Karatsuba Square Root Algorithm in Detail

Sebastian Mesow

April 4, 2023

We implement the recursive Karatsuba Square Root Algorithm by Paul Zimmermann as described in the following two papers:

1. https://hal.inria.fr/inria-00072854/document
2. https://hal.inria.fr/inria-00072113/document

Hello Hello
– Hello Hello Hello Hello Hello
– Hello Hello Hello Hello Hello
– Hello Hello Hello Hello Hello

Hello Hello

1 Implementation Function

1.1 1st-level Pseudo Code

Syntax: $S, R := \texttt{SqrtRem\_Impl}(X)$

Input:
1. normalized input integer X

Preconditions:
1. Normalization Condition
   – with m the minimal length of X
   1.1. $X \geq \frac{B^m}{4}$
   1.2. $X < B^m$

Output:
1. square root S
2. square root remainder R

Postconditions:
1. Integer Square Root Condition:
   1.1. $S^2 \leq X$
   1.2. $(S+1)^2 > X$
2. Integer Square Root Remainder Condition: $S^2 + R = X$

Deduced Postconditions:
1. Upper Bound for Square Root:
   – $S^2 \leq X \wedge X < B^m \implies S^2 < B^m$ is equivalent to
   1.1. $S < B^{\frac{m}{2}}$
2. Upper Bounds for Square Root Remainder
   – $(S+1)^2 > X \iff S^2 + 2S + 1 > X \iff 2S + 1 > X - S^2 \iff 2S \geq X - S^2$
   – and $S^2 + R = X \iff R = X - S^2$ implies
   2.1. $R \leq 2S$
   – $R \leq 2S \wedge S < B^{\frac{m}{2}}$ implies
   2.2. $R < 2B^{\frac{m}{2}}$

Algorithm:

Statement	Comment
1. compute H, L such that	
1.1. $HL = N$	
1.2. $H \geq L$	
1.3. $X = X_{23}L^2 + X_1 L + X_0$	
2. $S_p, R_p := \texttt{SqrtRem\_Wrap}(X_{23})$	
3. $Q, U := \texttt{DivRem}(R_p L + X_1, 2S_p)$	
4. $R_t := UL + X_0 - Q^2$	
5. $S_t := S_p L + Q$	
6.	correct return values
6.1. if $R_t < 0$ then	
6.1.1. $R := R_t + 2S_t - 1$	
6.1.2. $S := S_t - 1$	
6.2. else	if $R_t \geq 0$
6.2.1. $R := R_t$	
6.2.2. $S := S_t$	

1.2 2nd-level Pseudo Code

Syntax: $S, R := \texttt{SqrtRem\_Impl}(X)$

Input:

1. normalized input number X

Preconditions:

1. Normalization Condition
   - with m the minimal length of X
   1.1. $X \geq \frac{B^m}{4}$
   1.2. $X < B^m$

Output:

1. square root S
2. square root remainder R

Postconditions:

1. Integer Square Root Condition:
   1.1. $S^2 \leq X$
   1.2. $(S+1)^2 > X$
2. Integer Square Root Remainder Condition: $S^2 + R = X$

Algorithm:

Statement	Comment
1. compute H, L such that	
1.1. $HL = N$	
1.2. $H \geq L$	
1.3. $X = X_{23}L^2 + X_1 L + X_0$	
2. $S_p, R_p := \texttt{SqrtRem\_Wrap}(X_{23})$	
3.	$Q, U := \texttt{DivRem}(R_p L + X_1, 2S_p)$
	it holds $R_p L + X_1 = Q \cdot 2S_p + U$ it holds $Q < L$ (see research paper) it holds $U < 2S_p$
3.1. $Q_t, U_t := \texttt{DivRem}(R_p L + X_1, S_p)$	
	it holds $R_p L + X_1 = Q_t S_p + U_t$
	it must will hold $R_p L + X_1 \stackrel{!}{=} Q \cdot 2S_p + U$
3.2. $Q := \lfloor \frac{Q_t}{2} \rfloor$	

