Approximation of a Differential Equation using a Combination of Euler and Taylor Methods in CILK Environment

Francisco Alvarado and Angel Tirado
Advisor: Dr. Jaime Seguel

Mathematics Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
franciscoalvarado@yahoo.com
tirado_a@hotmail.com

ABSTRACT

In this paper we introduce a parallel method for computing a discrete set of approximate values of the solution of an initial value problem. A differential equation is said to be an equation containing the derivatives of one or more dependent variables, with respect to one or more independent variables.

Using CILK to find solutions to differential equations will result in more exact approximations and faster results.

1. INTRODUCTION

The purpose of this paper is to find a quick and exact set of approximations to the solution of a given differential equation. The methods to be used are Euler and Taylor modified to work together in a CILK environment. Since we have the real solution to the differential equation we can calculate the error that generates each method or any combination of them.

An initial value problem consists of three main elements:

(1) an ordinary differential equation
(2) an interval on which the solution of (1) is to be computed
(3) an initial value, which is the true value of the solution of (1) at the left boundary of (2)

Because of the nature of initial value problems, most numerical methods used to approximate their solutions are "step methods", this is, methods that use previously computed approximations to produce the following ones. This implies inherently serial algorithms. The parallel method introduced here is obtained by deriving first (2) into a fixed number of subintervals, using a high precision step method for computing the solution of (1) on each of these points, and using then a simpler step method on each of the subintervals. The simpler method takes as initial value the corresponding outputs of the high precision method. Thus, the overall precision of the solution will depend on two parameters the step sizes of the high precision method, and the step size of the true lower precision method. Parallelism is obtain by computing the approximate solutions on each subinterval, in parallel.

This parallel algorithm is implemented in Cilk 5.2 , and run on a symmetric multiprocessor with four Xeon 400Mhz multiprocessors. In order to test the accuracy of the computed results, we used an initial value problem with a well known analytical solution.

This paper is organized as follows: section 2 summarizes the features and capabilities of our program as a software product, section 3 describes the Cilk environment, section 4 describes the methods employed in difference equations, section 5 describes the algorithm, and section 6 provides tables with the experimental results.

2. FEATURES AND CAPABILITIES

Our program was designed to accomplish the following objectives:

- Calculate the approximate solutions to a differential equation.
- Combine Taylor's and Euler's methods to find a more exact set of approximations of the solution.
- Calculate solutions with different step sizes.
- Change the intervals in which the equation is going to be evaluated.
- Calculate the error of the approximate solution with the real solution.

3. SOFTWARE

This program is implemented in a CILK environment. CILK is a language for multithreaded parallel programming based on ANSI C. The Cilk runtime system automatically manages the low level details of executing parallel Cilk code by implementing an efficient work-stealing scheduler.

The CILK runtime system takes care of details such as load balancing, paging, and communication protocols. CILK is algorithmic in that the runtime system guarantees efficiency and predictable performance.

The main features of the language are exhibited in the following Cilk implementation of the familiar recursive Fibonacci program. In this program the recursive calls are executed in parallel.

```
cilk int fib (int n)
{
  if (n<2) return n;
  else
  {
    int x, y;

    x = spawn fib (n-1);
    y = spawn fib (n-2);

     sync;

    return (x+y);
  }
}
```

By deleting the three Cilk keywords (shown in **bold** and **underline**), a C program, called the *serial elision* or *C elision* of the Cilk program is obtained. Cilk is a *faithful* extension of C in that the C elision of any Cilk program is always a legal C implementation.

The keyword **cilk** identifies a *Cilk procedure*. A Cilk procedure has an argument list and body just like a C function. Some of the work in a Cilk procedure is executed serially. Parallelism is created when the invocation of a Cilk procedure is immediately preceded by the keyword **spawn**. A spawn is the parallel analog of a C function call, and just as a C function call, when a Cilk procedure is spawned, execution proceeds to the child. A Cilk procedure cannot safely use the return values of the children it has spawned until it executes a **sync** statement.

In Cilk, multithreding is at the level of algorithm's design and implementation. An algorithm is multithreaded if it can be described as a directed acyclic graph (dag) (see figure 1).

