УДК 519.688
DOI: 10.24160/1993-6982-2018-6-96-102

Об использовании техники отложенного переноса в арифметике длинных целых чисел

М.Е. Куляс

Описаны варианты использования техники отложенного переноса (ТОП) для ускорения вычисления произведения длинных целых чисел. Применение ТОП позволяет сократить накладные расходы, связанные с необходимостью обработки межразрядных переносов, возникающих в процессе вычислений.
Проведено исследование эффективности данной техники для модифицированного метода сдвигов и сложений и метода Карацубы. Разработаны методы программной реализации алгоритмов умножения, обладающие высокой эффективностью и переносимостью на различные вычислительные средства. Получены теоретические оценки числа инструкций умножения, сложения и доступа к памяти, необходимых для программной реализации алгоритмов. Выполнена экспериментальная оценка эффективности методов с ТОП по сравнению с базовыми методами сдвигов и сложений и методом Карацубы. Эксперименты показали высокую эффективность алгоритмов с ТОП, в том числе и в сравнении с уже существующими решениями на базе программной библиотеки GMP. Применение техники отложенного переноса позволило увеличить производительность вычислений по модифицированному методу сдвигов и сложений (для 1024-битных чисел) на 18%, а по методу Карацубы на 31% (по сравнению с линейной реализацией модифицированного метода сдвигов и сложений).
Рассмотренные алгоритмы умножения длинных целых чисел с применением ТОП могут быть эффективно использованы при создании переносимых (без ассемблерных вставок) программных библиотек компьютерной алгебры.

Ключевые слова: умножение длинных чисел, алгоритм Карацубы, метод сдвигов и сложений, отложенный перенос.

Для цитирования: Куляс М.Е. Об использовании техники отложенного переноса в арифметике длинных целых чисел // Вестник МЭИ. 2018. № 6. С. 96—102. DOI: 10.24160/1993-6982-2018-6-96-102.

On Using the Carry-Save Technique in the Arithmetic of Long Integer Numbers

M.E. Kulyas

Possible versions of using the carry-save technique (CST) for speeding up multiplication of long integer numbers are described. Application of the CST makes it possible to reduce the overhead computing associated with performing intermediate carryover operations the necessity of which arises in the course of computations.
The efficiency of the technique for the modified shift and add method and for the Karatsuba method was investigated. Methods for implementing the multiplication algorithms by means of software featuring high efficiency and portability to various computing platforms have been developed. Theoretical estimates of the number of multiplication, addition, and memory access instructions needed for implementing the algorithms by means of software are obtained. The efficiency of the methods utilizing the CST in comparison with the basic shift and add methods, and the Karatsuba method is experimentally evaluated. The experiments showed high efficiency of the algorithms utilizing the CST including comparison with the existing solutions based on the GMP software library. Application of the CST made it possible to increase the performance of computing by 18% for the modified shift and add method (for 1024-bit numbers) and by 31% for the Karatsuba method (in comparison with the linear implementation of the modified shift and add method).
The considered algorithms for multiplying long integer numbers with application of the CST can be effectively used in elaborating portable (without assembler-based insertions) computer algebra software libraries.
It is concluded that the optimal choice of a library depends essentially on both the particular application and on the used computing architecture.

Key words: multiplication of long integer numbers, Karatsuba algorithm, shift and add method, carry-save technique.

For citation: Kulyas M.E. On Using the Carry-Save Technique in the Arithmetic of Long Integer Numbers. MPEI Vestnik. 2018;6: 96—102. (in Russian). DOI: 10.24160/1993-6982-2018-6-96-102.

Введение

Операция умножения длинных целых чисел является составной частью более сложных алгоритмов современной компьютерной алгебры. Эффективность выполнения данной операции существенно влияет на производительность алгоритмов, основанных на вычислениях в конечных алгебраических структурах, широко применяемых в области защиты информации [1, 2]. При использовании позиционной системы счисления для представления длинных чисел возникают расходы, связанные с необходимостью обработки возникающих в процессе вычислений межразрядных переносов. Это сказывается на производительности вычислений. Один из путей ускорения операции умножения — использование техники отложенного переноса, позволяющего снизить накладные расходы на обработку межразрядных переносов. Возникает задача разработки эффективного и переносимого программ-

ного кода, реализующего операцию умножения длинных целых чисел.

