ASYMPTOTICALLY FAST DIVISION FOR GMP

PAUL ZIMMERMANN

Abstract. Until version 4.2.1, GNU MP (GMP for short) division has complexity $O(M(n)\log n)$, which is not asymptotically optimal. We propose here some division algorithms that achieve $O(M(n))$ with small constants, with corresponding GMP code.

1. Previous work

1.1. **Karl Hasselström's Master Thesis.** Previous work on fast division was done by Karl Hasselström in his master thesis (`http://www.treskal.com/kalle/exjobb/`), whose abstract says:

I have investigated, theoretically and experimentally, under what circumstances Newton division (inversion of the divisor with Newton's method, followed by division with Barrett's method) is the fastest algorithm for integer division. The competition mainly consists of a recursive algorithm by Burnikel and Ziegler.

For divisions where the dividend has twice as many bits as the divisor, Newton division is asymptotically fastest if multiplication of two n-bit integers can be done in time $O(n^c)$ for $c < 1.297$, which is the case in both theory and practice. My implementation of Newton division (using subroutines from GMP, the GNU Multiple Precision Arithmetic Library) is faster than Burnikel and Ziegler's recursive algorithm (a part of GMP) for divisors larger than about five million bits (one and a half million decimal digits), on a standard PC.

Hasselström implemented $1/x$ with Newton's algorithm, following the description in Knuth, and Barrett's division with remainder, given the inverse of the divisor. (His implementation does not give an approximate quotient as intermediate step.)

2. Approximate Inverse

We describe here algorithms that compute an approximate inverse of some positive real a. We denote $\rho = 1/a$ the exact inverse.

2.1. **Mathematical Analysis.** In that section, we assume all computations are done with infinite precision, thus neglecting roundoff errors.

Lemma 1. *Let $x > 0$, and $x' = x + x(1 - ax)$. Then:*

$$0 \le \frac{1}{a} - x' \le \frac{x^2}{\theta^3}(\frac{1}{a} - x)^2,$$

Date: First version in October 2005. Revised August 2006, October 2006, and February 2015 after feedback from Bill Hart.

for some $\theta \in (x, \rho)$.

Proof. Newton's iteration consists in approximating the function by its tangent. Let $f(t) = a - 1/t$, with $\rho = 1/a$ the root of f. The second-order expansion of f at $t = \rho$ with explicit remainder is:

$$f(\rho) = f(x) + (\rho - x)f'(x) + \frac{(\rho - x)^2}{2} f''(\theta),$$

for some $\theta \in (x, \rho)$. Since $f(\rho) = 0$, this simplifies to

$$\rho = x - \frac{f(x)}{f'(x)} - \frac{(\rho - x)^2}{2} \frac{f''(\theta)}{f'(x)}.$$

Substituting $f(t) = a - 1/t$, $f'(t) = 1/t^2$ and $f''(t) = -2/t^3$, it follows:

$$\rho = x + x(1 - ax) + \frac{x^2}{\theta^3}(\rho - x)^2,$$

which proves the claim. □

2.2. **An implementation in $3M(n)$ or $2.5M(n)$.** The following code computes an approximate inverse.

The break-even point between `mpn_invert` and `mpn_divrem` (which performs division with remainder) is around 25 limbs on a Pentium 4 with gmp-4.2.1.

Let β be the GMP internal base, usually $\beta = 2^{32}$ or 2^{64}. The input of `mpn_invert` is the integer A represented in GMP notation by `{ap,n}`. It is assumed that A is normalized, i.e., $A \geq \beta^n/2$; we thus have $\beta^n/2 \leq A < \beta^n$. The output of `mpn_invert` is the integer X represented by `{xp,n}` plus one implicit most significant bit of weight β^n; in other words $X = \beta^n + \{\mathrm{xp}, \mathrm{n}\}$.

```
void
mpn_invert2 (mp_ptr xp, mp_srcptr ap, mp_size_t n)
{
  if (n == 1)
    invert_limb (xp[0], ap[0]);
  else if (n == 2)
    /* omitted special code that guarantees A*X < B^4 <= A*(X+1) */
  else
    {
      mp_size_t l, h;
      mp_ptr tp, up;
      mp_limb_t cy, th;
      TMP_DECL;

      l = (n - 1) / 2;
      h = n - l;

      mpn_invert2 (xp + l, ap + l, h);

      TMP_MARK;
      tp = TMP_ALLOC_LIMBS (n + h);
      up = TMP_ALLOC_LIMBS (2 * h);
      mpn_mul (tp, ap, n, xp + l, h);
      cy = mpn_add_n (tp + h, tp + h, ap, n);
      while (cy)
        {
          mpn_sub_1 (xp + l, xp + l, h, ONE);
          cy -= mpn_sub (tp, tp, n + h, ap, n);
        }

      mpn_com_n (tp, tp, n);
      mpn_add_1 (tp, tp, n, ONE);
      mpn_mul_n (up, tp + l, xp + l, h);
      cy = mpn_add_n (up + h, up + h, tp + l, h - l);
      mpn_add_nc (xp, up + 2*h - l, tp + h, l, cy);
      if (up[2*h-l-1] + 3 <= 2)
        {
          /* omitted special code that checks if A*(X+1) < beta^(2n),
             and if so adds 1 to {xp, l} */
        }
      TMP_FREE;
    }
}
```

