mplrs/plrs/lrs test results

David Avis and Charles Jordan

August 27, 2015

We describe here some preliminary experimental results for three codes in lrslib-060 and two codes based on the double description method.

- *lrs*: C vertex enumeration code based on reverse search developed by D. Avis [1].
- *plrs*: C++ wrapper for *lrs*, using the Boost library, developed by G. Roumanis [2]. It runs on a single shared memory multicore machine.
- *mplrs*: C wrapper for *lrs*, using the MPI library, developed by C. Jordan. It runs on a network of multicore machines.
- *cddr+*: Developed by K. Fukuda [5].
- *ppl_lcdd*: Developed by the Parma Polyhedral Library project [3].

Details of the implementation of *mplrs* and more extensive tests will follow in a forthcoming paper. The input files are described in Table 1 and range from simple polyhedra to extremely degenerate polyhedra. They are contained in the lrslib-060 distribution [1] in subdirectory `lrslib-060/ine/test-060`. Five of the problems were described in [2]. The new problems are:

- $fq48$-19: related to the travelling salesman problem for $n = 5$, created by F. Quondam.
- $m6.ine$: the metric polytope for $n = 6$.
- $mit71$: a correlation polytope, related to problem mit, created by G. Garbulsky [4].
- $cp6$: the cut polytope for $n = 6$ solved in the 'reverse' direction: from an H-representation to a V-representation.

The time limit set was one week (604,800 seconds). Programs *cddr+*, *lrs*, *ppl_lcdd* were used with no parameters. For *mplrs* and *plrs* we used the default settings (see User's guide [1] for details):

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

The tests were performed using three computers:

- *mai12*: Xeon X5640, 2.66GHz, 12 core, 24GB memory, 60GB hard drive
- *mai20*: Xeon E5-2690 X 2, 3.0GHz, 20 core, 128GB memory, 3TB hard drive
- *mai64*: Opteron 16core 6272 X 4, 2.1GHz, 64 core, 64GB memory, 500GB hard drive

Table 2 contains results for low scale parallelization and were run on the single workstation *mai20*. With up to 12 cores available *plrs* usually outperforms *mplrs*.

Table 3 contains results for medium scale parallelization. Since *plrs* only runs on a single workstation, *mai64* was chosen and there is no data available for 96 processes. Runs for for *mplrs* were on the full cluster of workstations. The weighted average processor speed over the cluster is 2.36GHz, so *lrs* runs on the median speed workstation *mai12* were used as the benchmark. This understates

Name	Input			Output		*lrs*			cddr+	ppl
	H/V	m	n	V/H	size	bases	depth	secs	secs	secs
bv7	H	69	57	5040	867K	84707280	17	8848	*	578
c30-15	V	30	16	341088	73.8M	319770	14	42	4652	3040
c40-20	V	40	21	40060020	15.6G	20030010	19	9870	*	*
fq48-19	H	48	19	119184	8.7M	7843390	24	367	124	1355
m6	H	80	16	554	41K	6651872	32	496	1	0.1
mit71	H	71	61	3149579	1.1G	57613364	20	21356	*	260347
mit	H	729	9	4861	196K	1375608	101	558	404	40644
perm10	H	1023	11	3628800	127M	3628800	45	2457	*	*
cp6	H	368	16	32	1.6K	4844923002	153	1762156	1463829	>3000000

Table 1: Polyhedra tested and baseline times: *=killed after 604800 secs (*mai20*)

plrs speedups by about 25% since it uses the slowest machine. MPI was configured to use the machines in decreasing order by speed and so the average processor speeds for $mt = 16, 32, 64, 96$ are respectively 3, 2.9, 2.5, and 2.36GHz. The speedups shown for *mplrs* with $mt = 16, 32$ are therefore slightly overstated and those for $mt = 64, 96$ slightly understated. In this range of processors *mplrs* usually outperforms *plrs*. Table 4 contains results on the large scale parallelization obtained by Kazuki Yoshizoe using the *Tsubame2* supercomputer at the Tokyo Institute of Technology.

