Компьютерная математика

Оптимизация вычислений

УДК 519.6

А.Н. ТЕРЕЩЕНКО, В.К. ЗАДИРАКА

ОЦЕНКА СЛОЖНОСТИ ОПЕРАЦИИ УМНОЖЕНИЯ МНОГОРАЗРЯДНЫХ ЧИСЕЛ В ПАРАЛЛЕЛЬНОЙ МОДЕЛИ ВЫЧИСЛЕНИЙ

Анализируется сложность по числу векторных однословных операций умножения при реализации многоразрядной операции умножения в параллельной модели вычислений. Предлагается эффективная схема перераспределения вычислений на основе использования циклических сверток меньшей разрядности.

Введение. Одним из ускорений операций многоразрядной арифметики является их реализация в параллельной модели вычислений. Может показаться, что сложность выполнения многоразрядной операции умножения в параллельной модели вычислений по числу однословных операций умножения равна 1, что можно задействовать такое количество параллельных процессоров, сколько однословных операций умножения необходимо выполнить для получения результата. Это возможно только теоретически. Например, нет смысла задействовать 65536 параллельных процессоров при умножении двух 256-разрядных чисел для того, чтобы каждый из них выполнил только одну операцию однословного умножения. На практике, чем больше параллельных процессоров задействуем, тем больше ограничений мы должны учитывать. Если говорить о GPU (Graphics Processing Unit), то одно из первых ограничений – это размер кэш-памяти рабочей группы процессоров. Вычисления на GPU разбиваются на рабочие группы параллельных процессоров. Каждая такая группа имеет свою кэш-память. Кэш-память рабочей группы значительно быстрее локальной или глобальной памяти GPU. Преимущество кэш-памяти в том, что если хотя бы один процессор группы читает данные из локальной или глобальной памяти, то всем остальным процессорам нет необходимости выполнять операцию чтения той же ячейки памяти. Они читают данные уже из кэш-памяти.

Следующее ограничение – это число параллельных процессоров в рабочей группе. Особенность выполнения операций в рабочей группе в том, что нет необходимости в синхронизации всех процессоров группы. Они все выполняют операции синхронно. Поэтому при увеличении числа задействованных процессоров все вычисления необходимо разбивать на несколько рабочих групп, между которыми необходимо выполнять синхронизацию, которая является очень затратной операцией. При дальнейшем увеличении параллельных процессоров возникает вопрос затрат энергии на выполнение одной многоразрядной операции. Каждый задействованный параллельный процессор потребляет энергию. Преимущество GPU в том, что они поддерживают векторные операции. Но векторные операции имеют ограничения по длине 2, 4, 8 и 16.

Зачем необходимо разрабатывать новые алгоритмы в последовательной модели, когда есть GPU? Разработка параллельных алгоритмов невозможна без тщательного изучения этих алгоритмов для одного процессора. Эффективными алгоритмами в параллельной модели являются те, которые используют кэш-память максимально, основаны на векторных операциях и задействуют небольшое число параллельных процессоров, чтобы снизить объем электроэнергии необходимой для выполнения одной многоразрядной операции. Чтобы соответствовать таким критериям эффективности, большая задача в параллельной модели вычислений должна разбиваться на подзадачи для выполнения на одном процессоре. В этом случае вычисление на одном процессоре можно рассматривать как вычисление в последовательной модели. Чем больше алгоритмов проанализировано для последовательной модели, тем больше вероятность того, что выбранная последовательная модель для каждого параллельного процессора даст максимальный эффект при поддержке баланса между кэш-памятью, векторными операциями и числом задействованных процессоров.

Рассмотрим реализацию операции умножения многоразрядных чисел на основе циклической свертки в параллельной модели вычислений. В начале работы рассматривается матричное умножение в последовательной модели вычислений [1]. Покажем, как это вычисление переносится на параллельную модель вычислений и каким образом можно перераспределить вычисления при меньшем числе процессоров без существенной потери по времени. Аналогично операции матричного умножения предлагается эффективная схема перераспределения вычислений для операции многоразрядного умножения. Сначала рассмотрим реализацию операции умножения на основе циклической свертки для одного процессора [2] и покажем, как эта операция переносится на параллельную модель. Аналогично операции матричного умножения, можно эффективно перераспределить вычисления при реализации операции многоразрядного умножения на основе циклической свертки.

