Squaring integers as a special case of multiplication

Using the methodology described in the previous article https://www.cambridge.org/engage/coe/article-details/5eb717c5d7fc6400199f2159 [1], namely the use of an array of products as an auxiliary multiplication table for calculating the squares of integers, we can obtain an additional significant positive performance effect, as the total number of terms decreases. If we compare the product of polynomials with squaring them, it is easy to see that when squaring in comparison with multiplying polynomials, half as many multiplications of elements are required, except for multiplications of elements with the same indices. Moreover, multiplication by 2 in the binary system is inherently the operation of binary shift to the left by one bit.

Multiplying polynomials	Squaring polynomials
$(a_1 + a_0)(b_1 + b_0) \quad = \boldsymbol{a_1 b_1} + \boldsymbol{a_0 b_0} + a_1 b_0 + a_0 b_1$	$(a_1 + a_0)^2 = \boldsymbol{a_1^2} + \boldsymbol{a_0^2} \quad + 2a_1 a_0$
$(a_2 + a_1 + a_0)(b_2 + b_1 + b_0)$ $= \boldsymbol{a_2 b_2} + \boldsymbol{a_1 b_1} + \boldsymbol{a_0 b_0}$ $+ a_2 b_1 + a_1 b_2 + a_2 b_0 + a_0 b_2 + a_1 b_0 + a_0 b_1$	$(a_2 + a_1 + a_0)^2 = \boldsymbol{a_2^2} + \boldsymbol{a_1^2} + \boldsymbol{a_0^2} \quad + 2a_2 a_1 + 2a_2 a_0 + 2a_1 a_0$
$(a_3 + a_2 + a_1 + a_0) \cdot (b_3 + b_2 + b_1 + b_0)$ $= \boldsymbol{a_3 b_3} + \boldsymbol{a_2 b_2} + \boldsymbol{a_1 b_1} + \boldsymbol{a_0 b_0} + a_2 b_3 + a_3 b_2 + a_1 b_3$ $+ a_3 b_1 + a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0 + a_0 b_2 + a_2 b_0$ $+ a_0 b_1 + a_1 b_0$	$(a_3 + a_2 + a_1 + a_0)^2$ $= \boldsymbol{a_3^2} + \boldsymbol{a_2^2} + \boldsymbol{a_1^2} + \boldsymbol{a_0^2} + 2a_3 a_2 + 2a_3 a_1 + 2a_3 a_0 + 2a_2 a_1 + 2a_2 a_0 + 2a_1 a_0$
….	
$(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)$ $\cdot (b_7 + b_6 + b_5 + b_4 + b_3 + b_2 + b_1 + b_0)$ $= \boldsymbol{a_7 b_7} + \boldsymbol{a_6 b_6} + \boldsymbol{a_5 b_5} + \boldsymbol{a_4 b_4} + \boldsymbol{a_3 b_3}$ $+ \boldsymbol{a_2 b_2} + \boldsymbol{a_1 b_1} + \boldsymbol{a_0 b_0} + a_7 b_6 + a_6 b_7 + a_7 b_5 + a_5 b_7$ $+ a_7 b_4 + a_4 b_7 + a_7 b_3 + a_3 b_7 + a_7 b_2 + a_2 b_7 + a_7 b_1$ $+ a_1 b_7 + a_7 b_0 + a_0 b_7 + a_6 b_5 + a_5 b_6 + a_6 b_4 + a_4 b_6$ $+ a_6 b_3 + a_3 b_6 + a_6 b_2 + a_2 b_6 + a_6 b_1 + a_1 b_6 + a_6 b_0$ $+ a_0 b_6 + a_5 b_4 + a_4 b_5 + a_5 b_3 + a_3 b_5 + a_5 b_2 + a_2 b_5$ $+ a_5 b_1 + a_1 b_5 + a_5 b_0 + a_0 b_5 + a_4 b_3 + a_3 b_4 + a_4 b_2$ $+ a_2 b_4 + a_4 b_1 + a_1 b_4 + a_4 b_0 + a_0 b_4 + a_2 b_3 + a_3 b_2$ $+ a_1 b_3 + a_3 b_1 + a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0 + a_0 b_2$ $+ a_2 b_0 + a_0 b_1 + a_1 b_0$	$(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)^2$ $= \boldsymbol{a_7^2} + \boldsymbol{a_6^2} + \boldsymbol{a_5^2} + \boldsymbol{a_4^2} + \boldsymbol{a_3^2} + \boldsymbol{a_2^2} + \boldsymbol{a_1^2} + \boldsymbol{a_0^2} + 2a_7 a_6 + 2a_7 a_5 + 2a_7 a_4 + 2a_7 a_3$ $+ 2a_7 a_2 + 2a_7 a_1 + 2a_7 a_0 + 2a_6 a_5 + 2a_6 a_4 + 2a_6 a_3 + 2a_6 a_2 + 2a_6 a_1 + 2a_6 a_0$ $+ 2a_5 a_4 + 2a_5 a_3 + 2a_5 a_2 + 2a_5 a_1 + 2a_5 a_0 + 2a_4 a_3 + 2a_4 a_2 + 2a_4 a_1 + 2a_4 a_0$ $+ 2a_3 a_2 + 2a_3 a_1 + 2a_3 a_0 + 2a_2 a_1 + 2a_2 a_0 + 2a_1 a_0$

