parallel GP with GP2C

B. Allombert

IMB
CNRS/Université Bordeaux 1

12/01/2015

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N° 676541

POSIX thread support

If you followed yesterday tutorial, GP is build with POSIX thread support. To check it, launch GP

```
./GP/bin/gp
```

and check for

```
threading engine: pthread
```

It is possible to test parallel support with

```
time make test-parallel
```

Number of threads

The number of secondary threads to use is controlled by `default(nbthreads)`. The default value of nbthreads when using POSIX thread is the number of CPU threads (i.e. the number of CPU cores multiplied by the hyperthreading factor). It can be freely modified.

```
default(nbthreads)
```

threadsize

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, $\texttt{threadsize} = \texttt{parisize}$.
It is possible to use `threadsizemax` to allow the size of each thread stacks to grow dynamically up to `threadsizemax`.

```
default(threadsizemax,"1G")
```

polmodular

The GP function `polmodular` takes advantage of parallelism.

```
? my(t=getwalltime());polmodular(101);getwalltime()
%4 = 2603
? ##
  ***   last result computed in 9,369 ms.
```

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`, `parforvec`, `parforprime`, `parsum`, `parvector`, `pareval`.

Simple examples

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

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.

Partial GP2C compilation

Sometimes, getting a whole GP script compile and work with GP2C can take time. Using partial GP2C compilation can be simpler. Create a file ismersenne.gp with

```
ismersenne(x)=ispseudoprime(2^x-1);
```

then compile it with

```
GP=true GP/bin/gp2c-run ismersenne.gp
```

this creates files :

```
ls ismersenne.gp*
ismersenne.gp  ismersenne.gp.c  ismersenne.gp.o
ismersenne.gp.run  ismersenne.gp.so
```

Partial GP2C compilation

Now you can do :

```
\r ismersenne.gp.run
fun(V)=parvector(#V,i,ismersenne(V[i]));
fun(primes(400))
```

Grouping small tasks

Create a file thuemorse.gp with

```
thuemorse(n)= my(V=binary(n)); (-1)^sum(i=1,#V,V[i])
```

and compile it with GP2C.

```
GP=true GP/bin/gp2c-run thuemorse.gp
```

```
\r thuemorse.gp.run
ti(f)=my(t=getwalltime());f();getwalltime()-t
default(timer,1);
ti(()->sum(n=1,2*10^6, thuemorse(n)/n*1.))
ti(()->parsum(n=1,2*10^6, thuemorse(n)/n*1.))
ti(()->parsum(N=1,200, \
  sum(n=1+(N-1)*10^4, N*10^4, thuemorse(n)/n*1.)))
```

Using parfor/parforprime/parforvec

```
\r ismersenne.gp.run
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)
```

```
\r ismersenne.gp.run
findmersenne(a)=
  parforprime(p=a,,ismersenne(p),c,if(c,return(p)))
findmersenne(4000)
findmersenne(8)
findmersenne(8)
```

```
\r ismersenne.gp.run
parfirst(fun,V)=
  parfor(i=1,#V,fun(V[i]),j,if(j,return([i,V[i]])))
parfirst(ismersenne,[4001..5000])
```

Large scale use

We added support parallelism in polmodular. 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 more parallel algorithms to GP.