Если в последовательной модели вычислений операциями учета знака переноса можно пренебречь, так как знак переноса учитывается в операции сложения, то в параллельной модели вычислений учет знака переноса требует добавления специальной логики, увеличивающей сложность вычисления не менее

чем в два раза. Кроме того, что знак переноса нужно учитывать между разрядами, его нужно переносить еще между процессорами. Чтобы упростить дальнейшее изложение материала, приведены далее алгоритмы и оценки сложности, не учитывающие знак переноса.

Постановка задачи. Пусть даны числа A_N и B_N, которые можно представить в виде $A_N = \sum_{i=0}^{N-1}(a_i \cdot 2^{\omega \cdot i})$, $B_N = \sum_{i=0}^{N-1}(b_i \cdot 2^{\omega \cdot i})$, где $0 \le a_i < 2^{\omega}$, $0 \le b_i < 2^{\omega}$, $i = \overline{0, N-1}$, $N = 2^n$. Необходимо построить эффективный алгоритм реализации операции умножения этих чисел $R_{2N} = \sum_{k=0}^{2N-1}(r_k \cdot 2^{\omega \cdot k})$, где $r_k = \sum_{k=i+j}(a_i \cdot b_j)$, $0 \le i \le k$, $0 \le j \le k$, $k = \overline{0, 2N-2}$, на основе циклической свертки в параллельной модели вычислений.

Алгоритм умножения двух квадратных матриц в последовательной модели вычисления. Пусть даны матрицы $A_{4,4}$ и $B_{4,4}$ размерностью 4×4. Результатом умножения матриц будет матрица $C_{4,4}$ таких же размеров. Вычисление матрицы показано схематично на рис. 1, где $c_{i,j} = \sum_{k=0}^{3}(a_{i,k} \cdot b_{k,j})$, $i, j = \overline{0,3}$. Если элементы матриц $a_{i,j}$, $b_{i,j}$, $c_{i,j}$, $i, j = \overline{0,3}$, содержат однословные выражения, то одному процессору необходимо выполнить 64 однословных операций умножения.

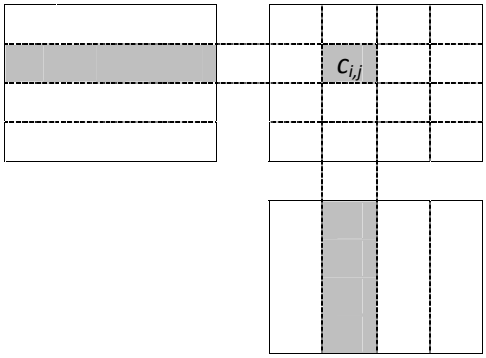

РИС. 1. Умножение матриц в последовательной модели

Реализация операции умножения двух матриц в параллельной модели вычислений. В параллельной модели можно задействовать 16 параллельных процессоров, где каждый из них вычислит сумму 4-х операций однословного умножения. При использовании большего числа параллельных процессоров, например 64, их работа не будет эффективной, поскольку добавлять

в одну ячейку $c_{i,j}$ можно только одно значение $a_{i,k} \cdot b_{k,j}$ за одну операцию и часть процессоров будет простаивать, ожидая окончания операции добавления значения в память другим процессорам.

Реализация операции умножения двух матриц в параллельной модели с использованием кэш-памяти. При чтении элементов $a_{i,j}, b_{i,j}, i, j = \overline{0,3},$ из локальной или глобальной памяти любым из параллельных процессоров их значения помещаются в кэш-память. Как было замечено ранее, операции с кэш-памятью значительно быстрее операций с локальной и глобальной памятью. При последующем использовании этих же элементов $a_{i,j}, b_{i,j}, i, j = \overline{0,3},$ значения элементов читаются напрямую из кэш-памяти. При небольшом числе обращений к памяти каждым параллельным процессором эффект от использования кэш-памяти не заметен. Для более эффективного использования кэш-памяти можно задействовать меньшее число параллельных процессоров, в нашем примере до 4, что увеличивает число обращений к памяти каждым из параллельных процессоров. В этом случае матрица $C_{4,4}$ разбивается на 4 квадратные секции размерностью 2×2. Каждый из процессоров вычислит по 4 элемента $c_{i,j}$, выполнив при этом 16 однословных операций умножения. Вычисление матрицы показано схематично на рис. 2.