	if Q_t is even: – it holds $2Q = Q_t$ – with $U := U_t$ it holds $$\begin{aligned} R_pL + X_1 &= Q_tS_p + U_t \\ &= 2Q \cdot S_p + U_t \\ &= Q \cdot 2S_p + U \end{aligned}$$
	if Q_t is odd: – it holds $2Q + 1 = Q_t$ – with $U := U_t + S_p$ it holds $$\begin{aligned} R_pL + X_1 &= Q_tS_p + U_t \\ &= (2Q+1)S_p + U_t \\ &= 2Q \cdot S_p + S_p + U_t \\ &= Q \cdot 2S_p + U \end{aligned}$$
3.3.	correct division remainder
3.3.1. if Q_t is odd then	
3.3.1.1. $U := U_t + S_p$	
3.3.2. else	if Q_t is even
3.3.2.1. $U := U_t$	
4.	$R_t := UL + X_0 - Q^2$
4.1. $Q_{sqr} := Q^2$	
4.2. $R_t := UL + X_0 - Q_{sqr}$	
5.	$S_t := SL + Q$
5.1. $S_t := S_pL + Q$	
6.	correct return values
6.1. if $R_t < 0$ then	
6.1.1.	$R := R_t + 2S_t - 1$
6.1.1.1. $R_{tt} := R_t + 2S_t$	
6.1.1.2. $R := R_{tt} - 1$	
6.1.2. $S := S_t - 1$	
6.2. else	if $R_t \geq 0$
6.2.1. $R := R_t$	
6.2.2. $S := S_t$	

1.3 3rd-level Pseudo Code

Syntax: $S, rB^n + R_o :=$ `mpn_SqrtRem_Impl`(X, n)

Input:

1. normalized input number X
2. the exact half of its minimal length n
   – Thus the minimal length of X must be even.

Preconditions:

1. Normalization Condition
   – with $2n$ the minimal length of X
   1.1. $X \geq \frac{B^{2n}}{4}$
   1.2. $X < B^{2n}$

Output:

1. square root S
   – will have length n
2. square root remainder without its overflow bit R_o
   – will have length n
3. square root remainder overflow bit r

Postconditions:

1. Integer Square Root Condition:

1.1. $S^2 \leq X$
1.2. $(S+1)^2 > X$
2. Integer Square Root Remainder Condition: $S^2 + rB^n + R_o = X$
3. S has length n:
3.1. $S \geq 0$
3.2. $S < B^n$
4. R_o has length n:
4.1. $R_o \geq 0$
4.2. $R_o < B^n$

Algorithm:

Statement	Comment
1.	compute H, L such that – $HL = N$ – $H \geq L$ – $X = X_{23}L^2 + X_1 L + X_0$
1.1. $l := \lfloor \frac{n}{2} \rfloor$	$L := B^l$
1.2. $h := n - l$	$H := B^h$
	it holds $X_0 = X \mod L$ it holds $X_1 = \lfloor \frac{X}{L} \rfloor \mod L$ it holds $X_{23} = \frac{X}{L^2}$
	X_0 has length l X_1 has length l X_{23} has the minimal length $2h$
2.	$S_p, R_p :=$ `SqrtRem_Wrap`(X_{23})
	X_{23} is normalized: – Its most significant limb is equal to that of X. – Its minimal length $2h$ is even.
2.1. $S_p, r_p H + R_{po} :=$ `mpn_SqrtRem_Impl`(X_{23}, h)	
	S_p has length $h \iff S_p < H$ R_{po} has length $h \iff R_{po} < H$
	$R_p = r_p H + R_{po}$
3.	$Q, U :=$ `DivRem`$(R_p L + X_1, 2S_p)$
3.1.	$Q_t, U_t :=$ `DivRem`$(R_p L + X_1, S_p)$
	We do not have R_p at hand. Also R_p can be H and thus may have a minimal length of $h+1$, which would make things a bit more complicate. We only have at hand r_p and $R_{po} < N$ seperated. Thus we like to execute something like `DivRem`$(R_{po}L + X_1, S_p)$.
	Suppose, we neglect the next subtraction. If $r_p = 1 \iff R_p \geq H$, then $R_{po} = R_p - H$ and $$\begin{aligned} Q_t, U_t &:= \texttt{DivRem}\left(R_{po}L + X_1, S_p\right) \\ &= \texttt{DivRem}\left((R_p - H)L + X_1, S_p\right) \\ \iff Q_t S_p + U_t &= (R_p - H)L + X_1 \\ &= R_p L - HL + X_1 \\ &= R_p L + X_1 - N \end{aligned}$$ which is not desired.
	How can we manipulate the dividend $R_{po}L + X_1$, such that manageable side effect $+ Q_t S_p + U_t =$ manipulated ?
	Observe: If we subtract $S_p L$ from the dividend, then the quotient is smaller by L (manageable side effect).

