Solving Differential Equations in Parallel

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 different parallel methods for solving differential equations. 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 different parallel algorithms to resolve this equations we develop quicker, faster and more exact approximations.

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 in parallel environments, specifically in MPI and CILK. We used initial value problems to implement our algorithms. 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. We developed 4 different methods that approximated the solutions using parallel methods. Two of the methods, one using MPI and other using Cilk, used a combinations of the Euler and Taylor methods to find the set of approximate values. The other two methods, also in MPI and Cilk, used a higher degree of approximation using Taylor order 4.

This parallel algorithm is implemented in Cilk 5.2 and in MPI, 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.

2. Software

This program is implemented in 2 environments, Cilk and MPI. MPI stands for Message Passing Interface. The purpose of MPI, in very terms, is to develop a widely used standard for writing message-passing programs. As such the interface attempts to establish a practical, portable, efficient, and flexible standard for message passing. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in Fortran 77 or C. MPI also forms a possible target for compilers of languages such as High Performance Fortran. MPI is used to specify the communication between a set of processes forming a concurrent program.

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.

3. The problem and its applications

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

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

The exact solution is:

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

The methods e used to find the approximations were Euler, Taylor of order 2, and Taylor of order 4.

Euler:

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

Taylor order 2:

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

Taylor order 4:

$$W_{i+1} = Wi + hf(t_i, W_i) + (h^2/2!)f''(t_i, W_i) + (h^3/3!)f'''(ti, Wi) + (h^4/4!)f^{IV}(ti, Wi)$$
$$W_0 = á$$

4. The algorithms

The program computes an approximation of a differential equation. In the first method 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 generate by Taylor, dividing it in h-low division. 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

In the second method we only use the Taylor approximation of order 4.

INPUT endpoints a,b, integer h;

Step 1 ss=(b-a)/h; t=a;

Step 2 set w_1 = w + hf(t,w)
set w_2 = h^2/2! f’(t,w)
set w_3 = h3/3! f’’(t,w)
set w_4 = h4/4! f’’’(t,w)

Step 3 for i = 1,2,…,h do steps 4, 5

Step 4 P_n do w_n

Step 5 w = w_1 + w_2+ w_3 + w_4;
t = a + ih;

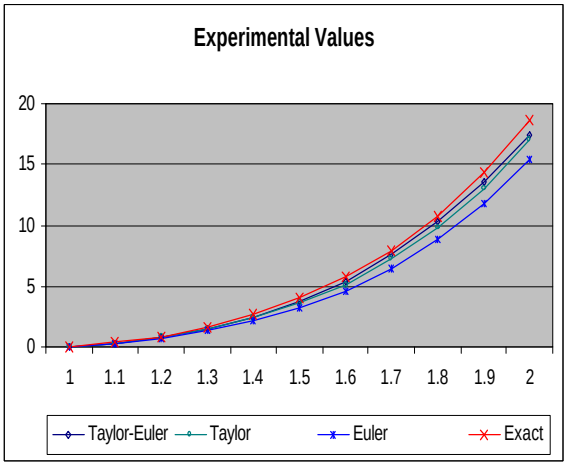
Step 6 Output (t,w)

Step 9 Stop

5. Experimental results

After analyzing the problem and finally achieving the correct source for the first 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 will depend exclusively on the N_low divisions on the Euler function. Since our Euler function is a serial code, hence almost no parallelism was achieved. In the case where N_high is 1, the bigger N_low is, the error between the exact solution and the approximation is smaller.

In the cases where N_high is greater than 1, the program starts to give results (in terms of reducing the error). We expected to find that the bigger both N's got, the error of the approximation would get smaller. After running the program several times, we found out that the value of N_high is the key to the approximation, the bigger the number, 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. The bigger the N_high, the effect that N_low has on the approximation decreases.



This graph represents the relation between the Exact differential equation, the approximate solution using the Taylor method, Euler method and our Taylor-Euler method.

Now we will discuss the results for the second algorithm. Because of the nature of the method (Taylor order 4), this parallel approach gives a more exact approximation of the differential equation than the previous method. When implemented in MPI, there occurs parallelism any time when two or more processes are been executed at the same time. Our algorithm divides the work to be done by the approximation in four functions. Theoretically each process computes one of the functions and returns the value to P_0, P_0 then adds up the individual values to obtain the approximation of that step size. In practice, the parallelism is not optimize since it needs all four values to continue to the next step size , hence the actual speed of the approximation depends on the slowest of the processes.

Cilk uses spawns to distribute the work between the processes. Using Cilk is not as easy to achieve parallelism. In MPI we can force parallelism assigning each process a specific work. Cilk works using a work-stealing scheduler. Assuming we have more than one processor, when a Cilk procedure spawns, its processor post the work locally by growing its stack of jobs. If a processor is idle it steals work from this stack.

6. Conclusions

After we analyze the results of both environments for the first algorithm, comparing the time, the parallelism, the precision and more important the performance, we conclude that in MPI the program depends on the value of N_high because if the N_high is a big number the processor 0 take a lot of time to send the array of initial values to the other processors. But if the N_high is a small number its calculation will be quicker and the result of it will be a faster approximation. In the Cilk environment we can observe a similar patron but with the difference that in Cilk the Taylor method doesn't affect the parallelism of the algorithm. The Euler method generates separate sets of jobs that depending on the work that one processor is doing the other processors steal part of these jobs creating an overlapping of some processes. In other words, the environment depends on the N_high or Taylor approximation.

The implementation of the Taylor order 4 algorithm in both Cilk and MPI was achieved. The Cilk implementation was quicker in calculating the approximations. Perhaps the work-stealing method was not optimal, if the primary processor does not generates the work for other processor the calculations are all done by only one process. In comparison with MPI, where the parallelism is forced, in Cilk the parallelism is left to the software. In the majority of the cases, most of the work was done by only one of the processors. The approximations using the Cilk where much much quicker than those done by MPI.

Both languages use very different methods to produce parallelism. The different implementations that can be accomplished with one of the languages doesn't necessarily can be implemented in the other. In our case we could

implement both methods in both MPI and Cilk, and we did get good results from them.

7. 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.