GRISK: An internet based search for K-optimal Lattice rules

Tor Sørevik*, Jan Frode Myklebust**

Dept. of Informatics,
University of Bergen,
NORWAY

Abstract. This paper describe the implementation and underlying philosophie of a large scale distributed computation of K-optimal lattice rules. The computation is huge corresponding to the equivalent of 36 years computation on a single workstation. In this paper we describe our implementation, how we have built in fault tolerence and our strategy for administrating a world wide computation on computer we have no account on.
We also provide a brief description of the problem, that of computing K-optimal lattice rules, and give some statistics to describe the extent of the search.

1 Introduction

Using idle cycles on Internet connected workstations for large scale computation is an appealing idea. The combined power of hundreds or thousands of workstations equals the computational power of a top-of-the-line supercomputer. Moreover, utilizing the idle cycles on networked workstations is an essential free-of-cost supercomputer, compared to the multi million dollar price tag on a high end HPC-system.

Nevertheless only a few heroic examples on successful large scale computation on Internet connected systems exists. The most noticeable being, the factorization of RSA 129 [AGLL94] and the record breaking Traveling salesman computation by Bixby et. al. [ABCC98].

Given todays hype about GRID-computing [FK] this fact may seems strange. There are however, good reasons for the situation. First of all the Internet has order of magnitude lower bandwidth and higher latency than the internal network in todays parallel computers, thus communication intensive parallel applications don't scale at all on WAN (Wide Area Network) connected computers. Secondly, one can't assume that each computational node in a NOW (Network of Workstations) has equal power. This might be due to difference in hardware as well a computational load imposed by other usage. Consequently one need a dynamical load balancing scheme which can adapt to available resources of the NOW and

* http://www.ii.uib.no/~tors
** http://www.ii.uib.no/~janfrode

the demand of the application. Most existing parallel program do apply static load balancing, and can for that reason not be expected to run well on NOWs.

These two reasons alone exclude most of the problems traditionally solved on parallel computers. There is however still important subclasses of large scale problems which possess a course grain parallelism and lends itself naturally to a dynamic scheduling approach. But only few of these problems are solved by this style of wide area, Internet base, computing. Again there is good reason for this. One is the problem of having heterogeneous platforms working together. Another is the administration problem; how to get your program running on far away computers where you don't have an account and without disturbing the interactive computation going on? Reliability and fault tolerance is also a problem which have to be dealt with. With hundreds of system running for weeks and even months, you can't expect them all to run without any interference. Thus you need a survival strategy in case of system crash on any of these

In this paper we report on our experiences with Internet based computation on a problem that satisfy the *Problem requirements:* (communication"thin" and dynamic load balancing). We explain our solution on how to deal with heterogeneous platforms, the administration problem and fault tolerance.

This paper is organized as follows. In section 2 we briefly describe our problem, that of finding K-optimal lattice rules in dimension 5. The underlying algorithm and how it is parallelized is explained in section 3. In section 4 we describe our implementation and our approach to heterogeneity, administration and fault tolerance. In section 5 we present statistics of our search to give a flavor of the size and involvement. In section 6 we give some references to related work, while we in section 7 sum up our experiences and indicate the direction in which we intend to develop GRISK in near future.

2 K-Optimal Lattice rules

Lattice rules are uniform weight, numerical integration rules for approximating multidimensional integrales over the s-dimensional unit cube. It takes its abscissaes as those points of an integration lattice, Λ, which lies within $[0, 1)^s$.

$$Qf = \sum_{\forall x \in \Lambda \cap [0,1)^s} f(x) \tag{1}$$

An integration lattice is a lattice which contains all integer points in the s-dimensional space. This makes its reciprocal lattice an integer lattice. The truncation error of a lattice rule is conventiently described by its error functional

$$Ef = If - Qf = \sum_{\forall r \in \Lambda^{\perp}} a_r(\hat{f}) \tag{2}$$

Where $a_r(f)$ is the Fourier coefficients of the integrand and $\Lambda^{\perp}$ is the reciprocal lattice. Since the Fourier coefficients decay rapidly for large $\mid r \mid$ for natural periodic integrands, we seek lattice rules, Q, for which $\Lambda^{\perp}$ has as few points as few points as possible for small $| r |$.