Пусть длинным n-битным целым числом X будет число, записанное в позиционной системе счисления по некоторому основанию $b = 2^w$ и имеющее разрядность n, превышающую разрядность машинного слова процессора w. Длинные числа удобно хранить и обрабатывать в виде массивов целых беззнаковых чисел из s элементов, где $s = \lceil n/w \rceil$.

$$X = \left(X[s-1]X[s-2]\ldots X[0]\right)_b = \sum_{i=0}^{s-1} X[i]b^i.$$

Диапазон возможных значений каждого элемента $X[i]$ от 0 до 2^w–1. Диапазон возможных значений произведения двух элементов — от 0 до 2^{2w}–2^{w+1}+1, т. е. для представления произведения двух элементов используются два машинных слова. Знаки чисел можно хранить и обрабатывать отдельно.

Длина чисел определяется областью применения конкретных алгоритмов работы с ними. Например, в области защиты информации используется модульная арифметика над числами с разрядностью, не превышающей нескольких тысяч бит. Для данного диапазона наиболее эффективными алгоритмами вычисления произведения целых чисел являются методы сдвигов и сложений и Карацубы [3]. Ограничимся их описанием и рассмотрим возможные пути повышения эффективности вычислений по ним.

Метод сдвигов и сложений

По методу сдвигов и сложений [2] произведение двух длинных чисел X и Y (в нашем случае одинаковой разрядности) вычисляется по формуле:

$$XY = \sum_{i=0}^{s-1} X[i]b^i \sum_{j=0}^{s-1} Y[j]b^j = \sum_{i=0}^{s-1}\sum_{j=0}^{s-1} X[i]Y[j]b^{i+j}.$$

Здесь вычисления идут «по строкам», т. е. для каждого i фиксируется элемент $X[i]$ и умножается (со сдвигом) на все $Y[j]$ с корректной обработкой возникающих при этом межразрядных переносов. Графическая интерпретация метода для $s = 3$ представлена на рис. 1, *а*.

Основу программной реализации метода составляют два вложенных цикла. Во внешнем загружается значение текущего разряда первого операнда, а во внутреннем выполняется операция умножения (с накоплением) текущего разряда первого операнда на все разряды второго операнда. Результаты промежуточных вычислений записываются в массив P на каждой итерации внутреннего цикла.

Существует модификация данного метода, в которой вычисления выполняются «по столбцам» [4]:

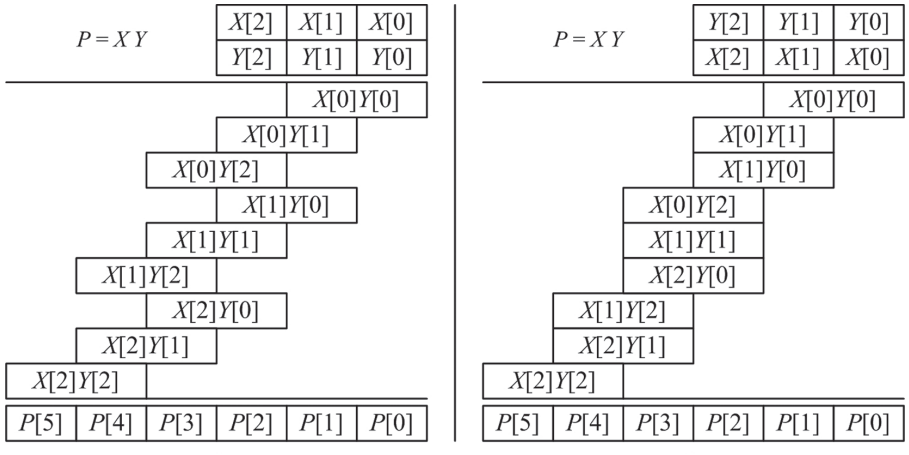
$$XY = \sum_{i=0}^{s-1} X[i]b^i \sum_{j=0}^{s-1} Y[j]b^j = \sum_{i=0}^{s-1} b^i \sum_{j+k=i;\, j,k\geq 0} X[j]Y[k]. \quad (1)$$

В модифицированном методе сдвигов и сложений также имеется два вложенных цикла. Во внешнем присваивается вычисленное значение текущего разряда $P[i]$, а во внутреннем вычисляется само значения с учетом переноса от предыдущего вычисленного разряда. Операция присваивания значения $P[i]$ вынесена во внешний цикл (по сравнению с предыдущим методом), что позволяет сократить число инструкций обращения к памяти. Графическая интерпретация метода дана на рис. 1, *б*. Асимптотическая сложность обоих методов оценивается как $O(s^2)$ [2].

