CoCoALib - Design #1377

CpuTimeLimit: limit "intervals" between full checks

13 Dec 2019 13:30 - John Abbott

Status:	Closed	**Start date:**	13 Dec 2019
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Improving	**Estimated time:**	5.33 hours
Target version:	CoCoALib-0.99700	**Spent time:**	5.30 hours

Description

A **CpuTimeLimit** object actually genuinely checks the time limit only every N calls to the check function. The value of N is changed heuristically during execution; however N can become too large. Find a better heuristic.

It may help to distinguish two use cases: where the calls to the check function are very regular, and where the calls are (or may not be) so regular.

Related issues:

Related to CoCoALib - Bug #1376: GBasisTimeout: not working as expected	**Closed**	**12 Dec 2019**
Related to CoCoA-5 - Support #1387: John's visit Feb 2020	**Closed**	**07 Jan 2020**

History

#1 - 13 Dec 2019 13:30 - John Abbott

- Related to Bug #1376: GBasisTimeout: not working as expected added

#2 - 13 Dec 2019 13:39 - John Abbott

In the current CVS version, intervals between genuine checks can be as long as 1000000 calls.

Quick reminder: issue #1181 arose because calling CpuTime can be quite costly, so it should not be called too often; as a consequence, I changed the impl of CpuTimeLimit so that a genuine check is made only every so often (and the frequency of the full checks is determined heuristically based on past computation speed).

In the problem reported in issue #1376 this interval was far too long, and a supposedly time-limited computation was effectively not time limited. The critical feature in that case was the calls to the check function did not occur a very regular intervals: *i.e.* the reductions inside a GB computation can require times which vary considerably.

A naive code change, limiting the longest interval to 512, produced satisfactory results. In my one test, allowing a limit of 1024 was less satisfactory, since the timeout was actually triggered after 38s when a limit of 30s had been requested.

#3 - 13 Dec 2019 13:46 - John Abbott

My current suggestions:

- **(A)** let the creator of the CpuTimeLimit object specify a maximum interval between genuine timeout checks (via an optional(?) extra arg).
- **(B)** impose a (global) lower maximum.

Approach **(A)** does put some extra onus on the programmer. I think there should be a reasonable default which works well even when the timeout-check function is called somewhat irregularly, but is not too costly.

Approach **(B)** is definitely KISS; my main doubt is that it might be too costly if checks are frequently made in a fast loop. *e.g.* one could imagine a loop which can be very fast for certain inputs, but also much slower for other inputs; it would be a shame to penalize the fast case unnecessarily.

#4 - 13 Dec 2019 14:33 - John Abbott

Here are the results of an extreme speed test for option **(B)** in the previous comment.

The simplistic test program is:

```
    long l = 0;
    const int mask = (1<<25)-1;
    CpuTimeLimit CheckForTimeout(1.0);
    for (long i=0; i < 2000000000L; ++i)
    {
      if ((i&mask)==0) cout << i << endl;
      CheckForTimeout("loop");
      l += i;
    }
```

Of course, one would normally expect CpuTimeLimit to be used with loops where each iteration is significantly more costly. This is intended to be an extreme case. The use of mask is just so that I can easily roughly how far the loop went before the timeout occurred.

Results:
without CheckForTimeout: 1.23s for whole loop (about 60*2^25 iters)

```
  Limit       NumIters
   256          7*2^25
   512         10*2^25
  1024         13*2^25
  4096         16*2^25
  8192         16*2^25
 16384         17*2^25
 65536         17*2^25
262144         17*2^25
```

Conclusion: little further gain beyond a limit of 4096, and still less beyond 16384; however, there is a significant penalty at 256, and still a penalty at 512.

#5 - 13 Dec 2019 18:17 - John Abbott

- Status changed from New to In Progress

- Assignee set to John Abbott

- % Done changed from 0 to 20

The experimental results above suggest that approach **(B)** is probably too simplistic if there is any risk of a CpuTimeLimit object being used for timeout checks in a fast-iterating loop.

How to implement some form of approach **(A)**?

- **(A1)** the ctor could have an optional "boolean" arg which the caller uses to choose between two hard-wired upper limits (say, 256 and 16384; the default is the lower limit!)
- **(A2)** the ctor could have an optional (machine int?) arg which the caller uses to set an upper limit explicitly (but no larger than 65535, say?); what should the default be? 256? 512?

#6 - 14 Dec 2019 12:37 - John Abbott

Now I'm thinking it may be simpler to use an approach like this:
measure actual CpuTime difference over the last interval;
compare this time with the time remaining;
if interval time is much smaller than time remaining then increase number of steps in an interval;
if interval time "close" to time remaining then reduce number of steps in an interval.

The idea is that "close to" can have a flexible definition: *e.g.*

- if user says iterations take nearly constant time then "close to" means perhaps "greater than 1/2 time remaining"
- if user says iterations may take highly variable amounts of time then "close" means perhaps "greater than 1/100 time remaining"

I hope to do some experiments later today.

#7 - 20 Dec 2019 16:34 - John Abbott

- % Done changed from 20 to 50

I have an improved impl, which I hope to check in fairly soon (later today?).

A problem with the description in comment 6 is that if the number of "iterations" in an interval is too low then the measured times can vary wildly, and the algorithm gets "confused". So I have modified it to take an average time of the last several iterations (where several > 15). This seems to produce satisfactory results.

A slight redesign has even made the impl a little bit faster (when testing inside an almost empty loop).
[using the technique for the table in comment 4, I measured 21*2^25 when the limit was 65536]

#8 - 20 Dec 2019 16:59 - John Abbott

- Target version changed from CoCoALib-0.99800 to CoCoALib-0.99700

- % Done changed from 50 to 60

I have checked in the code (even though it is not quite finished yet). It worked tolerably well in my two tests.

#9 - 09 Jan 2020 11:32 - John Abbott

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

#10 - 17 Jan 2020 22:20 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 60 to 90

- Estimated time set to 4.55 h

I think the current impl is acceptable; so moving to *feedback*. Doc seems OK too.

#11 - 11 Feb 2020 12:09 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

- Estimated time changed from 4.55 h to 5.33 h

Improved doc with Anna's help. Closing.