It can be seen from the table that when squaring an 8-bit integer divided into 4-bit components, as compared to multiplying the ALU adder, it is enough to perform one addition instead of 2x, which saves one assembly of the term and one addition, i.e., a performance increase of more than 33 %

Below is a comparison of the operations of multiplying 2 numbers and squaring an integer. Let it be necessary to square a certain integer, where the highest 4 bits of this number are - a_1, and the least significant bits are - a_0, using the auxiliary table (see Fig. 4 previous article) we determine the multiplications $a_1 a_1$, $a_1 a_0$ and $a_0 a_0$.

a_1				a_0			
7	5	5	4	3	2	1	0

1	1	0	0	0	1	1	1

Fig.1. Separation of an integer

Digit			15							8	7			4				0
Step1	a_1a_1+ a_0a_0	F.overflow	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1
+Step2	a_1a_0						0	1	0	1	0	1	0	0				
+Step3	a_1a_0						0	1	0	1	0	1	0	0				
	result	**X**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**

Fig.2. Squaring as a multiplication of two numbers.

Namely, squaring $(a_1 2^4 + a_0)^2 = a_1a_1 2^8 + 2a_1a_0 2^4 + a_0a_0$, taking into account that the double multiplication of integers 2*a1*a0 = a1*a0<< with a shift of one digit to the left, i.e. instead of Fig.1 gives $(a_1 2^4 + a_0)^2 = a_1a_1 2^8 + a_1a_0 2^5 + a_0a_0$:

Digit			15							8	7		5					0
Step1	a_1a_1+ a_0a_0	F.overflow	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1
+Step2	$2a_1a_0$					0	1	0	1	0	1	0	0					
	result	**X**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**

Fig.3. Squaring of an integer.

Consider squaring a 32-bit number, divided into four 8-bit elements, and comparing using the method of calculating the product of integers described inhttps://www.cambridge.org/engage/coe/article-details/5eb717c5d7fc6400199f2159

