Maple

+ Computer Grid

Solutions to Hard Problems

Dave Hare, Maplesoft, March 2012

▼ The setup

Maple has a *Grid Computing Toolbox*. This allows you to send the work required to solve a problem out to as many computers as you can link together.

To illustrate how this is done, we'll look at the problem of computing a high-precision approximation to a mathematical constant. Specifically, we will look at how to compute 125,000 digits of the *Landau-Ramanujan constant.*

▼ The problem

Let $N(x)$ be the number of positive integers less than or equal to x which can be written as the sum of exactly 2 squares. Then $K = \lim_{x \to \infty} N(x) \frac{\sqrt{\ln(x)}}{x}$. The convergence is extremely slow (at $x = 10^{10}$ get 1 digit).

Landau (1908) proved $K = \frac{1}{\sqrt{2}} \prod_{\substack{p \in P \\ p \equiv 3 \pmod 4}} \left(1 - \frac{1}{p^2}\right)^{-\frac{1}{2}}$, where P is the set of primes. Better, but still very slow: using primes < 100000 gives 6 digits.

Ramanujan (1910): K ≈.764.

▼ Better formula

Flajolet & Vardi (1996):

$$K = \frac{1}{\sqrt{2}} \prod_{k=1}^{\infty} \left(\left(1 - \frac{1}{2^{2^k}}\right) \frac{\zeta(2^k)}{\beta(2^k)} \right)^{\frac{1}{2^{k+1}}}$$

$\zeta(x)$ is the *Riemann zeta function* and $\beta(x)$ is the *Dirichlet beta function.*

$$\beta(s) = \frac{1}{4^s} \left(\zeta\left(s, \frac{1}{4}\right) - \zeta\left(s, \frac{3}{4}\right) \right)$$

$$\zeta(n) = \frac{1}{2} (2\,\pi)^n \frac{|B_n|}{n!} \text{ for even n}$$

B_n is the n^{th} *Bernoulli number*.

$\zeta(s, q)$ is the *Hurwitz (or generalized) zeta function*:

$$\zeta(s, q) = \sum_{n=0}^{\infty} \frac{1}{(n+q)^s} \quad , \qquad \zeta(s) = \zeta(s, 1)$$

The Flajolet-Vardi form is extremely rapidly converging: 9 terms give nearly 500 digits, 10 terms nearly 1000 digits, 11 terms nearly 2000 digits, and so on.

▼ Cost of computing it

The bulk of the work is the Hurwitz zeta function computations. These are done using the Euler-MacLaurin summation formula:

$$\zeta(s, q) = \sum_{n=0}^{N} \frac{1}{(n+q)^s} + \frac{(N+q)^{1-s}}{s-1} + \frac{(N+q)^{-s}}{} + \sum_{m=1}^{M} \frac{B_{2m}}{(2m)!} s\,(s+1)\,\ldots\,(s+2m-2)\,(N+q)^{-s-2m+1} + error\_term(M)$$

▼ Bernoulli numbers

$$B_n = [n = 0] - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k}{n - k + 1}$$

Note $B_n = 0$ for odd $n > 1$. B_n, $n > 0$ even, is rational (with typically a small denominator), not integer.

Really only usable up to about B_{10000}. We'll need a lot more.

Algorithm of Fee & Plouffe (2007)

Turn around the formula for B_n, n even, from above:

$$\zeta(n) = \frac{1}{2}(2\pi)^n \frac{|B_n|}{n!} \quad \Rightarrow \quad |B_n| = \frac{2\,\zeta(n)\,n!}{(2\pi)^n}$$

Progress? Yes, because

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} = \prod_{p \in P} \frac{1}{1 - p^{-s}}$$

Converges slowly for the exact value of $\zeta(s)$, but in the formula for $|B_n|$ it converges very quickly to the integer part of $|B_n|$. The fractional part is also quickly computable for even n:

$$\operatorname{frac}(|B_n|) = \sum_{\substack{p \in P \\ (p-1)|n}} \frac{1}{p}$$

Advantages:
- Can get B_n for any n without computing any previous B_k.
- Algorithm to compute sequences of Bernoulli numbers is fully parallelizable.

