www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 7 July, 2014 Page No. 7203-7212

Complexity & Performance Analysis of Parallel Algorithms of Numerical Quadrature Formulas on Multi Core system Using Open MP

D.S. Ruhela[1] and R.N.Jat[2]
1 Department of Computer Application, S.M.L. (P.G.) College, Jhunjhunu (Raj.)
12Department of Mathematics, University of Rajasthan, Jaipur
E-mail:1dsruhela@yahoo.com, 2khurkhuria_rnjat@yahoo.com

Abstract:

Analysis of an algorithm is to determine the amount of resources such as time and storage necessary to execute it. The efficiency or complexity of an algorithm is based on the function relating the input length to the number of operations to execute the algorithm. In this paper, the computational complexities and execution time for sequential and parallel algorithms used Numerical Quadrature Formulas on Multi Core system Using Open MP are analyzed.To find the integral value of various function using Trapezoidal Rule, Simpson 1/3 Rule, Simpson's 3/8 Rule, Boole's Rule.We have to calculate estimated execution time taken by the programs of sequential andparallel algorithms and also computed the speedup. Accuracy of thequadrature formulas has been found in the order- Simpson's three-eighth rule > Simpson'sone-third rule > Boole's rule > Trapezoidal rule.

Keywords: Complexity of Algorithms, Sequential and parallel execution time,OpenMP

1 Introduction

The field of numerical analysis predates the invention of modern computers by many centuries. Linear interpolation was already in use more than 2000 years ago. Many great mathematicians of the past were preoccupied by numerical analysis, as it obvious from the names of important algorithms like Newton's method, Lagrange interpolation polynomial, Gaussian elimination, or Euler's method (Burden et. all 2000), To facilitate computations by hand, large books were produced with formulas and tables of data such as interpolation points and function coefficients **(**Gilat et. all 2004), MATLAB, (Hildebrand et. all 1974). Using these tables, often calculated complexity and error out to 16 decimal places or more for some functions, one could look up values to plug into the formulas given and achieve very good numerical estimates of some functions. The canonical work in the field is the NIST publication edited by Abramowitz and Stegun (Abramowitz et. all 1972),a 1000-plus page book of a very large number of commonly used formulas and functions and their values at many points. The function values are no longer very useful when a computer is available, but the large listing of formulas can still be very handy. The mechanical calculator was also developed as a tool for hand computation. These calculators evolved into electronic computers in the 1940s, and it was then found that these computers were also useful for administrative purposes. But the invention of the computer also influenced the field of numerical analysis, since now longer and more complicated calculations could be done. Numerical quadrature is another name for numerical integration, which refers to the approximation of an integral $\int f(x)dx$ of some function f(x) by a discrete summation $\Sigma w_i f(x_i)$ over points x_i with some weights w_i. There are

