Peter Zinterhof and Peter Zinterhof jun.
RIST++, Salzburg

Searching for Good Lattice Points in parallel

(Contribution No 2 to NEWTON)

0 Goals

The NEWTON–(New Technology of Numerics–) Project is devoted to the development and practical applicability of NTN (Number Theoretical Numerics). NEWTON encompasses theoretical investigations and implementations onto parallel computers. This report is No 2 of a series which is intended to cover most of the relevant high dimensional NTN–algorithms both in theory and with respect to practical implementations.

1 Mathematical Background

The mathematical background of NTN is founded in a celebrated paper of Hermann Weyl [?] on the uniform distribution of sequences. Subsequently a vast literature arose in the field of equidistribution of sequences and the importance of the theory for Number Theory, Harmonic Analysis, Statistics, Topology and others became clear. In the sixties the austrian mathematician E. Hlawka and the russian mathematician N. M. Korobow first applied these concepts to high dimensional numerics. One basic question of Number Theoretic Numerics is the following: Given a class E of functions $f(x_1, \ldots, x_s)$ in the s–dimensional unit cube $I^s = [0,1)^s$

$$f : I^s \longrightarrow \not\subset, \tag{1}$$

one has to find points $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N$, N beeing large, such that

$$R_N(f) = \frac{1}{N}\sum_{k=1}^{N} f(\underline{x}_k) - \int_{I^s} f(\underline{x})d\underline{x} \tag{2}$$

is small. For analytical reasons the classes E of functions under consideration have to be (pre–)compact in a suitable topology, e. g. in the topology induced by the max–norm for continuos functions. It is an interesting fact that this classical problem of high dimensional numerics is solved by means of number theoretical methods. So number theory provides optimal solutions of the above problem and as a spin off best quasirandom sequences for the different kinds of simulation are supplied. The known techniques for the construction of good highdimensional integration rules come from continued fraction theory, diophantine approximations, geometry of numbers and other branches of number theory. Because of their historical rôle and because of both simplicity of application and good practical results we concentrate in this paper on the so called “good lattice points” or “optimal coefficients”. It is very well known that many classical problems of applied mathematics can be solved by means of high dimensional integrations. So the methods described in this paper apply for example to integral equations, approximation theory, stochastic processes, PDE’s and other.

Now we describe in a more or less expository way the principles of the method of optimal coefficients:
Given an absolute convergent Fourier series

$$f(x_1,\dots,x_s)=\sum_{m_1,\dots,m_s=-\infty}^{\infty} C(m_1,\dots,m_s)\exp(2\pi i(m_1x_1+\dots+m_sx_s)), \tag{3}$$

find good nodes $\underline{x}_k=(x_1^k,\dots,x_s^k)$, $k=1,\dots,N$ such that

$$R_N(f)=\frac{1}{N}\sum_{k=1}^{N} f(x_1^k,\dots,x_s^k)-\int_0^1\dots\int_0^1 f(x_1,\dots,x_s)dx_1\dots dx_s \tag{4}$$

will be small. However the class of functions having an absolute convergent Fourier series is not compact even in L_2–topology and no ball in this class is (pre–)compact in any topology suitable for purposes of numerical mathematics. So one has to specialize to more suitable classes of functions with useful smoothness conditions. There is apparently a transfinite set of possibilities in defining "reasonable" classes of functions. So we adopt in this paper the classes $E_s^\alpha(C)$ which minimize the technical problems and which maximize insight and practical usefulness. Let $\overline{m}=max(1,|m|)$, then one defines for $\alpha>1$, $C>0$

$$f(x_1,\dots,x_s)\in E_s^\alpha(C) \tag{5}$$

iff

$$|C(m_1,\dots,m_s)|\le\frac{C}{(\overline{m}_1\dots\overline{m}_s)^\alpha} \tag{6}$$

holds for all integer lattice points $(m_1,\dots,m_s)$. Obviously the convention for $\overline{m}$ prevents division by zero. It is well known that the classes $E_s^\alpha(C)$ have close connections to the classes of α–times differentiable functions. Differentiability can be understood even in the sense of Weyl's derivatives. Whatsoever, we give two important examples:

Example 1:

$$H_2(x_1,\dots,x_s)=\prod_{l=1}^{s}\left(1-\frac{\pi^2}{6}+\frac{\pi^2}{2}(1-2x_l)^2\right)=$$
$$\sum_{m_1,\dots,m_s=-\infty}^{\infty}\frac{1}{(\overline{m}_1\dots\overline{m}_s)^2}\exp(2\pi i(m_1x_1+\dots+m_sx_s)) \tag{7}$$

So we have $H_2(x_1,\dots,x_s)\in E_s^2(1)$. This Fourier representation is the result of classical properties of the second Bernoulli — polynomial.