▼ Putting it all together

So, we need to compute

$$K \approx \frac{1}{\sqrt{2}} \prod_{k=1}^{N} \left(\left(1 - \frac{1}{2^{2^k}}\right) \frac{\zeta(2^k)}{\beta(2^k)} \right)^{\frac{1}{2^{k+1}}}$$

$$= \frac{1}{\sqrt{2}} \prod_{k=1}^{N} \left(\left(1 - \frac{1}{2^{2^k}}\right) \frac{4^{2^k} (2\pi)^{2^k} \cdot \frac{\left|B_{2^k}\right|}{(2^k)!}}{2\left(\zeta\left(2^k, \frac{1}{4}\right) - \zeta\left(2^k, \frac{3}{4}\right)\right)} \right)^{\frac{1}{2^{k+1}}}$$

For each term in the product we need 2 Hurwitz zeta computations, one extra Bernoulli number and a factorial. All the cost is in the Hurwitz zeta computations. Computations are independent.

The Hurwitz zeta computations will reuse many of the same Bernoulli numbers, so it makes sense to precompute and store these.

▼ Using the Grid

We have a 24-node system running Microsoft's HPC o/s.

Using this, I precomputed 57,000 non-zero Bernoulli numbers in about 2.5 days. Stored in *.m* format, this requires ~ 6GB. ($B_{2\cdot 56999}$ has a numerator of 435981 digits and its denominator is 6.)

We pass a script file containing Maple commands to the nodes (each node gets exactly the same script). Nodes can know who they are, which can control the flow through the script.

Nodes communicate by sending messages using Grid toolbox commands. Grid commands in Maple (used in this work):

Setup(...)	Setup
MyNode()	Who am I?
NumNodes()	Number of nodes available
Send(node, msg)	Send *msg* to *node*
Receive(src)	Receive a message from *src* (omit to receive from anywhere)

For maximum computation efficiency, the precomputed Bernoulli numbers were stored multiple times, so that each node has access to the Bernoulli numbers it needs from a local disk.

▼ **Replacing Maple's** bernoulli **routine**

```
# replace bernoulli routine with one which uses stored values
 bernoulli := proc( m )
   local idx, n;
   global BTbl;

   if m = 1 then
      return -1/2;
   end if;

   n := m/2;

   try
      # values are stored in .m format; must translate
      return op( sscanf( BTbl[ n ], "%m" ) );

   catch:
      try
         # get more bernoulli numbers
         idx := 1000 * iquo( n, 1000 );

         # allow previous values to be collected
         BAry || idx := NULL;

         idx := idx + 1000;
         read cat( DIR, "bernoulli/BAry_", idx, ".m" );
         BTbl := BAry || idx;

      catch:
         # no more bernoulli numbers available
         error "bernoulli number %1 not available", m;
      end try;
   end try;

   # values are stored in .m format; must translate
   return op( sscanf( BTbl[ n ], "%m" ) );
 end proc: # bernoulli
```

Initialization

```
Grid:-Setup( "mpi", 'mpidll' = "msmpi", 'host'= "host name" );

ThisNode := Grid:-MyNode( ) :

Digits := 128000 :
nTerms := 18 :

DIR := "working directory" :
```

