UNIVERSITÄT SALZBURG

Gridification of the Solution of highdimensional improper Integration and Integral Equations

Peter Zinterhof Peter Zinterhof jun.

Technical Report 2010-01 March 2010

Department of Computer Sciences

Jakob-Haringer-Straße 2
5020 Salzburg
Austria
`www.cosy.sbg.ac.at`

Technical Report Series

Gridification of the Solution of highdimensional improper Integration and Integral Equations

P. Zinterhof + P. Zinterhof jun. - Salzburg University

Abstract. *The scope of the paper is the development of new methods for high dimensional numerical integration of unbounded functions, the solution of integral equations and the direct search for good lattice points in a grid environment. From the mathematical point of view we use the Zinterhof sequences for the removal of singularities by means of certain integral transforms. The algorithms for numerical integration (by summation and convolution) and a brute force search for good integration numbers have been parallelized for use in the grid environment. We also show the high suitability of novel computing machinery, such as the STI Cell processor and CUDA-enabled devices (NVIDIA GPUs) for these tasks.*

1. The mathematical Methods

1.1. Numerical Integration Methods avoiding the Curse of Dimensionality

Consider any classical integration method

$$\sum_{n=1}^{N} W_n f(x_n) - \int_0^1 f(x)dx = R_N(f) \tag{1}$$

A straight forward generalization of (1) to s dimensions is the cartesian product rule

$$\sum_{n_1=1}^{N}\sum_{n_2=1}^{N}\cdots\sum_{n_s=1}^{N} W_{n_1}W_{n_2}\dots W_{n_s} f(x_{n_1},x_{n_2},\dots,x_{n_s}) - \int_0^1\cdots\int_0^1 f(x_1,\dots,x_s)dx_1\dots dx_s = R_N(f) \tag{2}$$

The computational time grows as N^s. If e. g. the computation of an onedimensional integral (1) using $N = 10^6$ nodes deserves one second, the computation of the correspondending 10-dimensional integral deserves 10^{60} seconds. This is called the "curse of dimensionality". The theory of uniform distribution of sequences provides best possible methods, essentially avoiding the curse of dimensionality.

We provide in this paper methods for the computation of improper integrals, where the singularities are on the boundary of the s-dimensional unit cube $E_s = [0, 1]^s$. It is supposed, that the integrand $f(x_1, \dots, x_s)$ is smooth (differentiable) in $(0, 1)^s$.

We propose in this summary some integration methods which are essentially best possible. A comprehensive expository source is [Drmota and Tichy, 1997]. The rather extended proofs will be presented in a forthcoming full paper.

Let $\{x\}$ be the fractional part of the real number x, $\{x\} = x - \lfloor x \rfloor$, $\lfloor x \rfloor$ being the integer part of x. We use the integration nodes (vectors $\in \mathbb{R}^s$).

$$\left(\{ke^1\}, \{ke^{1/2}\}, \ldots, \{ke^{1/s}\}\right), \; k = 0, \pm 1, \ldots, \pm(N-1) \tag{3}$$

The proof of the quality and efficiency of nodes of this type, which are frequently called Zinterhof sequences [Zinterhof, 1969], [Hofbauer et al., 2006], [Zinterhof, 2002], uses deep results of the Fields medallist Allan Baker e. g.

Let $s(x)$ be a growing and differentiable function, such that $s(0) = 0,\ s(1) = 1,\ s'(0) = s'(1) = 0$. The functions $s_1(x) = \frac{1}{2}(1 - \cos \pi x),\ s_1'(x) = \frac{\pi}{2}(\sin \pi x)$ and $s_2(x) = x - \frac{1}{2\pi} \sin \pi x,\ s_2'(x) = 2 \sin^2 \pi x$ are useful examples of such functions.

