with mutex

- We finally decide to use a mutex (mutex-integrate.c)
- We get the same results as the sequential code.

mutex-integrate (mutex & thread)

```
pthread_mutex_t  mutex = PTHREAD_MUTEX_INITIALIZER;

void *IntegratePart(void *i) {

  double a,b,area;
  int rc;

  a = (int)i * H ;
  b = a + H;
  area = trapezoid(a,b);

  // critical section with mutex
  rc = pthread_mutex_lock(&mutex);
    checkResults("pthread_mutex_lock()\n", rc);

  answer=answer+area;

  rc = pthread_mutex_unlock(&mutex);
    checkResults("pthread_mutex_lock()\n",rc);

    pthread_exit(NULL);
}
```

Body is same as for nom-integrate

integration example wrap-up

- We can parallelise integration, but mutexes are required
- If we omit them, then errors may occur
- The worst case is when such errors are rare
  - such errors may not be revealed by testing
  - e.g. Mars Rover flash memory bug.
- What about speed?