Iraqi Journal of Statistical Sciences

www.stats.mosuljournals.com

Parallel algorithm for calculating the integration

Ehab Abdulrazak Alasadi

Department of Quranic Science, Faculty of Islamic Science, Kerbala University, Kerbala, Iraq.

Article information

Article history:
Received July 1, 2024
Revised: August 15, 2024
Accepted August 25, 2024
Available June 1, 2025

Keywords:
Parallel Program,
Pthread, Shared memory
Mutex

Correspondence:
Ehab Alasadi
ehabalasadi@gmail.com

Abstract

Analysis and implementation of a parallel algorithm to calculate the integration of the function y=1/e to the x with a specified time interval. Design and implementation using C/C++ is based on the sharing of memory among "THREADS." The "Pthread" library has been used. Use of the output file to print information and the purpose of using the POSIX library

It is to implement the program faster than the one nucleus, when it involves a set of processors (THREADS) where each thread is considered to be a processor. This accelerates the solution of the complex problems in the system that need a large memory, where time sharing is used by Mutex. Lock and unlock through research prior to the use of parallel programs and its memory sharing technique to solve complex and large issues that require a long time to be implemented. Using parallel programs, each thread carries a particular issue and solves it, and the results are combined by reducing the time execution and increasing the speed of the system speedup according to the speed equation S=T1/Tn

DOI 10.33899/iqjoss.2025.187732

1. Introduction

Threads are a key technology in the further development of Linux and Unix. Programming models that allow the use of these threads provides benefits for both client and server programs. Programs like Netscape Navigator are proof of the power of multithreaded programming using POSIX threads. Threads provide a natural programming model for applications requiring parallel execution.[1]

In order to achieve the maximum potential offered by fibers, it was necessary to standardize the programming interface. In UNIX operating systems, such an interface was specified by the IEEE POSIX 1003.1c-1995 standard, which culminated in "POSIX threads" or "PThreads". PThreads define a set of types and functions for the C programming language that are implemented in the pthread.h header file and the library.[1],[3].

The advantages of threads are compared to processes in the price of creation and maintenance, threads can be created at much lower system costs. Thread maintenance requires less system resources than process maintenance. Another advantage is that all threads in a process share the same address space. Communication between threads is more efficient and, in many cases, easier to implement than communication between processes[4].

The first argument in pthread_create() is a variable that holds the address that reference to the thread want to create, and the second argument is which typed to pthread_attr_t is a variable that let us address the attributes of the thread such as priority. The third argument need to supply to this system call is a function pointer that let the kernel knows where the thread should begin from and the fourth argument is the parameter would like to pass to the function that we are going call from this thread.

Previous works

During the past five years, most of the research has focused on using parallel programs to process large amounts of information and problems and solve complex problems that require large memory by using PTHREADS and PVM (**Parallel Virtual machine**) for sharing memory and memory synchronization among computers and return the results to master computer and that allows programmers to run large-scale projects that require speed and accuracy because its will reduce the execution time and increase the system Acceleration and the results show increase the numbers of thread will increase the system speedup and efficiency.

1.2 Problem define

To calculate a definite integral [1].

$$I = \int_a^b f(x)dx \quad (1)$$

When [a,b] specify the interval

f(x), algebraic expression in variable 'x'

I= Integral

functions given by functional values at equidistant points (nodes), Newton-Cotes quadrature formulas are most often used. The most famous representatives of this class of methods are the trapezoidal method and Simpson's rule.

To calculate the definite integral of the function f(x) in the interval <a,b> using equidistant points, the following is set: where n is the number of equal parts into which we divide the interval <a,b>. This is how equidistance points are obtained:

x0 = a, xi = x0 + ih (i = 1, 2, ..., n-1), xn = b,

for which functional values *yi = f(xi), i = 1, 2, ... n* are calculated.