We use the weights $W_{N,k} = (N - |k|)/N^2,\ k = 0, \pm 1, \pm 2, \ldots, \pm(N-1)$ and consider the integration method

$$I_N(f) - I(f) = R_N(f), \tag{4}$$

$$I_N(f) = \sum_{k=-(N-1)}^{N-1} \frac{N - |k|}{N^2} f(s(\{ke^1\}), s(\{ke^{1/2}\}), \ldots, s(\{ke^{1/s}\})) \prod_{l=1}^{s} s'(\{ke^{1/l}\}). \tag{5}$$

If the function $f(x_1, \ldots, x_s),\ 0 < x_i < 1,\ i = 1, \ldots, s$ fulfills mild differentiability conditions, then the error term $R_N(f)$ of the integration process tends fast to zero:

$$R_N(f) = \mathcal{O}\left(\frac{1}{N^{\alpha - \varepsilon}}\right), \tag{6}$$

where α is depending on the smoothness of $f(x_1, \ldots, x_s)$ and arbitrary $\varepsilon > 0$. The result can not be better than $R_N = \mathcal{O}(\frac{1}{N^\alpha})$. So, the proposed method is essentially best possible. We propose the following set of test functions:

1.1.1 $f_1(x_1, \ldots, x_s) = (\max(x_1, \ldots, x_s))^\beta,\ \beta > -1$

with $I(f) = s/(s + \beta)$. We remark that the function $f_1(x_1, \ldots, x_s)$ is not separable, i. e. it is not a product of functions of fewer variables.

1.1.2 $f_2(x_1, \ldots, x_s) = (x_1 x_2 \ldots x_s)^{-0.5},\ I(f_2) = 2^s$

1.1.3 $f_3(x_1, \ldots, x_s) = \prod_{l=1}^{s} (\ln(\frac{1}{x_l}))^{-0.5},\ I(f_3) = \pi^{\frac{s}{2}}$

1.1.4 $f_4(x_1, \ldots, x_s) = \prod_{l=1}^{s} (1 - \frac{\pi^2}{6} + \frac{\pi^2}{2}(1 - 2\{x_l\})^2),\ I(f) = 1$

The function f_4 is the product of normalized Bernoulli polynomials of other two. It is essential for the definition of the so-called diaphony of a sequence. The diaphony measures the uniform distribution of a sequence and is a very useful estimator for the integration error.

The (Zinterhof-) nodes $(\{ke^1\}, \{ke^{1/2}\}, \dots, \{ke^{1/s}\}),\ k = 0, \pm 1, \pm 2, \dots$ have three very valuable advantages: the nodes provide fast integration methods. The previos nodes are not to be altered, if one increases the number N of nodes to be useful. And, last but not least, the nodes are readily computed for each N and every dimension s. On the other hand, the so-called good lattice points (i. g. optimal coefficients) or the so-called (t, m, s)-nets provide slightly smaller error terms, at least in theory. Unfortunately, both types of nodes deserve in general remarkable computing effort to be computed. So, there exist extensive tables of good lattice points and of (t, m, s)-nets, which are necessarily restricted to special N and s. The nodes used by us need no tabulation, because they are computed for any number N of nodes and any dimension. Nevertheless we present a grid version for brute force search for good lattice points.

1.2. Application of the proposed Integration Methods to Integral Equations Fredholm II

We consider the approximate solution of the Fredholm Integral Equation of second type in dimensions:

$$\varphi(P) = f(P) + \lambda \int_{G_s} K(P, Q)\varphi(Q)dQ, \tag{7}$$

where $G_s = [0,1]^s$ and $\lambda \in \mathbb{C}$ is the parameter. The function $f(P)$ and the kernel $K(P, Q)$ are known. We choose nodes $Q_1, \dots, Q_N$ and solve the linear system

$$\psi_k = f(Q_k) + \lambda \frac{1}{N} \sum_{n=1}^{N} K(Q_k, Q_n)\psi_n,\ k = 1, \dots, N. \tag{8}$$

The solution is unique for small $|\lambda|$ at least. The approximate solution $\varphi_Q(P)$ is defined by

$$\varphi_Q(P) = f(P) + \lambda \frac{1}{N} \sum_{n=1}^{N} K(P, Q_n)\psi_n. \tag{9}$$

It is well known, that the error term $R_N = \varphi(P) - \varphi_Q(P)$ has the same order of magnitude as the error term in the corresponding integration process, where good estimations for the $\mathcal{O}$-constants are available. Nevertheless, in higher dimensions one needs many nodes $Q_1, \dots, Q_N$ to achieve a reasonable small error term in (9). So, the solution of the linear system (8) is crucial, say for $N = 10^6$. We consider therefore two subtypes of Fredholm II, namely the convolution type and separable type

1.2.1 The convolution type

$$\varphi(P) = f(P) + \lambda K(P)\varphi(P) \tag{10}$$

In this case the linear system (8) is circulant and will be solved by three FFT. So the computing time reduces from $\mathcal{O}(N^3)$ to $\mathcal{O}(N \ln N)$.

