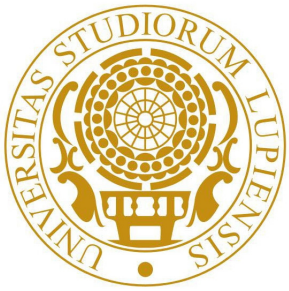
UNIVERSITÀ DEL SALENTO



Facoltà di Ingegneria

Corso di Laurea Magistrale in Computer Engineering

Parallel algorithms Project

Parallel root finding

Professor: **Massimo Cafaro**

Student: **Davide Basile**

Academic year 2020/2021

Contents

Introduction

The problem of computation of roots in a polynomial function is a very hard problem, there no exist a real formula that let us to compute the exact value of the roots except for some special kind of polynomial like quadratic ones as, $ax^2 + bx + c = 0$ some of third grade and for one kind of fourth grade, where the exact solution are known as

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

So finding roots about higher polynomial or transcendental polynomials like

$$x - \arctan x = 0$$

could be more difficult.
There exist different kind of algorithm that compute a root of the equation, all of this can handle only one root each time, all of this algorithms just move around the variable axis and guess the solution.

The Algorithm

We have started to analyze different sequential algorithms, this algorithms analyze a little segment of input where possibly the function change its sign.

Sequential algorithms

Bisection method

In order to do that we choose two numbers x_0 and x_1, with $x_0 < x_1$ and the function is continuous in $[x_0, x_1]$ then compute $f(x_0)$ and $f(x_1)$ if this values have different sign we can do next move, and compute c as $c = \frac{x_0+x_1}{2}$. Then we have to choose which point between x_0 and x_1 will be replaced so now we have to check which value of $f(c)$ has the same sign so if

$$\begin{cases} sign(f(c)) = sign(f(x_0)), & x_0 \leftarrow c \\ sign(f(c)) = sign(f(x_1)), & x_1 \leftarrow c \end{cases}$$

In order to have a finished time algorithm we have inserted a value that we will use as precision of the number, so the algorithm will run until it will reach the value that will change from the previous one by the precision. Now we have found our first root, in order to find all roots we can iterate this method several times and we need hope to find all the solutions.

Algorithm 1 BISECTION

Input: F, x_0, x_1

1: $f_0 = F(x_0)$
2: $f_1 = F(x_1)$
3: **if** sign(f_0) = sign(f_1) **then**
4: **return**
5: **end if**
6: $c = \frac{x_1+x_0}{2}$
7: **while** $F(c) \neq 0$ **do**
8: $f_c = F(c)$
9: **if** $sign(f_c) = sign(f_1)$ **then**
10: $f_1 = f_c$
11: **else**
12: $f_0 = f_c$
13: **end if**
14: $c = \frac{x_1+x_0}{2}$
15: **end while**
16: **return** c

Regula falsi

Like the bisection method this analyze the section in order to find a root but, the regula falsi method chose the next point with a different function,

$$\frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$$

This is meant to find a faster way in order to choose a point closer to the root. However, it may slow down the rate of convergence.

Algorithm 2 FALSI

Input: F, x_0, x_1

1: $f_0 = F(x_0)$
2: $f_1 = F(x_1)$
3: **if** sign(f_0) = sign(f_1) **then**
4: **return**
5: **end if**
6: $c = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$
7: **while** $F(c) \neq 0$ **do**
8: $f_c = F(c)$
9: **if** $sign(f_c) = sign(f_1)$ **then**
10: $f_1 = f_c$
11: **else**
12: $f_0 = f_c$
13: **end if**
14: $c = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$
15: **end while**
16: **return** c

Parallel algorithm

In order to compute all the roots and find all of them we need to iterate the algorithm inside the x axis, but we can't analyze the entire axis so we need to set a maximum size of it. We choose the input size as a portion of the x axis so we split the axis to all of the processes p, let's call the chosen portion of the axis as n so each process receive $\frac{n}{p}$ size. To do that we have analyzed the easiest function: $y = mx + q$, the zero in this function is at $x = \frac{-q}{m}$ so we choose the max value between $max = \max(m, q)$. When we analyze much complex functions the results will be closer to the zero so we can chose the bound as $\pm max$, obviously this can be a bound to the algorithm and some root could not be found. For transcendental function, the max value has to be chosen differently cause of frequency. In functions like $\sin(fx)$ the a of the argument of sin give us the frequency so in order to find all the roots we need to chose $max = f$. We can send to the algorithm the input size. Then we can split the input into max sections, and after have splitted the $\frac{n}{p}$ input max times, we can start the computation of the root for each split with one of this two sequential algorithms (Bisection, Regula Falsi). As described in sequential algorithm we will analyze only the segment where the function has different sign for x_0 and x_1, so not all of the split will be analyzed but only where there is a root.

