Comparative computational results for some vertex and facet enumeration codes

David Avis
School of Informatics, Kyoto University, Kyoto, Japan and
School of Computer Science, McGill University, Montréal, Québec, Canada

Charles Jordan
Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan

April 17, 2016

Abstract

We report some computational results comparing parallel and sequential codes for vertex/facet enumeration problems for convex polyhedra. The problems chosen span the range from simple to highly degenerate polytopes. We tested one code (*lrs*) based on pivoting and three codes (*cddr+*, *ppl*, *normaliz*) based on the double description method. *normaliz* employs parallelization as do the codes *plrs* and *mplrs* which are based on *lrs*. We tested these codes using various hardware configurations with up to 1200 core. Major speedups were obtained by parallelization, particularly by the code *mplrs* which uses MPI and can operate on clusters of machines.

1 Background and polytopes tested

A convex polyhedron P can be represented by either a list of vertices and extreme rays, called a V-representation, or a list of its facet defining inequalities, called an H-representation. The vertex enumeration problem is to convert an H-representation to a V-representation. The computationally equivalent facet enumeration problem performs the reverse transformation. For further background see G. Ziegler [12].

In this note we consider only polytopes (bounded polyhedra) so extreme rays will not be required. Furthermore, for technical simplicity in this description, we assume that all polytopes are full dimensional. Neither condition is required for the algorithms tested and in fact some of our test problems are not full dimensional. The input for either problem is represented by an m by n matrix. For the vertex enumeration problem this is a list of m inequalities in $n-1$ variables whose intersection define P. For a facet enumeration problem it is a list of the vertices of P each beginning with a 1 in column one[1]. So in either case, under our assumption, the dimension of P is $n-1$.

One of the features of this type of enumeration problem is that for given input parameters m and n, the output size varies widely. This is shown explicitly by McMullen's Upper Bound Theorem (see, e.g., [12]) which is tight. It states that for a vertex enumeration problem with parameters m, n we have:

$$|V| \le \binom{m - \lfloor \frac{n}{2} \rfloor}{m-n+1} + \binom{m - \lfloor \frac{n+1}{2} \rfloor}{m-n+1} \tag{1}$$

where $|V|$ is the number of vertices that are output. For a facet enumeration problem, by polarity of polytopes, the same inequality holds if we replace $|V|$ by $|F|$, the number of facets. By inverting the formula we can get lower bounds on the output size.

A class of polytopes for which the bound (1) is tight are the *cyclic polytopes* which are usually given as a V-representation consisting of m points on the $(n-1)$-dimensional moment curve. So, for example, a cyclic polytope with $m = 40$ and $n = 21$ has $|V|$=40 vertices in dimension 20 and $|F|$=40,060,020 facets. This implies that if we started with its H-representation, i.e., m =40,060,020

[1]Extreme rays would be indicated by a zero in column one.

and $n = 21$, then the output would consist of only 40 vertices! Problems of this second type are called *highly degenerate* since each vertex may be described by many different combinations of facets. This contrasts with a *simple* polytope where each vertex is given by the intersection of exactly $n-1$ facets. Dually a *simplicial* polytope is one where each facet contains precisely $n-1$ vertices. Cyclic polytopes are simplicial.

The polytopes we tested are described in Table 1 and range from simple polyhedra to highly degenerate polyhedra. This table includes the results of an *lrs* run on each polytope as *lrs* gives the number of cobases in a symbolic perturbation of the polytope, showing how degenerate the polytope is. The corresponding input files are contained in the lrslib-062 distribution [2] in subdirectory `lrslib-062/ine/test-062`. Four of the problems were previously used in [5]:

- *c30-15, c40-21*: cyclic polytopes described above.
- *mit*: a configuration polytope used in materials science, created by G. Garbulsky [7].
- *perm*10: the permutahedron for $n = 10$ whose vertices are the 10! permutations of $(1, 2, 3, ..., 10)$. It is a 9-dimensional simple polytope. For general n this polytope is described by $2^n - 2$ facets and one equation.
- *bv*7: an extended formulation of the permutahedron based on the Birkhoff-Von Neumann polytope. It is described by n^2 inequalities and $3n - 1$ equations in $n^2 + n$ variables and has $n!$ vertices. It is highly degenerate.