Метод Карацубы

В основу метода Карацубы заложен принцип декомпозиции (рекурсивного снижения сложности задачи путем ее разбиения на несколько более простых — принцип «разделяй и властвуй») [3].

Для определенности положим, что n (число бит, необходимых для хранения операндов) является степенью двойки, тогда любое n-битное целое число X можно представить по основанию $2^{n/2}$:



а *б*

Рис. 1. Метод сдвигов и сложений (*а*) и его модификация (*б*) для $s = 3$

$$X = X_0 + X_1 2^{n/2}.$$

По формуле Карацубы произведение двух таких чисел сводится к трем операциям умножения $n/2$-битных чисел и нескольким дополнительным операциям сложения/вычитания, имеющим линейную сложность:

$$XY = X_0 Y_0 (1 + 2^{(n/2)}) + \\ +(X_1 - X_0)(Y_0 - Y_1)2^{(n/2)} + X_1 Y_1 (2^n + 2^{(n/2)}).$$

Определенную сложность представляет необходимость учета знака среднего компонента. Существует аддитивный вариант формулы Карацубы, в котором знаки среднего члена всегда определены [5]:

$$XY = X_0 Y_0 \left(1 - 2^{n/2}\right) + \\ +\left(X_1 + X_0\right)\left(Y_0 + Y_1\right)2^{n/2} + X_1 Y_1 \left(2^n - 2^{n/2}\right).$$

Данные формулы можно применять рекурсивно, снижая, таким образом, разрядность операндов до некоторой величины n_0, называемой порогом рекурсии. По его достижению умножение n_0-битных чисел может выполняться по методу сдвигов и сложений или, используя непосредственно инструкцию умножения (если n_0 совпадает с размером машинного слова процессора). Асимптотическая сложность такого метода оценивается как $O(n_0^2 3^{\log(n/n_0)})$ [2].

При рекурсивной реализации метода необходимо минимизировать число операций выделения памяти при прямом ходе, поскольку это сильно снижает производительность вычислений [6]. Таким образом, требуется разработка эффективной схемы размещения операндов и результатов промежуточных вычислений в памяти. Декомпозиционную схему при такой реализации можно строить в автоматическом режиме. Для этого записывают последовательности базовых операций (сложения, умножения, вычитания) в процессе прямого и обратного проходов по уровням рекурсии для каждого разряда произведения (в соответствии с разработанной схемой размещения операндов в памяти). Основной недостаток данного метода в том, что практически не задействуются средства автоматической оптимизации памяти, встроенные в современные компиляторы.

Альтернативным подходом является использование аналитического метода построения декомпозиционной схемы вычислений. В публикации [7] получены формулы для вычисления произведения полиномов произвольной степени по методу Карацубы, в [8] приводится их обобщение для целых чисел.

$$XY = \sum_{i=1}^{s-1}\sum_{j=0}^{i-1}\left(X[i] - X[j]\right)\left(Y[j] - Y[i]\right)b^{i+j} + \\ +\sum_{i=0}^{s-1} b^i \sum_{j=0}^{s-1} X[j]Y[j]b^j. \qquad (2)$$

Аддитивный вариант:

$$XY = \sum_{i=1}^{s-1}\sum_{j=0}^{i-1}\left(X[i] + X[j]\right)\left(Y[j] + Y[i]\right)b^{i+j} + \\ +2\sum_{i=0}^{s-1} X[i]Y[i]b^{2i} - \sum_{i=0}^{s-1} b^i \sum_{j=0}^{s-1} X[j]Y[j]b^j. \qquad (3)$$

Программа для вычисления произведения двух длинных чисел по (2), (3) содержит последовательность операций сложения, вычитания и умножения отдельных разрядов входных операндов. При этом задача оптимизации памяти полностью ложится на компилятор. Данный вариант предпочтительнее, поскольку современные компиляторы отлично справляются с задачами оптимизации памяти и исполняемого кода. Существенным недостатком является трудность определения необходимого размера выделяемой памяти для хранения результатов промежуточных вычислений.

Техника отложенного переноса

При создании эффективного и одновременно переносимого программного кода можно воспользоваться техникой отложенного переноса [9]. Для этого операнды записывают по укороченному основанию $B = 2^W$, где $W < w$, а оставшиеся $(w - W)$ бит используют для накопления и обработки межразрядных переносов, возникающих в процессе вычислений (рис. 2).