many methods of numerical quadrature corresponding to different choices of methods such as Gaussian quadrature, (Gil et. All 2007)**,** with varying degrees of accuracy for various types of functions f(x). *Popular methods use one of the Newton–Cotes formulas (like the midpoint rule or Simpson's rule) or Gaussian quadrature. These methods rely on a "divide and conquer" strategy, whereby an integral on a relatively large set is broken down into integrals on smaller sets. In higher dimensions, where these methods become prohibitively expensive in terms of computational effort, one may use Monte Carlo or quasi- Monte Carlo methods or, in modestly large dimensions, the method of sparse grids.* More accurate integration formulas with smaller truncation error can be obtained by interpolating several data points with higher-order interpolating polynomials. For example, the fourth order interpolating polynomial P4(t) between five data points leads to the Boole's rule of numerical integration. The Boole's rule has the global truncation error of order $O(h^6)$. However, the higher-order interpolating polynomials often do not provide good approximations for integrals because they tend to oscillate wildly between the samples (polynomial wiggle). As a result, they are seldom used past Boole's rule. Another popular numerical algorithm is used instead to reduce the truncation error of numerical integration. This is Romberg integration based on the Richardson extrapolation algorithm In numerical analysis, the Newton–Cotes formulas are a group of formulas for numerical integration (also called quadrature) based on evaluating the integrand at n+ 1 equally spaced point. Newton–Cotes formulas can be useful if the value of the integrand at equally-spaced points is given. If it is possible to change the points at which the integrand is evaluated, then other methods such as Gaussian quadrature(Gil et. All 2007), (William 1988), and (Roland Bulirsch, 1980) are probably more suitable. It is assumed that the value of a function f is known at equally spaced points xi, for i = 0, , n. There are two types of Newton–Cotes formulas, the "closed" type which uses the function value at all points, and the "open" type which does not use the function values at the endpoints. The closed Newton- Cotes formula (Josef et. All 1980), (Atkinson et. All 1989) and (Douglas, 2000)of degree n is stated as where $x_i = h_i + x_0$, with h (called the step size) equal to $(x_n - x_0)/n$. The wi are called weights. As can be seen in the following derivation the weights are derived from the Lagrange basis polynomials. This means they depend only on the xi and not on the function f. Let L(x) be the interpolation polynomial in the Lagrange form for the given data points $(x_0, f(x0))$, …, (xn, f(xn)), then The open Newton–Cotes formula of degree n is stated as A Newton–Cotes formula of any degree n can be constructed. However, for large n a Newton–Cotes rule can sometimes suffer from catastrophic Runge's phenomenon where the error grows exponentially for large n. Methods such as Gaussian quadrature and Clenshaw–Curtis quadrature with unequally spaced points (clustered at the endpoints of the integration interval) are stable and much more accurate, and are normally preferred to Newton–Cotes. If these methods cannot be used, because the integrand is only given at the fixed equi-distributed grid, then Runge's phenomenon can be avoided by using a composite rule, as explained in next section.

The current multi-core architectures have become popular due to performance, and efficient processing of multiple tasks simultaneously. Today's the parallel algorithms are focusing on multi-core systems. The design of parallel algorithm and performance measurement is the major issue on multi-core environment. If one wishes to execute a single application faster, then the application must be divided into subtask or threads to deliver desired result. Numerical problems, especially the solution of linear

system of equation have many applications in science and engineering. This paper describes and analyzes the parallel algorithms for computing the solution of dense system of linear equations, and to approximately compute the value of _ using OpenMP interface. The performances (speedup) of parallel algorithms on multi-core system have been presented. The experimental results on a multi-core processor show that the proposed parallel algorithms achieves good performance (speedup) compared to the sequential.

We have calculated the complexity of definite integral by dividing the interval of integration [.2 1.4 1] into 10 to 1000000 equal parts in trapezoidal rule, Simpson's 1/3 rule, Simpson's 3/8 rule Boole's Rule and Wedles Rule by developing computer programs in Visual C++ language. Error in the values of integral calculated by quadrature formulas is minimum when upper limit is in the neighborhood of zero and lower limit is .2. Accuracy of the quadrature formulas has been found in the order- Simpson's three-eighth rule > Simpson's one-third rule > Boole's rule > Trapezoidal rule>Weddle's Rule.

We have to calculate estimated execution time taken by the programs of sequential and parallel algorithms and also computed the speedup. We find Sequential Execution and Parallel Execution Time, Compare and Analyze Result.

2 Programming in OpenMP

An OpenMP Application Programming Interface (API) was developed to enable shared memory parallel programming. OpenMP API is the set of compiler directives, library routines, andEnvironment variables to specify shared-memory parallelism in FORTRAN and C/C++ programs(Barbara et. all 2008). It provides three kinds of directives: parallel work sharing, data environment and synchronization to exploit the multi-core, multithreaded processors. The OpenMP provides means for the programmer to: create teams of thread for parallel execution, specify how to share work among the member of team, declare both shared and private variables, and synchronize threads and enable them to perform certain operations exclusively (Barbara et. all 2008). OpenMP is based on the fork-and-join execution model, where a program is initialized as a single thread named master thread (Barbara et. all 2008). This thread is executed sequentially until the first parallel construct is encountered. This construct defines a parallel section (a block which can be executed by a number of threads in parallel). The master thread creates a team of threads that executes the statements concurrently in parallel contained in the parallel section. There is an implicit synchronization at the end of the parallel region, after which only the master thread continues its execution (Barbara et. all 2008).

