Print ISSN: 1312-2622; Online ISSN: 2367-5357
DOI: 10.1515/itc-2017-0030

Quick Implementation of Definite Integrals Computation with Very High Precision

V. Dzhambov

Key Words: *High precision computation; parallel computations; numerical quadrature; computational mathematics; .NET Framework; X-MPIR.*

Abstract. *An exceptionally effective implementation using parallel computations with two perspective quadrature schemes in a specific environment, namely .NET Framework, is discussed. Comparative tests are carried out demonstrating the possibility for effective computations with very high precision with specific tools of the environment being used. This paper describes a part of the research undertaken for the purpose of creating a tool library for arbitrary precision calculations [1]. The main goal is to demonstrate that very useful and mutually complementing computational tools, solving non-trivial problems with high precision computations may be implemented in a concrete environment, which is wide spread for personal computers, but underestimated by the software developers of scientific applications.*

Introduction

High precision (and the more so as arbitrary precision) computation is not of engineering purpose. Such computations, in particular, concerning definite integrals, are used in so called "experimental mathematics". One of the main reasons is the possibility of applying to the result an algorithm of type "identification of constant", including as an integral part "integer relation detection" algorithm (such as PSLQ), the latter requiring very high precision. Methods here presented are especially designed to use resources and tools, provided by .NET Framework. Effective implementation of two perspective quadrature schemes is discussed – *Clenshaw-Curtis* and *tanh-sinh* schemes. Our goal is to show that effective solution of non-trivial problems can be performed in this environment, which is wide spread for desktop and laptop computers, but underestimated as developing environment for scientific software. The general structure description of the program system involved and some extensions of the underlying main library, using .NET Framework C# + XMPIR, can be found in [1], but also shortly described below as suggested by one of referees. More details can be found in [1], related to quadrature schemes used, as well as key fragments of program code, related to the specific implementations, using parallel computations.

About the Quadrature Schemes Used in the Program System

From practical point of view the problem of finding the numerical value of a given definite integral with a specified precision is almost never reduced to a single-time application of one particular quadrature formula. Probably the greatest advantage for a specific quadrature scheme is the possibility of using data from calculations already previously done. In this sense the quadrature schemes of type tanh-sinh, a variant of the *Double Exponential Transformation*, or of type *Clenshaw-Curtis* have some advantage before the schemes using Gaussian type quadrature. Former may use already calculated abscissae and the values of the integrand in the next levels. At the *tahh-sinh* quadrature already calculated before weights may be used also but this is not of crucial importance. In Gaussian type quadratures rules like the Gauss-Kronrod are a step ahead, nevertheless if repeated calculation with greater and greater accuracy is needed this is not sufficient. In the general case, for different particular problems, the most appropriate type of quadrature scheme is different. *Tanh-sinh* is the best scheme for numerical integration of functions with singularities in the interval ends with requirements for high precision (1000 and more digits, [8]). It and its implementation without parallel computations have been already discussed in [2]. Here will be considered in some more details the Clenshaw-Curtis formula. The heuristic foundation of its exclusive effectiveness for some classes of integrands is the fact, that it may be interpreted like an interpolation quadrature formula not with algebraic polynomials, but with trigonometrical ones. In case when the integrand may be reduced to a smooth periodic function integrated for the interval equal to the period a result of the type "Paley-Wiener theorem" gives reasons exponential convergence to be expected with regard to the number of points in the quadrature formula. Besides if the integrand has n-th derivative with bounded variation (V) in the interval the convergence speed for both – Gaussian and Clenshaw-Curtis formulae for N points is the same, $O\ (V(2N)^{-n})$. This fact refutes existing prejudices, related to the effectiveness of this formula, perceived only as algebraic order of accuracy [4,5]. An up-to-date concept is presented in [3]

In general both schemes used give exponential convergence, when applicable, concerning the number of points in the quadrature formula but the *tanh-sinh* scheme is of greater scope. It so to say regularizes the singularities in at the ends of the interval, transferring them to infinity. On the other hand the *Clenshaw-Curtis* scheme, where applicable, is much more effective. It is not surprising – narrower class of admissible functions – smoothness and bounded variation of derivatives are required.