Такой формат хранения позволяет эффективно реализовывать цепочки сложений W-битных чисел стандартными средствами языка Си (без задействования дополнительных ассемблерных инструкций и выделения памяти для учета возможных переполнений). Основным недостатком метода считается больший объем памяти, необходимой для хранения операндов и результатов промежуточных вычислений.

Рассмотрим использование данной техники при реализации модифицированного метода сдвигов и сложений.

В процессе вычислений по (2) многократно выполняются операции умножения с накоплением и учетом бита переноса в старшие разряды. В общем случае, для реализации накопления требуются три инструкции сложения. Если же операнды X и Y представить по укороченному основанию $B = 2^W$, где $W < w$, то произведение $X[j]Y[i-j]$ займет $2W$ бит. Оставшиеся $(2w - 2W)$

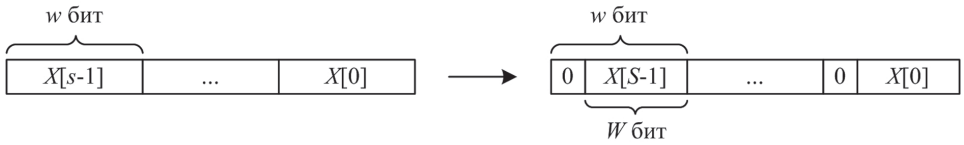


Рис. 2. Формат представления чисел по укороченному основанию

бит можно использовать для накопления и обработки межразрядных переносов. Такое решение требует уже две инструкции сложения на каждой итерации внутреннего цикла, но число инструкций при этом увеличивается и составляет $\left(\lceil n/W \rceil\right)^2$. Эффективность метода повышается при наличии процессорной инструкции сложения 2w-битных слов или векторных инструкций, позволяющих распараллеливать вычисления.

При реализации модифицированного метода сдвигов и сложений с использованием техники отложенного переноса внутренние циклы (пп. 2.1, 3.1 алгоритма 1) можно развернуть и записать в виде цепочки сложений $X[j]Y[i-j]$.

Алгоритм 1. Умножение длинных целых чисел модифицированным методом сдвигов и сложений с использованием техники отложенного переноса.

ВХОД: два массива X и Y с операндами, размером S элементов каждый X.

ВЫХОД: массив P из 2S элементов, содержащий результаты операции умножения: $P = XY = (P[2S-1], \ldots, P[0])$.

1. Установить $C \leftarrow 0$.
2. Для всех i от 0 до $S-1$ выполнить:

$$2.1.\ \left(C, P[i]\right) \leftarrow C + \sum_{j=0}^{i} X[j]Y[i-j].$$

3. Для всех i от S до $2S-2$ выполнить:

$$3.1.\ \left(C, P[i]\right) \leftarrow C + \sum_{j=i-S+1}^{S-1} X[j]Y[i-j].$$

4. Установить $P[2S-1] \leftarrow C$.

Пункты 2.1 и 3.1 алгоритма 1 достаточно просто реализовать стандартными средствами языка Си (без использования ассемблерных вставок), используя вспомогательную 2w-битную переменную с наложением битовой маски.

В процессе вычисления цепочки умножений с накоплением возникает переполнение. Этого можно избежать, если сформулировать критерии выбора значения параметра W.

Максимальная длина цепочки сложений частичных произведений $X[j]Y[i-j]$ достигается при $i = S-1$. Максимальное значение $X[j]Y[i-j]$ не превышает $(2^{2W} - 2^{W+1}+1)$, а C на текущей итерации цикла не превышает $(2^{2w-W}-1)$. Таким образом, для исключения переполнения 2w-битного компонента C, $P[i]$ необходимо выполнение следующего условия:

$$S(2^{2W} - 2^{W+1} + 1) + (2^{2w-W} - 1) < 2^{2w}.$$

Оценим число операций, необходимых для реализации алгоритма 1. В пунктах 2.1 и 3.1 алгоритма S^2 раз выполняется умножение с накоплением и задействуется суммарно $N_{ADD} = 2\cdot(S^2-1)$ инструкций сложения (для $i = 0$ сложение с компонентом C не требуется). Предварительно требуется загрузить значения $X[j]$ и $Y[i-j]$ ($2S^2$ инструкций чтения из памяти). Оценку можно улучшить до $N_{LD} = 2S^2-(2S-2)$, если заметить, что для каждого значения i кроме $i = 0$ загружается только один из сомножителей $X[j]$ или $Y[i-j]$, а значение второго сомножителя уже загружено на предыдущей итерации цикла. Значение $P[i]$ присваивается 2S раз ($N_{ST} = 2S$). Общее число инструкций умножения машинных слов составляет $N_{MUL} = S^2$, и сложность алгоритма оценивается как $O(S^2)$.