3 Creating an OpenMP Program

OpenMP's directives can be used in the program which tells the compiler which instructions to execute in parallel and how to distribute them among the threads (Barbara et. all 2008). The first step in creating parallel program using OpenMP from a sequential one is to identify the parallelism it contains. This requires finding instructions, sequences of instructions, or even large section of code that can be executed concurrently by different processors. This is the important task when one goes to develop the parallel application. The second step in creating an OpenMP program is to express, using OpenMP, the parallelism that has been identified (Barbara et. all 2008). A huge practical benefit of OpenMP is that it can be applied to incrementally create a parallel program from an existing sequential code. The developer can insert directives into a portion of the program and leave the rest in its sequential form. Once the resulting program version has been successfully compiled and tested, another portion of the code can be parallelized. The programmer can terminate this

process once the desired speedup has been obtained (Barbara et. all 2008).

4.Performance of Parallel Algorithm

The amount of performance benefit an application will realize by using Open MP depends entirelyon the extent to which it can be parallelized. Amdahl's law specifies the maximum speed-up thatcan be expected by parallelizing portions of a serial program [8]. Essentially, it states that themaximum speed up (S) of a program is

S = 1/ (1-F) + (F / N)

Where, F is the fraction of the total serial execution time taken by the portion of code that can beparallelized and N is the number of processors over which the parallel portion of the code runs.The metric that have been used to evaluate the performance of the parallel algorithm is the

Speedup [8]. It is defined as

Sp = T1 / TP

Where, T1 denotes the execution time of the best known sequential algorithm on single processormachine, and Tp is the execution time of parallel algorithm onIn other word, speedup refers to how much the parallel algorithm is faster than the csequential algorithm. The linear or ideal speedup is obtained.

5. Proposed Methodology

Calculation: to find the integral value of various function using following methods:

(A) Trapezoidal Rule

(B) Simpson 1/3 Rule

(C) Simpson's 3/8 Rule

(D) Boole's Rule

6.Trapezoidal Rule:

We learned that the definite integral $\int_a^b f(x) = F(b) - F(a).$ can be evaluated from the anti-derivative we learned that the definite integral area between the curves of f(x) and the x-axis. That is the principle behind the numerical integration we divide the distance from x=a to x=b into vertical strips and add the area of these strips.

$\int_{xi}^{xi+1} f(x)dx \approx fi + fi+1/2(xi+1-xi)$ we will usually write h=(xi+1-xi) for the width of interval.

6.1. Algorithm

1. Read a,b,n
2. Set h=(b-a)/n
3.Set sum=[f(a)+f(b)]/2
4 for i=1 to n-1
   i. Set sum=sum+f(a+i*h)
5. End for
6. Set Integral=h/2*sum
7 write Integral

6.2. Counting Primitive Operation

1. Reading the value of variable Read a,b,n contributes three unit of count.
2. Assigning values h= (b–a)/n contributes three unit of count.
3. Set sum= [f (a) +f (b)]/2 contributes five unit of count.
4. The body of loop executes n-1 times with count five operations in loop. So 5(n -1) unit of count.
5. Step 6 requires two operations
6. Write integral contributes one operation.

To summarize, the number of primitive operations t (n) executed by algorithm is at least.

t(n) = 3+3+5+5(n-1)+2+1
=11+5n-5+3
t(n)=5n+9.

So the complexity of algorithm used for Trapezoidal method is O (n). In fact, any polynomial $a^k n^k + a^{k-1} n^{k-1} \ldots\ldots + a^0$ will always be O (n^k)

Using program in c for Trapezoidal Rule we find the result for Integral of sin(x)-log(x)-exp(x) with range .2 to 1.4 with no of intervals.

