The bit-burst algorithm

Paul Zimmermann, INRIA LORRAINE, Loria, Nancy

July 21, 2006

Brent's Algorithm for exp

The Complexity of Multiple-Precision Arithmetic, in *The Complexity of Computational Problem Solving*, edited by R. S. Anderssen and R. P. Brent, Univ. of Queensland Press, 1976.

Theorem 6.2 *If* $M(n)$ *satisfies* $2M(n) \leq M(2n)$ *then*

$$t_n(\exp) \leq c_{32} M(n) \log^2(n)$$

Idea of Brent's Algorithm

Write $x = x_1 + x_2 + \cdots + x_j + \cdots + x_k$, where $x_j = \frac{r_j}{2^{2^j}}$, with r_j integer with at most 2^{j-1} bits.

Example:

$$x = 0.0\underbrace{1}_{r_1}\underbrace{10}_{r_2}\underbrace{1101}_{r_3}101\ldots$$

Evaluate each $\exp(x_j)$ using binary splitting.

We can stop when $x_j^k/k! < 2^{-n}$, i.e. when $2^n < k!2^{k2^{j-1}}$.

Thus the last term of the Taylor series has size $O(n)$, and the cost of each binary splitting tree is $O(M(n)\log n)$.

The total cost is thus $O(M(n)\log^2 n)$.

Brent's Lost Algorithm for sin

Theorem 6.2 continues with:

Theorem 6.2 *If $M(n)$ satisfies $2M(n) \leq M(2n)$ then*

$$t_n(\exp) \leq c_{32} M(n) \log^2(n) \tag{6.27}$$

and

$$t_n(\sin) \leq c_{33} M(n) \log^2(n) \qquad (6.28).$$

and the proof ends with:

[. . .] This establishes (6.27), and the proof of (6.28) is similar.

Brent's Lost Algorithm for sin

Write $x = x_1 + x_2 + \cdots + x_k$ as for exp.

$$\sin(x_j + r) = \sin x_j \cos r + \cos x_j \sin r$$

$$\cos(x_j + r) = \cos x_j \cos r - \sin x_j \sin r$$

1. Get $\sin x_j, \cos x_j$ by binary splitting: $O(M(n) \log n)$.

2. Compute $\sin r$ and $\cos r$ recursively.

3. Reconstruct $\sin(x_j + r)$ and $\cos(x_j + r)$ with the above formulae.

Total cost $O(M(n) \log^2 n)$.

Holonomic (D-finite) functions

Definition: f is *holonomic* iff it satisfies a linear differential equation with polynomial coefficients in x:

$$a_k(x)f^{(k)}(x) + \cdots + a_1(x)f'(x) + a_0(x)f(x) = b(x)$$

Running example: $f = \arctan$

$$(1 + x^2)f'(x) = 1$$

Holonomic functions are closed under sum, product, Hadamard product, right composition with an algebraic function, and algebraic functions are holonomic.

Their Taylor coefficients (a_n) satisfy a linear recurrence with polynomial coefficients in n:

$$na_n + (n-2)a_{n-2} = 0, a_0 = 1, a_1 = 1$$

Computations on holonomic functions can be performed with the Maple `gfun` package developed with B. Salvy:

```
> with(gfun):
> deq := (1+x^2)*diff(f(x), x) - 1:
> diffeqtorec({deq, f(0)=0}, f(x), a(n));
    {a(0) = 0, n a(n) + (n + 2) a(n + 2), a(1) = 1}
> rectodiffeq(%, a(n), f(x));
                  2  /d        \
         {(1 + x ) |-- f(x)| - 1, f(0) = 0}
                   \dx       /
```

The problem

Input: a holonomic function f, given by its differential equation, and a n-bit floating-point number $x \in [0, 1/2]$.

Output: n-bit approximation of $f(x)$.

D. V. Chudnovsky and G. V. Chudnovsky, *Computer Algebra in the Service of Mathematical Physics and Number Theory*, Computers in Mathematics (Stanford, CA, 1986), Lecture Notes in Pure and Applied Mathematics, 1990.

Joris van der Hoeven, *Fast evaluation of holonomic functions*, Theoretical Computer Science, 2000.

Sums of holonomic series

If (a_n) is holonomic, so is $(\sum a_n)$.

Proof: let $b_n = \sum_{k=0}^{n} a_k$.

Substitute a_n by $b_n - b_{n-1}$ in the recurrence for a_n: we get a recurrence of order one more for b_n.

Alternate Proof: if $f(x) = \sum a_n x^n$, then $g(x) = \sum b_n x^n$ is $f(x) \cdot \frac{1}{1-x}$.

