CoCoALib - Design #268

Exponent range (in power products)

18 Oct 2012 13:47 - John Abbott

Status:	Closed	**Start date:**	18 Oct 2012
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Safety	**Estimated time:**	0.00 hour
Target version:	CoCoALib-0.9953	**Spent time:**	4.85 hours

Description

The C++ type used for represent exponents in powerproducts is defined in the file **config.H**. It has to be an unsigned integral type. The possibilities are:
(A) unsigned char
(B) unsigned short
(C) unsigned int
(D) unsigned long
(E) unsigned long long (using C++11)

Which one to use?

Obviously the smaller types give a more limited exponent range and more compact representations in memory (and probably slightly faster computation).

Related issues:

Related to CoCoA-5 - Bug #267: Cyclotomic(106743) fails	**Closed**	**18 Oct 2012**
Related to CoCoALib - Feature #269: PPMonoids: check for exponent overflow in...	**Closed**	**18 Oct 2012**
Related to CoCoALib - Design #683: Module index component in internal compres...	**Closed**	**14 Apr 2015**
Related to CoCoA-5 - Bug #275: Unhelpful error messages when SmallExponent_t ...	**Closed**	**15 Nov 2012**
Related to CoCoALib - Slug #1521: Unexpectedly slow example with larger types...	**New**	**26 Oct 2020**

History

#1 - 18 Oct 2012 14:13 - John Abbott

- Status changed from New to In Progress

The real candidates are (B) and (C) -- I interpret these as meaning 16 bits and 32 bits.

Option (A) would limit exponents to 255; this is fine for many computations (esp. with many indeterminates), but is too low for general use.

Options (D) and (E) are necessary only if exponents reach 2^32 or higher. I think it is extremely unlikely that such high exponents will ever occur in general use. CoCoALib does also offer PPMonoidEvZZ for unlimited exponents (presumably with a significant speed and space penalty).

The real question is how likely is one to encounter degrees greater than 2^16, and how onerous is the speed/space penalty of 32-bit exponents.

Regarding the space penalty, let me give an example. Imagine we are computing in a polynomial ring with 6 indeterminates and coefficients in a small finite field. I shall assume use of the most compact representation, namely DistMPolyInFpPP. Assuming that the representation of each summand contains no "holes" (*i.e.* unused chunks of memory), with 16-bit exponents each summand occupies 4+6*2+8=24 bytes of memory, whereas with 32-bit exponents each summand occupies 4+6*4+8=36 bytes of memory. So the space penalty is at most 50% when using 32-bit exponents compared to 16-bit exponents. Of course, the (relative) penalty will be greater if there are more indeterminates, and less if there are fewer; but in any case the relative penalty will never reach 100%.

Note that I have chosen the setting which most favours the smaller exponents; with any other representation, the (relative) space penalty would be much less, probably to be point of becoming negligible.

#2 - 20 Oct 2012 11:08 - John Abbott

- Assignee set to John Abbott

- Target version set to CoCoALib-0.9953

- % Done changed from 0 to 20

I ran the (old) benchmark set (inside src/server/).

There was **no real difference in speed** between a compilation using (B) and one using (C).
In one case (B) was about 3% faster; there were a few cases where the relative speed difference was about 2%, but in general the difference was only about 1%.

After looking through the CoCoAServer source code, it seems that in the computations modulo 32003 the polynomial ring used was indeed a RingDistrMPolyInInFpPP -- *i.e.* the situation which should exhibit the most evident difference in computation times.

At this point I propose: we make (C) the definition in the standard distribution, adding somewhere in the documentation (where exactly?) that if there are mild problems of space, one can try recompiling using definition (B); or if the exponent limit of 4*10^9 is too low then one can use definition (D).

#3 - 21 Feb 2013 16:31 - John Abbott

Any objections to adopting option **(C)** *i.e.* the default type for "small" exponents in power products is unsigned int?
NB Currently CVS uses unsigned short. This caused "mysterious" problems when trying to compute cyclotomic polynomials with index greater than 65535.

I note here also that BOOST has a means of specifying an integral type signed or unsigned and having at least a certain number of bits. See boost_integer.
For instance, if, by default, we want at least N bits in the small exponents then we would specify the default type as boost::uint_t<N>::fast.
At the moment we are still avoiding any BOOST dependency in CoCoALib.

#4 - 21 Feb 2013 16:41 - Anna Maria Bigatti

John Abbott wrote:

> Any objections to adopting option **(C)** *i.e.* the default type for "small" exponents in power products is unsigned int?
> **NB** Currently CVS uses unsigned short.

ok. at some point we should run some speed tests, but for the time being I prefer being "safer" than "faster"

20130221 JAA has already done this (see post 2). Perhaps Anna could confirm JAA's results?

#5 - 26 Feb 2013 17:54 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 20 to 80

JAA has modified config.H so that SmallExponent_t is now unsigned int.

Waiting for confirmation from Anna that the change does not incur any significant speed penalty. If she confirms, we can close this issue.

#6 - 15 Mar 2013 17:05 - John Abbott

2013-03-14 Soeger+Bruns report via email about 10% slow down when using **(C)** rather than **(B)**.

#7 - 22 May 2013 16:42 - John Abbott

- % Done changed from 80 to 90

Anna and John discussed the matter (in the light of Soeger+Bruns's experience).

The current proposal is to leave the default as **unsigned int** which favours safety over speed/compactness. Clear documentation must be added (**where??**) explaining how and why someone might want to change the default.

#8 - 31 May 2013 16:50 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

Added a comment in the newly organized installation guides (see INSTALL-advanced.txt). There is no perfect answer to this issue.

Closing this issue.

#9 - 08 Oct 2015 12:10 - John Abbott

- Subject changed from Exponent range to Exponent range (in power products)

#10 - 26 Oct 2020 12:05 - John Abbott

- Related to Slug #1521: Unexpectedly slow example with larger types for SmallExponent_t added