Lemma 2. *If $\beta \geq 3$ is a power of two, the output X of the function* `mpn_invert` *satisfies:*

$$AX < \beta^{2n} \leq A(X+1).$$

Remark 1: the reasoning below requires $\beta \geq 3$, but the lemma might hold for $\beta = 2$ too.

Note that the upper inequality $\beta^{2n} \leq A(X+1)$ cannot be replaced by a strict inequality. Indeed, if $A = \beta^n/2$, the largest value of X satisfying the lower inequality $AX < \beta^{2n}$ is $X = 2\beta^n - 1$, which gives $A(X+1) = \beta^{2n}$.

Proof. For $n = 1$ the `invert_limb` internal GMP function returns $X = \lfloor \frac{\beta^2}{A} \rfloor$, except when $A = \beta/2$ where it returns $X = 2\beta - 1$. In all cases we have $AX < \beta^2 \leq A(X+1)$, thus the lemma holds.

For $n = 2$ the function uses special code — omitted here for brievity — that guarantees $AX < \beta^{2n} \leq A(X+1)$, like for $n = 1$.

Now consider $n \geq 3$. We have $\ell = \lfloor (n-1)/2 \rfloor$ and $h = n - \ell$, thus $n = h + \ell$ and $h > \ell$. The idea of the algorithm is to first compute an approximate inverse of the upper h limbs of A, and then to update it to n limbs using Newton's iteration.

After the recursive call

```
mpn_invert2 (xp + l, ap + l, h);
```

we have by induction

$$A_h X_h < \beta^{2h} \leq A_h(X_h + 1), \tag{1}$$

where $A_h = \{ap+\ell, h\}$ and $X_h = \beta^h + \{xp+\ell, h\}$. Let T be the value of $cy \cdot \beta^{n+h} + \{tp, n+h\}$. After

```
mpn_mul (tp, ap, n, xp + l, h);
cy = mpn_add_n (tp + h, tp + h, ap, n);
```

we have $T = AX_h$. After the while-loop

```
    while (cy)
      {
        mpn_sub_1 (xp + l, xp + l, h, ONE);
        cy -= mpn_sub (tp, tp, n + h, ap, n);
      }
```

we still have $T = AX_h$, where T and X_h may have new values, and in addition $T < \beta^{n+h}$. Note that since the initial value of AX_h satisfies $AX_h < \beta^{n+h} + 2\beta^n$, and $A \geq \beta^n/2$, the number of loops is at most 4. (This is achieved for example for $A = 542$, $\beta = 4$ and $n = 5$.) We also have $\beta^{n+h} < T + A$; we prove the latter by distinguishing two cases. Either we entered the while-loop, then since the value of T decreases by A at each loop, the previous value $T + A$ was necessarily larger or equal to β^{n+h}. If we didn't enter the while-loop, the value of T is the original one $T = AX_h$. Multiplying Eq. (1) by β^ℓ gives: $\beta^{n+h} \leq A_h\beta^\ell(X_h + 1) \leq A(X_h + 1) = T + A$. We thus have:

$$T < \beta^{n+h} \leq T + A.$$

It follows $T \geq \beta^{n+h} - A > \beta^{n+h} - \beta^n$. In summary, $\beta^{n+h} - \beta^n < T < \beta^{n+h}$, then $0 < \beta^{n+h} - T < \beta^n$: thus $\beta^{n+h} - T$ — when considered as a $n+h$-limb number — has its h upper limbs equal to zero. This is what we compute with the following lines (the `mpn_add_1` call cannot yield any carry since $\beta^{n+h} - T < \beta^n$):

```
    mpn_com_n (tp, tp, n);
    mpn_add_1 (tp, tp, n, ONE);
```