S. No	No. of Intervals	Sequential Execution time & Difference between Result			Parallel Execution time & Difference between Result			Performance/Speed Up(s)
		Integral	Error/ Difference	Time in ms	Integral	Error/ Difference	Time in ms	Difference
1.	10	4.05845 2	0.00009 8	.00023 0	4.05845 2	0.00009 8	.00019 6	1.173469
2.	100	4.05102 4	0.00009 4	.00030 0	4.05102 4	0.00009 4	.00025 7	1.167315
3.	200	4.05096 7	0.00001 7	.00053 0	4.05096 7	0.00001 7	.00022 6	2.345133
4.	300	4.05095 6	0.00000 6	.00054 5	4.05095 6	0.00000 6	.00047 6	1.144958
5.	400	4.05095 3	0.00000 3	.00055 6	4.05095 3	0.00000 3	.00050 8	1.094488
6.	500	4.05095 1	0.00000 1	.00059 3	4.05095 1	0.00000 1	.00051 2	1.158203
7.	600	4.05095 0	0.00000 0	.00061 6	4.05095 0	0.00000 0	.00059 8	1.0301
8.	700	4.05094 9	-0.00000 01	.00071 0	4.05094 9	-0.00000 01	.00058 2	1.219931
9.	800	4.05094 9	-0.00000 01	.00077 0	4.05094 9	-0.00000 01	.00061 6	1.25
10	900	4.05094 9	-0.00000 01	.00080 8	4.05094 9	-0.00000 01	.00071 5	1.13007
11.	1000	4.05094 9	-0.00000 01	.00086 8	4.05094 9	-0.00000 01	.00074 8	1.160428
12.	10000	4.05094 9	-0.00000 01	.00614 6	4.05094 9	-0.00000 01	.00379 4	1.619926
13.	100000	4.05094 9	-0.00000 01	.06	4.05	-	.05	1.24
14.	1000000	4.05094 9	-0.00000 01	.61	4.05	-	.37	1.648
15.	10000000	4.05094 9	-0.00000 01	6.1	4.05	-	3.7	1.648
16.	100000000	4.05094 9	-0.00000 01	61	4.05	-	37	1.648
17.	1000000000	4.05094 9	-0.00000 01	614	4.05	-	379	1.648

Table-1 Performance comparison of Serial and Parallel Algorithm fir integration of sin(x)-log(x)+exp(x)

7.Simpson's 1/3 Rule:

The Trapezoidal rule is based on approximating the function with a linear polynomial. We can fit the function better if we approximate it with a quadratic or a cubic interpolation polynomial. Simpson rules are based on this approximation. We get the Simpson 1/3 Rule by integrating the second degree Newton-Gregory forward polynomial, which fits f(x) a x-value of x0,x1,x2…. Which are evenly spaced a distance h apart.

$$\int_{xi}^{xi+1} f(x)dx \approx$$

$$\int_{x0}^{x2} f(x)dx \approx \int_{x0}^{x2} (f0 + s\Delta f0 + \frac{s(s-1)}{2}\Delta 2f0)dx$$

=h

$$\int_{0}^{2} (f0 + s\Delta f0 + \frac{s(s-1)}{2}\Delta 2f0)ds$$ = h(2f0+2Δ f0+1/3 Δ 2f0) $= \frac{h}{3}(f0 + 4f1 + f2)$

We get the error by integrating the error of polynomial:

$$\text{Error} = -\frac{1}{90} h^5 f^{(4)}(?) \quad x0 < ? < x2$$

It is convenient to think that the strips defined by successive x-values as panel. For Simpson 1/3 Rule there must be even number of panels.

7.1 Algorithm

1. Read a,b,n
2. Set h=(b-a)/n
3. Set s=[f(a)+f(b)]
4. Set S2=0
5. Set S4=0

6. For I=1 to n-2 step 2

i. Set s4=s4+f(a+ I *h)

ii. Set s2=s2+f(a+(i+1)*h)

7. End forS

6. Set Integral=h/3*(s+2*s2+4*s4)

7 write Integral

7.2. Counting Primitive Operation

1. Reading the value of variable Read a,b,n contributes three unit of count.
2. Assigning values h=(b–a)/n contributes three unit of count.
3. Set sum= [f (a) +f(b)]/2 contributes five unit of count.
5. Set s2=0 and S4=0 contributes two operations
4. The body of loop executes n-2/2 times with count twelve operations in loop. So 12(n-1) /2= 6(n-1) unit of count.
5. Step 7 requires seven operations
6. Write integral contributes one operation.