Последующим развитием идеи применения техники отложенного переноса может служить синтез алгоритма умножения по методу Карацубы с представлением операндов по укороченному основанию [10]. Декомпозиционные схемы по (2), (3) представляют собой цепочки сложений частичных произведений с учетом возникающих переносов. При этом (2) имеет меньшую сложность.

Введем обозначения $\Delta X_{ij} = (X[i] - X[j])$; $\Delta Y_{ji} = (Y[j] - Y[i])$, тогда (2) можно записать в следующем виде:

$$XY = \sum_{i=1}^{S-1}\sum_{j=0}^{i-1}\left(\Delta X_{ij}\Delta Y_{ji}\right)B^{i+j} + \sum_{i=0}^{S-1}B^i\sum_{j=0}^{S-1}X[j]Y[j]B^j.$$

Компоненты $X[j]Y[j]$ многократно используются в процессе вычислений. Их целесообразно вычислить и сохранить заранее. На рис. 3 представлена графическая интерпретация метода для $S = 3$. Для эффективного использования техники отложенного переноса вычисления следует проводить «по столбцам» (алгоритм 2), аналогично модифицированному методу сдвигов и сложений.

Пунктиром на рис. 3 обозначены значения $X[j]Y[j]$, вычисленные заранее. Их нужно сохранить в массиве T, состоящем из S элементов (2w-битных). Чтобы избежать многократного повторения вычисления $\sum X[j]Y[j]$ для текущего разряда $P[i]$, предлагается использовать соответствующие значения, вычисленные для предыдущего разряда $P[i-1]$. Для хранения $\sum X[j]Y[j]$ используется 2w-битный компонент (U, V).

Алгоритм 2. Умножение длинных целых чисел по методу Карацубы с использованием техники отложенного переноса.

ВХОД: два массива X и Y с операндами размером S элементов каждый.

ВЫХОД: массив P из 2S элементов, содержащий результаты операции умножения: $P = XY = (P[2S-1], \ldots, P[0])$.

1. Для всех j от 0 до $S-1$ выполнить:
   1.1. $T[j] \leftarrow X[j]Y[j]$.
2. Установить $(U, V) \leftarrow T[0]$.
3. Установить $P[0] \leftarrow V$.
4. Установить $C \leftarrow U$.
5. Для всех i от 1 до $S-1$ выполнить:
   5.1. $(U, V) \leftarrow (U, V) + T[i]$.

$$5.2.\ \left(C, P[i]\right) \leftarrow C + (U, V) + \sum_{j=1+i/2}^{i}\left(X[i-j] - X[j]\right)\left(Y[j] - Y[i-j]\right).$$

6. Для всех i от S до $2S-2$ выполнить:
   6.1. $(U, V) \leftarrow (U, V) - T[i-S]$.