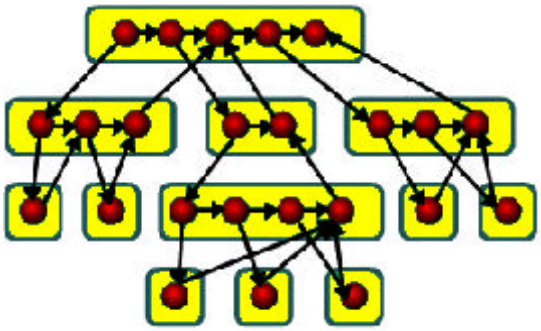

Figure 1. Example of a Cilk algorithm.

The graph is acyclic because the arrows never returns. The rounded squares represent a given procedure, the nodes represent threads, the downward edges represent spawns and the upward edges represent syncs.

4. THE PROBLEM AND ITS APPLICATION

We used a differential equation with a known analytical solution, so the exact values could be compared with the compuer outputs. The differential equation was:

$$y' = (2/t)y + t - (e^t) \; ; \; [1,2]$$
$$y(1) = 0$$

The exact solution is:

$$y(t) = t^2 (e^t - e)$$

The methods we used to find the approximations were Taylor of order 2 and Euler. We can define the Taylor function as:

$$W_{i+1} = Wi + hf(t_i, W_i) + (h^2/2!)f'(t_i, W_i)$$
$$W_0 = 0$$

The Euler function can be defined as:

$$W_{i+1} = Wi + hf(t_i, W_i)$$
$$W_0 = 0$$

The Taylor method obtains an approximation to the initial value problem. In reality, a continuous approximation to the solution y(t) will not be obtained; instead, a discrete set of approximations to y(t) will be generated at various points, called mesh points, in the interval [a,b]. Once the approximate solution is obtained at the points, the approximate solution at other points in the interval can be obtained. The common distance between the points h=(b-a)/N is called the step size, where N is a positive integer {1, 2, …., N}. In Euler's method the error grows slightly as the value t increases, but is expected to grow in no worse than a linear manner.

5. THE ALGORITHM

The program computes an approximation of a differential equation. First it calculates a high precision approximation using the Taylor method from point a to point b in step sizes of h-high. When the first step size of the approximation is calculated the Euler function calculates a low precision approximation of that first step size generated by Taylor, dividing it in h-low divisions. The Euler function uses as initial value an approximated value calculated by the Taylor function. Thus the final approximation is an approximation of another approximation.

INPUT endpoints a,b; integer N_high; integer N_low.

Step 1 set ss_high = (b-a)/N_high;
t=a;

Step 2 For i1=1,2,…., N_high do steps 3, 4, 5, 6.
Step 3 set w=w + hf(t,w)+(h^2 / 2!)f'(t,w) ; Compute w_i
t=a + ih;

Step 4 Call function Euler(w, t, (t+ss_high), N_low, i)

Step 5 set ss_low = ((t+ss_low) – t) / N-low;

Step 6 For i2=1,2,….,N_low do step 7,8.

Step 7 set w=w + hf(t,w) ; Compute w_i .
t=a + ih;

Step 8 OUTPUT (t, w)

Step 9 Stop

6. EXPERIMENTAL RESULTS

After analyzing the problem and finally achieving the correct code for our algorithm, we started to calculate the approximations of our differential equation. When we entered N_high as 1, the Taylor function will only be executed 1 time and the approximation of will depend exclusively on the N_low divisions on the Euler function. Since our Euler function is a serial code, no parallelism was achieved. In the case where N_high is 1, the bigger the N_low is, the smaller the error between the exact solution and the approximated one (see table 1).

Table 1.

N_high	N_low	Error
1	10	3.2848610
1	30	1.1778160
1	50	0.7174600
1	100	0.3628680
1	200	0.1824850
1	400	0.0915070
1	1000	0.0366670
1	10000	0.0036710
1	100000	0.0003670
1	1000000	0.0000370

In the cases where N_high is greater than 1, the program starts to give more accurate results. We expected to find that the error will diminish by increasing the values of N_high and N_low. After running the program several times, we found out that the value of N_high is the key to the approximation. The larger the number N_high is, the better the approximation. Since our program calculates first with Taylor and then with Euler the error produced by the Taylor approximation is carried over to the Euler calculation of the approximate solution. Because of this, the value of N_low is not as important as N_high. For example, with N_high equal to 100 and N_low equal to 10, the error was of 0.177086. If we use the both N_high and N_low equal to 100, the error is 0.1766470, which is a difference of 0.000439 with 90 more step sizes.