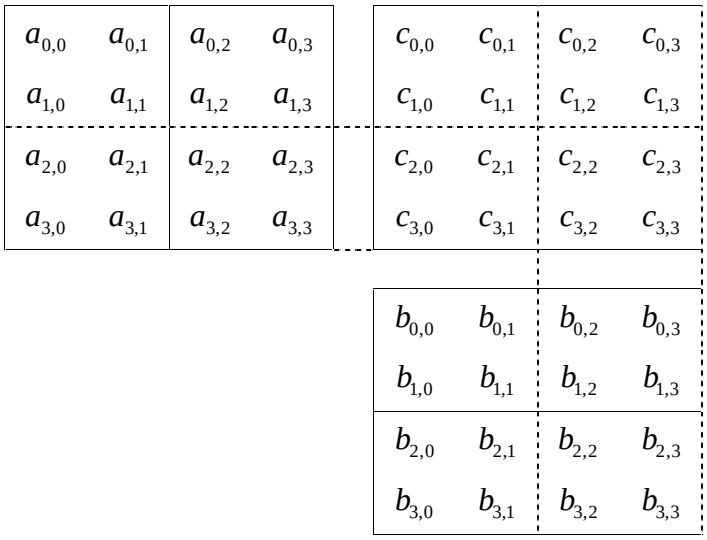

РИС. 2. Умножение матриц в модели из 4 параллельных процессоров

Реализация операции умножения многоразрядных чисел на основе циклической свертки в последовательной модели вычислений. Пусть числа A_8 и B_8 представлены в виде $A_8 = \sum_{i=0}^{7} (a_i \cdot 2^{\omega \cdot i}), \quad B_8 = \sum_{i=0}^{7} (b_i \cdot 2^{\omega \cdot i}),$

где $0 \le a_i < 2^{\omega}$, $0 \le b_i < 2^{\omega}$, $i = \overline{0,7}$. На рис. 3 показано вычисление свертки $R_8 = A_8 \otimes B_8$, где $r_k = \sum_{i=0}^{7} (a_i \cdot b_{\langle i+k \rangle_8})$, $k = \overline{0,7}$. Многоразрядная операция умножения реализуется на основе циклической свертки [2 – 6]. Рассмотрим ее на примере умножения двух 4-разрядных чисел A_4 и B_4, представленных в виде $A_4 = \sum_{i=0}^{3} (a_i \cdot 2^{\omega \cdot i})$, $B_4 = \sum_{i=0}^{3} (b_i \cdot 2^{\omega \cdot i})$, где $0 \le a_i < 2^{\omega}$, $0 \le b_i < 2^{\omega}$, $i = \overline{0,3}$. Операция умножения $R_8 = A_8 \cdot B_8$, где $r_k = \sum_{k=i+j} (a_i \cdot b_j)$, $k = \overline{0,7}$, реализуется на основе 8-разрядной циклической свертки. Реализация схематично показана на рис. 4. Пустые клетки содержат нулевые значения. Разряды числа B_4 берутся в прямом порядке, и к ним добавляются три младших нулевых разряда и один старший нулевой разряд. Разряды числа A_4 берутся в инверсном порядке, и к ним добавляются четыре старших нулевых разряда.

a_0	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7
a_1	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_0
a_2	b_2	b_3	b_4	b_5	b_6	b_7	b_0	b_1
a_3	b_3	b_4	b_5	b_6	b_7	b_0	b_1	b_2
a_4	b_4	b_5	b_6	b_7	b_0	b_1	b_2	b_3
a_5	b_5	b_6	b_7	b_0	b_1	b_2	b_3	b_4
a_6	b_6	b_7	b_0	b_1	b_2	b_3	b_4	b_5
a_7	b_7	b_0	b_1	b_2	b_3	b_4	b_5	b_6
	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7

РИС. 3. Вычисление 8-разрядной циклической свертки

Реализация операции умножения двух многоразрядных чисел на основе циклической свертки в параллельной модели вычислений. Для вычисления 8-разрядной свертки необходимо выполнить 64 однословные операции умножения (рис. 4). Неэффективно задействовать 64 параллельных процессора для того, чтобы все операции умножения выполнились за один такт. Кроме того, половина операций умножения дает ноль как результат. Можно задействовать только 7 процессоров $r_i, i = \overline{0,6}$, выполняющих суммирование результатов операций умножения по каждому столбцу. В этом случае загрузка всех процессоров будет несбалансированной. Параллельные процессоры, которые будут вычислять столбцы r_0 и r_6, выполнят по одной операции однословного умножения,

