The PARI-GNUMP library

Andreas Enge

LFANT project-team
INRIA Bordeaux–Sud-Ouest
andreas.enge@inria.fr
http://www.math.u-bordeaux.fr/~aenge

Atelier PARI-GP, Lyon
13 January 2017

Motivation

- Use C code for evaluating two-dimensional ϑ-functions from inside GP.

Motivation

- Use C code for evaluating two-dimensional ϑ-functions from inside GP.
- Use C code from the GNU multiprecision universe inside GP:
  - GMP
  - MPFR
  - MPC
  - MPFRCX, CM, CMH
  - FPLLL
  - ...

Motivation

- Use C code for evaluating two-dimensional ϑ-functions from inside GP.
- Use C code from the GNU multiprecision universe inside GP:
  - GMP
  - MPFR
  - MPC
  - MPFRCX, CM, CMH
  - FPLLL
  - ...
- Create a bridge between the GNU MP universe and PARI-GP.

The PARI-GNUMP library

1. Installation
2. Transformations of numbers
3. Higher level functions

Installation

https://pari-gnump.multiprecision.org/

Version 0.0.1 of 2014.

- Adapt the Makefile.

```
DIR=/usr/local
PARI=${DIR}/pari-dev
GMP=${DIR}/gmp-5.1.3
MPFR=${DIR}/mpfr-3.1.2
MPC=${DIR}/mpc-1.0.1
```

- make
- make check
- Copy libpari-gnump.so to your project directory.
- Use the available functions.

The PARI-GNUMP library

1 Installation

2 Transformations of numbers

3 Higher level functions

Memory management

- PARI: `t_INT`, `t_FRAC`, `t_REAL`, `t_COMPLEX`
  - stores numbers on the PARI stack
  - allocates sort of automatically:

    ```
    GEN c;
    c = gadd (a, b);
    ```

  - frees by moving the stack pointer (`avma`, `gerepile`)
- GMP, MPFR, MPC: `mpz_t`, `mpq_t`, `mpfr_t`, `mpc_t`
  - store numbers on the heap
  - require explicit allocation (`mpz_init`, `mpc_init2` → `malloc`)

    ```
    mpz_t c;
    mpz_init (c);
    mpz_add (c, a, b);
    ```

  - require explicit freeing (`mpz_clear`, `mpc_clear` → `free`)

    ```
    mpz_clear (c);
    ```

Precision

- PARI
    - has a global precision for the creation of variables
    - each variable implicitly has a given precision
    - works on a best-effort basis for rounding
- MPFR, MPC
    - assign a separate precision to each variable

      ```
      mpc_init2 (c, 200);
      ```
    - accept a rounding mode per operation and guarantee the result

      ```
      mpc_mul (c, a, b, MPC_RNDND);
      ```

Edianness

- Both store numbers as arrays of `unsigned long int`.
- `t_INT` and `mpz_t` have the same endianness.
- `t_REAL` has the other endianness.

Conversion functions provided by Karim Belabas

Conversion functions: pari-gnump.h

- From PARI to MP*
  - `void mpz_set_GEN (mpz_ptr z, GEN x);`
  - `void mpq_set_GEN (mpq_ptr q, GEN x);`
  - `int mpfr_set_GEN (mpfr_ptr f, GEN x, mpfr_rnd_t rnd);`
  - `int mpc_set_GEN (mpc_ptr c, GEN x, mpc_rnd_t rnd);`

  `x` of type `t_INT`, `t_FRAC`, `t_REAL`, `t_COMPLEX`, as suitable

  Semantics: consider `x` as exact, round and return inexact value

Conversion functions: `pari-gnump.h`

- From PARI to MP*
  - `void mpz_set_GEN (mpz_ptr z, GEN x);`
  - `void mpq_set_GEN (mpq_ptr q, GEN x);`
  - `int mpfr_set_GEN (mpfr_ptr f, GEN x, mpfr_rnd_t rnd);`
  - `int mpc_set_GEN (mpc_ptr c, GEN x, mpc_rnd_t rnd);`

  `x` of type `t_INT`, `t_FRAC`, `t_REAL`, `t_COMPLEX`, as suitable

  Semantics: consider `x` as exact, round and return inexact value

- From MP* to PARI
  - `GEN mpz_get_GEN (mpz_srcptr z);`
  - `GEN mpq_get_GEN (mpq_srcptr q);`
  - `GEN mpfr_get_GEN (mpfr_srcptr f);`
  - `GEN mpc_get_GEN (mpc_srcptr c);`

