# Parallel GP interface tutorial

B. Allombert

IMB
CNRS/Université Bordeaux 1

15/01/2019

# 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.

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.

# POSIX threads

- To use POSIX threads, add `--mt=pthread`.
- `gp-sta` is generally 20% faster than gp-dyn.

```
./Configure --mt=pthread
make test-parallel
make test-rnfkummer
```

Compare `gp-sta` against `gp-dyn`.

```
ln -s Olinux-x86_64/gp-sta .
./gp-sta
```

Check for "threading engine : pthread"

# 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)
```

# Parallel algorithms

A number of PARI functions will use parallelism when available :

```
? default(timer,1);
? isprime(2^600+187)
cpu time = 1,244 ms, real time = 197 ms.
%2 = 1
? nbthreads = default(nbthreads);
? default(nbthreads,1)
? isprime(2^600+187)
time = 660 ms.
%5 = 1
? default(nbthreads,nbthreads);
```

We see that the parallel version is three time faster than the serial one. When using pthread, the CPU time is the sum of the time used by all threads. The realtime is smaller than the CUP time due to use of parallelism.

# Simple examples

```
? ismersenne(x)=ispseudoprime(2^x-1);
? apply(ismersenne,primes(400))
cpu time = 1,248 ms, real time = 1,247 ms.
%7 = [1,1,1,1,0,1,1,1,0,0,1,0,0,0,0,0,0,1,0,...
? parapply(ismersenne,primes(400))
cpu time = 2,253 ms, real time = 298 ms.
%8 = [1,1,1,1,0,1,1,1,0,0,1,0,0,0,0,0,0,1,0,...
? select(ismersenne,primes(400))
cpu time = 1,192 ms, real time = 1,199 ms.
%9 = [2,3,5,7,13,17,19,31,61,89,107,127,521,607,127
? parselect(ismersenne,primes(400))
cpu time = 2,248 ms, real time = 299 ms.
%10 = [2,3,5,7,13,17,19,31,61,89,107,127,521,607,12
```

Compare the real time.

# Concept

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

- must be free of side effect.
- cannot access global variables or local variables declared with `local()` (but `my()` is OK),
- instead access variables exported to the parallel world with `export`.

# The parallel world

`export` is used to set values in the parallel world.

```
? ismersenne(x)=ispseudoprime(2^x-1);
? fun(V)=parvector(#V,i,ismersenne(V[i]));
? fun(primes(400))
  *** parvector: mt: please use export(ismersenne)
> break
? export(ismersenne)
? fun(primes(400))
```

# Silly example

```
? export(f=25);
? f
%2 = f
? parsum(i=1,1,f)
%3 = 25
```

# exportall

`exportall` exports all current global variables.

```
? V=primes(400);
? parvector(#V,i,ispseudoprime(2^V[i]-1))
  *** parvector: mt: please use export(V).
> break
? exportall()
? parvector(#V,i,ispseudoprime(2^V[i]-1))
```

# Using polynomial indeterminates

```
? fun(n)=bnfinit(x^n-2).no;
? parapply(fun,[1..30])
  *** parapply: mt: please use export(x).
> break
? fun(n)=bnfinit('x^n-2).no;
? default(timer,1);
? default(parisize,"16M");
? apply(fun,[1..30])
cpu time = 5,284 ms, real time = 5,281 ms.
%26 = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
?  parapply(fun,[1..30])
cpu time = 9,084 ms, real time = 2,045 ms.
%27 = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
?  parapply(fun,-[-30..-1])
cpu time = 9,281 ms, real time = 1,905 ms.
```

# Grouping small tasks

```
? thuemorse(n)=
 my(V=binary(n)); (-1)^sum(i=1,#V,V[i]);
? export(thuemorse);
? default(timer,1);
? sum(n=1,2*10^6, thuemorse(n)/n*1.)
cpu time = 4,861 ms, real time = 4,862 ms.
%33 = -1.1962837643252564372222916332008191772
? parsum(n=1,2*10^6, thuemorse(n)/n*1.)
cpu time = 17,245 ms, real time = 4,426 ms.
%34 = -1.1962837643252564372222916332008191772
? parsum(N=1,200, \
  sum(n=1+(N-1)*10^4, N*10^4, thuemorse(n)/n*1.))
cpu time = 11,836 ms, real time = 1,526 ms.
%35 = -1.1962837643252564372222916332008191811
```

# Using parfor

```
? ismersenne(x)=ispseudoprime(2^x-1);
? export(ismersenne)
? parfor(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)
cpu time = 108 ms, real time = 31 ms.
%13 = 637764906056784026430
```

# Using parforprime

```
? 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)
%15 = 637764906056784026430
```

# parforvec

```
? parforvec(v=[[1,3],[1,3]],factorback(v) \
            ,f,print(v,":",f))
```

# parploth

```
? parploth(x=-4.5,6,intnum(t=0,x,if(t,1/gamma(t))))
```

# return

```
? ismersenne(x)=ispseudoprime(2^x-1);
? export(ismersenne)
? findmersenne(a)=
  parforprime(p=a,,ismersenne(p),c,if(c,return(p)))
? findmersenne(4000)
cpu time = 2,600 ms, real time = 366 ms.
%17 = 4253
? findmersenne(8)
cpu time = 4 ms, real time = 1 ms.
%18 = 13
? findmersenne(8)
%19 = 13
```

# return

```
? parfirst(fun,V)=
  parfor(i=1,#V,fun(V[i]),j,if(j,return([i,j])));
? parfirst(ismersenne,[4001..5000])
cpu time = 3,104 ms, real time = 442 ms.
%21 = [253,1]
```