A standard way of meassuring the quality of an integration rule for periodic intergrands is the trigonometric degree. For lattice rules this correspond to The 1-norm of the point on $\Lambda^{\perp}$ closest to the origin, δ, defines the trigonometric degree, d, as $d = \delta - 1$.

A cubature rule needs not only be accurate, a good rule is also economical. A convenient measure of the cost of the rule is $N(Q)$, the number of function evaluation needed to carry out the calculation. This depends directly on the density of the points in the lattice $\Lambda^{\perp}$.

An optimal rule of degree, d, is optimal if its abscissas count is less or equal to the abscissa count of any other rule of degree d. Optimal rules are known for all degrees in dimension 1 and 2. For higher dimension optimal rules are only known for the degree 0,1,2 and 3. Since the number of lattice rules with $N(Q)$ less or equal a fixed number is finite. It is, in theory possible to do a computer search through all these lattices to find optimal lattices. The number of potenial interesting lattice rules is however huge and increases exponentially with the dimension. Thus to conduct such a search in practice the search needs to be restricted and even then lots of computing resources is needed.

Cools and Lyness [CL99] have recently conducted a computer search for good lattice rules in dimension 3 and 4. They define a class of lattices $K(s, \delta)$ and restrict their search to either this class, or well defined subclass of this. There are good reasons to believe that any optimal lattice rule belongs to class $K(s, \delta)$, but this has not been proved. They have termed the rules found in their search K-optimal.

We have modified their code so it can do the search in any dimension and parallelized it in a master slave style. The parallelism is very coarse grained. A typical task takes anything from 1 to 100 hours to do, and the amount of data exchange between the master and a slave is very small.

3 Brief outline of the algorithm

A set of s linearly independent vectors provides a basis for an s-dimensional lattice. If we arrange these vectors as rows in an $s \times s$ matrix. This matrix is called a generator matrix for the lattice. There is a beautiful theory [Lyn89] stating that a lattice is an integration lattice if and only if the generator matrix of its reciprocal lattice is an integer matrix. The idea behind the set $K(s, \delta)$ is that lattices of this class have a generator matrix for which the 1-norm of each row is exactly δ. We can than carry out a search by creating all such lattices, checking their abscissas count, and if it is a potential good one, compute its degree. To speed up the search a number of short cuts are taken. These are described in the paper by Cools and Lyness [CL99].

Think about the entire search as s nested loops. In each we loop over all possible $\mathbf{b}_i$, such that $\| \mathbf{b}_i \|_1 = \delta$. This will generate a huge, but finite number of lattices with degree at most $\delta - 1$. We could now compute the degree of each of these lattices as well as the abscicca number of the associated lattice rule. This is in principal what we do, though in pratice a number of short cuts designed to speed up the computation are taken. These are described in the paper by Cools and Lyness [CL99].

3.1 Parallelization

In the naive search described above the computation for each lattice is totally independent. The speed-up techniques do however try to eliminate computation in the $s-2$ inner loops. To take full advantage of the speed-up techniques we can't parallelize over these loops. However, the iteration of the two outer loops remains totally indepent and thus we are parallelizing over these. A parallel 'task' is identified by the indeces of the 2 outer loops.

Solving a 'task' is equivalent to creating all possible combination of the $s-2$ last rows and do the above described computation. The time needed to to compute a task shows huge differences. In the simplest case some simple checksums proves that none lattices with the two upper rows of the specified task as basis vector can have a degree equal $\delta - 1$ thus we can return immidiately. In the computaion reported in section 5 as much as 40 % of the tasks belong to this class of "quick-return". This computation is done within milliseconds. If the task is not in the class of "quick-return" it typically might need hours. But also within this class we find huge differences.

The computational work of the different tasks is impossible to predict in advance. Thus to efficiently load balancing the computation, the distribution of tasks needs to be done dynamically. In our case this is quite trivial. The different tasks are totally independent, thus as soon as a client return with an answer the next task in line is forward to that client.

4 Implementation

4.1 The distributed search

A central server is keeping the pool of tasks, keeping track of what's done and not done, and updates the current best solution. Each client is running a small java program that connects to the server, retrieves a task and start the calculations.