1.3 Trapezoidal method

The trapezoidal method is characterized by slow convergence of the numerical process with relatively low accuracy $O(h)$^2 [10].

$$I = I_L + R_L, \;\; I_L = h.\left(\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right) \quad (2)$$

This results in a large calculation error:

$$R_L = -\frac{(b-a)h^2}{12}f^n(\xi) = -\frac{(b-a)^3}{12n^2}f^n(\xi) \quad (3)$$

where ξ belongs to <a,b>

Its well-known geometric interpretation is the sum of the contents of **trapezoids**, which are formed by nodal points and their functional values. The integral is calculated using a piecewise linear approximation, which does not take into account the nature of the curvature of the function between nodal points **[9].**

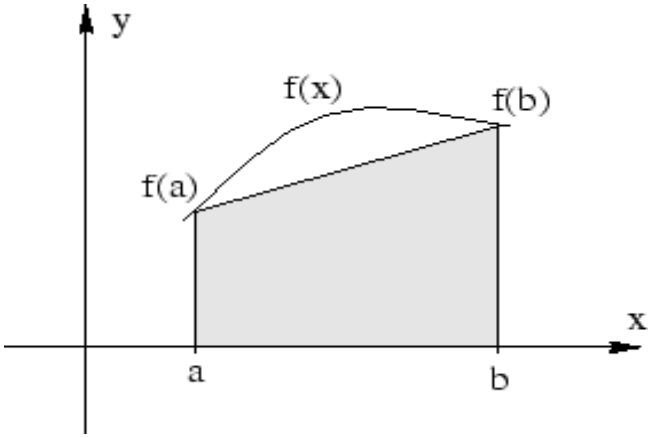


Fig (1) Trapezoidal formula

2 Proposed designs

To calculate the integral of the function y = 1/ e^x on the given interval. For the calculation of the integral, I

The trapezoidal method was chosen from among the numerical methods because it is an easy-to-understand method that provides the possibility of performing the calculation in parallel. The input to the program will be user-specified interval bounds and x, which are numbers of type "double"

2.1 Description of proposed solution

1) Read the interval and the number x, which are entered as program arguments.

2) calculation of the number h, according to h = (b-a)/N, where N is the number of equal parts into which we divide the given interval, and in the program, N is also equal to the number of running threads performing the calculation,

3) the main program starts N threads with arguments specifying the boundaries of sub-intervals and the value of x,

4) each running thread performs the calculation of the integral of the function y = 1/e^x by the trapezoidal method, where N is much larger this time (on the order of 1000) and performs the calculation independently without creating additional threads,

5) individual threads calculate integrals on partial intervals and add the result to a common variable determining the total value of the integral, access to this variable represents a critical area in the program,

6) the program prints the result of the integral.