Example 2:

$$H_1(x_1,\dots,x_s)=\prod_{l=1}^{s}(1-2\ln(2\min\pi x_l))=$$
$$\sum_{m_1,\dots,m_s=-\infty}^{\infty}\frac{1}{(\overline{m}_1\dots\overline{m}_s)}\exp(2\pi i(m_1x_1+\dots+m_sx_s)) \tag{8}$$

So we have $H_1(x_1,\ldots,x_s) \in E_s^1(1)$. This follows from a (not) straight forward calculation. Despite of the fact that this function $H_1(x_1,\ldots,x_s)$ has no absolut convergent Fourier–series it is very useful for the general theory and for the implementation of algorithms for good lattice methods.

Now let

$$\delta_N(b) = \begin{matrix} 1 & for\, b \equiv 0(N) \\ 0 & otherwise \end{matrix} \tag{9}$$

A straight forward and easy calculation shows for integers $a_1,\ldots,a_s$ and $m_1,\ldots,m_s$ the relation

$$\delta_N(a_1 m_1 + \ldots + a_s m_s) = \frac{1}{N}\sum_{k=1}^{N} \exp(2\pi i(m_1 a_1 + \ldots + a_s m_s).k/N) \tag{10}$$

After some elementary manipulations with the Fourier–series of $f \in E_s^\alpha(C)$ and with the above trigonometric sums one gets the error estimation.

$$|R_N(f)| \leq \sum_{m_1,\ldots,m_s \neq 0,\ldots,0} \frac{C}{(\overline{m}_1 \ldots \overline{m}_s)^\alpha} \delta_N(a_1 m_1 + \ldots + a_s m_s) \tag{11}$$

for the lattice nodes

$$\underline{x}_k = (a_1,\ldots,a_s)\frac{k}{N}, k = 1,\ldots,N \tag{12}$$

From this it becames clear even from an heuristic point of view: The coefficients $a_1,\ldots,a_s$ will deliver good integration rules if the number theoretic function δ_N is frequently zero and if the lattice point $(m_1,\ldots,m_s)$ leading the first time to $\delta_N(a_1 m_1 + \ldots + a_1 m_s) = 1$ has maximal possible "hight" $\overline{m}_1 \overline{m}_2 \ldots \overline{m}_s$. But this question is a pronounced number theoretic question, which must be considered as a problem of linear diophantine equations. There are rather deep results on this topic, all of the type of the following proposition.

Theorem: Given N, s, there exist $a_1,\ldots,a_s$ depending on N such that for a $\beta(\alpha, s) > 0$ the estimation

$$R_N(f) = 0\left(\frac{\ln^\beta N}{N^\alpha}\right) \tag{13}$$

holds for $f \in E_s^\alpha(C)$.

We will not show the hitherto known best results. In this context we will show merely a slightly weaker and different result in order to motivate the subsequent described algorithms and their implementation onto parallel architectures.

Theorem: Let $f \in E_s^\alpha(C)$ with $\alpha > 2$. Then the error estimation

$$|R_N(f)| \leq C R_N(H_2)^{\frac{\alpha}{2}} \tag{14}$$

holds.

Proof: Using the well known inequality for $u_n > 0, \alpha \geq 1$,

$$\sum u_n^\alpha \leq \left(\sum u_n\right)^\alpha, \tag{15}$$

we get immediately

$$|R_N(f)| \leq C \sum_{m_1,\ldots,m_s \neq 0,\ldots,0} \frac{\delta_N(a_1 m_1 + +a_s m_s)}{(\overline{m}_1 \ldots \overline{m}_s)^\alpha} \leq C \left(\sum \frac{\delta_N}{(\overline{m}_1 \ldots \overline{m}_s)^2}\right)^{\frac{\alpha}{2}}$$

$$= C[R_N(H_2(x_1 \ldots x_s))]^{\frac{\alpha}{2}} \tag{16}$$

Remark: The result is stronger than it seems at the first glance, because it can be proven that

$$R_N(H_2) = 0\left(\frac{\ln^{2s} N}{N^2}\right) \tag{17}$$

holds.

So it becomes clear: The best integration rule for the special case $H_2(x_1, \ldots, x_s)$ is a good rule for general functions $f(x_1, \ldots, x_s) \in E_s^\alpha(C)$.

