0.0.1 Последовательные программы умножения многочленов и их синтез

Изложенная в предыдущем разделе идея схемной реализации метода Карацубы может быть осуществлена и в виде последовательной (не содержащей циклов и операторов рекурсии) программы. В данном разделе рассмотрены два способа построения последовательных программ – аналитический, имитирующий строение декомпозиционной схемы, и способ, основанный на использовании рекурсивной программы умножения. Функциональность программы, построенной по первому способу очевидна. Программа, построенная по второму способу экономна по объему требуемой памяти. Описывается метод сравнения программ, полученных двумя разными способами для доказательства их функциональной эквивалентности [**?**].

Аналитический способ построения последовательных программ умножения. Многочлены $A(X)$ и $B(X)$ над полем $GF(p)$ будем рассматривать как многочлены $A'(Y)$ и $B'(Y)$, $Y = X^s$, $s = 2^k, k > 1$, над полем $GF(p^s)$, полагая, что элементарные многочлены, являющиеся коэффициентами при таком представлении, могут перемножаться как базовые элементы с использованием некоторой программной функции. Пусть

$$A(Y) = A_0(Y) + A_1(Y)Y^n,\ B(Y) = B_0(Y) + B_1(Y)Y^n.$$

По формуле Карацубы [**?**] произведение многочленов вычисляется следующим образом

$$A(Y)B(Y) = A_0(Y)B_0(Y)(1 + Y^n)+$$
$$+(A_1(Y) - A_0(Y))(B_0(Y) - B_1(Y))Y^n+$$
$$+A_0(Y)B_0(Y)(y_n + Y^{2n}).$$

Таблица 1: Схема перемножения многочленов степени менее двух

$[a;b]_4$	$=$	$[a;b]_2$	$[a+1;b+1]_2$	$[a.a+1;b,b+1]_2$
		$c\ c+1$	$c+1\ c+2$	$c+1$

Полагая, что многочлены размещаются в памяти как последовательности базовых слов, размер которых соответствует элементарным многочленам – коэффициентам многочленов, при известном адресе коэффициента при одночлене нулевой степени, будем использовать относительные адреса для указания коэффициентов или их последовательностей (относительный адрес коэффициента при одночлене нулевой степени – нулевой). Будем обозначать $[a;b]_{2k}$ произведение многочленов, записанных в последовательностях из k единиц, начинающихся с относительных адресов a и b. При этом произведения $[a;b]_2$ вычисляются встроенной функцией. Тогда программа для вычисления $[a;b]_4$ с размещением результата по некоторому относительному адресу c схематически может быть представлена схемой в табл. 1. В первой строке табл. 1 в квадратных скобках указаны относительные адреса множителей или слагаемых, образующих эти множители, а во второй строке – относительные адреса блоков из двух единиц, куда прибавляются произведения. По данной схеме осуществляется три умножения, и произведения последовательно накапливаются в блоке из четырех единиц с относительным адресом с. Первое и второе произведения прибавляются дважды, по двум адресам каждое, в последнем случае перемножаются две разности многочленов: форма $x.y$ соответствует разности $[y]-[x]$, а форма x,y – разности $[x]-[y]$, где $[x]$ и $[y]$ – многочлены, записанные по относительным адресам x и y.

Этой схеме соответствуют цепочки действий, изменяющих

Таблица 2: Схема перемножения многочленов степени менее двух в относительных адресах

$[0;0]_4$	=	$[0;0]_2$	$[1;1]_2$	$[0.1;0,1]_2$
$a\ b\ c$		0 1	1 2	1

первоначально нулевые состояния блока из четырех единиц с относительным базисным адресом c (в квадратных скобках с индексом k указывается содержание блока из k базовых слов с базисным адресом c) :

$$[c]_2 := [c]_2 + [a;b]_2;$$

$$[c+1]_2 := [c+1]_2 + [a;b]_2 + [a+1;b+1]_2 + [a.a+1;b,b+1]_2;$$

$$[c+2]_2 := [c+2]_2 + [a+1;b+1]_2.$$

При этом отдельные адресуемые единицы изменяются следующим образом:

$$[c]_1 := [a;b]_2^0;$$

$$[c+1]_1 := [a;b]_2^1 + [a;b]_2^0 + [a+1;b+1]_2^0 + [a.a+1;b,b+1]_2^0;$$

$$[c+2]_1 := [a;b]_2^1 + [a+1;b+1]_2^1 + [a.a+1;b,b+1]_2^1 +$$
$$+[a.a+1;b,b+1]_2^0;$$

$$[c+3]_1 := [a.a+1;b,b+1]_2^1.$$

Здесь верхние индексы означают взятие младшей (индекс 0) или старшей (индекс 1) половины слова из двух единиц.