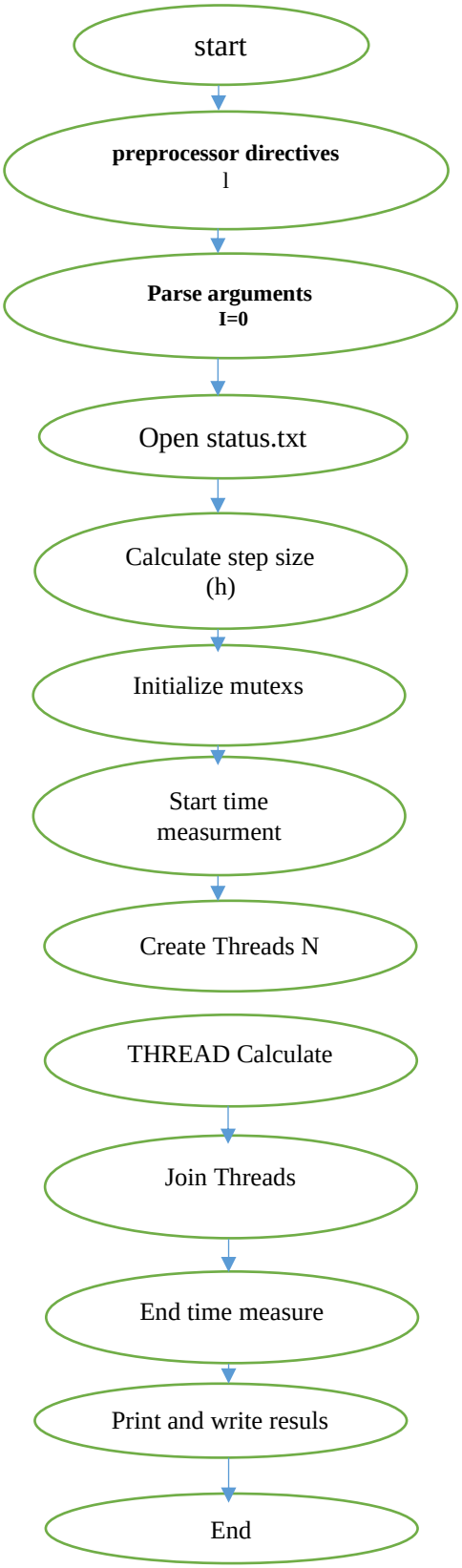


Fig (2) Program flowchart

2.2 Divide tasks among the nodes

Since the total calculation of the integral by the trapezoidal method consists of the sum of the functional values of the divided original interval, the calculation of the sum can be performed in parallel. This can be done in two ways:

1) The specified interval will be divided into a large number of smaller ones, and the sum calculation will be performed by starting the same number of threads as the number of intervals. Then each thread computes a function f(xi) where xi = a + ih, i = 0, 1, ..., N. The disadvantages are that a large number of threads would have to be created to implement a trivial function, causing unnecessary overhead system and high lag.

2) The second way is to divide the specified interval of the integral into a smaller number and start the same number of threads. Each thread will now perform the calculation of the integral using the trapezoidal method on the subinterval determined by the main program. This way of doing it is better because it creates a smaller number of threads that can calculate the partial integral fast enough since it is a simple cycle. This will also reduce the overhead and maintenance of access to the critical area.

2.3 Communication and synchronization

Synchronization and communication between the main program and threads is implemented using the "PThread" library. The input values of individual threads are specified as arguments when creating threads with the pthread_create command, and their output is the resulting partial integrals, which are calculated in a common variable that represents the shared memory. This variable and the common file for auxiliary outputs are critical areas in the program as data inconsistencies can occur. The solution to this question is offered by the synchronization functions of the "PThread" library: pthread_mutex_lock, pthread_mutex_trylock and pthread_mutex_unlock**.[7].**

During computation, the program enters two critical areas when threads access a shared variable and a shared file for auxiliary statements. I solved it using two locks I_mutex (for a variable) and file_mutex (for a file), the use of which in the thread is as follows:

```
pthread_mutex_lock (&I_mutex); //.  lock variable I
pthread_mutex_unlock (&I_mutex); //unlock variable I
pthread_mutex_unlock (&file_mutex); //unlock the file.
```

Program compilation

Compilation of the program is done with the command:

Arguments: pthread_dps2e [-h] [a b N {number of threads}]

```
// -h : help for the program usage
// a : lower limit of the interval of the integral
// b : upper limit of the interval of the integral
// N : division of the interval in the Threads
```

pthread dps2e [-h] [a b N {10}]

Calculation of the integral on the interval (0.000000, 2.000000) on 10 fibers. THREAD 0: entering the critical region
THREAD 0: It = 0.181269
THREAD 0: leaves the critical region THREAD 1: entering the critical region THREAD 1: It = 0.148411
THREAD 1: leaves the critical region THREAD 2: entering the critical region THREAD 2: It = 0.121508
THREAD 2: leaves the critical region THREAD 3: entering the critical region THREAD 3: It = 0.099483
THREAD 3: leaves the critical region THREAD 7: entering the critical region THREAD 7: It = 0.044700
THREAD 7: leaves the critical region THREAD 8: entering the critical region THREAD 8: It = 0.036598
THREAD 8: leaves the critical region THREAD 9: entering the critical region THREAD 9: It = 0.029964
THREAD 9:
leaves the critical area