The last lines compute the product of the h most significant limbs of $\beta^{n+h} - T$ by X_h, and put its ℓ most significant limbs in the low part X_ℓ of the result X:

```
    mpn_mul_n (up, tp + l, xp + l, h);
    cy = mpn_add_n (up + h, up + h, tp + l, h - l);
    mpn_add_nc (xp, up + 2*h - l, tp + h, l, cy);
```

Now let us perform the error analysis. Compared to Lemma 1, x stands for $X_h\beta^{-h}$, a stands for $A\beta^{-n}$, and x' stands for $X\beta^{-n}$. The while-loop ensures that we start from an approximation $x < 1/a$, i.e., $AX_h < \beta^{n+h}$. Then Lemma 1 guarantees that $x \leq x' \leq 1/a$ if x' is computed with infinite precision. Here we have $x \leq x'$, since $X = X_h\beta^h + X_\ell$, where $X_\ell \geq 0$. The only differences with infinite precision are:

- the low ℓ limbs from $1 - ax$ are neglected, and only its upper part $(1 - ax)_h$ is considered;
- the low $2h - \ell$ limbs from $x(1 - ax)_h$ are neglected.

Those two approximations make the computed value of x' smaller or equal to the one which would be computed with infinite precision, thus we have for the computed value x':

$$x \leq x' \leq 1/a.$$

The mathematical error is bounded from Lemma 1 by $\frac{x^2}{\theta^3}(\rho - x)^2 < \beta^{-2h}$ since $\frac{x^2}{\theta^3} \leq 1$ and $|\rho - x| < \beta^{-h}$. The truncation from $1 - ax$, which is multiplied by $x < 2$, produces an error less than $2\beta^{-2h}$. Finally the truncation of $x(1-ax)_h$ produces an error less than β^{-n}. The final result is thus:

$$x' \le \rho < x' + \beta^{-n} + 3\beta^{-2h}.$$

Assuming $3\beta^{-2h} \le \beta^{-n}$, which holds as soon as $\beta \ge 3$ since $2h > n$, this simplifies to:

$$x' \le \rho < x' + \beta^{-n}(1 + 3/\beta),$$

which gives with $x' = X\beta^{-n}$ and $\rho = \beta^n/A$:

$$X \le \frac{\beta^{2n}}{A} < X + 1 + 3/\beta. \tag{2}$$

Since β is assumed to be a power of two, equality $X = \beta^{2n}/A$ can hold only when A is a power of two itself, i.e., $A = \beta^n/2$. In that case there is only one value of X_h that is possible for the recursive call, namely $X_h = 2\beta^h - 1$. In that case $T = \beta^{n+h} - \beta^n/2$ before the while-loop, which is not entered. Then $\beta^{n+h} - T = \beta^n/2$, whose upper part `{tp+l, h}` is $\beta^h/2$, which multiplied by X_h gives $\beta^{2h} - \beta^h/2$, whose ℓ most significant limbs are all $\beta - 1$. Thus $X_\ell = \beta^\ell - 1$, and $X = 2\beta^n - 1$: equality does not occur either in that case. □

Remark 2: the condition `up[2*h-l-1] + 3 <= 2` is fullfilled when the limb `up[2*h-l-1]` is $\beta - 3$, $\beta - 2$, or $\beta - 1$. When that condition does not hold, taking into account the smaller limbs from `up`, the neglected part is bounded strictly by $\beta - 3$, thus when adding the maximal mathematical error and that from the truncation of $1 - ax$, which amounts to at most 3 — $3/\beta$ to the next more significant limb — no carry can occur at the next limb. This condition happens with very low probability with say $\beta = 2^{64}$. This condition is optimal since for example with $A = 8888$, $\beta = 4$, $n = 7$ we need a correction for $up_{2h-\ell-1} = 1 = \beta - 3$.