1.2.2 The separable type

Here are kernels $K(P, Q) = \sum_{l=1}^{L} A_l(P)B_l(Q)$ considered, where $L \ll N$. The crucial point of the numerical solution in that case is the numerical integration of the products $A_l(P)B_m(Q),\ l,m = 1,\dots,L$, and the solution of a $L \times L$ linear system. The computational effort is in this case of order $N \times (L^2 + L^3)$, which is much less than N^3 in the general case.

1.3. Optimal nodes for integration

Let

$$H(z) = \frac{1}{p}\sum_{k=1}^{\frac{p-1}{2}} (1 - 2\{\frac{k}{p}\})^2 (1 - 2\{\frac{kz}{p}\})^2 (1 - 2\{\frac{kz^{s-1}}{p}\})^2 (11)$$

with $\{x\}$ =fracpart$(x), 0 \leq \{x\} < 1,\ \forall x$
now, find the minium of $H(z)$ within the range of integer values of z with $1 \leq z \leq \frac{p-1}{2}$. If this minimum is considered at $z = a$, $H(a) = \min_{1 \leq z \leq \frac{p-1}{2}} H(z)$, then the P vectors
$(\{\frac{k}{p}\}, \{\frac{ka}{p}\}, \{\frac{ka^2}{p}\}, ..., \{\frac{ka^{s-1}}{p}\}),\ k = 1, .., P$ are optimal nodes for numerical integration and related problems.

2. Implementation for the grid

As shown above, numerical integration requires large amounts of floating-point operations. Furthermore, the quality of the results is proportional to the number of integration nodes, which is the motivation for the use of very high numbers of integration nodes. To make the computation of numerical integration more feasible for users without excessive hardware resources, we provide a framework of computer programs, which allow the computation of such tasks within a grid environment. Thus, the user can easily make use of the greater hardware resources offered by the grid, while being freed from the hassles of programming and tuning his algorithms for parallel execution. Another benefit of our implementation of numerical integration routines for the grid is the user's possibility to do more computations per time interval (e.g. repeat an integration for many different numerical settings) and to get better precision of these integrations, due to the possibly higher numbers of integration nodes. During the project we encountered many new hardware developments, such as the Sony/Toshiba/IBM Cell Broadband Engine Architecture (Cell processor) and NVIDIA'a general purpose graphics processing units (gpgpu, [NVIDIA, 2008]). Besides many other interesting accelerator plattforms (e.g. Convey's systems of tightly intergrated Xeon/fpgas and Sicortex systems), those two plattforms, while currently not being in the mainstream of typical grid hardware, look rather promising to us for two reasons: First, they are affordable, which makes them excellent candidates for building larger high performance cluster systems, that can be intergrated into grid environments. Second,they offer tremendous performance for single precision floating-point operations and very good performance on double precision codes, compared to the ubiquitous Intel x86-based systems. We therefore ported most of our algorithms to Cell and GPU platforms and show the beneficial role they can play in numerical integration tasks.

In the following we give an overview of the concepts of gridification and implementation details for the three main chapters (1.1 - 1.3). Each section is accompanied by practical results (e.g. precision) and benchmarks.

2.1. Numerical integration

The numerical integration of Fredholm type II integrals requires large summations of function evaluations. Due to the fact, that the error decreases in proportion to the number of integration nodes, we aim for very large - usually in the range of billions - integration node numbers. To achieve reasonable execution times we "gridified" the former sequential algorithm by dividing the summation into smaller blocks, that can be executed in parallel within the grid environment. During the test runs we encountered the - practically anticipated - effect of uneven load-balance. This effect results from different computational power, different hardware architectures or different utilization of the grid hardware and it leads to sub-optimal execution times, in that some grid nodes finish their blocks of work earlier than others and staying idle for the rest of the time. We therefore implemented a simple but rather effective scheme of automatic load-balancing, that doesn't require any previous knowledge of the current utilization of the grid nodes, at the expense of a slightly increased network traffic and very decent administrative overhead at the main node.

