CoCoALib - Bug #1522

SEGV: avoid long linked lists of loaves in MemPools

26 Oct 2020 13:00 - John Abbott

Status:	Closed	**Start date:**	26 Oct 2020
Priority:	High	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Safety	**Estimated time:**	3.99 hours
Target version:	CoCoALib-0.99850	**Spent time:**	3.95 hours

Description

See comment 4 below, where I understand the root cause.

Original report:
I get a SEGV reading about 4500 random lin forms in ZZ/(2)[x[1..1000]]

Reading the first 4000 works OK.

Investigate & fix.

Related issues:

Related to CoCoA-5 - Bug #1514: Cocoa crashes when calling RingElems	**Closed**	**22 Oct 2020**
Related to CoCoALib - Feature #142: Improve threadsafety	**In Progress**	**30 Apr 2012**
Related to CoCoALib - Design #786: MemPool: review min and max loaf sizes	**Closed**	**12 Oct 2015**

History

#1 - 26 Oct 2020 13:00 - John Abbott

- Related to Bug #1514: Cocoa crashes when calling RingElems added

#2 - 26 Oct 2020 13:08 - John Abbott

- Status changed from New to In Progress

- Assignee set to John Abbott

- % Done changed from 0 to 10

Triggered by the example from issue #1514.
The SEGV seems to arise during destruction at the end of the program!

Current situation: (running on my linux box with 32Gbytes RAM)

- get SEGV reading 4500 polys (each about 500 terms) in ZZ/(2)[x[1..1000]], so will be using RingDistrMPolyInlPPFp. SmallExponent_t was unsigned int (took about 112s)
- No SEGV with SmallExponents_t being unsigned char (took about 58s);
- SEGV with SmallExponents_t being unsigned short (took about 75s)
- No SEGV with SmallExponents_t being unsigned short and NO debugging

#3 - 26 Oct 2020 13:53 - John Abbott

I got a SEGV with debugging on and MemPool disabled. Let's see what valgrind says...
BTW it was even slightly faster without MemPool :-/

#4 - 26 Oct 2020 14:38 - John Abbott

Ahhh! Perhaps I should have guessed: valgrind reports that the problem is **stack overflow**.
Now, why are we using so much stack space??? But why did that not happen when SmallExponent_t is unsigned char???

Time for a break, then I can investigate.

#5 - 26 Oct 2020 17:07 - John Abbott

- % Done changed from 10 to 20

I think I have found the cause... it is in "invisible code" written by the compiler. The invisible code is correct, but can trigger deep recursion.

I think there are (at least) two aspects to the problem:

1. a loaf is an element of a linked list with an explicit (raw) pointer to the next loaf; destruction is a recursive process, so if the linked list becomes long then the recursion can become too deep --> SEGV
2. the linked list will become long if there are many "active" allocated slots, and the max loaf size is small (currently 64k bytes)
3. the MemPool and loaf mechanism works well only if the individual slots are not too large; for larger slots it is probably better just to less the system allocator handle the requests.

In the example which triggered the problem, the linked list probably contained over 100000 entries... hardly surprising that the stack overflowed!

#6 - 26 Oct 2020 19:49 - John Abbott

I now think it is probably a good idea to abandon the current design with explicit "NextLoaf" pointers (currently std::unique_ptr).
Instead we could maintain a std::vector of loaves, and use push_back; so that this would work well, there should be a move-ctor for loaves (I suppose).

I'm hoping this won't be too troublesome to implement.

#7 - 26 Oct 2020 19:50 - John Abbott

- Subject changed from SEGV: RingElems with long list and many indets and debugging ON to SEGV: avoid long linked lists of loaves in MemPools

#8 - 26 Oct 2020 19:51 - John Abbott

- Description updated

#9 - 09 Nov 2020 21:14 - John Abbott

- Related to Feature #142: Improve threadsafety added

#10 - 03 Dec 2020 20:28 - John Abbott

- % Done changed from 20 to 30

I have made a first design improvement:
There is now **MaxSliceSize** (in bytes); if the requested slice size is greater than this limit then all alloc/free calls are forwarded to the system mem mgr (namely, ::operator new and ::operator delete).

NOTE: both MemPoolFast and MemPoolDebug have been updated.

Currently the limit is set to 128 bytes -- no idea if this is really a good value to use.

This does not solve the original problem, but should make it less likely to occur.

#11 - 03 Dec 2020 20:58 - John Abbott

- Related to Design #786: MemPool: review min and max loaf sizes added

#12 - 16 Feb 2021 18:46 - John Abbott

- % Done changed from 30 to 50

I have just run the test from issue #1337.
With MemPool the program took 473s; disabling MemPool by setting the max slice size very low (to 12), the test took 489s.

I suppose I'm slightly disappointed that MemPool does not make a bigger difference, but probably memory allocation was only a small proportion of the total computation time.

Further test:

```
/**/ use R ::= ZZ/(101)[x[1..30]];
/**/ L := [RandomLinearForm(R) | i in 1..2000000];  TimeFrom(0);
46.460
/**/ t0 := CpuTime(); L := []; TimeFrom(t0);
1.713
```

If I reduce the number of indets to 24 (which should allow MemPool to operate) then the times become 33s and 0.76s.
Also memory consumption was considerably greater with 30 indets: 8.4Gbytes against 5.3Gbytes with just 24 indets.

This makes me inclined to increase max slice size to 256bytes.

Still, the system allocator is not bad at all (and is presumably even thread-safe).

#13 - 19 Mar 2021 13:15 - John Abbott

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

#14 - 08 Mar 2023 21:37 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 50 to 90

I'm not convinced the test in comment 12 is really a good test.
Anyway, i have reproduced results which suggest that 256 is a reasonable limit.

Propose closing this issue (even though it is not truly resolved... but I don't want to mess about inside MemPools).

#15 - 09 Mar 2023 21:50 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

- Estimated time set to 3.99 h