The new problems are:

- *fq*48-19: related to the travelling salesman problem for $n = 5$, created by F. Quondam (private communication).
- *mit*71: a correlation polytope related to problem *mit*, created by G. Garbulsky [7].
- *zfw91*: $0, \pm 1$ polytope based on a sensor network that is extremely degenerate and has large output size, created by Z.F. Wang [10]. There are three non-zeroes per row.
- *cp*6: the cut polytope for $n = 6$ solved in the 'reverse' direction: from an H-representation to a V-representation. The output consists of the 32 cut vectors of K_6. It is extremely degenerate, approaching the lower bound of 19 vertices implied by (1) for these parameters.

Name	Input			Output		*lrs*		
	H/V	m	n	V/H	size	bases	depth	secs
bv7	H	69	57	5040	867K	84707280	17	8848
c30-15	V	30	16	341088	73.8M	319770	14	42
c40-20	V	40	21	40060020	15.6G	20030010	19	9870
fq48-19	H	48	19	119184	8.7M	7843390	24	367
mit71	H	71	61	3149579	1.1G	57613364	20	21356
mit	H	729	9	4862	196K	1375608	101	558
perm10	H	1023	11	3628800	127M	3628800	45	2457
zfw91	H	91	38	2787415	205M	10819289888124[†]		*
cp6	H	368	16	32	1.6K	4844923002	153	1762156

Table 1: Polytopes tested and *lrs* times(*mai20*): *=time > 604800 secs

2 Algorithms, implementations and machines used

There are basically two approaches to this problem: pivoting using reverse search [3] and the Fourier-Motzkin double description method, see [12]. The conventional wisdom is to use the double description

[†]Computed by *mplrs1* in 2144809 seconds using 289 cores (see Section 2).

Name	*lrs*	*cddr+*	*ppl*	*normaliz*		*porta*	
	secs	secs	secs	secs(hybrid)	secs(GMP)	secs(64bit)	secs(extended)
bv7	8848	*	578	62	1030	315	310
c30-15	42	4652	3040	**	**	**	**
c40-20	9870	*	*	**	**	**	**
fq48-19	367	124	1355	2	9	5103	4561
mit71	21356	*	260347	503564	364354	108993	107689
mit	558	404	40644	175	2174	**	47553
perm10	2457	*	*	1025	33240	*	*
zfw91	*	*	*	189763	*	31348	30787
cp6	1762156	1463829	>6570000	138162	1518785	**	>959291

Table 2: Single processor times(*mai20*): *=time > 604800 secs **=abnormal termination

method if the polytope is highly degenerate and use a pivoting method if it is simple or has low degeneracy (see e.g. [1], Section 3). Results below shed some doubt on this, especially when parallel processing is used. We tested 5 sequential codes, one based on pivoting and four based on the double description method.

- *lrs* (v. 6.2): C vertex enumeration code based on reverse search developed by D. Avis [2].
- *cddr+* (v. 0.77): Double description code developed by K. Fukuda [9].
- *ppl* (v. 1.1): Double description code developed by the Parma Polyhedral Library project [6].
- *normaliz* (v. 3.0.0): Hybrid parallel double description code developed by the Normaliz project [11].
- *porta* (v. 1.4.1): Double description code developed by T. Christof and A. Lobel [8].

Of these 5 codes *lrs* and *normaliz* offer parallelization. For *normaliz* this occurs automatically with the standard implementation if it is run on a shared memory multicore machine. The number of cores used can be controlled with the -x option, which we use extensively in our tests. For *lrs* two wrappers have been developed:

- *plrs* (v. 6.2): C++ wrapper for *lrs* using the Boost library, developed by G. Roumanis [5]. It runs on a single shared memory multicore machine.
- *mplrs* (v 6.2): C wrapper for *lrs* using the MPI library, developed by the authors [4]. It runs on a network of multicore machines.