Fig (3) Running program

The argument {number of threads} is optional, if not specified the program will be executed with 10 threads. Help is displayed to the user even if the number of arguments is incorrect. The program creates a "status.txt" file, where records of the calculation status, or threads, if the file already exists records are added to it.

3 Program testing

The program was tested for different values of the input intervals (a, b), the result was verified using the relation that realizes the calculation of the integral. The accuracy of the calculation varies according to the entered N.

The program is treated for the inputs a, b when they are given in the wrong order and the input indicating the number of executable threads when it is given as 0.

The calculation of the integral is realized in an acceptable time if N is not greater than 2,000,000.

4 Model PRAM

In order to calculate the acceleration in the parallel calculation of the integral by the **trapezoidal method** on N processors, he

The time complexity, work, and cost of the sequential and parallel algorithm need to be calculated**.[8].**
The sequential approach to task implementation is shown by the RAM model:

```
h = (b-a)/N;
I = h* ((pow(e, -a) + pow(e, -b)) / 2);
for (int j=1; j<N; j++)
```

Then, for the resulting time complexity Ts(N):

int thr double a

double b int N

mydata->thr; mydata->a; mydata->b;

mydata->N; //rozdelenie intervalu

double h;

double e = exp (1); //cislo e

double It=0; //vypocitany integral vlakna

double suma=0; //pomocna suma

h=(b-a)/N;

//calculate integral LICHOBEZNIKOVOU METODOU

It h*((pow(e, -a) + pow(e, -b)) / 2); // h*[(fa+fb)/2]

for (int j=1; j<N; j++) // suma (1..N-1) [f(xj)] suma + pow(e, - (a+ j*h)); //xj= a+ h*j

It+=h*suma;

//enter to critical area

pthread_mutex_lock (&I_mutex); //lock I pthread_mutex_lock (&file_mutex); //lockfile

if ((file = fopen("status.txt", "a"))

{

== NULL)

fprintf(stderr, "Cannot open input file. \n");

number of processors P(N, 1) = 1

number of steps T(N, 1) = O(1) + O(1) + O(N) + O(1) = O(N)

work W(N, 1) = O(N)

price C(N, 1) = T(N, 1) . P(N, 1) = O(N) . 1 = O(N)

A parallel integral calculation algorithm based on a CRCWPRi type PRAM model is being considered

h = (b-a)/N;

I = h* ((pow(e, -a) + pow(e, -b)) / 2);

for (int j=1; j<N; j++)

pardo sum += pow (e, - (a+ j*h));

I += h * sum;

Then for the resulting time complexity T(N):

number of processors P(N) ≡ p = N

number of steps T(N, p) = O(1) + O(1) + O(1) + O(1) = O(1)

work W(N, p) = O(N)

price C(N, p) = T(N, p) . P(N, p) = O(1) . N = O(N)

From this we get the theoretical acceleration=(Sp)=$\frac{Ts}{Tp}$ = (4)

When **Ts:**Time at 1 process

Tp: Time at n process

Time calculated see **Table (1)**

Sp=T(1)/Tp=4.837/4.297=1.125

5 Evaluation of the implementation

The program is implemented in the Linux environment in the C++ language and with the g++ compiler according to the assignment specification. The program uses the libraries "sys/time.h" for time calculation, "math.h" for mathematical functions necessary for calculation, and "PThread.h", which defines a set of types and functions for the C programming language for working with threads.