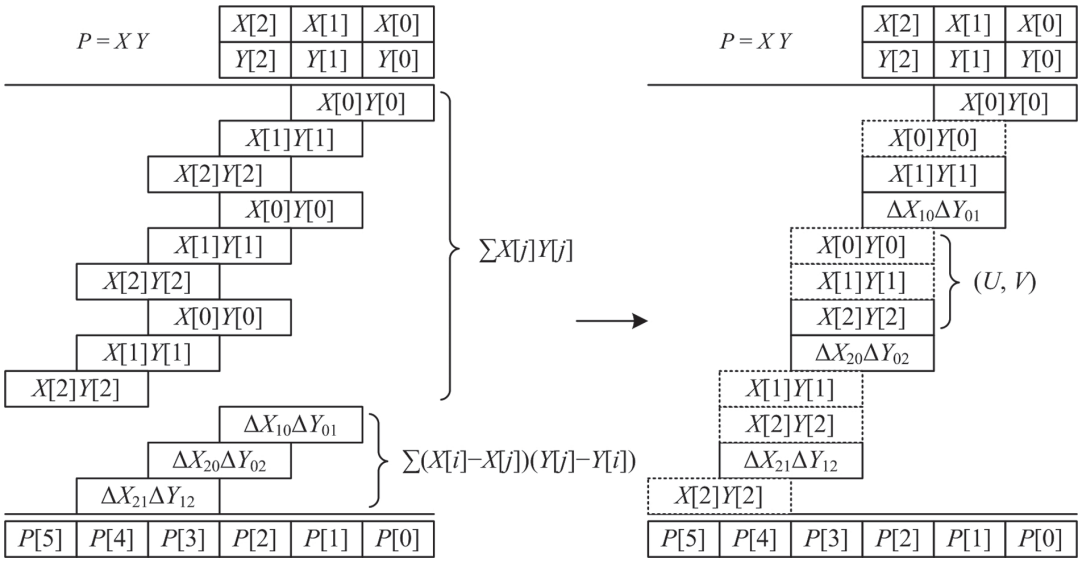


Рис. 3. Метод Карацубы с применением техники отложенного переноса для $S = 3$

6.2. $$\left(C,P[i]\right) \leftarrow C + (U,V) + \sum_{j=1+i/2}^{S-1} \left(X[i-j]-X[j]\right)\left(Y[j]-Y[i-j]\right).$$

7. Установить $P[2S-1] \leftarrow C$.

Оценим число операций, необходимых для реализации алгоритма 2. В пункте 1 алгоритма выполняется S операций умножения и S операций записи в память. Обозначим эту величину как $N_T = S$. В циклах 5 и 6 значения компонента (U, V) накапливаются $2S - 2$ раз ($N_{UV} = 2S - 2$), при этом выполняется $2N_{UV}$ инструкций сложения/вычитания. Кроме того, в пунктах 5.2, 6.2 вычисляется цепочка сложений с учетом переноса ($N_C = 2S - 2$). Обозначим число компонентов $(X[i - j] - X[j])(Y[j] - Y[i - j])$ как N_Δ. Каждый такой компонент входит в цепочку сложений только один раз, и сочетания индексов i и j не повторяются. Таким образом, N_Δ определяется как число сочетаний без повторений из S элементов по 2: $N_\Delta = C_S^2 = (S - 1)S/2$. Число операций записи в массив P определяется величиной $2S$ ($N_P = 2S$). Общее число инструкций сложения $N_{ADD} = 4N_{UV} + 2N_C + 4N_\Delta = 2S^2 + 10S - 12$; записи в память $N_{ST} = N_T + 2S = 3S$; чтения из памяти $N_{LD} = 2N_T + N_{UV} + 4N_\Delta = 2S^2 + 2S - 2$; умножения $N_{MUL} = N_T + N_\Delta = S^2/2 + S/2$.

В процессе вычислений может возникнуть ошибка переполнения $2w$-битного компонента $(C, P[i])$. Сформулируем критерии выбора значения параметра W, позволяющие ее избежать. Как следует из описания алгоритма 2, максимальная длина цепочки сложений частичных произведений $(X[i - j] - X[j])(Y[j] - Y[i - j])$ достигается при $i = S - 1$. Максимальное значение $(X[i - j] - X[j])(Y[j] - Y[i - j])$ не превышает $(2^W - 2^{W+1}+1)$, значение (U, V) при $i = S - 1$ не больше $S(2^W - 1)$, а C на текущей итерации цикла не превышает $(2^{2w-W} - 1)$. Таким образом, для исключения переполнения (с учетом знака) необходимо выполнение следующего условия:

$$(S-1)(2^{2W} - 2^{W+1} + 1)2^{-1} + S(2^{2W} - 1) + (2^{2w-W} - 1) < 2^{2w-1}.$$

Эффективность методов

Теоретический анализ числа операций, необходимых для реализации описанных методов, носит примерный характер. Это обосновано тем, что для конкретной вычислительной системы операции сложения, умножения и доступа к памяти могут выполняться за разное число тактов. Таким образом, для более точного определения эффективности методов умножения требуется их экспериментальное исследование.

На тестовом стенде (Intel(R) Xeon(R) CPU E5-1620 3.70GHz, Linux 4.4 64-bit, gcc 5.4.0–O2–m64) при фиксированных значениях параметров n, w, W проводили измерения среднего числа тактов процессора при выполнении вычислений по рассматриваемым алгоритмам. Измерения выполняли по методике, рекомендованной компанией Intel [11]. Результаты эксперимента представлены на рис. 4.

В процессе эксперимента измеряли эффективность линейной реализации (без циклов и ветвлений) метода сдвигов и сложения и его модификации (при $w = 32$), а также модифицированного метода сдвигов и сложений и метода Карацубы с использованием техники отложенного переноса (ТОП). Алгоритмы реализованы на языке Си без использования ассемблерных вставок.

