CoCoALib - Bug #1376

GBasisTimeout: not working as expected

12 Dec 2019 17:02 - John Abbott

Status:	Closed	**Start date:**	12 Dec 2019
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Improving	**Estimated time:**	5.23 hours
Target version:	CoCoALib-0.99800	**Spent time:**	5.00 hours

Description

It seems that sometimes GBasisTimeout does not timeout as one would expect.

I have just interrupted (via C-c C-c) a GBasisTimeout(J,30) call after almost 6000s CPU.
Why does a C-c interrupt work but not a timeout interrupt?

Investigate, and correct.

Related issues:

Related to CoCoALib - Slug #1375: Radical 0-dim: varied timings	**Closed**	**09 Dec 2019**
Related to CoCoALib - Slug #1181: CpuTime is costly!	**Closed**	**23 Apr 2018**
Related to CoCoALib - Design #1377: CpuTimeLimit: limit "intervals" between f...	**Closed**	**13 Dec 2019**
Related to CoCoALib - Design #1558: CpuTimeLimit: more frequent clock checks	**Closed**	**08 Jan 2021**

History

#1 - 12 Dec 2019 17:02 - John Abbott

- Related to Slug #1375: Radical 0-dim: varied timings added

#2 - 12 Dec 2019 17:04 - John Abbott

I noticed the problem while using a simple program to look for ideals where it was costly to compute the RGB of the radical.

I do have a vague recollection that I changed the code which checks for timeout, so that it calls CpuTime only rarely; perhaps too rarely?

#3 - 12 Dec 2019 21:44 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 40

It seems that my guess was good.

In CpuTimeLimit.C around line 67, the code allowed the interval between full checks to increase to once every 1000000.
I have now limited it to 1024, and the timeout works more or less as expected.

I'm still considering a further minor improvement.

#4 - 12 Dec 2019 21:45 - John Abbott

- Related to Slug #1181: CpuTime is costly! added

#5 - 13 Dec 2019 12:45 - John Abbott

- Assignee set to John Abbott

- % Done changed from 40 to 50

An important point to note is that the design of CpuTimeLimit works best when the times between successive calls to the "check-for-timeout" function are roughly equal.

However, the cost of reductions inside a GB computation can vary considerably (and probably rather unpredictably), so the times between successive calls are not all roughly equal.

If the times between calls are all roughly equal (or change only slowly and steadily) then it will still work well if the CpuTimeLimit object makes full checks only very rarely (*e.g.* one every million). If the times between calls are quite variable then full checks should not happen too rarely.

My tests yesterday evening produced reasonably good results when I limited the interval between full check to a maximum of 512; allowing 1024 already permitted a significant overshoot (38s instead of the requested 30s).

I am now contemplating letting the creator of a CpuTimeLimit object specify that larger intervals may be used; so the default would be to limit intervals to a maximum of 512 (say), but an optional ctor arg could permit a larger value -- it is then the creator's responsibility to set a higher interval limit only when appropriate.

#6 - 13 Dec 2019 13:26 - Anna Maria Bigatti

I was thinking of an alternative approach since the timeout mechanism is applied in so different cases:
The interval should adapt to the expected time of a single loop. If the estimate is small (<0.001s?), increment the interval (to minimize overhead), if big decrease the interval (giving better estimates with relatively small overhead).
Maybe it already does that?

Of course an input from the user for the first interval size would help for an optimal start.

#7 - 13 Dec 2019 13:30 - John Abbott

- Related to Design #1377: CpuTimeLimit: limit "intervals" between full checks added

#8 - 14 Dec 2019 12:32 - John Abbott

Reply to comment 6:
yes, CpuTimeLimit does already attempt to adapt automatically. I'm discussing (with myself?) some improvements in issue #1377.

#9 - 20 Dec 2019 16:58 - John Abbott

The main problem with checking for timeout during GB computation is that the times for each single reduction can vary enormously (fast may be just a few microsecs, long could be 10+ secs).

I think it may be appropriate to put a check for time-out even inside the reduction code (with the hope that the extra overhead incurred is negligible). It is also true that each single step of the reduction can have quite widely varying times, but I'm hoping the variation is not so pronounced).

I have not yet made any experiments.

#10 - 21 Dec 2019 17:22 - John Abbott

I have just spent quite a while trying to work out where I could insert a time-out check inside the reduction code.

TBH I failed to work out which part of the code is relevant :-(
[perhaps someone else can have a look?]

#11 - 21 Dec 2019 17:31 - Anna Maria Bigatti

John Abbott wrote:

> I have just spent quite a while trying to work out where I could insert a time-out check inside the reduction code.
> TBH I failed to work out which part of the code is relevant :-(
> [perhaps someone else can have a look?]

I will

#12 - 14 Jan 2020 21:06 - John Abbott

- Related to Support #1387: John's visit Feb 2020 added

#13 - 14 Feb 2020 10:55 - John Abbott

- Related to deleted (Support #1387: John's visit Feb 2020)

#14 - 22 Dec 2020 16:16 - John Abbott

What is the status of this issue?

#15 - 07 Jan 2021 20:19 - John Abbott

Why are there no test cases for this issue? Without them how can i know if it has been satisfactorily solved?

Using the "french students' example" it seems that this has **not** been satisfactorily solved :-(

```
use QQ[x,y,z],Lex;
I := ideal(x^2*y*z + x*y^3*z - 1, x^4*y*z - 1 , x*y^4 + x*y*z-1 );
SetVerbosityLevel(100);
t0 := CpuTime(); GB := GBasisTimeout(I, 30); println "Intr after ", TimeFrom(t0);
--> printed out 95s
```

Taking more than 3 times as long as requested is disappointing (but perhaps not wholly unacceptable).

What to do?

#16 - 07 Jan 2021 20:42 - John Abbott

Setting the timeout to 20s worked very well. Setting it to 25s triggered an interruption after 95s.
Setting it to 97s triggered an interruption after >900s

I must investigate further; this is not really acceptable.

#17 - 07 Jan 2021 20:54 - John Abbott

The relevant source is probably in **SparsePolyOps-reduce.C** around line 133.

The timer object is constructed in *BuiltinFunctions-CoCoALib.C around line 567.

#18 - 08 Jan 2021 09:48 - John Abbott

Now I think it should be possible to make CpuTimeLimit check the time more frequently without much overhead (by checking ElapsedTime rather than CpuTime for most checks).
This should avoid the problems reported here.

I have added a new issue (#1558) for exploring this idea.

#19 - 08 Jan 2021 16:09 - John Abbott

- Related to Design #1558: CpuTimeLimit: more frequent clock checks added

#20 - 08 Jan 2021 18:46 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 50 to 80

Using just the one example "french students" example, the current (2021-01-08) version of CpuTimeLimit works acceptably well (in a number of tests).

#21 - 16 Sep 2021 13:51 - John Abbott

- Status changed from Resolved to Closed

- % Done changed from 80 to 100

- Estimated time set to 5.23 h