	Observe: – Because $R_p \leq 2S_p = S_p + S_p$ and $S_p < H$, is $\begin{aligned} & R_p < S_p + H \\ \iff & R_p - H < S_p \\ \iff & R_p - H - S_p = R_{po} - S_p < 0 \end{aligned}$ – At this subtraction occurs an underflow $(+H)$. Thus $R_{po} - S_p$ is in reality $R_p - H - S_p + H = R_p - S_p$.
	Conclusion: – If $R_p \geq H \iff r_p = 1$, then we replace R_{po} by $R_{po} - S_p$ and the *obtained* quotient $q_t L + Q_{to}$ is smaller than the *required* quotient Q_t by L . – The $+H$ by the underflow propagates a borrow bit to the left and – as a nice side effect – mathematical precisely cancels r_p .
	Check of this Idea: $\begin{aligned} q_{t,ob}L + Q_{to}, U_t &:= \mathtt{DivRem}\big((R_p - S_p)L + X_1, S_p\big) \\ \iff (q_{t,ob}L + Q_{to})S_p + U_t &= (R_p - S_p)L + X_1 \\ &= R_pL - S_pL + X_1 \\ \iff (q_{t,ob}L + Q_{to})S_p + S_pL + U_t &= R_pL + X_1 \\ = \big((q_{t,ob} + 1)L + Q_{to}\big)S_p + U_t & \\ \iff (q_{t,ob} + 1)L + Q_{to}, U_t &= \mathtt{DivRem}(R_pL + X_1, S_p) \\ &=: q_tL + Q_{to}, U_t \end{aligned}$
3.1.1. if $r_p > 0$ then	
3.1.1.1. $R_{po} := R_{po} - S_p$	actually we would must also $r_p := 0$
	$R_{po}L + X_1$ has length $h + l = n$
3.1.2. $q_{t,ob}L + Q_{to}, U_t :=$ $\mathtt{mpn\_DivRem}(R_{po}L + X_1, S_p)$	
	according to the postconditions of $\mathtt{mpn\_DivRem}()$ – $Q_{to} < L$, because Q_{to} has length $l = n - h = (h + l) - h$ – $U_t < S_p \implies U_t < H$, because U_t has length h (same as the divisor S_p) – $q_{t,ob} = 0$ or 1
	If $r_p > 0 \iff r_p = 1 \iff R_p \geq H$, then the *required* q_t is bigger than the *obtained* $q_{t,ob}$ by 1 .
3.1.3. $q_t := q_{t,ob} + r_p$	To reflect this in reality we, just add $r_p = 1$ to $q_{t,ob}$. If $r_p = 0 \iff R_p = R_{po}$, then the addition is useless.
	memorize $Q_t = q_tL + Q_{to}$
	memorize $q_t = 0$ or 1 or 2
	it holds $\begin{aligned} Q_tS_p + U_t &= R_pL + X_1 \\ \iff (q_tL + Q_{to})S_p + U_t &= (r_pH + R_{po})L + X_1 \end{aligned}$
3.2.	$Q := \lfloor \frac{Q_t}{2} \rfloor$
3.2.1. $Q_o := Q_{to}$ `>>` 1	Q_o has length l, because Q_{to} has length l .
3.2.2. $Q_o[l-1] := \big((q_t \mathbin{\&} 1) \ll (b-1)\big) \mathbin{\vert} Q_o[l-1]$	The least significant bit of q_t as it shifts right floats into the most significant bit of the last limb of Q_o. Because Q_o has length l, the last limb is at index $l - 1$. (zero-based index)
3.2.3. $q := q_t$ `>>` 1	memorize $q = 0$ or 1 .
	memorize $Q = qL + Q_o$
3.3.	correct division remainder
3.3.1. if $Q_{to} \mathbin{\&} 1 = 1$ then	if Q_t is odd then
3.3.1.1. $uH + U_o := U_t + S_p$	$U := U_t + S_p$
3.3.2. else	if Q_t is even $\iff$ if $Q_{to} \mathbin{\&} 1 = 0$