Algorithm 3 PARALLELROOTFINDING

Input: F, x_0, x_1

```
1: max = getmaxoffunction(F)
2: splittedarray = split(x_1, x_0, max)
3: k = 0
4: for i in splittedarray do
5:     c[k] = BISECTION(F, i[0], i[1])
6:     k = k + 1
7: end for
8: return c
```

The main problem of this algorithm is the computation of $f(x) = 0$ because the computation could never end due to the floating point of the computer. In order to resolve this we can use a precision number which can determine the total amount of iteration of the algorithm. The Bisection or Regula Falsi method can run in less than this iterations, this algorithms could run in $\Theta(\lg n)$, cause at each iteration can split the size of the input by an half so the running time is $T(n) = T(n/2) + O(1)$ where for the master theorem is $T(n) = \Theta(\lg n)$.

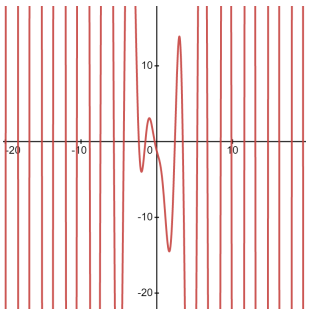
Results

Simulations

The simulation have been carried out in a laptop with:

- Intel Core I7-7700HQ, quad core 8 thread, 6MB cache and frequency of 2.80GHz up to 3.80GHz
- 16GB of DDR4 up to 2133MHz
- 500GB SSD NVMe
- OS: Windows Subsistem for Linux v.2 on Windows 10 Home x64

We are analyzing the following function $3x^2 * sin(2x + 2) - 4x - 1$in this case we will chose to give manually the input size in order to evaluate the algorithm and it's running time.



As shown in the next table computing this function with a lot of results cause a running time much slower and as we can see with a single process the running time increase a lot but with parallel version the running time goes down a lot.

input size\processes	p=1	p=2	p=4	p=8
100	0,001s	0,001s	0,0008s	0,0006s
1000	0,01s	0,0095s	0,006s	0,004s
10000	0,1s	0,095s	0,05s	0,03s
30000	557s	189s	102s	59s

Parallel Complexity

To know how much is the complexity is we need to analyze all the communications. We have 3 communication 2 gather and one scatter for a total communication time of $3\Theta(n \lg p)$. In order to partition the array we need $O(\frac{n}{p} \times max)$. Then to compute all the roots and the non roots we can assume that non roots are at most $O(\frac{n}{p} \times max)$ if no roots can be found, then to find all roots calling m the number of roots, we can compute it in mean $\Theta(m \lg(\frac{n}{p} \times \frac{max}{precision}))$. Adding all of this we can assume that the total mean running time is $\Theta(n \lg p + \frac{n}{p} \times max + m \lg(\frac{n}{p} \times \frac{max}{precision}))$, where we can assume that max is much smaller than p, m and n so we can rewrite it like $\Theta(n \lg p + \frac{n}{p} + m \lg(\frac{n}{p} \times \frac{1}{precision}))$.

Conclusions

Computing with the Bisection method we have found that the algorithm works well with low number of roots and small precision, otherwise with a lot of roots to find the algorithm runs slow based on the precision. Regula Falsi should works better but practically not.

Appendix A

List of Commands

Conditions in order to start the program:

- $x_0 > x_1$
- if $-a$ used x_0 and x_1 does not need to be used
- if want to run the sequential version you can run the sequential version

Compiling

The compile command for parallel and sequential version are:

```
mpicc -O3 -Wall functions.c read_input.c falsi_par.c -o ./falsi_par -
    lm
gcc -O3 -Wall functions.c read_input.c falsi_seq.c -o ./falsi_seq -lm
```

If you want to run the compilation with Makefile this command can be runned:

```
make compile
make compile-seq
```

Running

The run command for parallel and sequential version are:

```
mpiexec -n 2 ./falsi_par -x0 -15000 -x1 15000 -p 0.001 -f exp
./falsi_par -x0 -15000 -x1 15000 -p 0.001 -f sin
```

If you want to run the programs with Makefile this command can be runned:

```
make run
make run-seq
```