Complexity Analysis: let $I(n)$ be the cost to invert an n-limb number. If we neglect the linear costs, we have $I(n) = I(n/2) + M(n, n/2) + M(n/2)$, where $M(n, n/2)$ is the cost of multiplying an $n \times (n/2)$ product, and $M(n/2)$ the cost of an $(n/2) \times (n/2)$ product. If the $n \times (n/2)$ product is performed via two $(n/2) \times (n/2)$ products, we have $I(n) = I(n/2) + 3M(n/2)$, which yields $M(n)$ in the quadratic range, $3/2M(n)$ in the Karatsuba range, $1.704\ldots M(n)$ in the Toom-Cook 3-way range, and $3M(n)$ in the FFT range. However, in the FFT range, a $(n/2) \times (n/2)$ product is computed by a FFT on n limbs, and a $n \times (n/2)$ product can be directly computed by a FFT on $3n/2$ limbs, which therefore amounts to $3/2M(n/2)$. In that case the complexity decreases to $2.5M(n)$ in the FFT range.

Here are some timings obtained on a 2.83Ghz Core 2 Q9550 with gmp-6.0.0 and gcc 4.9.1. The `mpn_mul_n` column gives real timings; the other columns give the ratio with respect to `mpn_mul_n`:

limbs	`mpn_mul_n`	`mpn_divrem`	`mpn_invert2`
1000	0.243ms	2.54	1.75
10000	4.84ms	2.54	1.95
100000	66.2ms	2.68	2.13
1000000	982.ms	2.22	1.92
10000000	12.6s	2.76	2.23

2.3. **An implementation in $2M(n)$.** In the product AX_h:

```
mpn_mul (tp, ap, n, xp + l, h);
cy = mpn_add_n (tp + h, tp + h, ap, n);
```

we know from Eq. (1) that the result will be very near from β^{n+h}, more precisely:

$$\beta^{n+h} - \beta^n < AX_h < \beta^{n+h} + 2\beta^n.$$

Assume we use an FFT algorithm that computes products modulo $\beta^m + 1$ like Schönhage-Strassen's algorithm. We use here $m > n$. Let $AX_h = U\beta^m + V$ with $0 \le V < \beta^m$. It follows from above that $U = \beta^{n+h-m}$ or $U = \beta^{n+h-m} - 1$. Let $T = AX_h \bmod (\beta^m + 1)$ the value computed by the FFT. We have $T = V - U$ or $T = V - U + (\beta^m + 1)$. It follows $AX_h = T + U(\beta^m + 1)$ or $AX_h = T + (U - 1)(\beta^m + 1)$. Taking into account the two possible values of U, we have $AX_h = T + (\beta^{n+h-m} - \epsilon)(\beta^m + 1)$ where $\epsilon \in \{0, 1, 2\}$. For $m > n$, we have $\beta^m \ge 3\beta^n$, thus only one value of ϵ yields a value in the interval $(\beta^{n+h} - \beta^n, \beta^{n+h} + 2\beta^n)$.

We summarize the algorithm:

```
Compute $T = AX_h \bmod (\beta^m + 1)$ using FFT
$T \leftarrow T + \beta^{n+h} + \beta^{n+h-m}$        $\{$ case $\epsilon = 0$ $\}$
**while** $T \ge \beta^{n+h} + 2\beta^n$ **do**
   $T \leftarrow T - (\beta^m + 1)$
```

3. Approximate Quotient

Assume we have a dividend $B < \beta^{2n}$, a divisor $\beta^n/2 \le A < \beta^n$, and we compute an approximate quotient Q with the following algorithm:

```
$X \leftarrow \mathrm{ApproxInvert}(A)$
Write $B = B_1\beta^n + B_0$ with $0 \le B_0, B_1 < \beta^n$
$Q \leftarrow \lfloor \frac{B_1 X}{\beta^n} \rfloor$
```

where the ApproxInvert function returns X such that:

$$AX < \beta^{2n} \le A(X + 1),$$

as for example the above `mpn_invert2` routine.

Lemma 3. *The approximate quotient Q satisfies:*

$$0 \le B - QA < 4A.$$

Proof. Let $S = \beta^{2n} - AX$, which satisfies $0 < S \le A$ from Lemma 2, and $R = B_1 X - Q\beta^n$, which satisfies $0 \le R < \beta^n$. We have:

$$\begin{aligned} B - QA &= B_1\beta^n + B_0 - \frac{B_1 X - R}{\beta^n} A \\ &= B_1\beta^n + B_0 - \frac{B_1}{\beta^n}(\beta^{2n} - S) + \frac{R}{\beta^n} A \\ &= B_0 + \frac{B_1}{\beta^n} S + \frac{R}{\beta^n} A. \end{aligned}$$

Now $B_0 < \beta^n \le 2A$, $B_1 < \beta^n$, $S \le A$ and $R < \beta^n$. □