CoCoALib - Slug #881

ReadExpr is too slow on large polys

09 May 2016 21:10 - John Abbott

Status:	Closed	**Start date:**	09 May 2016
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Improving	**Estimated time:**	12.12 hours
Target version:	CoCoALib-0.99560	**Spent time:**	12.20 hours

Description

Recently Mario has passed me some large polys in their printed form (*i.e.* sum of terms). Reading them by pasting the printout as input to CoCoA-5 is impractically slow (because when the interpreter reads a poly it effectively uses a quadratic algorithm).

Reading is faster with ReadExpr, but still not fast enough (*e.g.* 15 mins to read just one low-deg poly with about 100000 terms in 100 variables).

Make ReadExpr significantly faster.

Related issues:

Related to CoCoALib - Slug #884: DistrMPolyInIPP::myPushFront and DistrMPolyI...	**New**	**24 May 2016**
Related to CoCoALib - Feature #801: Test whether a symbol is in a ring	**New**	**09 Nov 2015**
Related to CoCoALib - Slug #1238: ReadExpr is too slow on long lists of mono...	**Closed**	**21 Jan 2019**
Related to CoCoA-5 - Slug #1629: RingElem slow with many indets	**Closed**	**08 Nov 2021**

History

#1 - 09 May 2016 21:13 - John Abbott

I have implemented a first improvement by implementing a new fn ReadExprInSparsePolyRing which uses geobuckets to store sums. This has already made quite a big difference (*e.g.* that big poly now takes 15mins instead of much longer (I don't recall exactly))

I hope to check this in shortly -- the code is fairly simple, so should be safe.

#2 - 09 May 2016 21:23 - John Abbott

Since my "stupid" MacBook cannot profile, I can only guess that reading a symbol is probably quite costly (see line 79 of RingElemInput.C which calls RingElem to convert the symbol into a RingElem value).

The conversion of a symbol into a RingElem should ideally be done using a std::map. This could be done inside the ring itself, or outside.

Currently it uses a shockingly inefficient technique: each call to RingElem(..., symbol) creates a std::vector, and checks that the symbol is there (error if not). In the case of a SparsePolyRing the actual conversion then recreates the same list of symbols, and checks again if it is there, and if so generates the answer.

The conversion table of symbol-to-ringelem should be created only once; perhaps it is best if this is inside the ring?

#3 - 09 May 2016 21:49 - John Abbott

I did a quick test:

```
QQa1 ::= QQ[a];
QQa100 ::= QQ[a[0..99]];
```

```
P1 ::= QQa1[x];
P100 ::= QQa100[x];

str := "(x^10000-1)/(x-1) expanded out";
t0 := CpuTime();
j1 := ReadExpr(P1, str);
t1 := CpuTime();

j2 := ReadExpr(P100,str);
t2 := CpuTime();

println "Time to read in P1: ", FloatStr(t1-t0);
println "Time to read in P100: ", FloatStr(t2-t1);
```

The times recorded were about 0.28s and 1.3s. So the presence of indets which are not used has a significant cost.

For comparison I tried the test again after disabling the improvement describe in comment 1; the times reported were identical! Aha that's because adding a new terms whose PP is smaller than those in the poly is handled specially.

Anyway, for Mario's poly with 84 indets the time dropped from about 17s to 9s.

#4 - 10 May 2016 13:25 - John Abbott

I have hacked the ReadExpr code so that it produces a table of symbols (of type std::map).
This improved the read time for Mario's poly from about 9s to less than 1s, so it is definitely worth doing.

I believe the clean way to do this is to put the std::map inside the ring; then every call to RingElem(R, sym) will be quick because it can use the symbol table :-) This should also mean that ReadExpr code already in CVS will suddenly become faster.

#5 - 10 May 2016 15:33 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

I have done a quick profile of the current version of the code (using the *ad hoc* impl with std::map). Now most (90+%) of the execution time is in DistrMPolyInIPP::myAddClear. I'm now sure how to improve this :-(Yet a further improvement would really be needed.

#6 - 10 May 2016 16:19 - Anna Maria Bigatti

John Abbott wrote:

I have done a quick profile of the current version of the code (using the *ad hoc* impl with std::map). Now most (90+%) of the execution time is in DistrMPolyInlPP::myAddClear. I'm now sure how to improve this :-(Yet a further improvement would really be needed.

Is it a very long polynomial? I suppose we then should use in ReadExpr
geobucket f(P); instead of RingElem f(P);

#7 - 10 May 2016 16:36 - John Abbott

I have already implemented (and checked in) a version which uses geobuckets when the ring is a sparsepolyring.