As N_high increases, the effect that N_low has on the approximation decreases (see table 2). For example, if N_high is 1000 and N_low is 10, the error is of 0.017994. With some N_high but with N_low equal 1000 the error is 0.017989. This gives a difference of 0.000005 between these two errors. Since we obtained paralellism we can calculate

speed-ups. The speed up is the quotient between the execution time in one processor and the execution time in all processors. Speed up can be calculated using this equation:

Table 2.

N_high	N_low	Error	Parallelism	Speed-up
100	100	0.1766470	2	1.3333
100	1000	0.1766030	21	3.3600
100	10000	0.1765980	67	3.7746
1000	10	0.0179940	3	1.7143
1000	100	0.0179900	11	2.9333
1000	100	0.0179900	22	3.3846
1000	500	0.0179890	101	3.8476
1000	500	0.0179890	61.5	3.7557
1000	1000	0.0179890	124	3.8750

S_p = (paralellism)*(number of processors)/
(paralellism) + (number of processors)

When S_p comes closer to the number of processors, the perfect paralellism is achieved.

When N_high reaches 3000 the value of N_low is irrelevant, because no matter what is the latter value the approximation is the same (tolerance kept to 6 decimal points). When N_high reaches 3000 the approximation given is the same as if the approximation was calculated by the Taylor function alone. This means that if we run a program that calculates the approximated results returned by the Taylor algorithm, the approximation when N > 3000 will be the same as the one generated by our program.

In the process of analysis of our investigation, we evaluated the function with several different values for N_high and N_low, thus generating several values of parallelism. The larger the values of N_high and N_low, the highest the paralellism. As paralellism grows, so does the speed-up. We achieved a maximum paralellism of 28,123.89 which gives us an almost perfect speed-up of 3.99943.

Here are some other significant values:

N_high	N_low	Error	Paralellism	Speed-Up
5000	5000	0.0036040	2524	3.994
10000	10000	0.0018020	6686	3.998
15000	15000	0.0012020	6676	3.998
20000	20000	0.0009010	13441.5	3.999
100000	100000	0.0001800	28123.89	3.999

Our method isn't always a better aproximation than the other two methods. When N_low is lower than 3, the approximation generated by our method is not as exact as the Taylor's, but it is more exact than Euler's independently from the values of N_high (see figure 2). In general our method does give a better approximation of the given initial value problem. This investigation can be continued using other high precision methods and probably giving a better approximation and decrease execution time with a better paralellism.

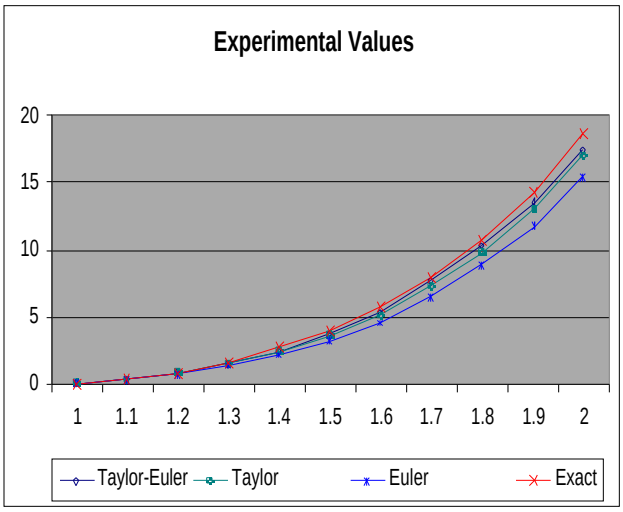

Figure 2. Relation between the Exact differential equation, the approximate solution using the Taylor method, Euler method and our Taylor-Euler method.

REFERENCES

[1] "The CILK Project" http://supertech.lcs.mit.edu/cilk/

[2] "The Cilk RunTime System" Charlie Patel, JavaNauts Research Project Department of Electrical & Computer Engineering, University of Alabama in Huntsville

[3] "Differential Equations with Boundary-Value Problems", 4th edition. Dennis G. Zill & Michael R. Cullen. 1997 Brooks/Cole Publishing Company.

[4] "Numerical Analysis", 6th edition. Richard L. Burden & J. Douglas Faires. 1997 Brooks/Cole Publishing Company.