The code consists of an spmd-style (single program, multiple data) program, that is based on the MPICH2 (message passing interface) library. It therefore can easily be run on local clusters and grid-enabled machine sets, that support some form of MPI. The program master is started on one machine and replicates itself within the grid as $N-1$ worker nodes, resulting in N active processes. Process 0 (master) partitions the sum consisting of S sumands (e.g. $S = 1.000.000.000$) in k evenly sized blocks b_k (intervals) of size l, such that $S = kl$ (figure 1, upper row). Each worker then keeps asking for the next block to compute its local sum. When the worker has finished its local job of summing up the function evaluations within the given interval, the sum is returned to the master which adds it to the global sum. In order to reduce communications overhead between master and worker, the returning of some local sum triggers the master to send out the next interval to the very same worker node that has just finished. The underlying rationale of the scheme is to provide faster nodes with more work than slower workers. Without complicated monitoring this can be achieved by the described demand-driven method of load balancing.

To further improve performance we added support for CUDA devices (Compute Unified Device Architecture). Currently, NVIDIA is offering these devices either as high performance computing nodes (accelerator boards) or in the form of CUDA-enabled general purpose graphics processing units (gpgpu, or 'GPU' in short). Both types differ in main memory size and computing capability. In this paper we refer to GPUs. CUDA devices are massively parallel systems, which offer hundreds of tightly integrated processing elements within a large hierarchical memory system. They are programmed by way of the SIMD (single program, multiple data) method, which imposes certain limitations, such as the need for a great amount of data parallelism. Also, in order to effectively use the computational capabilities of GPUs, one has to employ thousands of parallel threads, so that the scheduler of the device can hide memory latencies and shortage of (fast) local memory. We chose a CUDA grid size of 256 blocks and 256 threads in order to supply enough tasks for the GPU hardware, which is capable of running more than 12000 tasks concurrently. For the most efficient usage of the GPU hardware, the size b_k of each distributed block within the mpi-layer has to be a multiple of $256 * 256 = 65536$. So the smallest possible number of integration nodes N on GPU type hardware would be $N = 65536$ (figure 1, bottom row).