Далее для упрощения записей в подобных таблицах будем указывать только приращения адресов, помещая относительные адреса a и b исходных операндов (сомножителей) и c результата (произведения) в первом столбце во второй строке. Табл. 1 после такого упрощения принимает вид табл. 2. В случае, когда

Таблица 3: Совмещенная схема умножения с прибавлением произведения по двум или более адресам

$[0;0]_4$		$[0;0]_2$	$[1;1]_2$	$[0.1;0,1]_2$
$a\ b\ c\ d$	$=$	$c+0\ c+1$ $d+0\ d+1$	$c+1\ c+2$ $d+1\ d+2$	$c+1$ $d+1$

Таблица 4: Совмещенная схема умножения с прибавлением произведения по двум или более относительным адресам

$[0;0]_4$		$[0;0]_2$	$[1;1]_2$	$[0.1;0,1]_2$
$a\ b\ c\ d$	$=$	0 1	1 2	1

произведение накапливается по двум или более адресам, будем использовать совмещенные схемы (табл. 3), где a и b - относительные адреса сомножителей, c и d - относительные адреса накопления результата.

После очевидного упрощения такие схемы принимают вид табл. 4 (приращения относительно c и d одинаковы).

Схема для умножения разностей двух многочленов дана в табл. 5, 6).

Над схемами рассматриваемого вида можно выполнять операции удвоения, удваивая все встречающееся в схеме числа и индексы (при этом относительные адреса исходных операндов

Таблица 5: Схема перемножения разностей многочленов

$[a.b;c,d]_4$	$=$	$[a.b;c,d]_2$	$[a+1.b+1;$ $c+1,d+1]_2$	$[(a.b).$ $(a+1.b+1);$ $(c,d),$ $(c+1,d+1)]_2$
		$e\ e+1$	$e+1\ e+2$	$e+1$

Таблица 6: Схема перемножения разностей многочленов в относительных адресах

$[0.0;0,0]_4$	$=$	$[0.0;0,0]_2$	$[1.1;1,1]_2$	$[(0.0).(1.1);(0,0),(1,1)]_2$
$a\ b\ c\ d\ e$		0 1	1 2	1

Таблица 7: Удвоение схемы из табл. 2

$[0;0]_8$	$=$	$[0;0]_4$	$[2;2]_4$	$[2,0;0,2]_4$
$a\ b\ c$		0 2	2 4	2

и результата в первом столбце сохраняются). Например, удвоением схемы из табл. 2) получим схему в табл. 7.

Операция детализации схемы заключается в том, что используемые для описания схемы обозначения подсхем, определяющих произведения $[a;b]_n,\ n>2$, заменяются схемами, выражающими их через произведения $[a;b]_{n/2}$. Так, детализируя схему из табл. 7 с учетом табл. 2 – 6, получим схему в табл. 8.

Как видим из табл. 8, произведение $[0;0]_8$ можно получить, выполнив 9 перемножений элементарных многочленов. Удвое-

Таблица 8: Детализация схемы из табл. 7

$[0,0]_8$	$=$	$[0;0]_2$	$[1;1]_2$	$[0.1;0,1]_2$
$a\ b\ c$		0 1 2 3	1 2 3 4	1 3
		$[2;2]_2$	$[3;3]_2$	$[2.3;2,3]_2$
		2 3 4 5	3 4 5 6	3 5
		$[0.2;0,2]_2$	$[1.3;1,3]_2$	$(0.2).(1.3);(0,2),(1,3)$
		2 3	3 43	3

Таблица 9: Удвоение схемы из табл. 8

$[0,0]_{16}$	$=$	$[0;0]_4$	$[2;2]_4$	$[0.2;0,2]_4$
$a\ b\ c$		0 2 4 6	2 4 6 8	2 6
		$[4;4]_4$	$[6;6]_4$	$[4.6;4,6]_4$
		2 6 8 10	6 8 10 12	6 10
		$[0.4;0,4]_4$	$[2.6;2,6]_4$	$(0.4).(2.6);(0,4),(2,6)$
		4 6	6 8	6

