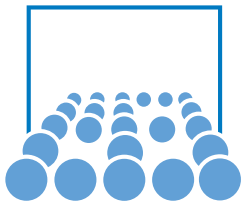
Technische Universität München

Assignment 2: Parallel Debugging with TotalView

Programming of Super Computers

Friedrich Menhorn, Benjamin Rüth, Erik Wannerberg
Team 12

December 3, 2015



Contents

Race Condition

Definition

- From:
  `stackoverflow.com/questions/34510/what-is-a-race-condition`

  ”A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don’t know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are ”racing” to access/change the data.”

Race Condition

Test Code

In **RaceCondition.cpp**

```
#include <iostream>
#include <omp.h>
int main(){
    int write,read;
    #pragma omp parallel shared(write) private(read)
    {
        write = omp_get_thread_num();
        if(omp_get_thread_num()==3){
            std::cout << "Thread: " << omp_get_thread_num() <<
            " Write: " << write << std::endl;
        }
        read = write;
        if(omp_get_thread_num()==3){
            std::cout << "Thread: " << omp_get_thread_num() <<
            " Read: " << read << std::endl;
        }
    }
    return 0;
}
```

Different outputs due to undefined order of access:

Output 1:

```
$ ./RaceCondition
Thread: 3 Write: 3
Thread: 3 Read: 3
```

Output 2:

```
$ ./RaceCondition
Thread: 3 Write: 1
Thread: 3 Read: 1
```

Output 3:

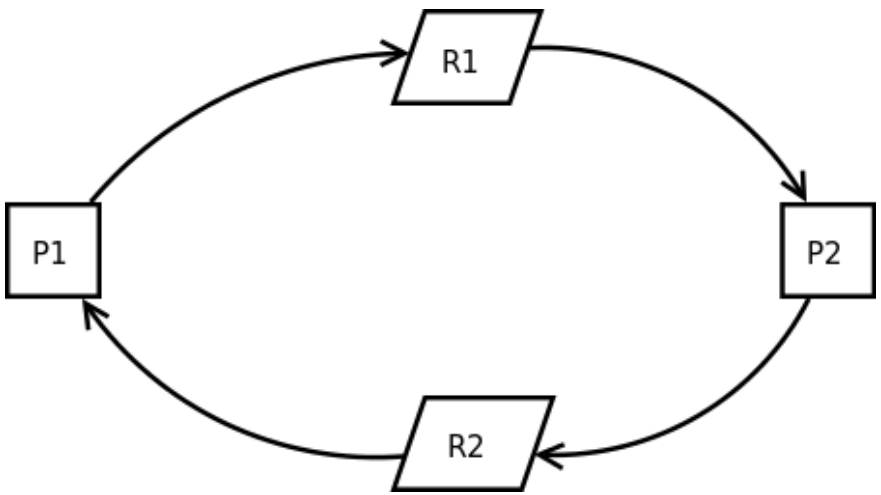
```
$ ./RaceCondition
Thread: 3 Write: 3
Thread: 3 Read: 0
```

Deadlock

Definition

- From: `https://en.wikipedia.org/wiki/Deadlock`

  ”In concurrent programming, a deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.”



Heisenbug

Definition

- From: `http://www.webopedia.com/TERM/H/heisenbug.html`

  ”In computer programming, heisenbug is a classification of an unusual software bug that disappears or alters its behavior when an attempt to isolate it is made.[...]”

- From: `https://en.wikipedia.org/wiki/Heisenbug`

  ”One common example of a heisenbug is a bug that appears when the program is compiled with an optimizing compiler, but not when the same program is compiled without optimization.[...]”