3.3.2.1.	$U := U_t$
3.3.2.1.1. $U_o := U_t$	
3.3.2.1.2. $u := 0$	
	U_o has length $h \iff U_o < H$
4.	$R_t := UL + X_0 - Q^2$
4.1.	$Q_{sqr} := Q^2$
	Recap $Q \leq L$ (see research paper) Thus either $q = 0$ or $Q_o = 0$ or both $= 0$, but never together > 0 . Thus always $qQ_o = 0$.
	$\begin{aligned} Q^2 &= (qL + Q_o)^2 \\ &= qL^2 + 2qLQ_o + Q_o^2 \\ &= qL^2 + Q_o^2 \\ &=: qL^2 + Q_{o,sqr} \end{aligned}$ See how q remains the same in Q^2 .
4.1.1. $Q_{o,sqr} := Q_o^2$	
	$Q_{o,sqr}$ has length $2l$, because Q_o has length l . $Q_o < L \iff Q_o^2 < L^2$
	Memorize $Q_{sqr} = qL^2 + Q_{o,sqr}$
4.2.	$R_t := UL + X_0 - Q_{sqr}$
	$U_oL + X_0$ has length $n = h + l$.
	$\begin{aligned} R_t &= (UL + X_0) - Q_{sqr} \\ = r_tN + R_{to} &:= (uH + U_o)L + X_0 - qL^2 - Q_{o,sqr} \\ &= uHL - qL^2 + U_oL + X_0 - Q_{o,sqr} \\ &= uN - qL^2 + (U_oL + X_0) - Q_{o,sqr} \end{aligned}$
4.2.1. $sub\_borrow, R_{to} :=$ `mpn_sub_n`$(U_oL + X_0,\ Q_{o,sqr},\ 2l)$	`mpn_sub_n`(X, Y, n) subtracts the n least significant limbs of X and Y. It leaves other limbs untouched. Thus a borrow is still returned.
	currently R_{to} has length $2l$
4.2.2.	assign borrows to their correct place
	r_t is has the meaning of a carry, but can be negative.
4.2.2.1. if $h = l$ then	$\iff$ if n is even
	$U_oL + X_0$ has length $h + l = n$. $Q_{o,sqr}$ has length $2l = n$. Thus in this case $U_oL + X_1$ and $Q_{o,sqr}$ have the same length. Thus in this case sub_borrow goes into the carry r_t . (Thus not n but $2l$ is provided to `mpn_sub_n`() .)
	Because R_{to} has length $2l$, $-qL^2$ also goes into the carry r_t .
4.2.2.1.1. $r_t := u - sub\_borrow - q$	
4.2.2.2. else	if $l + 1 = h \iff$ if n is odd
	$U_oL + X_1$ has length $h + l = n = 2l + 1$. $Q_{o,sqr}$ has length $2l$. So $U_oL + X_1$ has one limb more than $Q_{o,sqr}$. Thus to complete the subtraction $(U_oL + X_0) - Q_{o,sqr}$ the sub_borrow must borrow from the last (and still untouched) limb of U_oL+X_0 forming the last limb of R_{to} . This may delivers the actual borrow.
	Because R_{to} has length $2l$, $-qL^2$ also goes into the last (and still untouched) limb of $U_oL + X_0$ forming the last limb of R_{to} .
4.2.2.2.1. $actual\_borrow, R_{to}[2l] :=$ $(U_oL{+}X_0)[2l]{-}sub\_borrow{-}q$	The last limb of $U_oL + X_0$ is at index $2l$. (zero-based index)
4.2.2.2.2. $r_t := u - actual\_borrow$	
	Now if r_t is negative, then this stands therefore, that $R_t = (UL + X_o) - Q^2$ is negative.