при этом процессору, вычисляющему столбец r_3 необходимо четыре операции умножения. Наибольший интерес, как и в случае перераспределения вычислений в операции матричного умножения (см. рис. 2), представляют схемы, в которых максимально используется кэш-память, что позволяет использовать меньшее число параллельных процессоров. При умножении двух 4-разрядных чисел на основе циклической свертки (см. рис. 4) необходимо добавлять три младших и один старший нулевой элемент к разрядам числа B_4 и четыре старших нулевых элементов к разрядам числа B_4, что требует дополнительной кэш-памяти для размещения восьми дополнительных элементов. При умножении N-разрядных чисел потребуется кэш-памяти в объеме необходимом для размещения $4N$ элементов. Для небольших чисел кэш-памяти хватает, но с увеличением разрядности перемножаемых чисел кэш-память быстро исчерпывается. Далее показан алгоритм, использующий кэш-память для размещения не более чем $2N + 2n$ элементов, где N – длина перемножаемых чисел и n – длина секции, на которые разбивается число.

Здесь и далее серым цветом выделим клетки, которые не учитываются. В таких клетках операции умножений не выполняются, так как они дают нулевой результат.

a_3				b_0	b_1	b_2	b_3	
a_2			b_0	b_1	b_2	b_3		
a_1		b_0	b_1	b_2	b_3			
a_0	b_0	b_1	b_2	b_3				
	b_1	b_2	b_3					b_0
	b_2	b_3					b_0	b_1
	b_3					b_0	b_1	b_2
					b_0	b_1	b_2	b_3
	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7

РИС. 4. Реализация операции умножения на основе циклической свертки

Рассмотрим циклическую свертку вида $R_n = A_{2n} \otimes B_{2n}$, где $a_k = 0$, $k = \overline{n, 2n-1}$ и $r_k = \sum_{i=0}^{2n-1} (a_i \cdot b_{\langle i+k \rangle_{2n}})$, $k = \overline{0, n-1}$, ($\langle i+k \rangle_{2n}$ – операция вычисления суммы чисел $i+k$ по модулю $2n$) на примере $n = 2$. На рис. 5 нулевые элементы a_2 и a_3 не показаны. Далее циклические свертки такого вида представляются в виде матриц размерностью $n \times n$.

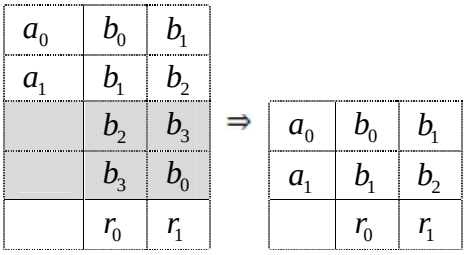

РИС. 5. Циклическая свертка вида $R_2 = A_4 \otimes B_4$, где a_2 и a_3 содержат нулевые значения

Далее на рис. 6 показана реализация операции умножения 4-разрядных чисел в виде схемы, состоящей из восьми секций.

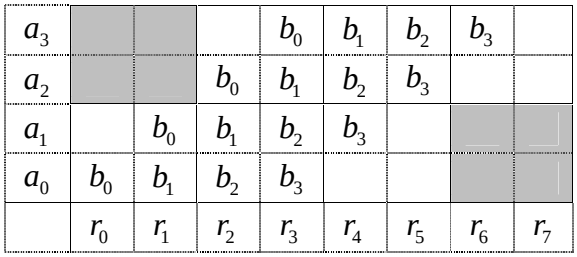

РИС. 6. Реализация операции умножения на модели из шести процессоров

Схема позволяет задействовать не более шести параллельных процессоров. В этой модели два процессора выполнят три операции умножения, два выполнят четыре операции умножения и еще два выполнят одну операцию умножения. Это является недостатком, так как необходимо добавлять дополнительную логику, чтобы каждый процессор выполнял свое число операций умножения. От дополнительной логики можно избавиться, если вычисление в каждой секции реализовать на основе циклических сверток вида $R_2 = A_4 \otimes B_4$ (см. рис. 5), хотя такие свертки будут выполнять операции с нулевым результатом в некоторых секциях. Например, свертка в правом верхнем углу (см. рис. 6) вычисляет r_7, дающее нулевой результат. Далее на рис. 7. показана схема перераспределения вычислений для шести процессоров в виде матрицы. Схема перераспределения вычислений – это упрощение рис. 6 до уровня секций, за исключением того, что ее строки показаны в инверсном порядке. Чтобы найти, какие процессоры должны быть задействованы в каждом столбце схемы, нужно сложить двухэлементные индексы процессора, и результат сложения должен равняться индексу по горизонтальной оси схемы. В схеме, например, процессоры с индексами (0, 1) и (1, 0) вычисляют промежуточные значения для получения результатов элементов r_2, r_3, находящихся в столбце 1.