Table 2 and the times for *cp6* in Tables 1–4 used timings from version 6.1. All of the above codes compute in exact integer arithmetic and with the exception of *porta*, are compiled with the GMP library for this purpose. However *normaliz* uses hybrid arithmetic, giving a very large speedup for certain inputs as described in the next section. Also *porta* can be run in either fixed or extended precision.

Finally, *lrs* is also available in a 64-bit version, *lrs1*, which does no overflow checking. In general, this gives unpredictable results that need independent verification. However, *lrs1* runs about 4–6 times faster than *lrs* (see Computational Results on the *lrs* home page[2]). The parallel version, *mplrs1*, was used to compute the number of cobases for *zfw91*, taking about 25 days on 289 cores.

The tests were performed using the following machines:

- *mai20*: 2x Xeon E5-2690 (10-core 3.0GHz), 20 cores, 128GB memory, 3TB hard drive
- *mai32abcd*: 4 nodes, each containing: 2x Opteron 6376 (16-core 2.3GHz), 32GB memory, 500GB hard drive (128 cores in total)
- *mai32ef*: 4x Opteron 6376 (16-core 2.3GHz), 64 cores, 256GB memory, 4TB hard drive
- *mai64*: 4x Opteron 6272 (16-core 2.1GHz), 64 cores, 64GB memory, 500GB hard drive

- *mai12*: Xeon X5640 (12-core 2.66GHz), 12 cores, 24GB memory, 60GB hard drive
- *mai24*: 2x Opteron 6238 (12-core 2.6GHz), 24 cores, 16GB memory, 600GB RAID5 array
- *Tsubame2*: supercomputer located at Tokyo Institute of Technology

The first 6 machines total 312 cores and are located at Kyoto University. They were purchased between 2011-15 for a combined total of 3.9 million yen ($33,200).

3 Computational results

Table 2 contains the results obtained by running the four sequential codes on the problems described in Table 1. The times for *lrs* shown in Table 1 are included for comparison. The time limit was one week (604,800 seconds) except for *cp6*. Programs *cddr+*, *lrs*, *ppl* were used with no parameters.

The program *normaliz* performs many additional functions, but was set to perform only vertex enumeration/facet enumeration for these tests. By default, it begins with 64-bit integer arithmetic and only switches to GMP arithmetic (used by all other programs except *porta*) in case of overflow. In this case, all work done with 64-bit arithmetic is discarded. For our test problems this happens on *c30-15, c40-20* and *mit71*, however the first two problems terminated abnormally after switching to GMP. Using the -B flag *normaliz* will do all computations using GMP arithmetic. We give times for the default hybrid arithmetic and also for GMP only arithmetic. Note that *mit71* runs significantly faster with the -B flag reflecting the time wasted in 64 bit arithmetic mode.

As mentioned above, *porta* supports arithmetic using either 64-bit integers or its own extended precision arithmetic package. The program terminates if overflow occurs. We tested both options on each problem, and found that the extended precision option outperformed the 64-bit option in all cases.

The results in Table 2 show that in fact it is rather hard to draw many general conclusions! The four double description implementations behaved remarkably differently on most of the problems. This could be due to the fact that the method is highly sensitive to the insertion order of the input and the codes may be using different orderings. One clear result was that none of these codes could solve the cyclic polytope *c40-20* problem and struggled even on *c30-15*. We also observed that the double description codes use a lot of memory, especially *normaliz*. In fact the machines with 32GB or less of memory were not able to solve either *mit71* or *cp6*, and even in single processor mode most of the 128GB available on *mai20* was required for some problems. Memory use by *lrs/plrs/mplrs* was negligible, making them good background processes. On the extremely degenerate problem *cp6*, *lrs* was in the middle of the pack, about 20% slower than *cddr+*. *ppl* was not able to solve the problem in 76 days, whereas *normaliz* was nearly 13 times faster than *lrs*. Only *porta* and *normaliz* could effectively solve the sparse 0,±1 polytope *zfw91*. A 289 core run with *mplrs1* was about 70 times slower. We note here that with about 151 million cobases/vertex *cp6* is far more degerate than *zfw91*, which has about 4 million cobases/vertex.