	finally R_{to} has a length of n
5.	$S_t := SL + Q$
5.1.	$\begin{aligned} S_t &:= S_pL + Q \\ = s_tN + S_{to} &:= S_pL + qL + Q_o \\ = s_tN + S_{to,high}L + Q_o &:= (S_p + q)L + Q_o \end{aligned}$
5.1.1. $s_tH + S_{to,high} := S_p + q$	Note: $N = HL \iff n = h + l$
	$S_{o,high}$ as length h, because S_p has length h .
5.1.2. $S_{to} := S_{to,high}L + Q_o$	
	S_{to} as length $n = h + l$, because $S_{to,high}$ has length h and Q_o has length l.
	We can only execute these statements after 3.3.1.1. , because until including 3.3.1.1. S_p (and not $S_p + q$) is still required and S_p is stored at memory, which will become S_{to} here.
6.	correct return values
6.1. if $r_t < 0$ then	if $R_t < 0$ then
6.1.1.	$R := R_t + 2S_t - 1$
6.1.1.1.	$R_{tt} := R_t + 2S_t$
6.1.1.1.1. $r_{tt}N + R_{tto} := R_{to} + 2S_{to}$	R_{tto} has length n .
6.1.1.1.2. $r_{tt} := r_{tt} + 2s_t$	
6.1.1.2.	$R := R_{tt} - 1$
6.1.1.2.1. $temp\_borrow, R_o := R_{tto} - 1$	return this R_o
6.1.1.2.2. $r := r_{tto} - temp\_borrow$	return this r
6.1.2.	$S := S_t - 1$
6.1.2.1. $temp\_borrow, S_o := S_{to} - 1$	
6.1.2.2. $s := s_t - temp\_borrow$	
6.2. else	else if $R_t \geq 0 \iff$ else if $r_t \geq 0$
6.2.1. $R := R_t$	
6.2.2. $S := S_t$	

- 6. correct return values: - 1if $R_t < 0$ then - 1. if $r_t < 0$ then - 1. $R := R_t + 2S_t - 1$ - 1. $R_{tt} := R_t + 2S_t$ - $= r_tN + R_{to} + 2(s_tN + S_{to})$¡br¿ $= (r_t + 2s_t)N + R_{to} + 2S_{to}$ - 1. $r'_{tto}N + R_{tto} := R_{to} + 2S_{to}$ - 2. $r_{tto} := r'_{tto} + 2s_t$ - 2. $R := R_{tt} - 1$ - $= r_{tt}N + R_{tto} - 1$ - 1. $temp\_borrow, R_o := R_{tto} - 1$ - 2. $r := r_{tto} - temp\_borrow$ - 2.$S := S_t - 1$ - 2. else - if $R_t \geq 0$ - 1. $R := R_t$ - 2. $S := S_t$