$A = a_3 \cdot 2^{24} + a_2 \cdot 2^{16} + a_1 \cdot 2^8 + a_0$ and $B = b_3 \cdot 2^{24} + b_2 \cdot 2^{16} + b_1 \cdot 2^8 + b_0$.
$A \cdot B = (a_3 \cdot 2^{24} + a_2 \cdot 2^{16} + a_1 \cdot 2^8 + a_0) \cdot (b_3 \cdot 2^{24} + b_2 \cdot 2^{16} + b_1 \cdot 2^8 + b_0) = a_3 b_3 \cdot 2^{48} + (a_2\, b_3 + a_3 b_2) \cdot 2^{40} + (a_2 b_2 + a_3 b_1 + a_1 b_3) \cdot 2^{32} +$
$(a_3 b_0 + a_0 b_3 + a_2 b_1 + a_1 b_2) \cdot 2^{24} + (a_1 b_1 + a_2 b_0 + a_0 b_2) \cdot 2^{16} + (a_0 b_1 + a_1 b_0) 2^8 + a_0 b_0$.

In this case, squaring is performed similarly to multiplication when A = B. When using the operation of multiplication, we have 7 additions of intermediate results:

	F.overflow	63	…	56	55	…	48	47	…	40	39	…	32	31	…	24	23	…	16	15	…	8	7	…	0
Factor A														a_3			a_2			a_1			a_0		
Factor A														a_3			a_2			a_1			a_0		
Step1		$a_3 a_3$						$a_2 a_2$						$a_1 a_1$						$a_0 a_0$					
+Step2					$a_2 a_3$						$a_0 a_3$						$a_0 a_1$								
+Step3					$a_3 a_2$						$a_1 a_2$						$a_1 a_0$								
+Step4								$a_3 a_1$						$a_2 a_0$											
+Step5								$a_1 a_3$						$a_0 a_2$											
+Step5											$a_2 a_1$														
+Step7											$a_3 a_0$														
RESULT																									

Fig.4. Squaring as a multiplication of a 32-bit integer.

When squaring a polynomial from 4 terms, we have:

$$A \cdot A = a_3 a_3 \cdot 2^{48} + a_2 a_2 \cdot 2^{32} + a_1 a_1 \cdot 2^{16} + a_0 a_0 + a_3 a_2 \cdot 2^{41} + a_3 a_1 \cdot 2^{33} + (a_3 a_0 + a_2 a_1) \cdot 2^{25} + a_2 a_0 \cdot 2^{17} + a_1 a_0 \cdot 2^9,$$

which allows you to square an integer in 4 steps instead of 7 when multiplying by itself.

Bits	F.overflow	63	……	55	55	……	48	47	……	40	39	……	32	31	……	24	23	……	15	15	……	8	7	……	0
Factor														a_3			a_2			a_1			a_0		
Step1		$a_3 a_3$						$a_2 a_2$						$a_1 a_1$						$a_0 a_0$					
+Step2				$a_3 a_2$						$a_3 a_0$						$a_1 a_0$									
+Step3							$a_3 a_1$						$a_2 a_0$												
+Step4										$a_2 a_1$															
RESULT																									

Fig.5. Squaring of a 32-bit integer.

Since the terms formed at each step are independent, it is possible to organize parallel calculations, and when using 2 adders, this action can be performed in 2 steps.

	Adder1	Adder2	Adder2

+Step1	Step1’		Step2’
+Step2			
+Step3		Step1”	
+Step4			

Fig.6. Parallelizing of the operation

Compare squaring a 64-bit number by dividing it into eight 8-bit parts by multiplication

$A = a_7 \cdot 2^{56} + a_6 \cdot 2^{48} + a_5 \cdot 2^{40} + a_4 \cdot 2^{32} + a_3 \cdot 2^{24} + a_2 \cdot 2^{16} + a_1 \cdot 2^8 + a_0$ and

