PARI/GP and Perl: Past, Present, Future

Charles Boyd

Houston Perl Mongers

March 14, 2013

PARI/GP - A Gentle Introduction

What is PARI/GP?

PARI/GP is a fast and portable computer algebra system, primarily for use in algebraic number theory.

- `PARI` is a C library for fast computations.
- `GP` is a scripting language and interpreter for PARI functions.
- `gp` is an interactive shell that provides an interface to the PARI/GP system.

Examples of PARI/GP

- `factor(2302984)`
- `factor(`$x^6 - 4x^3 + 7x^2 - 9x + 3$`)`
- `primes(100)`
- `for(i=0,10,print(fibonacci(i))`
- `taylor(sin(x),x)`

Other uses of PARI/GP

- Fast linear algebra library for computations with vectors and matrices.
- General-purpose mathematical functions for summations, series, derivatives and integrals.
- Number Theoretic functions for special computations over the ring of integers $\mathbb{Z}$,the ring of univariate polynomials over integers $\mathbb{Z}[x]$, p-adic number fields, finite fields, more general number fields, Galois Theory, ...
- Computing elliptic curves and their properties, applications to cryptography.

Why use PARI/GP with Perl?

- To use PARI/GP for quick computation from other Perl programs.
- Create new interfaces for computer algebra.
- For PARI developers to test new features quickly and easily.
- For researchers to parse/verify/manipulate data before or after evaluation by PARI/GP.
- Cryptography? (See the CPAN module for `RSA` encryption)
- Because we can!

What about Math::Pari?

Note on Math::Pari

The CPAN module `Math::Pari` satisfies the use case of writing a GP program in Perl. This is achieved by overloading Perl's arithmetic operators, conversion between Perl and PARI data structures, and importing PARI functions (as barewords) to be used in a Perl script.

GPP - Introduction

Basic Goals

My goal is quite different from what `Math::Pari` achieves:

- No overloading of Perl's operators.
- Keep the `PARI` stack completely separate from Perl's stack.
- Clean and simple interface for communication between Perl programs and `PARI` library.
- No (implicit) conversion of data structures, strings are the universal language.
- Make it extremely simple to write a `gp` clone in Perl.

GPP - Overview

Design Principles

Our goals suggest the basic design concepts.

- Call `PARI` functions through a simple wrapper library that evaluates a string and returns a string.
- Strict separation between Perl and `PARI` stacks.
- Make the Perl interface as simple and lean as possible, any "heavy lifting" should be done in the `C` wrapper library or by patching `PARI/GP`.
- Don't reify `PARI/GP` - design should be general enough to easily support inclusion of other (sufficiently wrapped) mathematics libraries.
- Keep `PARI`-specific code under the `GPP::Pari` namespace.

GPP - Communicating with Pari

Important Structures and Functions

Pari uses the `long *GEN` structure as an internal representation of all mathematical objects. The following functions are used by `GPP::Pari` to communicate with `libpari`.

- `GEN gp_read_str(char *in)`
- `char *GENtostr(GEN z)`
- `long typ(GEN z)`
- `const char *type type_name(long n)`

Technical note

In `parisv.c`, every `GEN` type is declared as `volatile` so we can trap amd recover from errors with `longjmp(jmp_buf env, long errnum)`.

GPP - Wrapper Library

So, I heard you like wrappers...

The C program `parisv.c` implements simple wrapper functions to facilitate passing strings between `libpari` and another program.

- `char *evaluate(const char *in)` - Evaluates input and returns result.
- We also handle a `pari_stack` structure so output from `libpari` can be redirected to controlling process rather than to `STDOUT`.
- `char *parisv_type(const char *in)` - Returns the type of resulting Pari object.
- Implementation for `init()`, `version()`, `help()`, `quit()` functions.
- Uses `swig` to generate XS wrapper code for `GPP::Pari::Native` module.

GPP - Perl Library

- `GPP` - Processes user input, handles metacommands, sends everything else to be evaluated by `GPP::Pari` and pushes results to `GPP::Stack` object.
- `GPP::Pari` - Provides high-level interface to `libpari` functions via `GPP::Pari::Native`.
- `GPP::Pari::Native` - Wrapper module for `Native.so` functions, generated by Swig.
- `GPP::Stack` - Really just an array of hashes, each element has a key for input, output and result type.
- `Native.so` - Shared library linked against `libpari.so` that contains symbols from `parisv.c` along with generated XS wrappers from `Native_wrap.c`.

primefactors.pl

Using GPP in a script

The `examples/primefactors.pl` script demonstrates using `GPP` to compute prime factors of 100 randomly generated integers $0 \leq n \leq 1000$. Runs in less than 1 second.

gpp

Using `GPP` to write an application

The `bin/gpp` script demonstrates using `GPP` to write an application that provides an interactive shell to `libpari`.

It emulates about 90% the functionality of the `gp` binary distributed with Pari/GP. Uses `Term::Readline::Zoid` for readline functionality (command history and emacs-like keybindings).

Ideas

- Graphical application using `XUL::Gui`. (Proof of concept has been done, but very incomplete/buggy)
- Web application with a live script editor, pretty printing with using `libpari` function `GENtoTeXstr()` to generate LaTeX output and MathML to render LaTeX in the browser.
- Ability to convert Pari data structures into "natural" Perl objects.
- Extend the `GP` language with pure-perl features.
- Possibly create similar bindings to other mathematics libraries.

Problems and Open Questions

- Build environment makes too many assumptions, not very robust.
- It would be nice to turn Pari `t_VEC, t_MAT` structures into Perl arrays (of arrays (of arrays...)) but turns out to be a tricky problem to solve in full generality.
- Export `libpari` constants and functions in a reasonable way.
- Need to check version of `libpari` on system before compiling wrapper library, this is also not particularly easy to do in a portable (across all unix variants) manner.

End

- GPP source code and wiki: `github.com/FreeMonad/GPP`
- E-mail: `charles.boyd@freemonad.org`

Thanks!

That's it. Thank you for listening.