The hardest problem solved was *cp*6, the 6 point cut polytope solved in the reverse direction, which is extremely degenerate. Its more than 4.8 billion bases span just 32 vertices! Normally such polytopes are considered out of reach for pivoting algorithms. In fact it it took *lrs* just under 3 weeks to complete the computation, about 4 days more than *cddr+* and faster than *ppl_lcdd*. With only 4 cores both *mplrs* and *plrs* were able to outperform both double description methods. With 1200 cores this problem was solved by *mplrs* in about 42 minutes, nearly 600 times faster than *cddr+*. We observe near linear speedup between 144 and 1200 cores. Solving in the 'reverse' direction is useful for checking the accuracy of a solution, and is usually extremely time consuming. For example, converting the V-representation of cp6 to an H-representation takes less than 2 seconds using any of the three single core codes.

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

[2] 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)

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

[4] 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)

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

Name	*lrs*	*mt*=4		*mt*=8		*mt*=12	
	secs/size	secs/speedup		secs/speedup		secs/speedup	
		mplrs	*plrs*	*mplrs*	*plrs*	*mplrs*	*plrs*
bv7	8848	5232	2444	1762	1249	1042	827
	867K	1.7	3.6	5.0	7.1	8.5	10.7
c30-15	42	65	16	16	12	10	10
	73.8M	0.65	2.6	2.6	3.5	4.2	4.2
c40-20	9870	19405	3779	6314	2616	3830	2276
	15.6G	0.5	3.6	1.6	4.7	2.6	4.9
fq48-19	367	147	102	49	53	41	37
	8.7M	2.5	3.6	7.5	6.9	9.0	9.9
m6	496	277	151	92	86	56	63
	41K	1.8	3.3	5.4	5.8	8.9	7.8
mit71	21356	12096	6492	3996	3311	2415	2243
	1.1G	1.8	3.3	5.3	6.5	8.8	9.5
mit	558	302	161	101	94	62	71
	196K	1.8	3.5	5.5	5.9	9.0	7.9
perm10	2457	1528	741	499	474	314	381
	127M	1.6	3.3	4.9	5.2	7.8	6.4
cp6	1762156	968550	1007184	331235	565915	199501	434390
	1.6K	1.8	1.7	5.3	3.1	8.8	4.1

Table 2: Small scale parallelization: *mt*=no. of processors (*mai20*)

Name	*lrs*	*mt* = 16		*mt*=32		*mt*=64		*mt*=96
	secs/size	secs/speedup		secs/speedup		secs/speedup		secs/speedup
		mplrs	*plrs*	*mplrs*	*plrs*	*mplrs*	*plrs*	*mplrs*
bv7	11851	1481	1356	680	725	310	507	223
	867K	8.0	8.7	17.4	16.3	38	23.4	53.1
c30-15	80	8	23	13	20	9	28	5
	73.8M	10	3.5	6.2	4.0	8.9	2.9	16.0
c40-20	22458	2672	4628	2420	4155	1198	6026	993
	15.6G	8.4	4.9	9.3	5.4	18.7	3.7	22.6
fq48-19	360	41	47	28	28	9	22	7
	8.7M	8.8	7.7	12.9	12.9	40	16.4	51.4
m6	679	76	111	46	89	18	89	13
	41K	8.9	6.1	14.8	7.6	37.7	7.6	52.2
mit71	21879	3236	4391	1547	2205	709	1710	528
	1.1G	6.8	5.0	14.1	9.9	30.9	12.8	41.4
mit	809	46	135	57	118	26	117	23
	196K	17.6	6.0	14.2	6.9	31.1	6.9	35
perm10	3193	225	687	202	571	101	592	87
	127M	14.2	4.6	15.8	5.6	31.6	5.4	36.7
cp6	1762156	142690	377857	81543	298408	54921	229713	44006
	1.6K	12.3	4.7	21.6	5.9	32.1	7.7	40.0

Table 3: Medium scale parallelization: *mt*=no. of processors *lrs*(*mai12*), *plrs*(*mai64*), *mplrs*(*cluster*)

Name	*mplrs*						
	mt=12	mt=36	mt=72	mt=144	mt=300	mt=600	mt=1200
cp6				20383	9782	4913	2487
	*	*	*	1	2.1	4.1	8.2
mit71	4207	1227	602	297	146	81	45
	1	3.4	7.0	14.2	28.9	52	93.5

Table 4: Large scale parallelization: secs/speedups, *= *Tsubame2* time limit exceeded