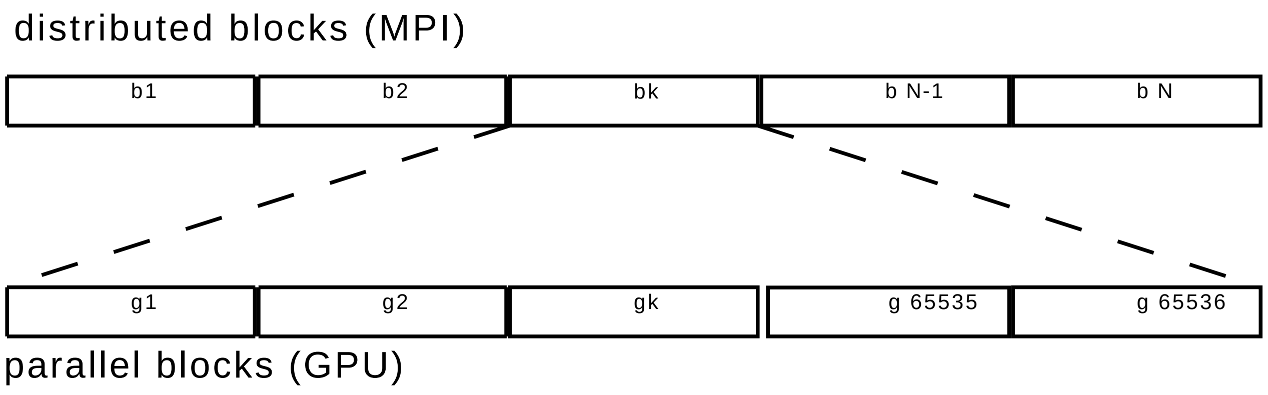

Figure 1. blocking scheme on global (MPI) and local (GPU) level

integration nodes	1 x86	4 x86	1 GPU (295)	4 GPU (295)	1 GPU (280)	error
65.536	0.15 s	0.5 s	4.65 s	5.05 s	5.6 s	$4.0 * 10^{-5}$
655.360	1.20 s	1.3 s	4.68 s	4.95 s	5.7 s	$1.02 * 10^{-6}$
8.388.608	19.13 s	5.1 s	4.89 s	5.07 s	6.04 s	$4.63 * 10^{-8}$
67.108.864	153.7 s s	39.7 s	7.05 s	5.76 s	8.54 s	$3.77 * 10^{-11}$
536.870.912	1240.5 s	315.1 s	22.59 s	10.07 s	28.71 s	$1.06 * 10^{-11}$
1.073.741.824	2455.7 s	622.8 s	40.42 s	14.34 s	51.72 s	$1.4 * 10^{-12}$
2.147.483.648	4925.0 s	1256.5 s	94.85 s	24.10 s	97.80 s	$9.1 * 10^{-13}$
4.294.967.296	9899.22 s	2467.2 s	147.19	41.84 s	158.80 s	$1.3 * 10^{-12}$

Table 1. execution times in seconds, example 1.1.2, s=5 (295=NVIDIA 295GTX, 280=NVIDIA 280GTX

The following performance results (table 1) were obtained on Intel Q6600 (2.4GHz) and NVIDIA GPU type machines. They indicate good scalability in the relevant range of node numbers ($N > 500.000.000$), which is key for the use within a dynamic grid environment. The sub-linear speedup for GPU-versions and low integration node numbers stems from CUDA-process startup times. For low integration node numbers the cpu versions show better (linear) scalability.

The mathematical precision of the integration is even better than expected. If we consider double precision numbers with a precision of 15 digits and the smallest possible error of 10^{-15} in each part of the computation, then by averaging out every second error term and a node number of approximately $2*10^9$ ($\approx 2^{31}$) we would expect an overall error of $2*\frac{10^9}{2}*10^{-15} = 10^{-6}$. Thus we would get about $15-5 = 10$ digits of precision, instead we achieve 12 digits of precision, which is two orders of magnitude better than expected.

2.2. Convolution type

The convolution type of numerical integration (10) is based on two large 1-dimensional discrete fourier transforms, which are inherently hard to parallelize on distributed memory systems. On typical grid sites we often encounter clusters of workstations, that are capable of massive numbers floating-point operations per second but suffer from rather weak system interconnects. So we didn't try to speed up the single FFT-operation but we built a framework to compute many such FFTs in parallel. This approach seams reasonable since many users might be interested in fast integration of many functions, rather than to solve a single integration in the

integration nodes	Intel Q6600 (2.4GHz)	280 GTX	speed up	error variance
1048576	2.58 s	0.617 s	4.18	$4.89*10^{-6}$
2097152	5.39 s	1.07 s	5.03	$1.31*10^{-6}$
4194304	11.51 s	2.03 s	5.66	$3.03*10^{-7}$
8388608	24.60 s	3.78 s	6.50	$1.26*10^{-7}$

Table 2. convolution (3x 1d-FFT): execution times in seconds

shortest possible time.

The performance values (table 2) were obtained for a 5-dimensional function on the Intel Q6600 (one core) and the NVIDIA 280 GTX GPU with 30 multiprocessors. The speedup calculation is based on the timing ratio (wall time)between the two compared systems.

We have to note that the GPU is only used for the FFT-part of the computations, which leaves the initial setup of the coefficient arrays to the cpu. In the final version of the code the setup process will be moved completely onto the GPU, which will further increase the speedup in favor of the GPU version.

