CoCoALib - Bug #413

OrdvArith: use of a single buffer is NOT THREADSAFE

21 Nov 2013 19:14 - John Abbott

Status:	Closed	**Start date:**	21 Nov 2013
Priority:	High	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Safety	**Estimated time:**	5.00 hours
Target version:	CoCoALib-0.99532	**Spent time:**	4.00 hours

Description

Fix OrdvArith fns so that they are threadsafe; at the moment they use a single buffer (myOrdvBuffer and myExpvBuffer) in a non-threadsafe manner.

PS this bug was originally reported by Bruns

Related issues:

Related to CoCoA - Support #425: Osnabrueck 2014-01	**Closed**	**27 Jan 2014**
Related to CoCoALib - Support #195: OrdvArith documentation needs rewriting	**Closed**	**22 Jun 2012**
Related to CoCoALib - Bug #428: PPMonoidOv is not threadsafe	**Closed**	**28 Jan 2014**
Related to CoCoALib - Feature #142: Improve threadsafety	**In Progress**	**30 Apr 2012**

History

#1 - 23 Nov 2013 17:09 - John Abbott

- % Done changed from 0 to 10

JAA ha now implemented a partial fix to the problem: the new design should allow the multithreaded case to be handled with ease (but currently it is restricted just to single-threaded).

JAA also wonders whether the use of a buffer constitutes a "premature optimization". We should try benchmarking an impl which uses a local variable for the buffer against the current (thread-unsafe) version which uses a "global" buffer. If the speed difference is minor then we can simply use a buffer local to each fn call. Personally I suspect that the repeated new/delete of buffer space will lead to a perceptible reduction in run-time performance (perhaps just because it messes up the heap?)

#2 - 25 Jan 2014 22:10 - John Abbott

- Status changed from New to In Progress

- Assignee set to John Abbott

We should discuss this in Osnabrueck next week.

(A) the solution using local buffers in every fn that needs one is fully safe and portable, but incurs significant overhead in the single-threaded case;

(B) JAA's solution using a vector of buffers requires special compilation for the threadsafe version, but it does localise the threadsafe special bits in just a few fns.

(C) we could revert to the old soln for single-threaded compilation, and make threadsafe compilation "see" code as described in **(A)** -- this way the threadsafe special bits are more spread about than in **(B)**, but still localised inside OrdvArith

Notes after a discussion with Christof:
(1) approach **(B)** surely looks the most promising, but we know how to do it only if OpenMP is being used to manage the multithreading; it requires 3 sections of code to know about multithreading (ctor, myExpvBuffer and myOrdvBuffer)

(2) Christof has run some tests: approach **(A)** works surprisingly well in a true multithreaded environment; no evidence of a speed penalty compared to **(B)** was observed!

#3 - 28 Jan 2014 18:04 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 10 to 90

Somewhat unexpectedly Christof's tests shows that approach **(C)** is as fast as approach **(B)** but it is also simpler and more general. So I have adopted approach **(C)**; the code has been cleaned anc checked in!

Changed status to "feedback" while Christof makes a few more tests (but we fully expect it the problems to be resolved).

#4 - 30 Jan 2014 18:59 - Christof Soeger

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

Several test were successful. I consider it as done.

#5 - 31 Jan 2014 20:17 - John Abbott

- Status changed from Closed to Feedback

@Christof, I have unclosed the issue because Anna proposed to redo your long computation using impl **(B)**, just to confirm that it really goes no better than **(C)**. Would you be willing to do this?

Do you still happen to have a copy of my code? I deleted my copy.

#6 - 03 Feb 2014 11:07 - Christof Soeger

Yes, I still have it and will give it a try.

#7 - 03 Feb 2014 11:47 - Christof Soeger

I did some tests and the timings were exactly the same. So I checked again. I only have a version which still includes a check for TSH ==1, but only at one place, at including the omp header.
In short, I don't have that version anymore.

#8 - 03 Feb 2014 12:43 - John Abbott

OK, I'll try to rewrite it -- maybe even this afternoon while invigilating!