To put the above results in perspective, we recall that the problem *mit* was a big challenge for the time, the early 1990s. At that time early versions of both *cddr+* and *lrs* took over a month to solve the problem. Combined hardware and software improvements since then have lead to speedups of over 5000 times and both codes now complete the job in less than 10 minutes. We will see that parallelization of *lrs* can lead to further dramatic reductions in running time: on our 312 core cluster the problem now requires only 12 seconds.

We move now to the three parallel codes. For *mplrs* and *plrs* we used the default settings (see User's guide [2] for details):

- *plrs*: -id 4
- *mplrs*: -id 2, -lmin 3 -maxc 50 -scale 100 -maxbuffer 500

For *normaliz* we used the default settings which imply that hybrid arithemtic is used. Table 3 contains results for low scale parallelization and all problems were run on the single workstation *mai20*. With 4 core available *plrs* usually outperforms *mplrs* and they break even with 8 core. *mplrs* is usually faster with 12 core with speedups in the range 7-12. *normaliz* obtained reasonable speedups on most problems that it could solve, in the range 3.7-9.9. The exception was *perm*10 where no speedups were obtained by parallelization.

Table 4 contains results for medium scale parallelization on the 64 core shared memory machine *mai64* which is considerably slower than *mai20*. We used 8,16,32,64 core and speedups are measured by comparing with the running time on 8 core. *mplrs* was the clear winner over *plrs* with speedups ranging from 4.3 to 7.2. *plrs* showed little improvement after 32 core, and *normaliz* little improvement after 16 core.

Table 5 contains results for medium scale parallelization on a cluster of computers with 312 core. Only *mplrs* is able to run in this heterogeneous environment. The machines were scheduled in the order given at the end of Section 2 (excluding *Tsubame2*). Due to the heterogeneous selection of machines we do not present speedups in this table. In fact we observed that *mai20* is substantially faster than the other machines than would be expected by just comparing clock speeds. For example it was more than twice as fast as *mai12* on *c40-20*. Jobs running in under a minute do not profit much, if at all, as extra core are added. However the longer running jobs show continuous improvement as core are added. To complete the full set of 9 problems *lrs* required about 3 weeks on the fastest machine, *mai20*. With the 312 core cluster this time is reduced to 4 hours and 40 minutes. These total times are dominated by *cp6*. Excluding this problem, the *lrs* total running time of 12 hours 13 minutes is improved to about 8 minutes on the cluster.

Table 6 contains results on the large scale parallelization obtained by Kazuki Yoshizoe using the *Tsubame2* supercomputer at the Tokyo Institute of Technology. We chose the two hardest problems: *mit*71 and *cp*6. We observe near linear speedup between 12 and 1200 core for both problems[2]. With 1200 core *mplrs* solved *cp*6 in about 42 minutes, nearly 600 times faster than *cddr+*, 55 times faster than *normaliz* in single processor mode and over 6 times faster than *normaliz* running on 64 core, the largest shared memory machine available. A supercomputer on the scale of *Tsubame2* may seem out of reach for most researchers. However, at current prices, a 1200 core cluster could be built for roughly $100,000 and would be considerably cheaper with used hardware. This price will certainly fall substantially in the near future making this amount of computing power readily available to most researchers. The problem will not be the availability of the hardware but the availability of software that can make use of it.