$B = b_7 \cdot 2^{56} + b_6 \cdot 2^{48} + b_5 \cdot 2^{40} + b_4 \cdot 2^{32} + b_3 \cdot 2^{24} + b_2 \cdot 2^{16} + b_1 \cdot 2^8 + b_0$.
$A \cdot B = (a_7 \cdot 2^{56} + a_6 \cdot 2^{48} + a_5 \cdot 2^{40} + a_4 \cdot 2^{32} + a_3 \cdot 2^{24} + a_2 \cdot 2^{16} + a_1 \cdot 2^8 + a_0) \cdot (b_7 \cdot 2^{56} + b_6 \cdot 2^{48} + b_5 \cdot 2^{40} + b_4 \cdot 2^{32} + b_3 \cdot 2^{24} + b_2 \cdot 2^{16} + b_1 \cdot 2^8 + b_0) = a_7 b_7 \cdot 2^{112} + (a_7 b_6 + a_6 b_7) \cdot 2^{104} + (a_7 b_5 + a_6 b_6 + a_5 b_7) \cdot 2^{96} + (a_7 b_4 + a_6 b_5 + a_5 b_6 + a_4 b_7) \cdot 2^{88} + (a_7 b_3 + a_6 b_4 + a_5 b_5 + a_4 b_6 + a_3 b_7) \cdot 2^{80} + (a_7 b_2 + a_6 b_3 + a_5 b_4 + a_4 b_5 + a_3 b_6 + a_2 b_7) \cdot 2^{72} + (a_7 b_1 + a_6 b_2 + a_5 b_3 + a_4 b_4 + a_3 b_5 + a_2 b_6 + a_1 b_7) \cdot 2^{64} + (a_7 b_0 + a_6 b_1 + a_5 b_2 + a_4 b_3 + a_3 b_4 + a_2 b_5 + a_1 b_6 + a_0 b_7) \cdot 2^{56} + (a_6 b_0 + a_5 b_1 + a_4 b_2 + a_3 b_3 + a_2 b_4 + a_1 b_5 + a_0 b_6) \cdot 2^{48} + (a_5 b_0 + a_4 b_1 + a_3 b_2 + a_2 b_3 + a_1 b_4 + a_0 b_5) \cdot 2^{40} + (a_4 b_0 + a_3 b_1 + a_2 b_2 + a_1 b_3 + a_0 b_4) \cdot 2^{32} + (a_3 b_0 + a_2 b_1 + a_1 b_2 + a_0 b_3) \cdot 2^{24} + (a_2 b_0 + a_1 b_1 + a_0 b_2) \cdot 2^{16} + (a_1 b_0 + a_0 b_1) \cdot 2^8 + a_0 b_0$.

Bits	127 ... 120	119 ... 112	111 ... 104	103 ... 96	95 ... 88	87 ... 80	79 ... 72	71 ... 64	63 ... 56	55 ... 48	47 ... 40	39 ... 32	31 ... 24	23 ... 16	15 ... 8	7 ... 0
Factor a									a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
Factor b									b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Step1	a_7b_7		a_7b_5		a_7b_3		a_7b_1		a_5b_0		a_4b_0		a_2b_0		a_0b_0	
+Step2		a_7b_6		a_7b_4		a_7b_2		a_7b_0		a_5b_0		a_3b_0		a_1b_0		
+Step3		a_6b_7		a_6b_5		a_6b_3		a_6b_1		a_4b_1		a_2b_1		a_0b_1		
+Step4			a_6b_6		a_6b_4		a_6b_2		a_4b_2		a_2b_2		a_0b_2			
+Step5			a_5b_7		a_5b_5		a_5b_3		a_5b_1		a_3b_1		a_1b_1			
+Step5				a_5b_6		a_5b_4		a_5b_2		a_3b_2		a_3b_0				
+Step7				a_4b_7		a_4b_5		a_4b_3		a_2b_3		a_0b_3				
+Step8					a_4b_6		a_4b_4		a_2b_4		a_0b_4					
+Step9					a_3b_7		a_3b_5		a_3b_3		a_1b_3					
+Step10						a_3b_6		a_3b_4		a_1b_4						
+Step11						a_2b_7		a_2b_5		a_0b_5						
+Step12							a_2b_6		a_0b_6							
+Step13							a_1b_7		a_1b_5							
+Step14								a_0b_7								
+Step15								a_1b_6								
RESULT																

Fig.7. Squaring as a multiplication of a 64-bit integer.

