The future of parallel GP

B. Allombert

IMB
CNRS/Université Bordeaux 1

15/01/2015

Summary from last year

Resources

The GP interface

Avoiding global variables

Using parfor/parforprime

Since the last year

Introduction

PARI now supports two common multi-threading technologies :

- POSIX thread : run on a single machine, lightweight, flexible, fragile.
- Message passing interface (MPI) : run on as many machine as you want, robust, rigid, heavyweight. Used by most clusters.

- To use POSIX threads, use `./Configure -mt=pthread`
- To use MPI, add `./Configure -mt=mpi`

However the parallel GP interface does not depend on the multithread interface : a properly written GP program will work identically with both. In this tutorial we will focus on POSIX threads.

Resources

The number of secondary threads to use is controlled by `default(nbthreads)`. The default value of nbthreads is the number of CPU threads (i.e. the number of CPU cores multiplied by the hyperthreading factor). The default can be freely modified.
The PARI stack size in secondary threads is controlled by `default(threadsize)`, so the total memory allocated is equal to $\texttt{parisize} + \texttt{nbthreads} \times \texttt{threadsize}$. By default, $\texttt{threadsize} = \texttt{parisize}$.

`default(nbthreads)`

The libpari interface

PARI provides an C interface for parallelizing PARI programs, with the function `mt_queue_start`, `mt_queue_submit`, `mt_queue_get`, `mt_queue_end`, see `examples/pari-mt.c`.

The GP interface

GP provides functions that allows parallel execution of GP code, subject to the following limitations : the parallel code

- must not access global variables or local variables declared with `local()` (but `my()` is OK),
- must be free of side effect.

The parallel functions are `parapply`, `parselect`, `parfor`, `parforprime`, `parsum`, `parvector`, `pareval`. The functions `inline` and `self` can be used to work around the limitation.

Simple examples

```
ismersenne(x)=ispseudoprime(2^x-1);
default(timer,1);
apply(ismersenne,primes(400))
parapply(ismersenne,primes(400))
select(ismersenne,primes(400),1)
parselect(ismersenne,primes(400),1)
```

Avoiding global variables

```
V=primes(400);
parvector(#V,i,ispseudoprime(2^V[i]-1))
  *** parvector: mt: global variable not
  *** supported: V.
break
fun(V)=parvector(#V,i,ispseudoprime(2^V[i]-1));
fun(V)
my(V=V);parvector(#V,i,ispseudoprime(2^V[i]-1))
```

Avoiding global functions

```
ismersenne(x)=ispseudoprime(2^x-1);
fun(V)=parvector(#V,i,ismersenne(V[i]));
fun(primes(400))
  *** parvector: mt: global variable not
  *** supported: ismersenne.
```

The simplest way to avoid that is to compile ismersenne with GP2C.

Using inline

```
inline(ismersenne);
ismersenne(x)=ispseudoprime(2^x-1);
fun(V)=parvector(#V,i,ismersenne(V[i]));
fun(primes(400))
```

Using parfor/parforprime

```
inline(ismersenne);
ismersenne(x)=ispseudoprime(2^x-1);
parforprime(p=1,999,ismersenne(p),c,if(c,print(p)))
prodmersenne(N)=
{ my(R=1);
  parforprime(p=1,N,
    ismersenne(p),
    c,
    if(c, R*=p));
  R;
}
prodmersenne(1000)
```

```
inline(ismersenne);
ismersenne(x)=ispseudoprime(2^x-1);
findmersenne(a)=
  parforprime(p=a,,ismersenne(p),c,if(c,return(p)))
findmersenne(4000)
findmersenne(8)
findmersenne(8)
```

```
inline(ismersenne);
ismersenne(x)=ispseudoprime(2^x-1);
parfirst(fun,V)=
  parfor(i=1,#V,fun(V[i]),j,if(j,return([i,j])));
parfirst(ismersenne,[4001..5000])
```

GP2C

GP2C is able to compile parallel code (with some limitation for `parfor` and `parforprime`). Since GP2C transforms GP functions to C functions, it does not store functions in global variables, which is the major usability issue with the GP interface. Thus in practice it is easier to use the parallelism with GP2C.

Large scale use

We experimented with large scale use of parallel GP on clusters using MPI.It performed correctly. We were able to compute the modular polynomial of degree 3001 in 3 hours on 96 cores.

The future

- Increasing portability.
- Improving the MPI interface.
- Improving the GP interface.
- Adding parallel algorithms to GP.