To summarize, the number of primitive operations t (n) executed by algorithm is at least.

T (n) =3+3+5+2+6(n-1) +7+1

=6n+21-6

=6n+15

So the complexity of algorithm used for Trapezoidal method is O (n). In fact, any polynomial $a^k n^k + a^{k-1} n^{k-1} \ldots\ldots + a^0$ will always be O (n^k)

1.	10	4.05115 8	0.00009 8	.00029 3	**4.05115 8**	0.00009 8	.000191	1.53403 1
2.	100	4.05094 8	-0.00000 2	.00025 8	4.05094 8	-0.00000 2	.000152	1.69736 8
3.	200	4.05094 8	-0.00000 2	.00026 1	4.05094 8	-0.00000 2	.000223	1.17040 4
4.	300	4.05094 8	-0.00000 2	.00039 4	4.05094 8	-0.00000 2	.000254	1.55118 1
5.	400	4.05094 9	-0.00000 1	.00043 1	4.05094 9	-0.00000 1	.000275	1.56727 3
6.	500	4.05095 1	0.00000 1	.00054 9	4.05095 1	0.00000 1	.000431	1.27378 2
7.	600	4.05095 0	0.00000 0	.00058 3	4.05095 0	0.00000 0	.000359	1.62395 5
8.	700	4.05094 9	-0.00000 1	.00062 0	4.05094 9	-0.00000 1	.000409	1.51589 2
9.	800	4.05094 9	-0.00000 1	.00037 5	4.05094 9	-0.00000 1	.000419	0.89498 8
10	900	4.05094 9	-0.00000 1	.00045 2	4.05094 9	-0.00000 1	.000487	0.92813 1
11.	1000	4.05095 4	0.00000 4	.00066 8	4.05095 4	0.00000 4	.000507	1.31755 4

S. No	No. of Intervals	Sequential Execution time & Difference between Result			Parallel Execution time & Difference between Result			Performance/Speed Up(s)
		Integral	Error/ Difference	Time in ms	Integral	Error/ Difference	Time in ms	Difference

Table-3 Performance comparison of Serial and Parallel Algorithm fir integration of sin(x)-log(x)+exp(x)

8. Simpson's 3/8 Rule:

We get The Simpson 3/8 rule by integrating the third degree Newton -Gregory Interpolating polynomial that fit four evenly spaced points and its error term:

$$\int_{x0}^{x3} f(x)dx \approx \int_{x0}^{x3} P3(x5)dx = \frac{3h}{8}(f0+3f1+3f2+f3)$$

$$\text{Error} = -\frac{3}{80} h^5 f^{(4)}(?) \quad X0 < (?) < x3$$

8.1 Algorithm

1. Read a,b,n
2. Set h=(b-a)/n
3. Set s=[f(a)+f(b)]
4. Set S2=0
5. Set S3=0
6. fori=1 to n-3 step 3
   i. Set s2=s2+f(a+(i+2)*h)
7. End for
6. Set Integral=3*h/83*(s1+2*s2+3*s3)
7 write Integral

8.2. Counting Primitive Operation

1. Reading the value of variable Read a,b,n contributes three unit of count.
2. Assigning values h= (b–a)/n contributes three unit of count.
3. Set sum= [f (a) +f (b)]/2 contributes five unit of count.
5. Set s2=0 and S4=0 contributes two operations
4. The body of loop executes n-3/3 times with count fifteen operations in loop. So 15(n-1) /3= 5(n-1) unit of count.
5. Step 7 requires seven operations
6. Write integral contributes one operation.

To summarize, the number of primitive operations t (n) executed by algorithm is at least.