Now consider separately squaring of an integer

$(a_7 \cdot 2^{56} + a_6 \cdot 2^{48} + a_5 \cdot 2^{40} + a_4 \cdot 2^{32} + a_3 \cdot 2^{24} + a_2 \cdot 2^{16} + a_1 \cdot 2^8 + a_0)^2 = \boldsymbol{a_7^2} \cdot 2^{112} + \boldsymbol{a_6^2} \cdot 2^{96} + \boldsymbol{a_5^2} \cdot 2^{80} + \boldsymbol{a_4^2} \cdot 2^{64} + \boldsymbol{a_3^2} \cdot 2^{48} + \boldsymbol{a_2^2} \cdot 2^{32} + \boldsymbol{a_1^2} \cdot 2^{16} + \boldsymbol{a_0^2} + a_7 a_6 \cdot 2^{105} + a_7 a_5 \cdot 2^{97} + (a_7 a_4 + a_6 a_5) \cdot 2^{89} + (a_7 a_3 + a_6 a_4) \cdot 2^{81} + (a_7 a_2 + a_6 a_3 + a_5 a_4) \cdot 2^{73} + (a_7 a_1 + a_6 a_2 + a_5 a_3) \cdot 2^{65} + (a_7 a_0 + a_6 a_1 + a_5 a_2 + a_4 a_3) \cdot 2^{57} + (a_6 a_0 + a_5 a_1 + a_4 a_2) \cdot 2^{49} + (a_5 a_0 + a_4 a_1 + a_3 a_2) \cdot 2^{41} + (a_4 a_0 + a_3 a_1) \cdot 2^{33} + (a_3 a_0 + a_2 a_1) \cdot 2^{25} + a_2 a_0 \cdot 2^{17} + a_1 a_0 \cdot 2^9$,

which allows you to square an integer in 8 steps instead of 15 when multiplying by itself.

Bits	F.over	127	.....	120	119	.....	112	111	.....	104	103	.....	96	95	.....	88	87	.....	80	79	.....	72	71	.....	54	53	.....	56	55	.....	48	47	.....	40	39	.....	32	31	.....	24	23	.....	15	15	.....	8	7	.....	0
Factor																										a_7			a_6			a_5			a_4			a_3			a_2			a_1			a_0		
Step1		a_7a_7						a_6a_6						a_5a_5						a_4a_4						a_3a_3						a_2a_2						a_1a_1						a_0a_0					
+Step2				a_7a_6						a_7a_4						a_7a_2						a_7a_0						a_5a_0						a_3a_0						a_1a_0									
+Step3							a_7a_5						a_7a_3						a_7a_1						a_5a_1						a_3a_1						a_2a_0											
+Step4										a_7a_4						a_5a_4						a_5a_2						a_3a_2						a_2a_1															
+Step5													a_6a_4						a_6a_2						a_4a_2						a_4a_0																		
+Step5																a_6a_3						a_4a_3						a_4a_1																					
+Step7																			a_5a_3						a_6a_0																								
+Step8																						a_6a_1																											
RESULT																																																	

Fig.8. Squaring of a 64-bit integer.

You can also increase productivity by organizing parallel calculations, for example, when using 2 adders, this operation can be performed in 4 steps, however, a further increase in the number of adders does not give a performance gain.

	Adder1	Adder2	Adder1	Adder2	Adder1	Adder2	Adder1
+Step1	Step1’		Step2’		Step3’		Step4’
+Step2							
+Step3		Step1”					
+Step4							
+Step5				Step2’’			
+Step6							
+Step7						Step3’’	
+Step8							

Fig.9. Parallelizing of the operation

Thus, if you use a separate micro-operation using the separate algorithm described in this article for squaring, you can achieve an additional increase in productivity.