Mario says his polys are printed out using the standard term ordering (StdDegRevLex presumably).
So maybe just using PushBack will make it faster still. I hope to investigate this evening.

Some polys have more than 100000 terms, and they seem to take several minutes to read.

#8 - 11 May 2016 12:09 - John Abbott

- Assignee set to John Abbott

I wrote a very hacked version which uses PushBack, and it went considerably faster (but it also gave SEGV on some examples).

I think this also explains what I observed with the profiler, namely that DistrMPolyInlPP::myAddClear was very expensive (because it was running through all the terms in the poly).

I am now trying to rewrite DistMPolyInlPP so that myAddClear can be fast: if the LPP of the poly to be added comes after the last term, then it is just concatenated. This needs a pointer to the last term. There are some complications in myRemoveSummand and myInsertSummand.

#9 - 11 May 2016 14:32 - John Abbott

- % Done changed from 10 to 20

I have now finished the main changes to DistrMPolyInlPP, and especially myAddClear. Time to read a correctly ordered polynomial has now decreased significantly (by a factor of about 20-30); time to read a "randomly ordered" polynomial is much the same as right before this change.

I can now do Mario's test case in around 10 mins, whereas yesterday it was 5+ hours. :-)

Next step: clean up all the changes, and check in (and maybe write some documentation).

#10 - 11 May 2016 14:48 - John Abbott

I reran the test to read Mario's polynomial: now it takes about 0.4s (with the "wrong" term-ordering), and about 0.25 s with the "right" term-ordering (lex in this case).

Probably the case of "wrong" term-ordering would be even faster if the terms were read into separate polynomials (one term in each) in a vector, and then sort the vector, and finally sum the terms.

Note that for larger polys the difference between "wrong" order (degrevlex when the poly was written in lex) and the "right" order is usually much more marked (typically 20-30 times slower).

#11 - 18 May 2016 13:44 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 20 to 70

I have modified ReadExprInSparsePolyRing so that sums of terms are handled specially: the terms are simply appended to a vector, then at the end the vector is sorted then summed. The impl is simple rather than superfast; anyway it works fairly well even if the terms in the input are not ordered the same way as in the ring.

Further ideas: also have a single polynomial to which terms may be appended (pre- or post-), so that if the input terms are in the right order we get the correct poly directly. This would require a function which can detect whether "fast polynomial concatenation" is possible (and perhaps actually do the concatenation when it is possible). This ought to minimise copying/reallocation and also comparisons between PPs.

In Mario's application, reading the poly is still much slower (*e.g.* 10x) than testing for irreducibility... which seems a bit ridiculous!

#12 - 24 May 2016 15:25 - John Abbott

- Related to Slug #884: DistrMPolyInIPP::myPushFront and DistrMPolyInIPP::myPushBack inefficient if arg is a PP added

#13 - 25 May 2016 17:01 - John Abbott

Using the profiler on a linux box I noticed that DistrMPolyInIPP::myPushFront was using surprisingly much time. I had added this as issue #884.

I wrote an *ad hoc* reader and writer of polynomials, and observed approximately 5 times speed up (should be more if the myPushFront problem can be improved). Now I wonder whether it is worth developing a better format for communication between processes (sacrificing human-readability); this is what the OpenMath-like interface was for (but it seems implausible that the OpenMath-like XML-based format could be as fast as the simple *ad hoc* format).

#14 - 16 Sep 2016 16:13 - John Abbott

- Target version changed from CoCoALib-0.99550 spring 2017 to CoCoALib-0.99560

#15 - 06 Nov 2017 13:56 - John Abbott

- Status changed from Resolved to Feedback

- % Done changed from 70 to 90

#16 - 06 Nov 2017 14:11 - John Abbott

- Related to Feature #801: Test whether a symbol is in a ring added

#17 - 08 Nov 2017 15:59 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

- Estimated time set to 12.12 h

I have just tested it now, and it seems adequately fast (until someone needs really big polys).

Reading ChebyshevPoly(10000,x) took less than 1sec; the string was about 15Mbytes long.

Reading (x^100-1)*(y^100-1)*(z^100-1)/((x-1)*(y-1)*(z-1)) tok about 9secd; the string was about 15Mbytes long, poly has 1000000 terms.

Closing.

#18 - 21 Jan 2019 16:00 - Anna Maria Bigatti

- Related to Slug #1238: ReadExpr is too slow on long lists of monomial with many indets: ---> use RingElems instead added

#19 - 26 Nov 2021 14:58 - John Abbott

- Related to Slug #1629: RingElem slow with many indets added