CoCoALib - Slug #1518

SLUG: Printing PPs with many indets

23 Oct 2020 16:32 - John Abbott

| Status: | In Progress | Start date: | 23 Oct 2020 | | | | | | |
|--|--|-----------------|-------------|-------------|--|--|--|--|--|
| Priority: | Normal | Due date: | | | | | | | |
| Assignee: | | % Done: | 20% | | | | | | |
| Category: | Improving | Estimated time: | 0.00 hour | | | | | | |
| Target version: | CoCoALib-0.99900 | Spent time: | 2.00 hours | | | | | | |
| Description | | | | | | | | | |
| It seems that printing polys in polyrings with many indets is slower than I would like: the example from issue <u>#1514</u> involves printing out many polys from a ring with 1000 indets, and printing takes surprisingly long (longer than re-reading the same polys!) | | | | | | | | | |
| Investigate and improve (if poss). | | | | | | | | | |
| Related issues: | | | | | | | | | |
| Related to CoCoALib - Des | ign #1538: RingElem from string (ReadExpr) | | Closed | 13 Nov 2020 | | | | | |
| | | | | | | | | | |

History

#1 - 23 Oct 2020 16:36 - John Abbott

According to the profiler: printing 250 polys each with about 500 terms (RandomLinearForm from polyring with 1000 indets) calls

| | | 0.21 | 9.65 | 124871/124871 | Co | CoA::operator<<(std::ostream&, | CoCoA:: | ConstRefPPMonoid | Elem) | |
|--|------|------|------|-----------------|--------|--------------------------------|----------|------------------|-------|--|
| [24] | | | | | | | | | | |
| [25] | 20.1 | 0.21 | 9.65 | 124871 | CoCoA: | :PPMonoidBase::myOutput(std::o | stream&, | CoCoA::PPMonoid | ElemC | |
| onstRawPtr) const [25] | | | | | | | | | | |
| | | 0.52 | 8.71 | 124871000/12487 | 1000 | CoCoA::PPMonoidOvImpl::myBig | Exponent | (CoCoA::BigInt&, | CoCo | |
| A::PPMonoidElemConstRawPtr, long) const [26] | | | | | | | | | | |
| | | 0.11 | 0.27 | 124871000/12487 | 1026 | CoCoA::IsZero(CoCoA::BigInt | const&) | [40] | | |

There are two points here:

1. why is it calling myBigExponent instead of myExponent (which should return a long)?

2. why is myBigExponent being called so many times? Isn't there a myBigExponentVec or similar?

#2 - 26 Oct 2020 16:59 - John Abbott

- Status changed from New to In Progress
- % Done changed from 0 to 10

I propose the following revision to the design:

- (A) each PPMonoid has a fn which says whether it can handle only "small" exponents (or whether big ones are possible); we can then have 2 print fns one which works only for small exps, and one which works for larger exps
- (B) the print fn should anyway use expvs (since computing one expv is surely no more expensive than computing each exp individually).
- (C) (not so sure about this) the "small" expv fn could return a boolean saying whether it succeeded -- this would allow using "small" expvs where possible (even if the PPMonoid could represent larger exps)

If we implement (C) then (A) may not longer be that useful.

#3 - 26 Oct 2020 20:06 - John Abbott

Here is a reference test:

N := 4*1024; use P ::= ZZ/(2)[x[1..N]]; f := sum(indets(P)); StartTime := CpuTime(); for i := 1 to 10 do str := sprint(f); endfor; println "sprint time: ", TimeFrom(StartTime); TimeFrom(0);

On my machine the loop took about 5.4s; altogether it took about 47s.

#4 - 26 Oct 2020 20:20 - John Abbott

I have just tried modifying the impl (PPMonoid.C around lines 215--230, PPMonoidBase::myOutput). The modified version calls myBigExponents and then runs through this vector. It takes 9.4s instead of 5.4.

Huh??? Time for a break!

#5 - 27 Oct 2020 15:21 - John Abbott

I have just repeated the experiment, but in CoCoALib. This is the test program:

```
ring P = NewPolyRing(NewZZmod(2), SymbolRange("x",1,4096));
RingElem f = sum(indets(P));
double t0 = CpuTime();
ostringstream out;
out << f;
double t1 = CpuTime();
cout << "len(out) = " << out.str().size() << endl;
cout << "Print time: " << t1-t0 << endl;</pre>
```

With myBigExponents printing took about 0.72s. With myBigExponent called several times, printing took about 0.56s

I tried running with profiling; then the times were reversed (i.e. myBigExponents was decidedly faster).

At the moment I cannot explain these results: they are contrary to my expectations. :-(

#6 - 27 Oct 2020 15:56 - John Abbott

- % Done changed from 10 to 20

I have just tried again but with SmallExponent_t being unsigned short (previously it was unsigned int). Printing times remain almost unchanged: 0.71s and 0.55s. For information: Total time was considerably lower: about 3.3s against 6.3s.

#7 - 29 Oct 2020 14:52 - John Abbott

Here is a guess as to why the observed times are as they are: with myBigExponents the ctor for BigInt is called NumIndets times, whereas with myBigExponent in a loop the ctor for BigInt is called just once.

As a test I could try replacing myBigExponents by myExponents (which is safe in the test I used)...

#8 - 30 Oct 2020 15:59 - John Abbott

Anna suggest using a virtual fn for printing which is specialized in PPMs which can have big exps.

JAA will think about it.

#9 - 03 Nov 2021 16:53 - John Abbott

- Target version changed from CoCoALib-0.99800 to CoCoALib-0.99850

#10 - 12 May 2023 07:21 - Anna Maria Bigatti

- Related to Design #1538: RingElem from string (ReadExpr) added

#11 - 15 Feb 2024 22:45 - John Abbott

- Target version changed from CoCoALib-0.99850 to CoCoALib-0.99900