Example: the coefficients of $\arctan x$ satisfy:

$$na_n + (n-2)a_{n-2} = 0$$

Those of $b_n := \sum_{k=0}^{n} a_k$ satisfy:

$$nb_n - nb_{n-1} + (n-2)b_{n-2} + (2-n)b_{n-3} = 0$$

More generally, if p/q is a rational, consider:

$$b_n = \sum_{k=0}^{n} a_k (p/q)^k$$

We have:

$$b_n - b_{n-1} = a_n \frac{p^n}{q^n}$$

$$a_n = \frac{q^n}{p^n}(b_n - b_{n-1})$$

Thus we get for $\arctan(p/q)$:

$$q^2 n b_n - q^2 n b_{n-1} + p^2(n-2)b_{n-2} - p^2(n-2)b_{n-3} = 0$$

Example: compute arctan(3/7).

```
b := proc(n) option remember;
  (49*n*b(n-1)-9*(n-2)*b(n-2)+9*(n-2)*b(n-3))/49/n
end:
b(0):=0:
b(1):=3/7:
b(2):=3/7:
> seq(b2(2*n+1), n=0..5);
      138  34053  11669244  81695643   44033065842
3/7, ---, -----, --------, ---------, -----------
      343  84035  28824005  201768035  108752970865

> evalf(b2(23)), evalf(arctan(3/7));
              0.4048917863, 0.4048917863
```

We have:

$$\begin{pmatrix} b_{n+3} \\ b_{n+2} \\ b_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & \frac{-9(n+1)}{49(n+3)} & \frac{9(n+1)}{49(n+3)} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} b_{n+2} \\ b_{n+1} \\ b_n \end{pmatrix}$$

i.e.

$$B_{n+1} = M_{n+1} B_n$$

with $B_n := \begin{pmatrix} b_{n+2} \\ b_{n+1} \\ b_n \end{pmatrix}$.

$B_n = M_n M_{n-1} \cdots M_2 M_1 B_0$ can be evaluated by applying the binary splitting algorithm to the matrix product

$$M_n M_{n-1} \cdots M_2 M_1,$$

possibly by taking out the denominators to work on integers only.

Now write:

$$x = \underbrace{r_0 + r_1 + \cdots + r_j}_{R_j} + \cdots + r_k$$

where the r_j are (small) rationals.

Define $f_0(x) = \arctan(x)$, $f_1(x) = f_0(r_0 + x) - f_0(r_0)$, ..., $f_{j+1}(x) = f_j(r_j + x) - f_j(r_j)$.

Then $f_j(x) = f_0(R_{j-1} + x) - f_0(R_{j-1})$, and $f_j(r_j) = f_0(R_j) - f_0(R_{j-1})$, thus

$$f_0(r_0) + f_1(r_1) + \cdots + f_k(r_k) = f_0(x_0)$$

The main point is that f_j is holonomic since $f_j(x) = f_0(R_{j-1} + x) - f_0(R_{j-1})$:

```
> deq := (1+(R+x)^2)*diff(f(x), x) - 1:
> diffeqtorec({deq, f(0)=0}, f(x), a(n));
{n a(n) + (2 R n + 2 R) a(n + 1)

              2             2
     + ((1 + R ) n + 2 + 2 R ) a(n + 2), a(0) = 0,

             1
    a(1) = ------}
                2
           1 + R
```

With $R = p/q$, we get

$$(p^2 + q^2)na_n + 2pq(n-1)a_{n-1} + q^2(n-2)a_{n-2} = 0$$

Since the recurrence for (a_n) is similar, that for $(b_n := \sum a_k p^k/q^k)$ is also similar.

The Algorithm (1/2)

1. Compute the differential eq. for $g(x) = f(p/q + x)$.

$$(1 + (p/q + x)^2)g'(x) - 1 = 0$$

2. Deduce the recurrence for the coefficients a_n of $g(x)$.

$$(p^2 + q^2)na_n + 2pq(n-1)a_{n-1} + q^2(n-2)a_{n-2} = 0$$

3. Deduce the recurrence for $b_n = \sum_{k=0}^{n} a_k (u/v)^k$.

$$v^2(p^2 + q^2)nb_n - v(2pqu + vp^2n - 2pqun + vnq^2)b_{n-1}$$

$$-qu(2qu - qun + 2vpn - 2vp)b_{n-2} - q^2u^2(n-2)b_{n-3} = 0$$

The Algorithm (2/2)

4. Split $x = r_0 + \cdots + r_k$. Define $R_j = r_0 + \cdots + r_j$, and $f_j = f(R_{j-1} + x) - f(R_{j-1})$.

5. For each $0 \le j \le k$, form the b_n recurrence for f_j at $x = r_j$, and approximate by binary splitting $y_j \approx f_j(r_j)$.

6. Compute $y_0 + \cdots + y_k$.

Example: consider $x_0 = 3/7$ (in binary form).

$$x_0 = 0.0\underbrace{1}_{r_0}\underbrace{10}_{r_1}\underbrace{1101}_{r_2}101\ldots$$

We have $r_0 = 1/4$, $r_1 = 2/16$, $r_2 = 13/256$.

