CoCoALib - Design #786

MemPool: review min and max loaf sizes

12 Oct 2015 15:44 - John Abbott

Status:	Closed	**Start date:**	12 Oct 2015
Priority:	Normal	**Due date:**	
Assignee:	Anna Maria Bigatti	**% Done:**	100%
Category:	Improving	**Estimated time:**	4.44 hours
Target version:	CoCoALib-0.99800	**Spent time:**	4.50 hours

Description

Currently each MemPool has a minimum and maximum loaf size set to 64K and 64M respectively.

Review these values; perhaps also making some performance checks?

A smaller minimum size may be desirable if we write any code which creates many rings (*e.g.* chinese remaindering).

Related issues:

Related to CoCoALib - Bug #1522: SEGV: avoid long linked lists of loaves in M...	**Closed**	**26 Oct 2020**

History

#1 - 12 Oct 2015 15:50 - John Abbott

I am reasonably sure that the upper limit is far too large: as it currently stands it can ask the system for 64M of contiguous RAM.

Probably the lower limit is needlessly large too: even 1K or 4K may be enough. Note that Singular's memory manager (called omalloc, I believe) works with 4K pages.

Note that if we do want to allow a MemPool to free some of its loaves then it is easier for a small loaf to be freeable than for a large one -- related to #437

#2 - 12 Oct 2015 15:58 - John Abbott

The problem arose when I experimented creating all (prime) finite fields up to characteristic 32767. There are about 3500 such fields. Finding all primes took less than 0.01s, but creating all the fields took about 0.35s. The time to create the fields was primarily due to the creation of one MemPool for each field. Also the space occupied was about 200Mbytes.

Reducing the smallest MemPool loaf to 4Kbytes, reduced the creation time to about 0.05s.

Independently of the above, in the end, I have decided to discard the idea of automatically creating all small prime finite fields at start up -- see #785.

#3 - 12 Oct 2015 16:01 - John Abbott

Someone should do some benchmarking to see if changing the lower/upper limits for loaf size has any measurable effect on run-time performance.
[JAA guesses that there will be almost no measurable effect]

#4 - 14 Oct 2015 09:21 - Anna Maria Bigatti

I think that implicit could be a good test.
I can do it, could you give me indications on the first tests to try? (and which lines I should change?)

#5 - 14 Oct 2015 15:04 - John Abbott

Thanks, Anna, for volunteering!

The relevant lines are in MemPool.C around line 100.

Change MinLoafBytes to 4*1024. Maybe you could also try 1024.
Change MaxLoafBytes to 64*1024. Maybe you could also try 1024*1024.

That's FOUR sets of experiments to try. My guess is that they will all
come out roughly the same for computation time.

#6 - 19 Oct 2015 18:09 - Anna Maria Bigatti

- Assignee set to Anna Maria Bigatti

#7 - 19 Nov 2015 15:06 - Anna Maria Bigatti

about to try..
(some compilation problems...)

#8 - 20 Nov 2015 14:38 - John Abbott

Are the problems my fault? Or some problem with the interface of MemPool? :-(

#9 - 20 Nov 2015 15:01 - Anna Maria Bigatti

John Abbott wrote:

> Are the problems my fault? Or some problem with the interface of MemPool? :-(

no, my problems. And then I was busy. Doing the tests again right now. Conclusions soon

#10 - 20 Nov 2015 17:26 - Anna Maria Bigatti

John Abbott wrote:

> Change MinLoafBytes to 4*1024. Maybe you could also try 1024.
> Change MaxLoafBytes to 64*1024. Maybe you could also try 1024*1024.

I made several combinations on an example taking approx 2 mins and creating 950 rings.
I cannot see any difference in the timings for 1, 4, 64...

#11 - 21 Nov 2015 11:27 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

Did you also try tests which create lots of ring elements? Maybe a non-trivial GBasis?

#12 - 23 Nov 2015 10:41 - Anna Maria Bigatti

John Abbott wrote:

> Did you also try tests which create lots of ring elements? Maybe a non-trivial GBasis?

I tried test-implicit2: now I see I tried only the "direct" algorithm.
Now I try with elimth as well.

#13 - 03 Dec 2020 20:58 - John Abbott

- Related to Bug #1522: SEGV: avoid long linked lists of loaves in MemPools added

#14 - 03 Dec 2020 20:59 - John Abbott

- Target version changed from CoCoALib-1.0 to CoCoALib-0.99800

#15 - 16 Feb 2021 18:03 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 10 to 80

After the design change noted in issue #1522#note-10, this issue is probably largely irrelevant now.

The current loaf size limits are 1k (**MinLoafBytes**) and 64k (**MaxLoafBytes**).
They have been like this for some time, and no one has complained; so advancing to 80% and "resolved".

#16 - 19 Mar 2021 13:29 - John Abbott

- % Done changed from 80 to 100

- Estimated time set to 4.44 h

This has been working for some time now... closing.
As mentioned above this is now largely irrelevant given the change from #1522#note-10.

#17 - 16 Sep 2021 13:38 - John Abbott

- Status changed from Resolved to Closed