Unfortunately there is no general simple and fast method for the computation of good lattice points for given dimensions s and given number of nodes N. In general only direct search methods are known in order to compute good lattice points $a_1(N), a_2(N), \ldots, a_s(N)$. So one has to test all the candidates $a_1, \ldots, a_s$ modulo N. The computational complexity of this problem is N^s, being an unsurmountable problem for the fastest computers even for modest N and smaller dimensions s. Fortunately it can be shown, that it is sufficient to consider only the candidates $(1, a, a^2, \ldots, a^{s-1})$ modulo N for $1 \in a \leq N/2$. Because we are interested in algorithms and implementations we will not tell the hole story about prime or not prime N's.

Algorithm 1
consists of the computation of the value and the location of

$$\min_{1 \leq a \leq N/2} R_N(H_2). \tag{18}$$

More explicitly: If a is the minimum of $R_N(H_2)$,

$$\sum_{k=1}^{N} H_2\left(\frac{k}{N}, \frac{ak}{N}, \ldots, \frac{a^{s-1}k}{N}\right) = \min_{1 \leq z \leq N/2} \sum_{k=1}^{N} H_2\left(\frac{k}{N}, \frac{zk}{N}, \ldots, \frac{z^{s-1}k}{N}\right) \tag{19}$$

then $(1, a, \ldots, a^{s-1})k/N, mod 1$, is a good lattice rule.

Remark1: This method was first introduced by Korobow for $N = p, N = p_1 p_2, p_1, p_2, p$ being primes.

Remark2: The method is obviously coarse grain parallelizable. The complexity of computation is $0(s.N^2)$.

Algorithm 2
Find by direct search the value and location of

$$\min_{1 \leq a \leq \frac{N}{2}} R_N(H_1) \tag{20}$$

More explicitly: If a is the minimum of $R_N(H_1)$,

$$\sum_{k=1}^{N-1} H_1\left(\frac{k}{N}, \ldots, \frac{a^{s-1}k}{N}\right) = \min_{1 \leq z \leq \frac{N}{2}} \sum_{k=1}^{N-1} H_1\left(\frac{k}{N}, \frac{zk}{N}, \ldots, \frac{z^{s-1}k}{N}\right) \tag{21}$$

then $(1, a, \ldots, a^{s-1})k/N, mod 1$, is a good lattice rule.

The above remarks apply in this case as well.
Algorithm 3:
Let $a_1 = 1$. Given the integers $1, a_2, \ldots, a_n$ define a_{n+1} as one of the possible values $z, 1 \leq z \leq N/2$ such that the minimum of

$$\sum_{k=1}^{N-1} H_1\left(a_1\frac{k}{N}, a_2\frac{k}{N}, \ldots, a_n\frac{k}{N}, z\frac{k}{N}\right) \tag{22}$$

is achieved. Then the numbers $1, a_2, a_3, \ldots, a_s, \ldots$ will be good lattice points modulo N. The complexity of computation is $0(s^2 N^2)$. This method has the advantage of delivering good rules for successively growing dimension s without loss of the previous computational work for smaller dimensions. The previous remarks apply accordingly.

Remark: The lattice rules of the considered type $(a_1, \ldots, a_s).k/N, k = 1, \ldots, N$, form obviously a (cyclic) group modulo 1. Therefore these rules apply not only in numerical integration but also in all higher–dimensional convolution problems, like integral equations or signal processing.

3 Implementation of search direct for s–dimensional good lattice points.

The basic algorithm typically employs the summation of a given function in the integer interval $[1..N]$. For each step of the interval the function has to be evaluated in all s dimensions, which may be accomplished by a for–loop.

Further, that part makes the core of a third loop, running over the interval $[1..N/2]$ to find the minimum of the two inner loops.

```
bestlatticepoint=0
```

```
bestsum=999999999. // sentinel value

FOR outer=1 TO N/2 DO
   sum=0
   FOR inner=1 TO N DO
      FOR dim=1 TO S DO
        sum=sum+H(outer,inner,dim)
      ENDFOR
   ENDFOR
 IF (sum<bestsum) THEN
    bestsun=sum
    bestlatticepoint=outer
 END
ENDFOR
```

Vectorization:
Version 1
A Convex C220 was used for vectorizing the standard sequential version of the program. As expected, the C–compiler on the Convex was not able to produce efficient vector code from the original C-source, because the process of vectorization starts with the most inner loop, which is very small in our case, depending on dimension s (usually about 10 iterations).
Thus, the program running in pure scalar mode performed accordingly bad.