$T(n)=3+3+5+2+5(n-1)+7+1$

$=5n+21-6$

$=5n+15$

So the complexity of algorithm used for Trapezoidal method is O (n). In fact, any polynomial $a^k n^k + a^{k-1} n^{k-1} \ldots\ldots + a^0$ will always be O (n^k)

S. No	No. of Intervals	Sequential Execution time & Difference between Result	Parallel Execution time & Difference between Result	Performance/Speed Up(s)

		Integral	Error/ Difference	Time in ms	Integral	Error/ Difference	Time in ms	Difference
1.	10	4.05115 8	0.00020 8	.00014 9	4.05115 8	0.00020 8	.000129	1.15503 9
2.	100	4.05094 8	0.00000 2	.00015 2	4.05094 8	0.00000 2	.000132	1.15151 5
3.	200	4.05096 7	0.00001 7	.00032 3	4.05096 7	0.00001 7	.000303	1.06600 7
4.	300	4.05095 6	0.00000 6	.00039 4	4.05095 6	0.00000 6	.000354	1.11299 4
5.	400	4.05095 3	0.00000 3	.00067 1	4.05095 3	0.00000 3	.000571	1.17513 1
6.	500	4.05095 1	0.00000 1	.00054 9	4.05095 1	0.00000 1	.000549	1
7.	600	4.05095 0	0.00000 0	.00066 1	4.05095 0	0.00000 0	.000601	1.09983 4
8.	700	4.05094 9	-0.00000 1	.00070 2	4.05094 9	-0.00000 1	.000692	1.01445 1
9.	800	4.05094 9	-0.00000 1	.00080 1	4.05094 9	-0.00000 1	.000701	1.14265 3
10	900	4.05094 9	-0.00000 1	.00081 0	4.05094 9	-0.00000 1	.000710	1.14084 5
11.	1000	4.05094 9	-0.00000 1	.00082 5	4.05094 9	-0.00000 1	.000755	1.09271 5

Table-3 Performance comparison of Serial and Parallel Algorithm fir integration of sin(x)-log(x)+exp(x)

9.Boole's Rule:

$$\int_{x1}^{xn} f(x)dx = \sum_{k=1}^{(n-1)/4} \int_{x4k-3}^{x4k+1} f(x)dx$$

=

$$\sum_{k=1}^{(n-1)/4} h[14 f(x4k-3) + 64(x4k-2) + 24 f(x4k) = 14 f(x4k+1)/45$$

where h = $x_{4k-3} - x_{4k-2} = x_{4k-2} - x_{4k-1} = \ldots = x_{4k+2} - x_{4k+1}$, (n-1)/4 are positive integers.

$$\int_{x1}^{x5} f(x)dx = \frac{2}{45}h(7f(x1)+32f(x2)+12f(x3)+32f(x4)+7f(x5)$$

And error=$-\frac{8}{945}h7f(6)(?)$ where x1<(?)<x5

Using program code in c for sequential and parallel execution we get the results:

	No of iterations	Lower Limit	Upper Limit	Integral Value	Sequential time In ms	Parallel Execution time in ms	Performance/ Speed Up(s)
1.	10	0.2	1.0	2.414793	0.000855	*0.000834*	0.975439
2.	100	0.2	1.36	3.865551	0.022117	*0.018794*	0.849754
3.	1000	0.2	1.396	4.0321589	0.330559	*0.321379*	0.972229
4.	10000	0.2	1.4	4.059064	1.408216	*1.169268*	0.830319

Table-4 Performance comparison of Serial and Parallel Algorithm fir integration of sin(x)-log(x)+exp(x)

Conclusion

There are two version of algorithm: sequential and parallel. The programs are executed on Intel@Core2-Duo processor machine. We analyzed the performance using results and finally derived the conclusions. The Visual C++ compiler 12.0 under Microsoft Visual Studio 12.0 used for compilations and executions. The Visual C++ compiler supports multithreaded parallelism with /Qopenmp flag. In the experiments the execution times of both the sequential and parallel algorithms have been recorded to measure the performance (speedup) of parallel algorithm against sequential.. The data presented in Table 1 to 5 represents the execution time taken by the sequential and parallel programs, Error between mathematica's values of and result and speed up. The result obtained shows a vast difference in time required to execute the parallel algorithm and time taken by sequential algorithm. Based on our study we arrive at the following conclusions:

(1) We see that parallelizing serialalgorithm using Open MP has increased the performance.