Name secs/size	core=4 secs/speedup			core=8 secs/speedup			core=12 secs/speedup			core=16 secs/speedup		
	mplrs	*plrs*	*normaliz*	*mplrs*	*plrs*	*normaliz*	*mplrs*	*plrs*	*normaliz*	*mplrs*	*plrs*	*normaliz*
bv7	5219	2444	43	1739	1249	24	1045	827	17	747	624	13
	1.7	3.6	1.4	5.1	7.1	2.6	8.5	10.7	3.7	11.8	14.2	4.8
c30-15	28	16	**	9	12	**	6	10	**	4	10	**
	1.5	2.6	-	4.7	3.5	-	7	4.2	-	10.5	4.2	-
c40-20	5979	3779	**	2033	2616	**	1219	2276	**	873	2066	**
	1.7	3.6	-	4.9	4.7	-	8.1	4.9	-	11.3	4.8	-
fq48-19	146	102	2	49	53	2	30	37	2	21	29	1.5
	2.5	3.6	-	7.5	6.9	-	12	9.9	-	17.5	12.7	-
mit71	11386	6492	107482	3983	3311	65507	2390	2243	50910	1709	1724	42916
	1.9	3.3	4.7	5.4	6.5	7.7	8.9	9.5	9.9	12.5	12.4	11.7
mit	293	161	70	99	94	42	61	71	33	44	57	29
	1.9	3.5	2.5	5.6	5.9	4.2	9.1	7.9	5.3	12.7	9.8	6
perm10	1422	741	1085	481	474	960	292	381	1090	215	320	1093
	1.7	3.3	.94	5.1	5.2	1.1	8.4	6.4	.94	11.4	7.7	.93
zfw91	*	*	46741	*	*	23885	*	*	15975	*	*	12110
	-	-	4.1	-	-	7.9	-	-	11.9	-	-	15.7
cp6	968550	486667	42774	331235	268066	23493	199501		18585	143006	169352	16980
	1.8	3.6	3.2	5.3	6.6	5.9	8.8		7.4	12.3	10.4	8.1

Table 3: Small scale parallelization(*mai20*): *=time > 604800 secs, **=abnormal termination

[2]The benchmark for *cp*6 was taken with *mai12* as it has a similar processor to those we used on *Tsubame2*.

Name	core=8			core=16			core=32			core=64		
	secs/speedup on 8 core			secs/speedup on 8 core			secs/speedup on 8 core			secs/speedup on 8 core		
	mplrs	*plrs*	*normaliz*	*mplrs*	*plrs*	*normaliz*	*mplrs*	*plrs*	*normaliz*	*mplrs*	*plrs*	*normaliz*
bv7	3238	5040	60	1478	1356	39	1206	725	29	515	507	21
	1	1	1	2.2	1.9	1.5	2.7	3.6	2.1	6.3	5.1	2.9
c30-15	17	23	**	9	23	**	5	20	**	4	28	**
	1	1	-	1.9	4.7	-	3.4	1.2	-	4.3	.82	-
c40-20	3882	4808	**	1876	4628	**	1141	4155	**	717	6026	**
	1	1	-	2.1	1.0	-	3.4	1.2	-	5.4	.8	-
fq48-19	89	86	4	42	47	3	23	28	3	14	22	1
	1	1	-	2.1	1.8	-	3.9	3.1	-	6.4	3.9	-
mit71	7395	6606	115088	3401	4391	77436	1900	2205	60694	1251	1710	51594
	1	1	1	2.2	1.5	1.5	3.9	3.0	1.9	5.9	3.9	2.2
mit	195	225	111	93	135	83	53	118	75	42	117	82
	1	1	1	2.1	1.7	1.3	3.7	1.9	1.5	4.6	1.9	1.4
perm10	909	1081	1951	432	687	1870	253	571	1840	171	592	1930
	1	1	1	2.1	1.6	1.1	3.6	1.9	1.1	5.3	1.8	1
zfw91	*	*	42409	*	*	24822	*	*	14452	*	*	7332
	-	-	1	-	-	1.7	-	-	2.9	-	-	5.8
cp6	727771	565915	38621	326214	377857	23773	171194	298408	17468	100676	229713	15480
	1	1	1	2.2	1.5	1.6	4.3	1.9	2.2	7.2	2.5	2.5