$f_1(x) = f_0(1/4 + x)$:

$$f_0 : na_n + (n-2)a_{n-2} = 0$$

$$f_0(1/4) : 16nb_n - 16nb_{n-1} + (n-2)b_{n-2} - (n-2)b_{n-3} = 0$$

$$b_0 = 0, b_1 = b_2 = 1/4$$

$f_2(x) = f_1(1/8 + x) = f_0(3/8 + x)$:

$$f_1 : 17na_n + 8(n-1)a_{n-1} + 16(n-2)a_{n-2} = 0$$

$$f_1(1/8): 68nb_n - (64n+4)b_{n-1} - (3n-2)b_{n-2} - (n-2)b_{n-3} = 0$$

$$b_0 = 0, b_1 = 2/17, b_2 = 33/289$$

$f_3(x) = f_2(13/256 + x) = f_0(109/256 + x)$:

$$f_2 : 73na_n + 48(n-1)a_{n-1} + 64(n-2)a_{n-2} = 0$$

$f_2(13/256)$:

$$74752nb_n - (72256n + 2496)b_{n-1} - (2327n - 2158)b_{n-2} - (169n - 338)b_{n-3}$$

$$b_0 = 0, b_1 = 13/292, b_2 = 29861/682112$$

Complexity Analysis (1/2)

The recurrence for the coefficients of $f_j(x)$ can be directly obtained from that of f_0 by a translation of $R_{j-1} = r_0 + \cdots + r_{j-1}$.

R_{j-1} has size $O(2^j)$, thus each a_i (and thus b_i) grows by $O(2^j \log n)$.

Thus a_i (and b_i) has size $O(i2^j \log n)$.

If f has a finite radius of convergence, we need $\Theta(n/2^j)$ terms to get an accuracy of n bits for $f(x)$ when $x < 2^{-2^j}$.

The largest term $a_{n/2^j}$ has thus size $O(n \log n)$.

Complexity Analysis (2/2)

Since $x^{n/2^j}$ has size $O(n)$, the root of the binary (splitting) tree has size $O(n \log n)$.

The cost of each binary splitting tree is:

$$O(M(n \log n) + 2M(n/2 \log(n/2)) + 4M(n/4 \log(n/4)) + \cdots)$$

$$= O(M(n \log n) \log n)$$

If we assume $M(n) = O(n \log^k n)$, this is $O(M(n) \log^2 n)$.

The total cost is $O(M(n) \log^3 n)$.

Another way to compute (b_n)

If (a_n) has order k, (b_n) has order $k+1$.

We have to multiply $(k+1) \times (k+1)$ integer matrices, plus the denominators, i.e. $(k+1)^3 + 1$ integer multiplications per node tree.

Alternate way: we want to compute $S(0, n)$ where

$$S(a, b) = \sum_{k=a}^{b-1} \frac{p(a)p(a+1)\cdots p(k-1)}{q(a)q(a+1)\cdots q(k-1)}$$

where $p(\cdot)$ is a $k \times k$ matrix, and $q(\cdot)$ an integer.

Write $P(a, b) = p(a)p(a+1)\cdots p(b-1)$,
$Q(a, b) = q(a)q(a+1)\cdots q(b-1)$, then

$T(a,b) = Q(a,b)S(a,b)$ can be written:

$$P(a,b) = P(a,c)P(c,b), Q(a,b) = Q(a,c)Q(c,b)$$

$$T(a,b) = T(a,c)Q(c,b) + P(a,c)T(c,b)$$

This needs one $k \times k$ matrix product for P, one integer product for Q, one matrix-scalar product for TQ, and another matrix product for PT. The cost is $2k^3 + k^2 + 1$ per node tree.

k	$2k^3 + k^2 + 1$	$(k+1)^3 + 1$
1	4	9
2	21	28
3	64	65
4	145	126

Conclusion: using $k \times k$ matrices is (theoretically) better for $k \leq 3$.

Is $M(n)\log^2 n$ better than $M(n)\log n$?

Some $O(M(n)\log^2 n)$ algorithms based on binary splitting are better in practice than theoretically optimal $O(M(n)\log n)$ algorithms.

Example: the `gmp-chudnovsky.c` program available on the GMP web page is more efficient than the `const_pi.c` program from the MPFR library, up to several million digits (1.867Mhz Pentium M with gmp-4.2, mpfr-dev):

digits	1M	2M	5M	10M
`const_pi.c`	21.8s	54.4s	180s	440s
`gmp-chudnovsky.c`	4.3s	10.5s	36.6s	95.6s
ratio	5.1	5.2	4.9	4.6

Can we explain that?

Save a Factor of 2

When analyzing Brent's algorithm for exp, or other binary splitting algorithms, we often write:

$$S(n) := M(n)\log n + 2M(\frac{n}{2})\log(\frac{n}{2}) + 4M(\frac{n}{4})\log(\frac{n}{4}) + \cdots$$

$$\approx M(n)\log^2 n$$

As pointed out by Damien Stehlé, this is wrong because the $\log(\frac{n}{2^j})$ terms linearly decrease to zero:

$$S(n) \approx \frac{1}{2}M(n)\log^2 n$$

And Another Factor of 2

If the binary splitting algorithm is optimal, the last multiply gives a result of size exactly n, i.e. the operands have size $n/2$.

This gives $S(n/2) \approx \frac{1}{4}M(n)\log^2 n$.