Master node code

```
if ThisNode = 0 then

   # Longest tasks are those near the middle of the index range, so send those out
   first
  Tasks := proc(m)
    local n, t, j;

    n := ceil(m/2);
    t := [n, 1], [n, 2];

    for j to n do
      if n + j ≤ m then
        t := t, [n + j, 1], [n + j, 2];
      end if;

      if n - j > 0 then
        t := t, [n - j, 1], [n - j, 2];
      end if;
    end do;

    return [ t ];
  end( nTerms ) :

  Res := Array( 1 ..nTerms, 1 ..2 );

  # "-1" because node 0 is the job manager and does not participate in
  # the term computations
  waiting := Grid:-NumNodes( ) - 1 :

  while waiting > 0 do
    # receive message
    msg := Grid:-Receive( );

    if msg[ 1 ] = 1 then
      # msg is request for task: [ 1, node ]
      if Tasks = [ ] then
```

```
            # no more tasks - signal to finish
            Grid:-Send( msg[ 2 ], [ -1 ] );

        else
            # send next task
            nexttask := Tasks[ 1 ];
            Tasks := Tasks[ 2 .. -1 ];
            Grid:-Send( msg[ 2 ], nexttask );
         end if;

    elif msg[ 1 ] = 2 then
        # msg is confirmation of node termination: [ 2, node ]
        waiting := waiting - 1;

    else # msg[ 1 ] = 3 then
        # msg is report of result: [ 3, node, k, term_code, value ], where k is the
        #   infinite product index
        t := [ msg[ 3 ], msg[ 4 ] ];

        # Save result
        Res[ msg[ 3 ], msg[ 4 ] ] := msg[ 5 ];
    end if;
end do:

# compute the final result
# columns of Res are [ Zeta(0, 2^k, .25), Zeta(0, 2^k, .75) ]
p2 := evalf(2 * Pi);
LR := 1.0;

for k to nTerms do
    k2 := 2^k;
    s := bernoulli( k2 );
    LR := LR * ( (1.-2.^( -k2) ) * .5 * p2^k2 * abs( s ) /
            GAMMA(k2 + 1.) * 4.^k2 /
            (Res[ k, 1 ] - Res[ k, 2 ]) ) ^ ( .5^(k + 1) );
end do;

LR := 2.^( -1/2) * LR;
save LR, cat( DIR, "LR_res.m" )
```

Worker node code

```
else
  do
    # request a task
    Grid:-Send( 0, [ 1, ThisNode ] );

    task := Grid:-Receive( 0 );

    if task = [ -1 ] then
      # no more tasks, so quit
      Grid:-Send( 0, [ 2, ThisNode ] );
      break;
    end if;

    try
      # compute the requested term
      # task = [ k, j ]
      k2 := 2.^task[ 1 ];

      if task[ 2 ] = 1 then
        r := Zeta( 0, k2, .25 );

      else
        r := Zeta( 0, k2, .75 );
      end if;

    catch "not enough" :
      r := -1.0;

    catch:
      printf( StringTools:-FormatMessage[ 2 .. -1 ] );
    end try;

    # send result
    Grid:-Send( 0, [ 3, ThisNode, op( task ), r ] );
  end do;
end if:
```

▼ Results from the Grid

Digits	# Terms	Accuracy of result	Max. (non-0) Bernoulli number	Time
1000	10	979	1200	1.0 sec
2000	11	1957	2272	1.8 sec
4000	12	3911	4262	8.5 sec
8000	13	7820	< 8000 (8192)	46 sec
16000	14	15637	< 16000 (16384)	4.5 min
32000	15	31272	< 28000 (32768)	33 min
64000	16	62541	< 54000 (65536)	3.4 hr
128000	18	125079	< 108000 (131072, 262144)	15.3 hr

Comptutation records

The fifth entry is an uncredited value which was found on the On-Line Encyclopedia of Integer Sequences, where the current record of 125,079 digits is now stored.

Ramanujan	3	~1910
Flajolet	1024	1996
Hare	10,000	1996
Sebah	30,010	2002
?	65,536	?
Hare	125, 079	2010

▼ Last things

Using Maple on a computer grid is very straightforward. It is likely more of a challenge to set up the grid in the first place.

Other than grid initialization, exactly the same code can be run on a "local grid", that is, a single computer with multiple CPU's. Once fully debugged, the code can then easily be deployed to a distributed grid.