Micro-GMP
(work in progress)

Paul Zimmermann

iRRAM/MPFR/MPC workshop, Trier, November 21-23, 2018

The GNU MP library

Created by Torbjörn Granlund.

First release GMP 1.3.2 in 1993.

GMP 2.0.2 in 1996, GMP 3.0 in 2000.

GMP 4.0 in 2001, GMP 5.0.0 in 2010.

GMP 6.0.0a in 2014, latest version is 6.1.2 (December 2016).

Now a standard. Contains several layers: `mpn` (internal functions), `mpz` (integers), `mpq` (rational numbers), `mpf` (floating-point numbers).

Mini-GMP

Written by Niels Möller (one of the GMP developers).

A small implementation of the basic GMP routines, in just two files: `mini-gmp.h` and `mini-gmp.c`.

Appeared in GMP 5.1.0 (2013).

Only the `mpn` and `mpz` layers.

Only schoolbook $O(n^2)$ algorithms.

Used to bootstrap GMP.

Using MPFR with Mini-GMP

With revision 13276:

```
$ ./configure --with-mini-gmp=/tmp/gmp-6.1.2/mini-gmp
$ make
$ make check
# PASS:  173
# SKIP:  10
# FAIL:  0
```

Tests skipped: `mpf_compat`, `mpfr_compat`, `tfprintf`, `tget_f`, `tget_q`, `tgmpop`, `tprintf`, `tset_f`, `tset_q`, `tsprintf`.

Limb

A *limb* is a word in the GMP language.

A GMP multiple-precision number is stored in n limbs:

$$a = a_0 + a_1\beta + \cdots + a_{n-1}\beta^{n-1}$$

In Mini-GMP, a limb is defined as `unsigned long`.

Generic multiple-precision bugs

A *generic* bug is a bug in a multiple-precision algorithm that does not depend on the limb size.

Example: Niels Möller recently found a generic bug in Algorithm SvobodaDivision from our book *Modern Computer Arithmetic*

How to find generic bugs?

1. Hoping to get lucky...

2. Generating inputs with GMP's `mpz_random2` function:

   void mpz_random2 (mpz_t ROP, mp_size_t MAX_SIZE)
   Generate a random integer of at most MAX_SIZE limbs, with long strings of zeros and ones in the binary representation. Useful for testing functions and algorithms, since this kind of random numbers have proven to be more likely to trigger corner-case bugs.

3. Forcing a small limb size.

Micro-GMP

A modified version of Mini-GMP that enables it to work with 32-bit, 16-bit, or even 8-bit limbs.

Motivations:

- exhaustive test of the MPFR low-level routines
- exhaustive test of cryptographic libraries?

How?

1. define `mp_limb_t` as `uint32_t`, `uint16_t` or `uint8_t`
2. fix the issues, until `make check` from both Mini-GMP and MPFR do pass

Difficulties

- GMP (and Mini-GMP) assume an `unsigned long` fits in a limb
- 27 functions from `mini-gmp.c` need to be modified
- even the test code of Mini-GMP had to be patched
- we also had to adapt MPFR to Micro-GMP
- need additional casts for 16- and 8-bit limbs (next slide)

```
$ cat e.c
#include <stdio.h>
#include <stdint.h>

int main()
{
  uint16_t x = 61270;

  printf ("~x = %u\n", ~x);
  printf ("x << 10 = %u\n", x << 10);
}

$ gcc e.c
$ ./a.out
~x = 4294906025
x << 10 = 62740480
```

Using Micro-GMP 16

```
$ cd /tmp
$ wget http://www.loria.fr/~zimmerma/16.tar.gz
$ tar xf 16.tar.gz

$ tar xf gmp-6.1.2.tar.bz2
$ cd gmp-6.1.2/mini-gmp
$ cp /tmp/16/mini-gmp.h .
$ cp /tmp/16/mini-gmp.c .
$ cd tests
$ patch -i /tmp/16/patch
patching file t-div.c
$ make check
```

One test (`t-div`) still fails, needs to investigate.

Using Micro-GMP 16 with MPFR

```
$ ./configure --with-mini-gmp=/tmp/16
...
checking for GMP_NUMB_BITS... 16 bits (from mini-gmp.h)
$ make
$ make check
...
[tversion] GMP_NUMB_BITS = 16, sizeof(mp_limb_t) = 2
```

Using Micro-GMP 8 with MPFR

```
$ cd /tmp
$ wget http://www.loria.fr/~zimmerma/8.tar.gz
$ tar xf 8.tar.gz

$ ./configure --with-mini-gmp=/tmp/8
...
checking for GMP_NUMB_BITS... 8 bits (from mini-gmp.h)
$ make
$ make check
...
[tversion] GMP_NUMB_BITS = 8, sizeof(mp_limb_t) = 1
```