#9 - 03 Feb 2014 17:37 - Christof Soeger

```
CVS
time nmzIntegrate -E CondEffPlurSymm-100
nmzIntegrate -E CondEffPlurSymm-100  218,09s user 0,69s system 385% cpu 56,729 total
nmzIntegrate -E CondEffPlurSymm-100  223,31s user 0,57s system 384% cpu 58,219 total
nmzIntegrate -E CondEffPlurSymm-100  211,24s user 0,63s system 383% cpu 55,199 total
nmzIntegrate -E CondEffPlurSymm-100  203,23s user 0,73s system 383% cpu 53,209 total

with the openmp "improvement"
time ~/source/genEhrhart-John/nmzIntegrate -E CondEffPlurSymm-100
213,50s user 0,70s system 377% cpu 56,780 total
245,01s user 0,68s system 375% cpu 1:05,45 total
233,61s user 0,74s system 371% cpu 1:03,00 total
230,74s user 0,76s system 377% cpu 1:01,40 total
```

will write more later

Note (JAA) looks like the vector of buffers is about 10% slower on average.

#10 - 04 Feb 2014 19:42 - Christof Soeger

On our compute server with 20 threads the performance was about 5% better with the OpenMP adaptation for some medium sized examples.

Times for our large example:
current cvs:
264661.90user 85.55system 3:42:47elapsed 1980%CPU
OpenMP adaption:
246289.10user 82.55system 3:29:14elapsed 1962%CPU
cvs with short as exponents:
232235.84user 82.66system 3:14:34elapsed 1990%CPU
OpenMP adaption with short as exponents:
207053.24user 79.09system 3:09:58elapsed 1817%CPU

We would like it, but the gain is not so big that we really need it. I guess the question is whether you want to have it in the CoCoALib.

#11 - 13 Feb 2014 12:02 - John Abbott

Christof could you make some further tests after modifying slightly the OpenMP version of the OrdvArith code I sent?

The changes are to two lines in the ctor for OrdvArith::base, about lines 65-66 of OrdvArith.C.

The lines are currently like this:

```
    myOrdvBuffers(omp_get_max_threads(), vector<OrdvElem>(NumOrdvEntries)),
    myExpvBuffers(omp_get_max_threads(), vector<long>(NumIndets))
```

Could you change them to this:

```
    myOrdvBuffers(omp_get_max_threads(), vector<OrdvElem>(std::max(32l,NumOrdvEntries))),
    myExpvBuffers(omp_get_max_threads(), vector<long>(std::max(32l,NumIndets)))
```

Note that the number is 32 with suffix l for long; it is **not** 321, three hundred and twenty one.

The motivation for making this strange change is what Scott Meyers wrote about cache memory.

Let me know if it improves the run times!

#12 - 18 Feb 2014 16:11 - Christof Soeger

I tried the change. It is really hard to tell if there is a difference. It seems to be slightly faster on some examples, e.g.
for the 3,5h example we now get 2 minutes less:
246462.30user 89.56system 3:27:23elapsed 1981%CPU
Done this twice, nearly same result. But still it is not very significant.

#13 - 18 Feb 2014 17:17 - John Abbott

Thanks Christof for trying the hacked code. I'm both surprised and relieved that there was no significant improvement. Maybe Meyers's comment about cache lines is not appropriate in this instance...

To respond to your question at the end of #note-10, my preference is to keep the code as clean as possible (without incurring a significant run-time overhead). To my mind a 5% reduction in speed is acceptable, but you're the ones who notice it.
[I recall also that the best gain was 5%, sometimes there was a loss, #note-9]

#14 - 02 Apr 2014 13:08 - John Abbott

- Status changed from Feedback to Closed

This has been in feedback for over a month, so closing.

As stated in my previous note, the OpenMP special version is not demonstrably superior to the fully portable version (just occasionally slightly better, but sometimes worse); so the official code is the fully portable version, but I have put the OpenMP special version in CVS (OrdvArith-OpenMP.C) though it is not used -- I just didn't want to lose it.

#15 - 03 Apr 2014 11:53 - Anna Maria Bigatti

- Estimated time set to 5.00 h