Таблица 10: Детализация схемы их табл. 9

$[0,0]_{16}$	$=$	$[0;0]_2$	$[1;1]_2$	$[0.1;0,1]_2$
$a\ b\ c$		0 1 2 3 4 5 6 7	1 2 3 4 5 6 7 8	1 3 5 7
		$[2;2]_2$	$[3;3]_2$	$[2.3;2,3]_2$
		2 3 4 5 6 7 8 9	3 4 5 6 7 8 9 10	3 5 7 9
		$[0.2;0,2]_2$	$[1.3;1,3]_2$	$(0.2).(1.3));(0,2),(1,3))$
		2 3 6 7	3 4 7 8	3 7
		$[4;4]_2$	$[5;5]_2$	$[4.5;4,5]_2$
		4 5 6 7 8 9 10 11	5 6 7 8 9 10 11 12	5 7 9 11
		$[6;6]_2$	$[7;7]_2$	$[6.7;6,7]_2$
		6 7 8 9 10 11 12 13	7 8 9 10 11 12 13 14	7 9 11 13
		$[4.6;4,6]_2$	$[5.7;5,7]_2$	$(4.6).(5.7));(4,6),(5,7)$
		6 7 10 11	7 8 11 12	7 11
		$[0.4;0,4]_2$	$[1.5;1,5]_2$	$(0.4).(1,5);(0,4),(1,5)$
		4 5 6 7	5 6 7 8	5 7
		$[2.6;2,6]_2$	$[3.7;3,7]$	$(2.6).(3.7);(2,6),(3,7)$
		6 7 8 9	7 8 9 10	7 9
		$[(0.4).(2.6);$ $(0,4),(2,6)]_2$	$[(1.5).(3.7);$ $(1,5),(3,7)]_2$	$[(0.4).(2.6)).((1.5).(3.7));$ $(0,4),(2,6)),((1,5),(3,7))]_2$
		6 7	7 8	7

нием схемы из табл. 8(см. табл. 9 и последующей детализацией получим схему для вычисления произведения $[0;0]_{16}$ (табл. 10, включающую 27 умножений элементарных многочленов степени меньше s. Выполняя последовательно операции удвоения и детализации, можно получить схему умножения $[0;0]_{2^m}$ (схему размерности 2^m) при любом m как последовательность бинарных схем (схем размерности два). При этом порядок следования бинарных схем несущественен.

Рекурсивный алгоритм синтеза последовательной программы умножения многочленов. Альтернативный способ построения последовательной программы умножения состоит в использовании рекурсивной реализации метода Карацубы. Именно в коде работающего рекурсивного метода умножения каждая операция выполнения элементарного действия заменяется операцией сохранения наименования действия и адресов операндов в файл. Затем ищется минимальный адрес операнда и принимается за нулевой адрес рабочего массива последовательной программы, соответствующая поправка вносится во все адреса операндов. Тем самым выполняется отображение массива данных исходных и промежуточных многочленов на рабочий массив программы. При этом программа строится так, что исходные перемножаемые полиномы расположены последовательно и занимают $2n$ базовых слов, начиная с нулевого адреса рабочего массива последовательной программы. А после выполнения последовательной программы первые $2n$ базовых слов занимает результирующий многочлен. Сложность заключается в том, что для обеспечения линейности отображения адресов операндов на рабочий массив последовательной программы необходимо чтобы все участвующие в вычислениях переменные принадлежали одной странице оперативной памяти компьютера. Эта задача решается заменой стандартных опера-

ций выделения памяти пользовательскими, которые учитывают данную особенность. Такой подход позволяет применять процедуру построения последовательной программы вообще к любому методу умножения, не зависящему от умножаемых многочленов. Однако при этом в данном подходе используются неочевидные преобразования и поэтому после реализации необходимо обосновывать эквивалентность последовательных программ, построенных по этому методу и по методу, изложенному выше.

Контролируемый автоматический синтез последовательных программ умножения. Последовательные программы умножения, синтезированные описанными двумя способами, принадлежат одному классу, при этом отличаются только способами адресации элементов и методами распределения памяти для хранения промежуточных результатов. Значит, в случае эквивалентности, оба алгоритма будут порождать одинаковые цепочки действий по формированию базовых слов результата умножения. Верно и обратное: в случае равенства цепочек действий алгоритмы эквивалентны. Рассмотрим пример формирования цепочек действий для двух итераций метода Карацубы в случае умножения бинарных многочленов. Последовательная программа, получаемая рекурсивным методом, имеет вид:

Прямой ход, уровень рекурсии:1

$$
\begin{array}{l}
equ(13,1,2),\\
add(13,3,2),\\
equ(15,7,2),\\
add(15,5,2),\\
exchg(3,5,2).
\end{array}
$$