In the program system being presented a special graphical program tool NQTS, described in [2], as well as two console tools – THSHPar, CCPar are implemented. Parallel computations are used in THSHPar and CCPar. THSHPar uses the Double Exponential Transformation method. CCPar uses the Clenshaw-Curtis quadrature scheme.

Using the Multi Core Architecture

Some preliminary details concerning base tools used for arbitrary precision computations. The possibility to use arbitrary precision computation library (MPIR [9], calling from C# in .Net Framework environment), generated from C source code is crucial (XMPIR, [10]). Detailed description can be found in [10]. As a result, dll files are generated (using mpir.c, mpir.h), enabling calls from C# via binding file (xmpir.cs). Unfortunately, this procedure don't include recently issued MPIR version. Details, below.

Specific modifications. Using quadrature schemes allowing parallel computations to be applied is one of the resources for speedup – the "theoretical" one. However the "practical" one, related to software adequacy, is important too. In this specific case some corrections had to be made in the generation of the dynamic library files of the MPIR library as well as in the wrapper file xmpir.cs.. In the original of 2010 Sergey Bochkanov uses versions of the static libraries generated by a quite old version of MPIR. The latest 2.7.0 is from June, 29, 2015. A possibility is provided libraries to be generated, which to be tuned and optimized for a particular variant of the processor. In this version however signatures of many functions are changed, e.g. in functions with mixed operands – integers with fixed and arbitrary precision, fixed integers with or without sign already correspond to C# types Int64 и UInt64. This imposed multiple changes in files 'mpir.c', used for dynamic library generation and 'xmpir.cs' used for linkage at call by C# code, so that they correspond to the head file 'mpir.h', received at the generation of the static library. The result is generation of dynamic library which is about 70% faster than the original for the particular processor.

Implementation for the tanh-sinh scheme. In the *tanh-sinh* scheme the nodes and weights do not depend on the integrand and the implementation of their parallel computation is almost straightforward. In such type of parallelization however the usage of independent objects and strict following for the manipulation of the current precision of the respective methods is a nicety. It should not handle with changes in the default precision – global variable for the MPIR library. If at some places computations with different precision are necessary this should be done locally at the initialization of the respective variable – mpf_init2 (*precision*).

An object of type *thsh* is in advance formed and includes the necessary lists, void, with abscissae and weights – *thsh.xc* и *thsh.wc*. Finally the effective number of abscissae and weights – *npmax*, necessary further is defined. A class *XW* is created whose basic method *xw* calculates the abscissae and weights in a given interval [*idx1*, *idx2*). It is realized quite economically. Abscissae and weights are

$$x_k = \tanh\left(\frac{1}{2}\pi\sinh(kh)\right),\quad w_k = \left(\frac{1}{2}h\pi\cosh(kh)\right)/\cosh^2\left(\frac{1}{2}\pi\sinh(kh)\right),$$

but *sinh* and *exp* are called only once in each cycle with different arguments. *GetPi()* is repeatedly called but it is implemented in such way that uses a static field in the class *MPMath* so a value is calculated in fact only if a greater precision is required than the one, fixed in the static field.

Then the values of the function are computed in the necessary *npmax* points. Here again parallel computing is used. Method *f_full()* for object of class *MFF* is used whose main purpose is to use this method namely with input data from a list *thsh.xc* already formed, with abscissae of length *npmax*.

A separate object is again used for each problem – *FPS*, very simple in the case, whose method *processing* uses an object of class *Integrand*, independent for each problem, containing the method computing the integrand.