  Semantics: Create `t_REAL` or `t_COMPLEX` with the minimal precision to store `f` or `c` without loss

Use the PARI heap for MP*: pari-gnump.h

- Allocate mpfr and mpc numbers on the PARI heap; do not free!
  - `void pari_mpfr_init2 (mpfr_ptr f, mpfr_prec_t prec);`
  - `void pari_mpc_init2 (mpc_ptr c, mpfr_prec_t prec);`

  -
    ```
    void pari_mpc_init3 (mpc_ptr c, mpfr_prec_t prec_re,
                         mpfr_prec_t prec_im);
    ```
- Emulate PARI precision handling
  - `void pari_mpfr_init_set_GEN (mpfr_ptr f, GEN x, mpfr_prec_t default_prec);`
  - `void pari_mpc_init_set_GEN (mpc_ptr c, GEN x, mpfr_prec_t default_prec);`

  Allocate on the PARI heap.

  For t_REAL components, use their own precision.

  For t_INT and t_FRAC components, use default_prec.

The PARI-GNUMP library

1 Installation

2 Transformations of numbers

3 Higher level functions

Adding a C function

See examples in pari-gnump-user.h; compiled into the library.

- MPC
  - `GEN pari_mpc_mul (GEN x, GEN y, long prec);`
- MPFR
  - `GEN pari_mpfr_mul (GEN x, GEN y, long prec);`
  - `GEN pari_mpfr_erf (GEN x, long prec);`
  - `GEN pari_mpfr_zeta (GEN x, long prec);`
- CMH
  - `GEN pari_cmh_I2I4I6I10 (GEN tau, long prec);`
  - `GEN pari_cmh_4theta (GEN tau, long prec);`
  - `GEN pari_cmh_10theta2 (GEN tau, long prec);`

pari_mpfr_zeta

```
GEN pari_mpfr_zeta (GEN x, long prec)
{
   mpfr_prec_t p = bit_accuracy (prec);
   mpfr_t z, z1;

   pari_mpfr_init2 (z, p);
   pari_mpfr_init_set_GEN (z1, x, p);

   mpfr_zeta (z, z1, MPFR_RNDN);

   return mpfr_get_GEN (z);
}
```

pari_mpfr_zeta

```
GEN pari_mpfr_zeta (GEN x, long prec)
{
   mpfr_prec_t p = bit_accuracy (prec);
   mpfr_t z, z1;

   pari_mpfr_init2 (z, p);
   pari_mpfr_init_set_GEN (z1, x, p);

   mpfr_zeta (z, z1, MPFR_RNDN);

   return mpfr_get_GEN (z);
}
```

Caveat: Pollutes the PARI stack, needs `gerepile`!

pari_mpfr_zeta: Installation into GP

Use the Foreign Function Interface of GP, see examples.gp.

```
install ("pari_mpfr_zeta", "Gp",
         "mpfr_zeta", "./libpari-gnump.so");
```

- Takes our new function pari_mpfr_zeta;
- with one argument of type GEN, and the default precision;
- calls it mpfr_zeta inside GP;
- from the just compiled library libpari-gnump.so copied into the working directory ./

Plans for the future

- Add PARI stack management.

Plans for the future

- Add PARI stack management.
- Creating a new wrapper function is not that easy.
  - Use autotools to detect available libraries MP*.
  - Write wrappers for all/important functions from MP*.

    Your input needed!

    macro generated?
  - Activate those corresponding to available libraries.
  - Provide `pari_gnump.gp` include file for GP.
  - Provide `make install`; can GP find the library?

Plans for the future

- Add PARI stack management.
- Creating a new wrapper function is not that easy.
  - Use autotools to detect available libraries MP*.
  - Write wrappers for all/important functions from MP*. Your input needed! macro generated?
  - Activate those corresponding to available libraries.
  - Provide `pari_gnump.gp` include file for GP.
  - Provide `make install`; can GP find the library?
- Wrap CM for use in ECPP.
- Wrap FPLLL to test our LLL implementation.
- Wrap ARB for real and complex interval arithmetic (Fredrik Johansson).
- Wrap your favourite library.