Table 4: Medium scale parallelization (*mai64*): *=time > 604800 secs, **=abnormal termination

Acknowledgements

We thank Kazuki Yoshizoe for kindly allowing us to use the results of his *Tsubame2* experiments and for helpful discussions concerning the MPI library which improved *mplrs*' performance. This work was partially supported by JSPS Kakenhi Grants 23700019 and 15H00847, Grant-in-Aid for Scientific Research on Innovative Areas, 'Exploring the Limits of Computation (ELC)'.

References

[1] Assarf, B., Gawrilow, E., Herr, K., Joswig, M., Lorenz, B., Paffenholz, A., Rehn, T.: Computing convex hulls and counting integer points with polymake. CoRR **abs/1408.4653** (2015). URL http://arxiv.org/abs/1408.4653

[2] Avis, D.: (2013). http://cgm.cs.mcgill.ca/~avis/C/lrs.html

[3] Avis, D., Fukuda, K.: A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. Discrete & Computational Geometry **8**, 295–313 (1992)

[4] Avis, D., Jordan, C.: mplrs: A scalable parallel vertex/facet enumeration code. CoRR **abs/1511.06487** (2015). URL http://arxiv.org/abs/1511.06487

[5] Avis, D., Roumanis, G.: A portable parallel implementation of the lrs vertex enumeration code. In: Combinatorial Optimization and Applications - 7th International Conference, COCOA 2013, *Lecture Notes in Computer Science*, vol. 8287, pp. 414–429. Springer (2013)

[6] Bugseng.org: (2013). http://bugseng.com/products/ppl

[7] Ceder, G., Garbulsky, G., Avis, D., Fukuda, K.: Ground states of a ternary fcc lattice model with nearest- and next-nearest-neighbor interactions. Phys Rev B Condens Matter **49**(1), 1–7 (1994)

[8] Christof, T., Lobel, A.: (2009). http://typo.zib.de/opt-long_projects/Software/Porta/

[9] Fukuda, K.: (2012). http://www.inf.ethz.ch/personal/fukudak/cdd_home

[10] Moran, W., Cohen, F.R., Wang, Z., Suvorova, S., Cochran, D., Taylor, T., Farrell, P.M., Howard, S.D.: Bounds on multiple sensor fusion. CoRR **abs/1410.3083** (2014). URL http://arxiv.org/abs/1410.3083

[11] Normaliz: (2015). http://www.home.uni-osnabrueck.de/wbruns/normaliz/

[12] Ziegler, G.M.: Lectures on Polytopes, *Graduate Texts in Mathematics*, vol. 152. Springer (1995)

Name	*mplrs*					
	core=16	core=32	core=64	core=128	core=256	core=312
	secs	secs	secs	secs	secs	secs
bv7	747	389	262	179	101	88
c30-15	4	3	2	3	2	2
c40-20	873	528	328	218	133	121
fq48-19	21	11	7	5	4	5
mit71	1709	993	625	421	228	199
mit	44	26	21	23	13	12
perm10	215	118	89	75	53	55
cp6	143006	75712	50225	33684	18657	16280

Table 5: Medium scale parallelization(cluster): *mt*=no. of processors

Name	*mplrs*						
	mt=12	*mt*=36	*mt*=72	*mt*=144	*mt*=300	*mt*=600	*mt*=1200
cp6	283403(*mai12*)	*	*	20383	9782	4913	2487
	1	-	-	14	29	58	114
mit71	4207	1227	602	297	146	81	45
	1	3.4	7.0	14	29	52	94

Table 6: Large scale parallelization: secs/speedups, *=*Tsubame2* time limit exceeded