After this preliminary preparation the basic algorithm *tanh-sinh* is used with an essential modification consisting in that, that all values being calculated there are taken from lists which are ready and are not calculated on the spot. These latter computations are reduced to multiplication and addition only and they take a very small part of the time, needed for the through calculation.

Implementation of the Clenshaw-Curtis scheme. First of all the points in the interval [-1,1] are necessary with coordinates $\cos\left(\frac{\nu\pi}{2^n}\right)$, where *n* is the corresponding level and $\nu = 0, 1, \ldots, 2^n$. An exclusively quick implementation is the recursive generation [6], p. 417, using only two calls to the *sin* function.

To avoid loss of precision a little bit higher working precision must be used in calculations. If *digits* correspond to the number of decimal digits, then *digits* + log(*digits*) is enough. No parallel computations are used here. Recursive generation is very quick and takes a very small part of the total calculation time.

For computing the weights a scheme with N*log(N) operations is used [7], where $N = 2^n$ and *n* is the level. The core idea is in generating a vector, using the abscissae received above, for which a quick inverse Fourier transform is used, giving the weights. Here parallel computations are also not used.

The parallel computing of the integrand looks like in the same way as the one described in the *tanh-sinh* scheme. And here again for the calculations giving the successive levels no parallel computing is needed.

Comparative Tests

Tests were carried out on a laptop with the following characteristics: processor Intel (R) Core (TM) i7-3610QM CPU @ 2.30GHz (4 physical processors, 8 logical processors (threads) through hyper-thraeding technologies; up to 3.1 GHz at 4 active processors through Turbo Boost). 16 GB RAM, 64-bit operating system Windows 7 Enterprise (Microsoft Windows NT 6.1.7601 Service Pack 1). In *tables* below it is marked as TL (test laptop).

For a base of the investigation is accepted [8]. There 14 exemplary integrals are given computed with precision to 2000 decimal digits. The results received are compared between the proposed and developed program tools THSHPar, a scheme with parallel computing for tanh-sinh quadratures and CCPar, a scheme with parallel computations for Clenshaw-Curtis quadrature to some of the results received in [8]. As the program CCPar uses a Clenshaw-Curtis quadrature scheme data for it are shown only where it is effectively applicable. A necessary condition is that the integrand and its derivatives are continuous, with finite values for the whole interval.

Table 1. Comparative results for THSHPar and the set of examples in [8]

		Number of processors			
Example	**Level required**	**4, [8]**	**16, [8]**	**1024, [8]**	**4, THSHPar+TL**
Initialization for level 13		1085.34	271.87	5.02	382.31
1	10	101.63	25.25	0.53	6.11
2	10	294.32	74.04	1.54	129.03
3	10	317.01	79.69	1.83	44.75
4	10	328.73	82.13	1.63	130.31
5	9	51.62	12.90	0.30	4.69
6	10	5.62	1.42	0.05	0.43
7	10	11.46	2.87	0.10	0.65
8	9	50.98	12.85	0.27	4.70
9	10	333.24	83.60	1.84	17.54
10	10	245.45	61.39	1.44	11.84
11	11	5.17	1.30	0.04	1.67
12	12	161.99	40.71	0.80	112.47
13	13	216.50	54.13	0.97	214.40
14	13	1826.02	457.02	7.87	309.41
Total, no initialization		3949.74	989.6		988.00
Correction for hardware in [8]	1/[(3.1/2)* 1.3]	1960.17	492.12		988.00

In the last row a correction is added concerning the maximum possible acceleration related to the hardware used in [8], having in mind the maximum achieved clock frequency with Turbo Boost and the maximum percentage of improvement of about 30% provided by the hyper-threading technology in TL. Here only evident factors are considered. The speed of the main RAM memory and the processor cache may influence, but we have no data to compare to [8]. At such data not including the initialization time in both implementations – the one from [8] and the presented here, a rough account (492,12*2 = 982,24 ~ 988.00) shows that results of the present research by parallelization to 4 processors of TL are equivalent to 8 processors of the basic scheme in [8]. Without the correction the result is equal to 16 processors from [8].