Прямой ход, уровень рекурсии:2

$$
\begin{array}{l}
equ(9,1,1),\\
add(9,2,1),\\
equ(10,4,1),\\
add(10,3,1),\\
exchg(2,3,1),\\
equ(11,5,1),\\
add(11,6,1),\\
equ(12,8,1),\\
add(12,7,1),\\
exchg(6,7,1),\\
equ(17,13,1),\\
add(17,14,1),\\
equ(18,16,1),\\
add(18,15,1),\\
exchg(14,15,1).
\end{array}
$$

Умножение:

$$
\begin{array}{l}
Mul(1,2,1),\\
Mul(3,4,1),\\
Mul(9,10,1),\\
Mul(5,6,1),\\
Mul(7,8,1),\\
Mul(11,12,1),\\
Mul(13,14,1),\\
Mul(15,16,1),\\
Mul(17,18,1).
\end{array}
$$

Обратный ход, уровень рекурсии:2

$$add(9,1,2),$$
$$add(9,3,2),$$
$$add(2,9,2),$$
$$add(11,5,2),$$
$$add(11,7,2),$$
$$add(6,11,2),$$
$$add(17,13,2),$$
$$add(17,15,2),$$
$$add(14,17,2).$$

Обратный ход, уровень рекурсии:1

$$add(13,1,4),$$
$$add(13,5,4),$$
$$add(3,13,4).$$

В программе использованы следующие элементарные операции: equ(P,Q,N) - операция присваивания,
add(P,Q,N) - операция сложения,
exchg(P,Q,N) - операция перемещения (элементы с адресами P и Q меняются местами),
Mul(P,Q,N) - операция умножения.
При этом параметры P и Q определяют адреса операндов, N – количество последовательно расположенных слов, над которыми выполняется операция. Результат выполнения каждой операции записывается, начиная с адреса P.

Соответственно и действия последовательной программа могут быть разделены на 3 этапа:

Прямой ход рекурсии.

$$
\begin{aligned}
C_1 &= A[0] + A[1],\\
C_2 &= B[1] + B[0],\\
C_3 &= A[2] + A[3],\\
C_4 &= B[3] + B[2],\\
C_5 &= A[0] + A[2],\\
C_6 &= B[2] + B[0],\\
C_7 &= A[1] + A[3],\\
C_8 &= B[3] + B[1],\\
C_9 &= A[0] + A[2] + A[1] + A[3],\\
C_{10} &= B[3] + B[1] + B[2] + B[0],
\end{aligned}
$$

где $C_i,\ i = 1, \ldots, 10$ – ячейки памяти с адресами выше 8.

Этап умножений. На этом этапе производится 9 умножений, при этом вычисляются слова D_1 – D_{18}, т.к. каждый ре-

зультат умножения занимает два базовых слова:

$$
\begin{aligned}
D_1 &= (A[0]) * (B[0])[0,] \\
D_2 &= (A[0]) * (B[0])[1], \\
D_3 &= (A[1]) * (B[1])[0], \\
D_4 &= (A[1]) * (B[1])[1], \\
D_5 &= (A[2]) * (B[2])[0], \\
D_6 &= (A[2]) * (B[2])[1], \\
D_7 &= (A[3]) * (B[3])[0], \\
D_8 &= (A[3]) * (B[3])[1], \\
D_9 &= C1 * C2[0], \\
D_{10} &= C1 * C2[1], \\
D_{11} &= C3 * C4[0], \\
D_{12} &= C3 * C4[1], \\
D_{13} &= C5 * C6[0], \\
D_{14} &= C5 * C6[1], \\
D_{15} &= C7 * C8[0], \\
D_{16} &= C7 * C8[1], \\
D_{17} &= C9 * C10[0], \\
D_{18} &= C9 * C10[1].
\end{aligned}
$$

При этом, поскольку операнды каждой операции умножения записаны последовательно, то для экономии памяти результат умножения записывается на место операндов. Т.е. D_1 – D_{18} – ячейки памяти с адресами, начиная с 1-го.

Обратный ход рекурсии. На этом этапе завершается вычисление произведения исходных многочленов. Формально эти дей-

ствия представимы в виде:

$$
\begin{aligned}
E_1 &= D_1,\\
E_2 &= D_1 + D_2 + D_3 + D_9,\\
E_3 &= D_1 + D_2 + D_3 + D_4 + D_5 + D_{10} + D_{13},\\
E_4 &= D_1 + D_2 + D_3 + D_4 + D_5 + D_6 + D_7 + D_9 + D_{11} +\\
&\quad + D_{13} + D)14 + D_{15} + D_{17},\\
E_5 &= D_2 + D_3 + D_4 + D_5 + D_6 + D_7 + D_8 + D_{10} + D_{12} +\\
&\quad + D_{13} + D_{15} + D_{16} + D_{18},\\
E_6 &= D_4 + D_5 + D_6 + D_7 + D_8 + D_{11} + D_{16},\\
E_7 &= D_6 + D_7 + D_8 + D_{12},\\
E_8 &= D_8.
\end{aligned}
$$

Здесь E_1 – E_8 – ячейки памяти с адресами, начиная с 1-го.