The calculation execution algorithm is implemented according to the design of the solution only with changed input data entered as arguments. The user must enter in order the limits of the interval (lower and upper), the number N, according to which the interval will be divided into N equal parts in individual threads, and the number of threads to perform the calculation. The argument specifying the number of threads is optional, if not specified by the user, 10 threads will be created.

Each thread performs the calculation of the integral using the trapezoidal method on the interval determined by the main program. These intervals are divided according to the number of threads that will solve the task. The input data for threads is given by a structure as its argument:

```
struct thread_data{
int thr; //thread number (index)
double a; //lower limit of subinterval
double b; //upper limit of subinterval
int N; //how many parts the interval will be divided into
```

During computation, the program enters two critical areas when threads access a shared variable and a shared file for auxiliary statements. I

It was solved by using two locks i_mutex (for a variable) and file_mutex (for a file), the use of which in the thread is as follows:

```
pthread_mutex_lock (&I_mutex); //lock variable I
  pthread_mutex_lock (&file_mutex); //lock the file
{critical area}
pthread_mutex_unlock (&I_mutex); //unlock variable I
```

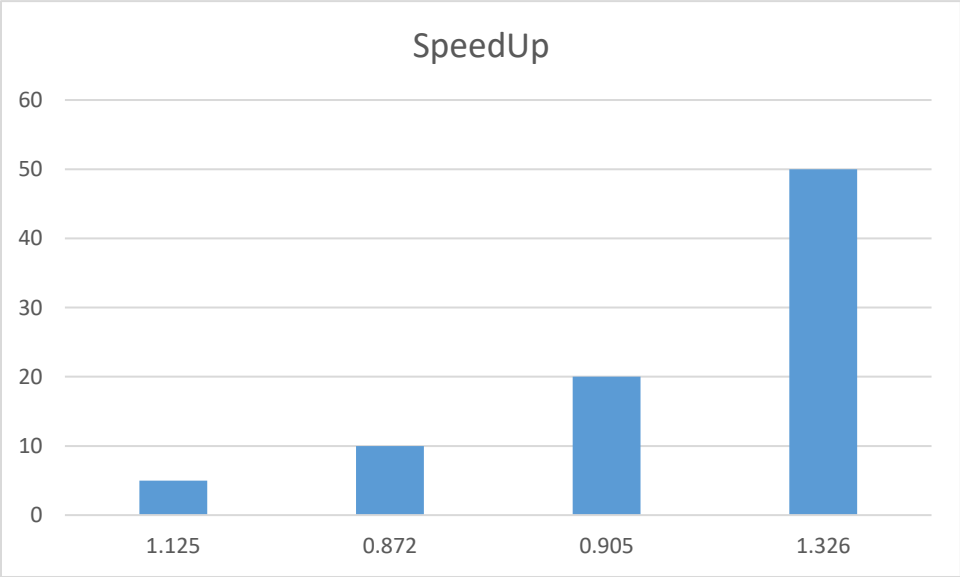
pthread_mutex_unlock (&file_mutex); //unlock the file.

In a thread, these critical areas overlap, so one lock might be enough, but with two locks, for example, the main program writes the entry to a common file, while some free thread "reserves" the result write by locking access to the variable It won't be overtaken by another thread ifhad to fall asleep that way.

The calculation time is measured using the gettimeofday(struct timeval &T, NULL) function of the "sys/time.h" library.

Table1 (Speedup at N process measurements)

Number of Threads	**1**	**5**	**10**
1	Real time	Real time	Real time
2	4.164	2.137	8.277
3	2.4044	3.54	6.293
4	8.709	4.835	2.629
5	4.074	6.676	2.5
Average	4.837	4.297	4.924
Speedup		1.125	1.072



Fig(4) Speedup vs No of Threads

6 Conclusion

Solving this task gave a better insight into the issue and the way to implement calculations on multiprocessor systems, as well as programming using shared memory.

From the achieved results for the PRAM models, it can be seen that the algorithm is designed optimally, since the prices of both approaches shown by the RAM and PRAM models have the same asymptotic bounds.