The initiative is in the hand of the clients. The server only reponds to request. A task consists of a task id which is used to construct the two basis vector constituting a task, and an integer vector, specifying problem parameters such as δ and upper and lower bounds for the abscissa count. The client-Java program then calls a fortran library which does the actual calculations of the task.

4.2 Implementation

Java is an ideal language when you program for an unknown number of different system. Being platform independent, Java program runs on whatever platform you happen to have. Moreover, there should be no possibility for nasty format mismatch when two Java program communicate, even if they run on totally different systems running totally different OS. In our case we've found that this holds not only in theory, but in practice as well.

However, of reasons such that history, laziness and computational speed, the core numerical number crunching is carried out in Fortran. The Fortran code has of course to be compiled for the different OS it is supposed to run under. We have compiled Fortran library for all the systems we have had access to, which is a wide variety of Unix-system. We have experienced some problem with the binary fortran-code on some system, even if it is compiled on the approriate version of the particular OS and the same compiler release. The problem is due to differences in the local installation of the runtime libraries. This is without any comparison the main barrier to seemless portability of our code. First it imposes a significant amount of extra work in moving the code around, compiling it for N different systems and keeping it updated with the latest OS and compiler version, secondly it is still no foolproof guaranty for portability.

By using RMI (Remote Method Invocation), we quickly had a simple and elegant method for communicating distributed java objects. One possible enhancement of our program might be to instead of using a native fortran library, we could have the routine for solving the problem included in the distributed objects. Then we would have a platform and problem independent distributed solver.

4.3 Administration

The administration burden we have distributed to our Internet-friends. What we do is to make a compressed tar-file available at our web-page `http://www.ii.uib.no/grisk`, which anyone can download, unzip, untar and kick-off on all systems that NSF-mounts the catalogue with the downloaded executable. This takes you about 10 minutes of which eight is needed to read the instruction. The run-script assure that the code runs with lowest possible priority, and since the code is pretty slim memory wise (less than 10 MB) you hardly notice that it runs on your desktop when you're processing your daily computing there. (Try it!!)

The administration duties left to us is those related to maintaining the web-page and keeping the server on the road. The server collects all the incoming data which, including statistics on the search. This add up to no more than 20 MB. The workload on the server is low. We run the server program on an O2-workstation which also process a full load of standard workstation tasks. The extra load imposed by our server program has been unnoticeable, even when more than 100 clients have been actively engaged in the search.

4.4 Fault tolerance

More than 250 systems have been taking turns as clients. Only few, if any, have been active all the time. The owner must have full freedom to abort the client-program any time he likes. If that happens the computation of that particular task is lost and have to be recomputed. The server keeps track of, not only which task he has sent out, but also which he receives answer to. After having waited for an answer a couple of days there is a high possibility that the client is down. But it might of course well be that the system has been occupied with more pressing computation, and only few cycles has been allocated to the low priority grisk-job. There is however no need to any immediate action. Thus recomputation of unfinished tasks are postponed to the very end of the computation. This is perfectly all right as there is no reason to believe that the order in which the tasks are completed has any significant influence on the overall running time, and it has definitely no influence on the computational results. For $\delta = 11$ there is 50505 tasks of these 5.6% (3030 tasks) where redistributed. The 5.6% provides an upper bound for the extra computational cost due to recomputation. We find this an acceptable cost, and our fallback scheme simple and reliable.

Cases of permanently network failure will be treated as if the client is aborted. Temporarily network failure are dealt with by letting the client resubmit its message to the server every 5. minute. If the server fails, it will not stop active clients from working, but when they finish with their current task they will have no place to deliver their answer, nor any possibility to get new ones. In the first place this will be treated by the client as a network failure. It simply resubmit the sending of results every 5. minute. If the server recover, we are back in business. To recover a server process we do however rely on manual supervision. Keeping in mind that the server runs 24 hour a day, 7 days a week, through holidays and vacations this is not foolproof. Even the grisk-administrator takes occational breaks. We had, however, no serious problems with the search in the 6 months it took place.

