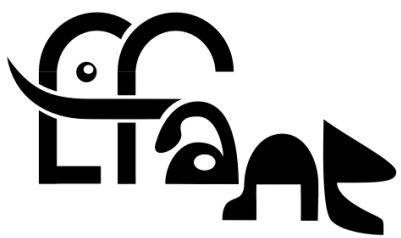
PARI and POSIX threads.

Pierre Lezowski
INRIA Bordeaux, team Lfant

Atelier PARI/GP,
January 8, 2014.



General aim

- You have a (working) C program using PARI library.
- Some functions can be run independently.
- You want to run these parts in parallel to use the processors of your machine.
- It is especially useful on a single machine with multiple CPUs.

To compute the Euclidean minimum of a point

In the number field K with ring of integers $\mathbf{Z}_K$, I want to compute the Euclidean miminum of some point $x \in K$,

$$m_K(x) = \inf_{y \in \mathbf{Z}_K} \left| N_{K/\mathbf{Q}}(x - y) \right|.$$

To this extent, we have some a bound on the y's, which may achieve this miminum.
In practice, we use an embedding of K into $\mathbf{R}^n$, where $n = [K : \mathbf{Q}]$. We identify y with some $(y_1, y_2, \ldots, y_n) \in \mathbf{Z}^n$. If $y_1, y_2, \ldots, y_r$ are fixed, then we have a bound on y_{r+1}.

To compute the Euclidean minimum of a point (2)

Summary

- The coordinate y_1 is bounded, let us assume that only $y_1 = 0$ and $y_1 = 1$ are possible.
- y_1 being fixed, there are only finitely many possibilities for $y_2, \ldots, y_n$, an efficient bound on these depend also on y_1.

Functions without PTHREAD

- `GEN compute_minimum_x_1(GEN y1)`
- `GEN compute_minimum_x(void)`
  which consists just in computing `GEN compute_minimum_x_1` in 0 and 1.

To compute the Euclidean minimum of a point (2)

```
GEN compute_minimum_x(void){
   GEN m,m0, m1;
   m0 = compute_minimum_x_1(gen_0);
   m1 = compute_minimum_x_1(gen_1);
   if( gcmp(m0,m1) < 0)
     m = m0;
   else
     m = m1;
   return m;
}
```

General ideas

You put the execution in a thread, you decide when to start and stop threads.

The following functions are available.

- `pthread_create()`
- `pthread_join()`

Generally, the memory is shared, but each thread has its own PARI stack.

Requirements

You need `Configure -enable-tls`. For each thread, there will be a pari stack.

Add `#include <pthread.h>` to the preamble of your C file.

Compilation

```
cc thread.c -o thread.o -lpari -lpthread
```

General functions

- For the parent thread: `pari_thread_alloc()` and `pari_thread_free()`.
- For the child thread: `pari_thread_start()` and `pari_thread_close()`.

Example

We write a new function to compute with the first coordinate fixed, which also starts a new thread for it.

```
void* my_compute_1(void* arg){
   GEN F,y;
   y = pari_thread_start((struct pari_thread*) arg);
   F= compute_minimum_x_1(y);
   pari_thread_close();
   return (void*)F;
}
```

```
GEN compute_minimum_x(void){
   pthread_t th0, th1;
   struct pari_thread pth0, pth1;
   GEN m,m0, m1;
   pari_thread_alloc(&pth0, 4000000, gen_0);
   pari_thread_alloc(&pth1, 4000000, gen_1);
   pthread_create(th0,NULL, &my_compute_1, (void*)&pth0);
   pthread_create(th1,NULL, &my_compute_1, (void*)&pth1);
   pthread_join(th0,(void*)&m0);
   pthread_join(th1,(void*)&m1);
   if( gcmp(m0,m1) < 0)
     m = gcopy(m0);
   else
     m = gcopy(m1);
   pari_thread_free(&pth0);
   pari_thread_free(&pth1);
   return m;
}
```

- The function passed in parameter admits itself one GEN parameter.
- A call to `pari_thread_free()` deletes the PARI stack of the thread, so the useful objects should be copied before.
- See Appendix B in the documentation.
- A readily testable example is available: `examples/thread.c`.

Advertisement

I used it in my program `euclid` to compute the Euclidean minimum $M(K) = \sup_{x \in K} m_K(x)$ of a number field K.

The source is available at
`http://www.math.u-bordeaux1.fr/~plezowsk/euclid/`.

Some tables of Euclidean minima, and therefore of (non) norm-Euclidean number fields are available.