Even if the parallel approach to the calculation of the integral on multiprocessors achieves a theoretical speedup, it will never be achieved when it is implemented using threads, as the program

runs on one computer, resulting in higher overhead costs than with the sequential approach.

Acknowledgment

The author is very grateful to the University of Mosul, and College of Computer and Mathematical Science, which helped improve this work's quality.

Conflict of interest

The author has no conflict of interest.

References

1- Ippolito, G.Home, YoLinux. Available at: http://www.yolinux.com/TUTORIALS/#google_vignette (Accessed: 17 August 2024).
2- Buttlar, D., Farrell, J.P. and Nichols, B. "Pthreads programming". Sebastopol, O'Reilly Media,2013.
3-David.B, "Programming with POSIX Threads, Addison-Wesley Professional,1997.
4- Github.io. (2016). RCS Workshop 3," Introduction to Parallel Computing using MATLAB" 5Limitations of Parallel Speedup. [online] Available at: https://researchcomputingservices.github.io/parallel-computing/02-speedup-limitations/#:~:text=One%20good%20way%20to%20measure [Accessed 17 Aug. 2024]..
5-Hesham El-Rewini , "Advanced Computer Architecture and Parallel Processing", Wiley-Interscience,2008.
6-Maplesoft.com. (2024). Newton-Cotes Formula - Maple Help. [online] Available at: https://www.maplesoft.com/support/help/maple/view.aspx?path=Student%2FCalculus1%2FNewtonCotes [Accessed 17 Aug. 2024].
7-Victor Alessandrini," Shared Memory Application Programming", Morgan Kaufmann,2015.
8-Michael L. Scott ," Shared-Memory Synchronization", Springer,2013
9-KHURMA, M., "Trapezoidal rule - formula: Trapezoidal formula", Cuemath. Available at: https://www.cuemath.com/trapezoidal-rule-formula/ (Accessed: 17 August 2024).
10- Mathematics, (2023)," Midpoint rule vs Trapezoidal rule accuracy", Mathematics Stack Exchange. Available at: https://math.stackexchange.com/questions/4646783/midpoint-rule-vs-trapezoidal-rule-accuracy (Accessed: 17 August 2024).

خوارزمية متوازية لحساب التكامل الرياضي

إيهاب عبد الزراق حسن،
قسم علوم القرآن ، كلية العلوم الاسلامية، جامعة كربلاء ، كربلاء ، العراق

الخلاصة :تحليل وتنفيذ خوارزمية متوازية لحساب تكامل الدالةy=1/e^x بفاصل الزمني محدد. تصميم وتنفيذ باستخدام لغة (C/C++)يعتمد على مشاركة الذاكرة بين "THREADS". تم استخدام مكتبة "PThread". واستخدام ملف الاخراج لطباعة المعلومات والغرض من المسائل المعقدة والكبيرة التي تحتاج وقت طويل للتنفيذ .باستخدام البرامج الموازية تقوم كل خيط بحمل مسالة معينه وحلها ويتم تجميع النتائج بذلك تقليل زمن التنفيذ Time execution وزيادة سرعة النظام system speedup حسب معادلة السرعة S=T1/Tn .استخدام مكتبة POSIX هو تنفيذ البرنامج بشكل أسرع من النواة الواحدة فعند اشراك مجموعة من المعالجات (THREADS) حيث يعتبر كل خيط بمثابة معالج وهذا يسرع من حل المشاكل المعقدة في النظام التي تحتاج والى ذاكرة كبيرة حيث يتم المشاركة الزمنية لكل واحدة باستخدام Mutex. Lock and unlock من خلال البحوث السابقة لاستخدام البرامج الموازية وتقنيه مشاركة الذاكرة للحل.

الكلمات المفتاحية: البرنامج الموازي، Pthead، الذاكرة المشتركة.