In case of a total disaster at the server, like serious hardware failure, the server might be replaced by another piece of hardware. The address for the clients message is just an IP-address and it does not care who is processing it at this address. Backup of the results are taken once every night, so in case of disk crash the latest results will have to be recomputed.

5 Some statistics on the search

A test version of the code was used to compute k-optimal rules for $\delta = 10$. Some bugs were fixed and improvements made for the final version which was used for the $\delta = 11$. The great search had kick-off November 11 1999 and the last task was recieved at the server at May YY 2000. In this periode 263 systems[1] asked for tasks to compute and 218 from 13 different countries delivered one or more answers.

[1] More precisely; We have recorded request from 263 IP-adresses. We know that in some cases this is a firewall, and don't know how many systems which have been running `grisk` behind this firewall.

The top ten systems were:

Name	processors	No. of tasks
parnass2.iam.uni-bonn.de 144 CPU linux cluster, University of Bonn	450 Mhz Pentium II	9423
cs.kotnet.kuleuven.ac.be 16 linux PC at a stuent lab at CS, K.U.Leuven	Pentium II(Deschutes)	8531
lolligo.mi.uib.no A 10 CPU Linux cluster at Math. dept. UoB	350 Mhz Pentium II	4032
dontask-20.ii.uib.no A 4 CPU Origin 2000 at Parallab, UoB	250 Mhz MIPS R10000	2178
dke.cs.kuleuven.ac.be Single CPU linux PC at CS, K.U.Leuven	Pentium III (Katmai)	1066
cke.cs.kuleuven.ac.be Single CPU linux PC at CS, K.U.Leuven	Pentium III (Katmai)	992
korkeik.ii.uib.no A 4 CPU Sun Ultra 450 at Dept. of Inf., UoB	300 Mhz Ultra Sparc-II	984
bke.cs.kuleuven.ac.be Single CPU linux PC at CS, K.U.Leuven	Pentium III (Katmai)	719
madli.ut.ee 'Something' at the University of Tartu, Estonia	??	706
pandora.cs.kuleuven.ac.be Single CPU linux PC at CS, K.U.Leuven	Pentium III (Katmai)	695

The number listed for the multiprocessor system is the total number of CPU they have. Not all of these have been used for GRISK.

The multiprocessors systems were running multiple clients. No effort was made to parallelize the client program. Thus at peak in January there might have been as many as 300-400 CPUs simulatiously running `grisk`.

The server got 53595 requests of these 3030 never was completed. The total amount of CPU time used for those completed was: 1140638417 CPU-seconds $\approx$ 316844 CPU-hours $\approx$ 36 CPUyears. Which shows that there is no way this computation could be performed on a single workstation.

For comparison we note the May 8th 2000 annoucement from SGI were they claim that: "...The Los Alamos' Blue Mountain supercomputer recently set a world record by running 17.8 years of equivalent single-processor computing in just 72 hours. The Blue Mountain system, one of the world's fastest supercomputers with a peak speed of three trillion floating point operations per second (TFLOPS)..." Our computation is twice as big, has been performed without any additional hardware investment and moderate extra administration work.

The following table shows the distribution of tasks as a function of their completion time.

	0-2 sec	2-10000 sec	10000-100000 sec	> 100000 sec	Sum
no of tasks:	20486	67	29542	761	50856
percentage	40.3	0.1	58.1	1.5	100
Total time	0	515982	1041182248	98940187	