2.3. Finding optimal nodes for numerical integration

As described in 1.3. optimal nodes for numerical integration can be found by a brute force computation of all $H(z)$ with $1 \leq z \leq \frac{p-1}{2}$. In other words: We have to compute a set of $\frac{p-1}{2}$ integrations and find the minimum value within this set. We chose the outer loop for parallelization, which reduces the communications overhead to $\frac{p}{2}$ task definition messages (master $\Rightarrow$ workers) and $\frac{p}{2}$ result messages (workers $\Rightarrow$ master). The algorithm has been implemented on standard x86 type cpu and the STI Cell processor, which is used in the SONY PS3 gaming console. The program for the Cell processor employs so-called spulets, which are stand-alone programs that can be executed directly on the synergistic processing elements and communicate with the calling code by way of a pipe. We now treat a single Cell system (PS3) as 6 separate processing elements, that is, from the MPI point of view, we launch 6 MPI-jobs on the ppu of every PS3. Each MPI-worker then acts as a gateway between the master node (by sending/receiving MPI-pakets) and one single spu-processor (Unix pipe). This approach enables seemless integration of the Cell's spu chips within our mpi-framework, including good load-balancing not only within the 6 on-chip spus, but also within distributed Cell chips.
Preliminary timing data show that the tremendous performance of GPUs can be fully utilizied for this brute force search algorithm, too.

Cell is especially well optimized for single precision calculations within streaming multimedia applications. Despite the fact that the brute force search requires double precision operations, Cell delivers deliberately higher performance than the Intel Xeon quad core chip that we also used for our benchmark runs. It seems especially noteworthy that a single PS3 gaming console with only 6 synergistic processing elements achieves a performance in the range of 2/3 of a dual quad-core Xeon server (table 3).

nodes p	1 Xeon 2.3 GHz	1 PS3 Cell	8x Xeon 2.3 GHz	8x PS3 Cell
10000	3.03 s	1.17s	1.03 s	2.97 s
20000	12.3 s	2.83 s	1.76 s	3.24 s
30000	27.3 s	5.21 s	3.90 s	3.56 s
40000	47.8 s	9.28 s	6.31 s	3.77 s
50000	74.0 s	13.66 s	9.75 s	4.29 s
60000	105.9 s	19.42 s	13.72 s	5.11 s
70000	143.4 s	26.80 s	18.62 s	5.88 s
80000	191.8 s	34.51 s	24.69 s	6.86 s
90000	235.5 s	43.18 s	30.03 s	8.07 s
100000	290.0 s	52.85 s	36.81 s	9.31 s
150000	647.9 s	117.78 s	81.42 s	17.47 s
200000	1147.7 s	207.18 s	144.34 s	28.62 s

Table 3. brute force search: execution times in seconds, dimension s=5

3. Conclusions

Regarding the application of our algorithms within the grid environment, we show good scalability and thus high suitability for the grid. Scalability is better for x86-type machines than for novel architectures such as the Cell chip and the GPU. This is a result of the more complicated startup routines, which include loading of drivers, calling of GPU-kernels and also from the rather low performance of Cell's ppu (power-pc unit). Nevertheless, the use of grid services along with novel computer architectures enables extremely high performance for the computation of numerical integrations and we believe that the integration of those architectures within the grid environment is a very promising step forward. Grid computing and the use of cutting-edge architectures significantly extends the area of practical computability of integration based numerical problems.

4. Acknowledgement

We want to thank G. Haase (University of Graz) for his support by giving access to his department's grid hardware as a very valuable extension of our local environment.

References

[Drmota and Tichy, 1997] Drmota, M. and Tichy, R. F. (1997). *Sequences, Discrepancies and Applications.* Springer.

[Hofbauer et al., 2006] Hofbauer, H., Uhl, A., and Zinterhof, P. (2006). A pragmatic view on numerical integration of unbounded functions. In Keller, A., Heinrich, S., and Niederreiter, H., editors, *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 512–528,. Springer.

[NVIDIA, 2008] NVIDIA (2008). *NVIDIA CUDA Compute Unified Device Architecture, Programming Guide.* NVIDIA Corporation.

[Zinterhof, 1969] Zinterhof, P. (1969). Einige zahlentheoretische methoden zur numerischen quadratur und interpolation. In *Sitzungsberichte der Österreichischen Akademie der Wissenschaften, math.-nat.wiss. Klase Abt. II*, pages 177:51–77.

[Zinterhof, 2002] Zinterhof, P. (2002). High dimensional improper integration procedures. In Technical University of Košice, C. E. F., editor, *Proceedings of Contributions of the 7th International Scientific Conference*, pages 109–115, Hroncova 5, 04001 Košice, SLOVAKIA. TULIP.