Для определения эффективности описанных методов по сравнению с известными программными библиотеками длинной арифметики выполняли измерения среднего числа тактов процессора при вычислении функции mpz_mul() библиотеки GMP версии 6.2.2 [12].

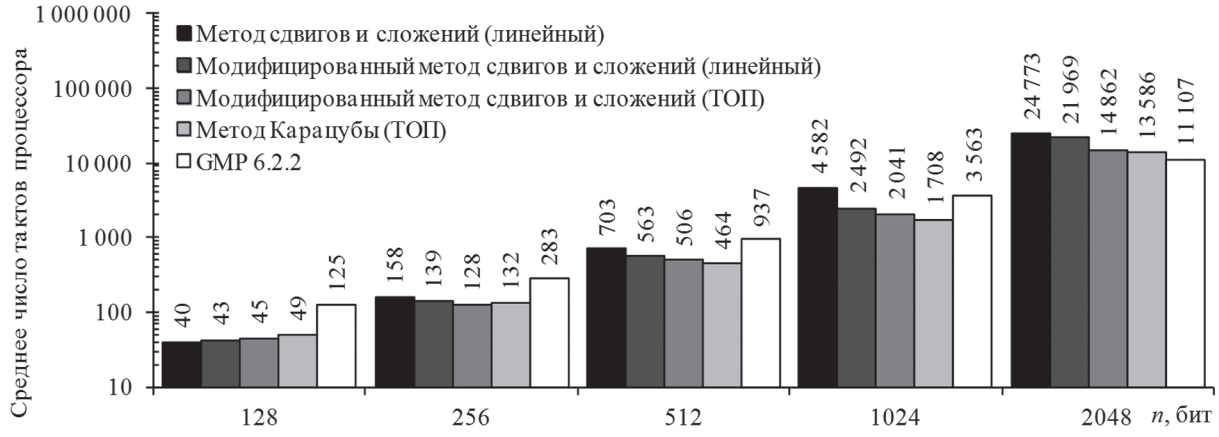


Рис. 4. Эффективность методов умножения для $w = 32$, $W = 28$

Данная функция вычисляет произведение двух длинных целых чисел, используя рекурсивную реализацию методов Карацубы и сдвигов и сложений на нижних уровнях рекурсии. Библиотека GMP скомпилирована для 64-битной ОС Linux (Linux 4.4 64-bit, gcc 5.4.0 – O2 –m64) без использования ассемблерных вставок (при реализации описанных алгоритмов ассемблерные вставки также не использовались). Для этого при сборке использовался ключ --disable-assembly.

Из результатов эксперимента следует, что для небольших чисел ($n \le 256$ бит) методы имеют примерно одинаковую производительность. В диапазоне $256 < n < 2048$ использование техники отложенного переноса дает существенное преимущество в скорости вычислений. Например, для 512-битных чисел применение ТОП для модифицированного метода сдвигов и сложений дает 10% прирост производительности, а для метода Карацубы — 18% (по сравнению с линейной реализацией модифицированного метода сдвигов и сложений). Для 1024-битных чисел эффективность ТОП еще выше — 18 и 31%, соответственно. Кроме того, в диапазоне $n < 2048$ алгоритмы умножения, реализованные с применением ТОП, эффективнее GMP почти в 2 раза.

Заключение

Рассмотрены алгоритмы умножения длинных целых чисел с применением техники отложенного переноса. Получены теоретические оценки числа инструкций умножения, сложения и доступа к памяти, необходимых для реализации алгоритмов. Проведена экспериментальная оценка эффективности методов с ТОП по сравнению с базовыми методами сдвигов и сложений и методом Карацубы. Эксперименты показали высокую эффективность алгоритмов с ТОП, в том числе и в сравнении с уже существующими решениями на базе программной библиотеки GMP. Применение техники отложенного переноса позволило увеличить производительность вычислений по модифицированному методу сдвигов и сложений (для 1024-битных чисел) на 18%, а по методу Карацубы на 31% (по сравнению с линейной реализацией модифицированного метода сдвигов и сложений).

Рассмотренные алгоритмы умножения длинных целых чисел с применением ТОП эффективны при создании программных библиотек компьютерной алгебры.

Литература

1. **Кнут Д.Э.** Искусство программирования. Т. 2. Получисленные алгоритмы. М.: Вильямс, 2007.

2. **Brent R.P., Zimmermann P.** Modern Computer Arithmetic. N.-Y.: Cambridge University Press, 2010.

3. **Карацуба А.А., Офман Ю.П.** Умножение многозначных чисел на автоматах // ДАН СССР. 1962. Т. 145. № 2. С. 293—294.