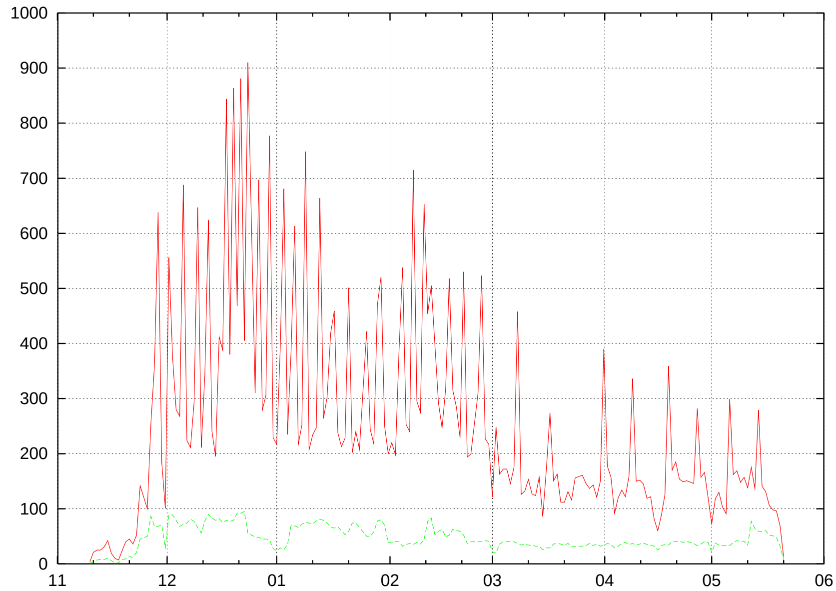

Fig. 1. The number of completed tasks pr. day in red and the number of contributing systems pr. day in green

The table shows that more than 40% of the tasks gave 'quick-return'. The fastest of the tasks, not included in the 'quick-return-class' used 6738 sec or almost 2 hours while the longest lasting task which was completed used 382383 sec or more than 100 CPU-hours. The huge bulk of the tasks use between 3 and 30 CPU-hours.

Note that there is no way to know the time consumption of a task in advance. Thus the table about provides a convincing arguement for the need of dynamic load balancing.

6 Related work

A number of other groups are working on system for pooling together distributed resources. In this section we briefly mention some of them. A more comprehensive list of distributed computing project might be found at **http://distcomp.rynok.org/dcprojects.htm**

SETI The Search for ExtraTerrestrial Intelligence is probably the largest project in terms of participants. Signal received by largest radio telescope in the world, the Arecibo Radio Telescope, The signals are chopped up in very narrow frequency/time bands and sent out to more than 100 000 partcipants where they analysed.

Condor The aim of the condor system is as for GRISK to utilize idle cycles. The difference is that it targets multiple jobs and works in many ways as a batch system. It does however have a very sophisticated check pointing and job migration system, which enable it migrate jobs from one workstation to another depending on the current load. This requires the server to have a much higher degree of control over its slave than in the GRISK system.

distributed.net is a loosely coupled organization of distributed individuals how share the common goal of "..development, deployment, and advocacy, to be pursued in the advancement of distributed computing...". Through their website they, very much like GRISK, invites everyone to contribute their cycles to solving large scale problems. They have successfully solve a number of code-cracking problems

7 Conclusion and future work

There is no way we could have carried out this computation on a traditional supercomputer. We don't have enough money to buy our own computer or sufficiently political influence to occupy enough computing time on a communitee system to accomplish the computation described above. We've found the results of this project very encouraging. Our problem seems tailored to large scale internet based computing. Using Java with RMI the implementation was rather easy and our strategy for fault tolerance and administration worked very well.

We will continue this work along two different axis. 1) Improvements on the core lattice-search program and 2) Improving the functionality of the internet search.

The lattice-search program will be extended to handle all different bases for k-optimal lattices rules. We also plan to implement a Java-only version of the entire system. Whether this will be used in next version depends on whether or not it can compete with FORTRAN in speed.

Planed improvements on the internet search include updating and improvements of the web-pages and better backup for the server program.

References

[ABCC98] D. Applegate, R. E. Bixby, V. Chvatal, and W. Cook. On the solution of traveling salesman problems. *Documenta Mathematica*, ICM(III):645–656, 1998.

[AGLL94] Derek Atkins, Michael Graff, Arjen K. Lenstra, and Paul C. Leyland. The magic words are squeamish ossifrage. In Josef Pieprzyk and Reihanah Safavi-Naini, editors, *Advances in Cryptology – ASIACRYPT '94*. Springer, 1994.

[CL99] Ronald Cools and James Lyness. Lattice rules of moderate trigonometric degree. Technical report, Math. and CS. dept. Argonne Nat. Lab., November 1999.

[FK] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. http://www.mkp.com/index.htm.

[Lyn89] James N. Lyness. An introduction to lattice rules and their generator matrices. *IMA J. of Numerical Analysis*, 9:405--419, 1989.