For example 14 at 13 levels (*table 1*) wanted 2000 digits are not reached but up to 1971, both in [8] and THSHPar+TL. It is stated in [8] that these 2000 digits are reachable with 14 levels showing no data. This is really true. THSHPar+TL reach this precision with 14 levels in 620.90 calculations plus 768.55 seconds initialization for 14 levels.

Table 2. Comparative results for CCPar and a part of the examples in [8]

			Number of processors					
Example	**Level required in [8]**	**Level required in CCPar**	4, [8]	16, [8]	64, [8]	256, [8]	1024, [8]	4, CCPar +TL
Initialization for level 13)/8, i.e. for level 10			135.67	33.98	8.61	2.22	0.63	0 Total time given below
1	10	12	101.63	25.55	6.45	1.65	0.53	1.84
2	10	12	294.32	74.04	18.83	4.99	1.54	10.95
3	10	10	317.01	79.69	20.42	5.24	1.83	1.23
4	10	12	328.73	82.13	20.84	5.52	1.63	10.99
Total, no initialization			1041.69	261.41	66.54	17.40	5.53	25.01
Total with init. each time, i.e. + (row #3)*4			1584.37	397.33	100.98	26.28	8.05	25.01
Total with hardware correction			786.29	197.19	50.11	13.04	4.00	25.01

Where the parallel Clenshaw-Curtis quadrature is applicable (*table 2*) CCPar + TL with 4 processors is equivalent to about 128 processors of the system used in [8]. This is with the hardware correction as above. Otherwise this equals to about 256 processors. Here however another scheme is applied and such a comparison is somewhat speculative.

There is a difference which is to be noted in the denotation of the levels in the two types of schemes. In the *tanh-sinh* scheme level k means $20*2^k$, and in the *Clenshaw-Curtis* scheme level k means 2^k. The computational method used here for weight calculation including fast inverse Fourier transform requires the number of points to be a power of two.

Discussion

Solving of a particular problem, as above mentioned, depends on many factors. At comparison some of them are unknown. Even the way of implementation of the elementary functions participating in the expression for the integrand is very important. As an example may be given the evolution of the program system considered here. It was mentioned in [2] about the program tool NQTS that the example given for integral $\int_0^{\pi/2} \frac{\arcsin\left(\sqrt{2}/2 \cdot \sin x\right)\sin x}{\sqrt{4-2\sin^2 x}}\,\mathrm{d}x$ whose analytical valuation hampers the wide spread specialized environments for mathematical calculations, after "long computations" gives the result with precision 1000 decimal digits. These "long computations" at that moment, about two and a half years before THSHPar and CCPar were developed, took more than 4 hours and a half. THSHPar solves this problem in 49.51 sec. (11 levels required, i.e. $N = 20*2^{11} = 40960$). From them – 9.53 sec for initialization of abscissae and weights. Computation of the integrand in the necessary *npmax*=8177 points – 39.90 sec. The difference is even greater for the achieved result with CCPar for the same problem – 3.87 sec total, 11 levels required, i.e. $N = 2^{11} = 2048$ points). The great difference is due to improvements of all factors taking part as multipliers in the result for time. For THSHPar no interpreter is used. Increased precision is used in the scheme when this is necessary and the number of the necessary points for computing the integrand is decreased – the condition for interruption in the basic scheme of the algorithm forming parameter *npmax*. Parallel computing is used. There are improvements in the implementation of the function atan, asin is calculated through it, and in the trigonometric functions. The library XMPIR itself implements a newer version of MPIR. Besides, what is even more important, it is optimized for the concrete processor. All this, except the improvement of the scheme for *npmax* calculation, is valid for CCPar also. Something more, this scheme is more appropriate for the particular example. This does not belittle the qualities of the *tanh-sinh* scheme. It is applicable where the *Clenshaw-Curtis* scheme is not. Where there are singularities at the end of the interval, the *tanh-sinh* transform "regularizes them sending them to infinity". Something the *Clenshaw-Curtis* scheme cannot do in its classic form. Going back to the basic list of examples it can be noted however, that CCpar can solve example 13 and 14 too. Using the transform $\int_{-1}^{1} f\left(\frac{2x}{1-x}+1\right)\frac{2}{(1-x)^2}\,\mathrm{d}x = \int_0^{\infty} f(x)\,\mathrm{d}x$, CCPar solves these examples for 60.62 and 652.43 sec respectively. 16 and 19 levels respectively are necessary. From formal point of view, the problem is of course in the fact that the interval for the examples is infinite and although the integrand and the derivatives behave well in it, the valuation of the error, including as a multiplier the interval length becomes undefined. Of course, transforming to finite interval can't resolve this problem, but simply move it elsewhere. For example 14 this is almost evident. At the transform the integrand's derivatives are not of bounded variation. Integrals of such type require a special approach for their effective solving.