Version 2
The first step of improvement included the removal of the most inner loop by explicitly writing down the 10 possible cases of this loop. Now the middle loop became the most inner loop and due to its large size the compiler was able to generate more efficient vector code. The achieved speedup was in the range of 2.24 to 3.60.

Version 3
In a second step it was tried to reduce the program's complexity by introducing lookup–tables for the numerous redundant evaluations of the function, which occur at each iteration up to dimension $s-1$. The main limitation of this method seemed to be the actual size of the

machine's memory. In the given configuration the upper limit of the interval $[1..N]$ was near 32500. Larger values of N caused segmentation faults.

Some timing results (measured in CPU–SECS):

N	S	Version1		Version2		Version3	
4999	2	269.00	(1.00)	119.96	(2.24)	92.04	(2.92)
4999	5	3551.21	(1.00)	987.27	(3.59)	370.17	(9.59)
4999	10	10370.20	(1.00)	4121.50	(2.51)	859.87	(12.06)
10007	2	1076.70	(1.00)	476.631	(2.25)	365.79	(2.94)
10007	5	14254.00	(1.00)	3959.00	(3.60)	1457.28	(9.78)
10007	10					3293.25	

The numbers in parentheses are the speedups.

Parallelization:
The algorithm was parallelized under the PVM2.4 system, which offers all necessary functions for interprocess communication (IPC) on distributed machines. Since the algorithm basically is a minimum–function of many data–independend equations it can be parallelized by 'divide & conquer'.

A first 'from scratch'–version distributed equal parts of the most outer loop of the searching algorithm onto each Workstation, collected their local results and combined them into one global solution.
Clearly, this version reduced the IPC overhead to a minimum, but provided no load–balancing at all. Execution times therefore varied largely, depending on other users'cpu–needs. By using a more fine–grained parallelism (by distributing much smaller parts) a better load balance could be achieved, resulting in the optimal performance.

The core of the searching algorithm was chosen from version 1 of the vectorization examples above, because it shows the highest level of data independence, which makes it very suitable for the 'bag–of–tasks'approach. This term reflects the dynamic distribution of tasks onto machines selected in the moment of distribution. The main goal in load balancing is to avoid overstrain of single processors and – e.g. in heterogenous workstation environments – load balancing. To achieve these two goals, the decision wether to accept another part of the computation or not, has been delegated to each 'worker'. (receiver–initiated load balance strategy)

Because of the unforseeable load situation caused by other users the load balancing was always dynamic. So the load balance strategy was under the given conditions dynamic as well. Confer for example to [?]

Every worker is able to determine the current load of the machine it ist running on, and therefore it is enabled to make the appropriate decision. The mechanism that proved working well can be defined as follows:

```
WHILE (there ist work left)
```

```
    get current load

    IF (0<CURRENT_LOAD_ON_MACHINE<115) THEN

        accept another junk of work

    ELSE

        DO

          go idle for 10 seconds

          get current load

        WHILE (CURRENT_LOAD_ON_MACHINE>50)

    ENDFIN

DO
```

The number that defines the current load situation on the machine ist simply derived from the output of the Unix command "w". This command returns the average number of jobs in the run queue during the last 5 seconds. This floating point number is then mulitplied by a factor of 100 to get a useful "percentage" number.

The percentages listed above where retrieved heuristically (means that the user's complaints were reduced to a minimum) and may vary on different Workstation clusters and users.

Some timing results are:
(9 DECStations, Ultrix, MIPS R3000, 40MHz)

N	S	Worker–CPU		Wall time	
4999	2	224.034	s	31.51	s
4999	5	1670.436	s	206.00	s
4999	10	6589.947	s	760.00	s
10007	2	907.493	s	97.00	s
10007	5	6666.087	s	779.00	s
10007	10	1418.027	s	2805.00	s
19997	2	3518.029	s	371.00	s
19997	5	—		2833.08	s
19997	10	—		11171.00	s

The codes are part of the NEWTON-Package which is a collection of implemented algorithms in high dimensional number theoretical numerics.

Conclusions and further research:

The methods described above are devoted to the systematic finding of new efficient integration rules. The application is usually not time critical and the problem ist large enough to be solved even on larger parallel systems. In workstation farms are very often bigger amounts of computing capacity freely available which can be used by the approach described in this paper. Further research will encompass other methods for the parallel computation of good lattice rules and the further tabulation of new rules in higher dimensions.