Далее, производя замену введенных обозначений C_i и D_i на формальное содержимое, получим цепочки действий по вычислению каждого слова результата умножения:

$$
\begin{aligned}
E[1] &= (A[0]) * (B[0])[0],\\
E[2] &= (A[0]) * (B[0])[1] + (A[0] + A[1]) * (B[1] + B[0])[0] +\\
&\quad + (A[0]) * (B[0])[0] + (A[1]) * (B[1])[0],\\
E[3] &= (A[1]) * (B[1])[0] + (A[0] + A[1]) * (B[1] + B[0])[1] +\\
&\quad + (A[0]) * (B[0])[1] + (A[1]) * (B[1])[1] +\\
&\quad + (A[0] + A[2]) * (B[2] + B[0])[0] +\\
&\quad + (A[0]) * (B[0])[0] + (A[2]) * (B[2])[0],\\
E[4] &= (A[1]) * (B[1])[1] + (A[0] + A[2]) * (B[2] + B[0])[1] +\\
&\quad + (A[0] + A[2] + A[1] + A[3]) *\\
&\quad * (B[3] + B[1] + B[2] + B[0])[0] +\\
&\quad + (A[0] + A[2]) * (B[2] + B[0])[0] + (A[1] + A[3]) *\\
&\quad * (B[3] + B[1])[0] + (A[0]) * (B[0])[1] +\\
&\quad + (A[0] + A[1]) * (B[1] + B[0])[0] + (A[0]) *\\
&\quad * (B[0])[0] + (A[1]) * (B[1])[0] + (A[2]) * (B[2])[1] +\\
&\quad + (A[2] + A[3]) * (B[3] + B[2])[0] +\\
&\quad + (A[2]) * (B[2])[0] + (A[3]) * (B[3])[0],
\end{aligned}
$$

$$
\begin{aligned}
E[5] = {} & (A[2]) * (B[2])[0] + (A[1] + A[3]) * (B[3] + B[1])[0] + \\
& + (A[0] + A[2] + A[1] + A[3]) * \\
& * (B[3] + B[1] + B[2] + B[0])[1] + \\
& (A[0] + A[2]) * (B[2] + B[0])[1] + \\
& + (A[1] + A[3]) * (B[3] + B[1])[1] + \\
& + (A[1]) * (B[1])[0] + \\
& + (A[0] + A[1]) * (B[1] + B[0])[1] + \\
& + (A[0]) * (B[0])[1] + (A[1]) * (B[1])[1] + \\
& + (A[3]) * (B[3])[0] + (A[2] + A[3]) * \\
& * (B[3] + B[2])[1] + (A[2]) * (B[2])[1] + \\
& + (A[3]) * (B[3])[1], \\
E[6] = {} & (A[2]) * (B[2])[1] + (A[2] + A[3]) * (B[3] + B[2])[0] + \\
& + (A[2]) * (B[2])[0] + (A[3]) * (B[3])[0] + \\
& + (A[1] + A[3]) * (B[3] + B[1])[1] + \\
& + (A[1]) * (B[1])[1] + (A[3]) * (B[3])[1], \\
E[7] = {} & (A[3]) * (B[3])[0] + (A[2] + A[3]) * (B[3] + B[2])[1] + \\
& + (A[2]) * (B[2])[1] + (A[3]) * (B[3])[1], \\
E[8] = {} & (A[3]) * (B[3])[1].
\end{aligned}
$$

Аналогичные цепочки действий получаются и для программы, сформированной описанным выше аналитическим методом. Соответствующие цепочки для этой программы имеют вид:

Метод автоматического контролируемого синтеза программ умножения многочленов заключается в том, что при заданных максимально возможных степенях $2^n - 1$ сомножителей автоматически синтезируются две последовательных программы умножения двумя рассмотренными методами. Затем из описаний первой и второй программы извлекаются цепочки действий по формированию базовых слов результата умножения. Формальными преобразованиями цепочки автоматическое приводятся к единому виду. Равенство полученных цепочек свидетельствует о корректности обеих последовательных программ.

Данная проверка отменяет необходимость тестирования программ при конкретных значениях исходных данных.