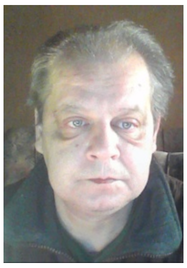
Conclusion

An effective implementation of very high precision computations of definite integrals in a wide spread environment is described. Speedup achieved is due to various factors, including: 1) Appropriate choice of quadrature schemes, allowing effective applying of parallel computations; 2) Optimization of the basic program instruments for high precision computations, adapting them to the specific hardware, and 3) Effective implementation of basic mathematical functions. Program tools are developed for computing definite integrals with high precision, THSHPar and CCPar. Tests are carried out for this implementation demonstrating the possibility for effective computations with very high precision with specific tools of the environment being used, including the multi-core processors massively built in in the modern desk top and lap top computers.

References

1. Dzhambov, V. http://www.iict.bas.bg/konkursi/2017/VDjambov/disertacia.pdf.
2. Dzhambov, V. High Precision Computing of Definite Integrals with .NET Framework C# and X-MPIR. – *Cybernetics and Information Technologies*, 14, 2014, No. 1, 172-182.
3. Reid, H., Clenshaw-Curtis Quadrature. Online, http://homerreid.dyndns.org/teaching/18.330/Notes/ClenshawCurtis.pdf.
4. Trefethen, L. N. Six Myths of Polynomial Interpolation and Quadrature. Online, https://people.maths.ox.ac.uk/trefethen/mythspaper.pdf.
5. Trefethen, L. N. Is Gauss Quadrature Better than Clenshaw–Curtis? – *SIAM Review*, 50, February 2008, No. 1, 67-87.
6. Arndt, J. Matters Computational. Springer, 2011.
7. Waldvogel, J. Fast Construction of the Fejér and Clenshaw–Curtis Quadrature Rules. – *BIT Numerical Mathematics*, 46, March 2006, No. 1, 195-202.
8. Bailey, D., J. Borwen. Highly Paralell, Highly-Precision Numerical Integration. 2008, http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf.
9. MPIR site, http://mpir.org/.
10. XMPIR Web Location. http://www.alglib.net/x/xmpir/xmpir-0.2.zip.

Manuscript received on 23.06.2017



Velichko Dzhambov *was born in 1960 in Yambol, Bulgaria. M.S. degree in 1985 in physics from Sofia University "St. Kliment Ohridski". Researcher in Institute of Engineering Cybernetics and Robotics – BAS since 1988 and its later heirs till the present Institute of Information and Communication Technologies. Ph.D. in informatics and computer sciences since 2017. Interests – high precision computation, numerical analysis, nonlinear optimization.*

Contacts:
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Acad. G. Bonchev St., bl. 2, 1113 Sofia
e-mail: vili_jambov@abv.bg