CoCoALib - Design #1804

Use long long (at least sometimes)?

25 Mar 2024 19:29 - John Abbott

Status:	In Progress	**Start date:**	25 Mar 2024
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	80%
Category:	Portability	**Estimated time:**	0.00 hour
Target version:	CoCoALib-0.99900	**Spent time:**	1.45 hour

Description

Winfried Bruns suggested in issue #1661 to consider using **long long** wherever we want more than 32-bits.
Investigate, discuss, and implement (if we decide to make the change).

Another option might be int64_t (but there are only optional... not good for portability).

NOTE mysteriously Brun's comment on issue 1661 is no longer there -- how did that happen??

Related issues:

Related to CoCoALib - Bug #1661: Microsoft: cannot compile with signal handling	**Closed**	**09 Feb 2022**
Related to CoCoALib - Support #1666: MachineInt: chase through ULL changes	**In Progress**	**16 Feb 2022**

History

#1 - 25 Mar 2024 19:29 - John Abbott

- Related to Bug #1661: Microsoft: cannot compile with signal handling added

#2 - 25 Mar 2024 19:31 - John Abbott

Personally I was hoping to drop support for 32-bit platforms, but long on MinGW is only 32-bits. Does MinGW offer long long? I suppose so.

#3 - 09 Apr 2024 21:53 - John Abbott

I am a bit concerned that **long long** may incur unnecessary overhead on some platforms.

We could also have a CoCoA typedef for a 64-bit integer (being one of int, long int or long long int). This would avoid the portability doubts related to int64_t. Not sure this is a such a good idea...?

#4 - 13 Apr 2024 22:21 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

Nico sent the following comment by email:

```
That’s a good question. Indeed, long longs perform considerably worse (on average apparently ~ 2x worse accord
ing to this guy: https://stackoverflow.com/q/33848357/5894824).
So, it might really be better to stay with longs, I think.
```

#5 - 13 Apr 2024 22:38 - Nico Mexis

> Personally I was hoping to drop support for 32-bit platforms, but long on MinGW is only 32-bits.

Actually, MinGW-w64-compiled code is still "64-bit code" - just with this difference in the 64-bit data model (LLP64 vs LP64). The original 32-bit MinGW implementation is also rather "dead", by the way.

> Does MinGW offer long long? I suppose so.

Yes, it does. And it is guaranteed to be 64 bits wide.

> We could also have a CoCoA typedef for a 64-bit integer

This is indeed exactly what I had also thought about, but I was unsure whether it is really necessary when one can also just use long long in the first place.

#6 - 14 Apr 2024 09:20 - John Abbott

The more I think about making a "typedef", the less I am convinced. For convenience I shall suppose it is called **CoCoA_LONG**

- **Pros** we can simply use CoCoA_LONG everywhere where 64-bit values may occur
- **Cons** interfacing to GMP might become a problem (similarly for any other library which has an API using long)

Indeed, if we want to use a non-standard type for 64-bit values, it'd probably be better to use int64_t (or similar), since these are at least officially documented (but also documented as *optional*).

Overall, I fear that a "typedef" would cause interfacing problems. Also, I'm never inclined to put in much effort to circumvent obstacles on Microsoft platforms...

#7 - 14 Apr 2024 20:44 - Nico Mexis

To be honest, I would also rather keep support for 32-bit platforms instead of using e.g., long longs which are maybe not compatible with GMP or int64_ts which, on the other hand, are not compatible with some (maybe even all... Haven't tested?) 32-bit platforms.

#8 - 15 Apr 2024 10:03 - John Abbott

- Related to Support #1666: MachineInt: chase through ULL changes added

#9 - 15 Apr 2024 10:14 - John Abbott

My current thoughts are that we should avoid using LL/ULL in any (normal) user interfaces, but we may use them internally *e.g.* they might be useful for some CRT-based methods (assuming LL-arithmetic is not much slower than for (unsigned) long) since there would be only about half as many iterations.

#10 - 15 Apr 2024 22:22 - John Abbott

Winfried Bruns sent the following response by email:

```
That GMP ignores long long (and most likely int_64) is indeed a problem. Normaliz uses the following functions
:

inline bool try_convert(long long& ret, const mpz_class& val) {
    if (val.fits_slong_p()) {
        ret = val.get_si();
        return true;
    }
    if (sizeof(long long) == sizeof(long)) {
        return false;
    }
    mpz_class quot;
    ret = mpz_fdiv_q_ui(quot.get_mpz_t(), val.get_mpz_t(), LONG_MAX);  // returns remainder
    if (!quot.fits_slong_p()) {
        return false;
    }
    ret += ((long long)quot.get_si()) * ((long long)LONG_MAX);
    return true;
}

inline bool try_convert(mpz_class& ret, const long long& val) {
    if (fits_long_range(val)) {
        ret = mpz_class(long(val));
    }
    else {
        ret = mpz_class(long(val % LONG_MAX)) + mpz_class(LONG_MAX) * mpz_class(long(val / LONG_MAX));
    }
    return true;
}
```

#11 - 15 Apr 2024 22:27 - John Abbott

- % Done changed from 10 to 50

I think we are close to a decision: not to use (unsigned) long long except perhaps internally.
I don't regard it as a bug that CoCoALib on Micro$oft platforms is "needlessly" limited by their choice of data model; in fact... >-}

#12 - 23 Apr 2024 22:05 - John Abbott

- Assignee set to John Abbott

- % Done changed from 50 to 80

While it might give slightly better performance to use (unsigned) long long in some internal chinese-remaindering functions, I would not expect the gain to be great (at most factor of 2). Right now I favour KISS.

#13 - 26 Apr 2024 21:09 - John Abbott

Nico Mexis sent the following by email (a few days ago):

```
That’s a good question. Indeed, long longs perform considerably worse (on average apparently ~ 2x worse accord
ing to this guy: https://stackoverflow.com/q/33848357/5894824).
```