4. **Comba P.G.** Exponentiation Cryptosystems on the IBM PC // IBM Systems J. 1990. No. 29. Pp. 526—538.

5. **Карацуба А.А.** Сложность вычислений // Труды МИАН. 1995. Т. 211. С. 186—202.

6. **Куляс М.Е.** О синтезе программ умножения длинных целых чисел // Программная инженерия. 2017. № 2. С. 66—75.

References

1. **Knut D.E.** Iskusstvo Programmirovaniya. T. 2. Poluchislennye Algoritmy. M.: Vil'yams, 2007. (in Russian).

2. **Brent R.R., Zimmermann P.** Modern Computer Arithmetic. N.-Y.: Cambridge University Press, 2010.

3. **Karatsuba A.A., Ofman YU.P.** Umnozhenie Mnogoznachnyh Chisel na Avtomatah. DAN SSSR. 1962; 145;2: 293—294. (in Russian).

4. **Comba P.G.** Exponentiation Cryptosystems on the IBM PC. IBM Systems J. 1990;29:526—538.

5. **Karatsuba A.A.** Slozhnost' Vychisleniy. Trudy MIAN. 1995; 211:186—202. (in Russian).

6. **Kulyas M.E.** O Sinteze Programm Umnozheniya Dlinnyh Tselyh Chisel. Programmnaya Inzheneriya. 2017;2: 66—75. (in Russian).

7. **Weimerskirch A., Paar C.** Generalization of the Karatsuba Algorithm for efficient Implementation // Cryptology ePrint Archive. 2006. Rep. 224 [Электрон. ресурс] http://eprint.iacr.org/2006/224 (дата обращения 10.10.2017).

8. **Khachatrian G., Kuregian M., Ispiryan K., Massey J.** Faster Multiplication of Integers for Public-key Applications // Selected Areas in Cryptography. 2001. V. 2259. P. 245—254.

9. **Kovtun V.Yu., Okhrimenko A.O.** Integer Squaring Algorithm with Delayed Carry Mechanism // Безпека інформації. 2013. V 19. No. 3. Pp. 188—192.

10. **Scott M.** Missing a Trick: Karatsuba Variations // Cryptology ePrint Archive. 2015. Rep. 1247 [Электрон. ресурс] http://eprint.iacr.org/2015/1247 (дата обращения 10.10.2017).

11. **Paoloni G.** How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures 2010. [Электрон. ресурс] http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf (дата обращения 12.10.2017).

12. **Granlund T.** The GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library [Электрон. ресурс] http://gmplib.org (дата обращения 12.10.2017).

7. **Weimerskirch A., Paar C.** Generalization of the Karatsuba Algorithm for Efficient Implementation. Cryptology ePrint Archive. 2006; 224 [Elektron. Resurs] http://eprint.iacr.org/2006/224 (Data Obrashcheniya 10.10.2017).

8. **Khachatrian G., Kuregian M., Ispiryan K., Massey J.** Faster Multiplication of Integers for Public-key Applications. Selected Areas in Cryptography. 2001;2259:245—254.

9. **Kovtun V.Yu., Okhrimenko A.O.** Integer Squaring Algorithm with Delayed Carry Mechanism. Bezpeka informatsii. 2013;19;3:188—192.

10. **Scott M.** Missing a Trick: Karatsuba Variations. Cryptology ePrint Archive. 2015;1247 [Elektron. Resurs] http://eprint.iacr.org/2015/1247 (Data Obrashcheniya 10.10.2017).

11. **Paoloni G.** How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures 2010. [Elektron. Resurs] http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf (Data Obrashcheniya 12.10.2017).

12. **Granlund T.** The GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library [Elektron. Resurs] http://gmplib.org (Data Obrashcheniya 12.10.2017).

Сведения об авторе:

Куляс Михаил Евгеньевич — аспирант кафедры математического моделирования НИУ «МЭИ», e-mail: KuliasMY@mpei.ru

Information about author:

Kulyas Mikhail E. — Ph.D.-student of Mathematical Modeling Dept., NRU MPEI, e-mail: KuliasMY@mpei.ru

Работа выполнена при поддержке: РФФИ (проект № 17-01-00485 А)
The work is executed at support: RFBR (grants No. 17-01-00485 A)

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов
Conflict of interests: the authors declare no conflict of interest

Статья поступила в редакцию: *21.11.2017*
The article received to the editor: *21.11.2017*