Floating point comparison

Test Code

In **FPComparison.cpp**

```
#include <iostream>
int main(){
    float a = 0.0;
    float b = 0.0;
    b = b - 0.1;
    b = b + 0.1;
    std::cout << "a: " << a << " b: " << b << std::endl;
    std::cout << "Comparison: " << (a==b) << std::endl;
    return 0;
}
```

Output:

```
$ ./FPComparison
a: 0 b: -1.49012e-09
Comparison: 0
```

Zero and signed zero

Definition

- From: `http://www.johndcook.com/blog/2010/06/15/why-computers-have-signed-zero/`
  "[...] computers have two versions of 0: positive zero and negative zero. Most of the time the distinction between +0 and -0 doesn't matter, but once in a while signed versions of zero come in handy. [...]"
- +0 and -0 to distinguish between the direction of underflowing to zero
  "[...] The IEEE floating point standard says 1/+0 should be +infinity and 1/-0 should be -infinity.[...]"

Test Code:

```
int main()
{
    double x = 1e-200;
    double y = 1e-200 * x;
    printf("Reciprocal␣of␣+0:␣%gn", 1/y);
    y = -1e-200*x;
    printf("Reciprocal␣of␣-0:␣%gn", 1/y);
}
```

Output:

```
Reciprocal of +0: inf
Reciprocal of -0: -inf
```

(Catastrophic) Cancellation

Test Code

- From `docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html`
- Calculate $b^2 - 4ac$ in $x_{1/2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- With floating point precision (here $\approx_{fp}$) to **first digit!**
- Example:
  - Values: $b = 3.34$; $a = 1.22$; $c = 2.28$
  - In one step: $b^2 - 4ac = 0.0292 \approx_{fp} 0.0$
  - In single steps:

$$\begin{aligned} y &:= b^2 = 11.1556 \approx_{fp} 11.2 \\ z &:= 4 * a * c = 11.1264 \approx_{fp} 11.1 \\ y - z &= 11.2 - 11.1 = 0.1 \end{aligned} \tag{1}$$

  - Error:

$$\begin{aligned} \text{One step:} \quad & |0.0292 - 0.0| = 0.0292 \\ \text{Single steps:} \quad & |0.0292 - 0.1| = 0.0708 \end{aligned} \tag{2}$$

Amplification and error propogation

Test Code

In **ErrorPropagation.cpp**

```
#include <iostream> #include <iomanip> #include <math.h>
int main(){
    float x = 0.0;
    float constant = 0.1;
    float exact;
    int curPow = 1;
    for(int i = 1; i <= 10000000; i++){
        x = x + constant;
        if( log10(i)==curPow){
            exact = i*constant;
            std::cout <<std::setprecision(8) << "# i: "<< i <<  " # x: " << x <<
            " Exact: " << exact << " Diff: " << exact-x << std::endl;
            curPow++;
        }
    }
    return 0;
}
```

Output:

```
$ ./ErrorPropagation
# i: 10 # x: 1.0000001 Exact: 1 Diff: -1.1920929e-07
# i: 100 # x: 10.000002 Exact: 10 Diff: -1.9073486e-06
...
# i: 100000 # x: 9998.5566 Exact: 10000 Diff: 1.4433594
# i: 1000000 # x: 100958.34 Exact: 100000 Diff: -958.34375
# i: 10000000 # x: 1087937 Exact: 1000000 Diff: -87937
```

Amplification and error propogation

Definition

- From: `http://floating-point-gui.de/errors/propagation/`

  ”While the errors in single floating-point numbers are very small, even simple calculations on them can contain pitfalls that increase the error in the result way beyond just having the individual errors “add up”.

  In general:

  - Multiplication and division are “safe” operations
  - Addition and subtraction are dangerous, because when numbers of different magnitudes are involved, digits of the smaller-magnitude number are lost.
  - This loss of digits can be inevitable and benign (when the lost digits also insignificant for the final result) or catastrophic (when the loss is magnified and distorts the result strongly).
  - The more calculations are done (especially when they form an iterative algorithm) the more important it is to consider this kind of problem.
  - [...] ”

Error exclusiveness to parallel programming

Notation:

- ✓ : Problem does arise in this field
- ✗ : Problem does not arise in this field

	Sequential	Parallel
FP arithmetic	✓	✓
Heisenbug	✓	✓
Race Condition	✗	✓
Deadlock	✗	✓

Task 4

dummy

dummy

dummy

dummy

dummy

dummy

dummy

dummy

Task 5

dummy

dummy

Task 6

dummy

dummy