CoCoALib - Design #1558

CpuTimeLimit: more frequent clock checks

08 Jan 2021 09:38 - John Abbott

Status:	Closed	**Start date:**	08 Jan 2021
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Improving	**Estimated time:**	6.99 hours
Target version:	CoCoALib-0.99800	**Spent time:**	7.00 hours

Description

The impl of CpuTimeLimit is a bit complicated because the CpuTime function is more expensive than I had expected.
The workaround I impl'd was to check the CpuTime only sometimes (other times the time limit check was just a no-op, with a decrement of a counter).

Unfortunately this imp/design does not work well for GBasis computation (see issue #1376).

A quick check shows that ElapsedTime (based on C++ "steady clock") is much faster than CpuTime.
So the idea is now to make CpuTimeLimit check ElapsedTime instead of CpuTime (except perhaps for a final check?)
This would allow checks to be made more frequently, and thus avoid the problem reported in issue #1377.

Related issues:

Related to CoCoALib - Bug #1376: GBasisTimeout: not working as expected	**Closed**	**12 Dec 2019**

History

#1 - 08 Jan 2021 10:07 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

Here is the code I used as a speed check:

```
    double MaxT = 0;
    BigInt witness;
    for (long n = 0; n < 10000000; ++n)
    {
      BigInt N = BigInt(n*n+n+163);
      double t0 = /*ElapsedTime();*/CpuTime();
      bool b = IsPrime(N);
      double t1 = /*ElapsedTime();*/CpuTime();
      if (t1-t0 > MaxT) { MaxT = t1-t0; witness = N; }
    }
    cout << "MaxT = " << MaxT << "   witness=" << witness << endl;
```

On my linux box the times were: 3.9s (just setting t0 and t1 to constants), 4.3s (using ElapsedTime) and 9.9s+6.5s (sys) using CpuTime.

#2 - 08 Jan 2021 16:09 - John Abbott

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

#3 - 08 Jan 2021 16:11 - John Abbott

Now I am wondering whether it may not make more sense to base CpuTimeLimit on the "steady clock" (but then the name would have to change since it no longer measures CPU time).

#4 - 08 Jan 2021 18:11 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 10 to 50

I now have a version of CpuTimeLimit, which actually uses only "steady clock" rather than CpuTime.

It seems to work OK in the two tests I have tried.

Quick loop test:

```
    CpuTimeLimit CheckTime(3.5, 2);
    long counter = 0;
    for (long n = 0; n < 10000000; ++n)
    {
      CheckTime("Loop");
      if (IsPrime(n*n+n+163)) ++counter;
    }
    cout << "counter = " << counter << endl;
```

With a time limit of 3.5s the timeout triggers; with a limit of 3.6s it does not. There is very little run-time overhead: it takes longer than 3.5s if I comment out the line CheckTime("loop");

GBasis test: this is trickier because the calls to check the time are quite irregular:

```
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, 40); println "Intr after ", TimeFrom(t0);
```

I have tried various time limits, and usually the computation is interrupted after no more than 10% longer than requested.

#5 - 11 Jan 2021 08:55 - Anna Maria Bigatti

John Abbott wrote:

> Now I am wondering whether it may not make more sense to base CpuTimeLimit on the "steady clock" (but then the name would have to change since it no longer measures CPU time).

I think we'd better leave automatic limit based on CPU because otherwise one could have different behaviour, and CPU time, based on his computer load (and that could be, at least, quite confusing)

#6 - 12 Jan 2021 14:32 - John Abbott

- Assignee set to John Abbott

- % Done changed from 50 to 70

I have modified the impl so that it triggers only when the CPU time limit has been exceeded, but it does try to call CpuTime only rarely.
It is hard to test whether the mechanism really works: I tried running 5 copies in parallel (my computer has 4 pseudo-cores), and the mechanism seems to work as hoped.

#7 - 26 Jan 2021 13:47 - John Abbott

While looking for examples for my course computer algebra I wanted to use GBasisTimeout. Sometimes the curr impl did not work so well.
I choose 30s timeout, but sometimes it took much longer...

```
use QQ[x,y,z],lex;
SLOW INTR: ["89.025",  ideal(3*x^2*z -4*x*y*z +4*y^2*z +3*y^2,  2*x^2*y +x*y*z +4*y^2*z +z^2,  -x^2*y -2*x -2*
y^3 +2*z^3)]
SLOW INTR: ["136.54",  ideal(-x^2*z -x*y^2 +2*x +2*z^2,  2*x*y +2*x*z^2 -x*z -2*y^3,  2*x^3 +y)]
SLOW INTR: ["216.64",  ideal(x^2 -2*y^3 -3*y^2 +2*z^2,  -4*x^3 -4*x^2 +5*x*y*z -3*y^2,  -x*y^2 -4*x*y +5*x -y*
z)]
SLOW INTR: ["378.31",  ideal(-2*x^2*z +4*x*z +4*x -3*y^3,  -2*x^2*y +3*x*y*z +3*z^2 +4,  -3*x^2*z +4*x*y*z +4*
x*y)]
```

These were with a 10s timeout:

```
SLOW INTR: ["49.475",  ideal(-3*x^3 -5*x^2*z +3*x -2*y,  5*x^2 -3*x -5*y*z^2 -3*z^2,  3*x^2*y -5*x*z -y^3 -3*y
^2*z)]
SLOW INTR: ["197.31",  ideal(3*x^2*z -2*x*y^2 -2*x*z +2*y^3,  -3*x^3 +5*x*y +4*y^3 -y*z,  -3*x*y*z +4*x +2*y*z
 +2*z^3)]
```

Perhaps investigate?

#8 - 17 Feb 2021 20:33 - John Abbott

I have done a speed test using the "quick" example from comment 4.
The speed penalty when I limited the number of iters between "elapsed" checks to 16 was probably about 1% (and no more than 3%).
I'm pleasantly surprised; of course, I do not know if "elapsed time" calls are so fast on all platforms (though it seems likely).

#9 - 18 Feb 2021 17:21 - John Abbott

I have revised the impl based on the results from yesterday's speed tests.

The current impl seems to work acceptably in all cases tried: I tried the examples from comment 7, I tried the speed test, I also tried the 10s tests from comment 7 under high load (there were 20 copies of CoCoA running the same examples). All tests worked satisfactorily.

I shall shortly check-in even though some cleaning is still necessary.

#10 - 20 Sep 2021 15:41 - John Abbott

- Status changed from Resolved to Closed

- % Done changed from 70 to 100

- Estimated time set to 6.99 h

No problems have been reported in 7 months. So closing.
Maybe the imminent CoCoA school will find some "interesting cases".