(2) For multi-core system Open MPprovides a lot of performance increase and parallelization can be done with careful small changes.

(3) The parallel algorithm is approximately twice faster than the sequential and the speedup is Linear.

References:

[1] Burden, Richard L.; J. Douglas Faires, (2000), NumericalAnalysis (7th Ed. ed.), Brooks/Cole.

[2] Gilat, Amos, (2004), MATLAB: An Introduction withApplications (2nd edition ed.), John Wiley & Sons.

[3] Hildebrand, F. B. ,(1974), Introduction to NumericalAnalysis (2nd edition ed.), McGraw-Hill.

[4] Leader, Jeffery J., (2004), Numerical Analysis andScientific Computation, Addison Wesley.

[5] M. Abramowitz and I. A. Stegun eds., (1972), Handbookof Mathematical Functions with Formulas, Graphs, andMathematical Tables. New York: Dover.

[6] Gil, Amparo; Segura, Javier; Temme, Nico M., (2007),"§5.3: Gauss quadrature", Numerical Methods for SpecialFunctions, SIAM

[7] Press, William H.; Flannery, Brian P.; Teukolsky, Saul A.;Vetterling, William T., (1988), "§4.5: GaussianQuadratures and Orthogonal Polynomials", NumericalRecipes in C (2nd ed.)

[8] Josef Stoer and Roland Bulirsch, (1980). Introduction toNumerical Analysis. New York: Springer-Verlag.

[9] Atkinson, Kendall A., (1989). An Introduction to NumericalAnalysis (2nd edition ed.), John Wiley & Sons.[10] Burden, Richard L. and Faires, J. Douglas, (2000).Numerical Analysis (7th edition ed.), Brooks/Cole.

[11] George E. Forsythe, Michael A. Malcolm, and Cleve B.Moler, (1977), Computer Methods for MathematicalComputations. Englewood Cliffs, NJ: Prentice-Hall.

[12] William H. Press, Brian P. Flannery, Saul A. Teukolsky,William T. Vetterling, (1988), Numerical Recipes in C.Cambridge, UK: Cambridge University Press.

[13] Jon M. Smith,(1974), Recent Developments in NumericalIntegration, J. Dynam. Sys., Measurement and Control96, Ser. G-1, No. 1, 61-70.

[14] Noronha R., Panda D.K., “Improving Scalability of OpenMP Applications on Muti-core Systems

Using Large Page Support, IEEE Computer, 2007.

[15] Kulkarni, S. G., “Analysis of Multi-Core System Performance through OpenMP”, National Conference on Advanced Computing and Communication Technology, IJTES, Vol-1. No.-2, Page.189-192, July – Sep 2010.

[16] Gallivan K. A., Plemmons R. J., and Sameh A. H.,"Parallel algorithms for dense linear algebra computations," SIAM Rev., vol. 32, pp. 54-135, March 1990.

[17] Gallivan K. A., Jalby W., Malony A. D., and Wijshoff H. A. G., "Performance prediction for parallelnumerical algorithms.," International Journal of High Speed Computing, vol. 3, no. 1, pp. 31-62,1991.International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol.2, No.5, October 201264

[18] Horton M., Tomov S., and Dongarra J., "A class of hybrid LAPACK algorithms for multi-core andGPU architectures," in Proceedings of the 2011 Symposium on Application Accelerators in High-Performance Computing, SAAHPC ’11, (Washington, DC, USA), pp. 150-158, IEEE ComputerSociety, 2011.

[19] Wilkinson, B., Allen, M., “Parallel Programming”, Pearson Education (Singapore), 2002.

[20] Barbara, C., Jost, G., Pas, R.V., “Using OpenMP: portable shared memory parallel programming”,The MIT Press, Cambridge, Massachusetts, London, 2008.

[21] Quinn, M. J, “Parallel Programming in C with MPI and OpenMP”, McGraw-Hill Higher Education,2004.