0	(0, 0)	(0, 1)	(0, 2)	
1		(1, 0)	(1, 1)	(1, 2)
	0	1	2	3
	r_0, r_1	$r_2,\ r_3$	r_4, r_5	$r_6,\ r_7$

РИС. 7. Схема перераспределения вычислений для шести процессоров

Преобразование входных данных. Схема перераспределения вычислений предполагает трансформацию входных данных. Рассмотрим метод, при котором добавляются младшие и старшие нулевые элементы к одному из многоразрядных чисел и не добавляются между секциями. Особенностью данного метода является то, что в инверсном порядке берется не все число (рис. 4), а только элементы каждой секции. Данный метод рассмотрим на примере реализации умножения двух чисел A_8 и B_8 при длине секции $n = 4$. Разобъем число A_8 на 2-е секции $(k = 2)$ и запишем элементы в каждой секции в обратном порядке:

$$(a_3, a_2, a_1, a_0 \mid a_7, a_6, a_5, a_4).$$

Увеличим разрядность B_8 до 16 элементов за счет добавления трех младших и пяти старших нулевых элементов:

$$(0,0,0, b_0 \mid b_1, b_2, b_3, b_4 \mid b_5, b_6, b_7, 0 \mid 0,0,0,0).$$

Из рис. 8 видно, что достаточно вычислить $2 \cdot (2+1) = 6$ циклических сверток разрядности 8 на 4 для вычисления результата. Две секции не задействованы. Каждая циклическая свертка может быть вычислена независимо от остальных сверток на отдельном параллельном процессоре.

a_3				b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7					
a_2			b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7						
a_1		b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7							
a_0	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7								
a_7								b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	
a_6							b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7		
a_5						b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7			
a_4					b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7				
	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{13}	r_{15}

РИС. 8. Реализация операции умножения на модели из шести процессоров

Векторные операции. Замечательным преимуществом GPU является поддержка векторных операций. Существующие GPU поддерживают векторы длины 2, 4, 8, 16. Для нас интерес представляет векторная операция умножения рис. 9. Если алгоритм ядра программы содержит несколько циклов, то использование векторных операций позволяет избежать не менее одного уровня циклов и накладных расходов, связанных с выполнением этого цикла. Векторная операция умножения разбивается на элементарные операции умножения, выполняющиеся поэлементно. Так, например, при векторном перемножении чисел X_4 и Y_4 получаем число Z_4 с элементами $z_i = x_i \cdot y_i,\ \ i = \overline{0,3}.$ Использование векторных операций умножения значительно ускоряет время необходимое на выполнение всех однословных операций умножения.

X_4	.	Y_4	=	Z_4
x_0		y_0		$x_0 \cdot y_0$
x_1		y_1		$x_1 \cdot y_1$
x_2		y_2		$x_2 \cdot y_2$
x_3		y_3		$x_3 \cdot y_3$

РИС. 9. Векторная операция умножения длины 4

Лемма 1. Для чисел C_{2n} и D_{2n}, представимых в виде $C_{2n} = \sum_{i=0}^{2n-1}(c_i \cdot 2^{\omega \cdot i})$, $D_{2n} = \sum_{i=0}^{2n-1}(d_i \cdot 2^{\omega \cdot i})$, где $0 \le c_i < 2^{\omega},\quad 0 \le d_i < 2^{\omega},\quad i = \overline{0,n-1};\quad c_i = 0,\quad i = \overline{n,2n-1},$ $n = 2^m,\ 1 \le m \le 4,$ при вычислении операции свертки $E_n = \sum_{k=0}^{n-1}(e_k \cdot 2^{\omega \cdot k})$, где $e_k = \sum_{i=0}^{2n-1}(c_i \cdot d_{\langle i+k \rangle_{2n}})$, $k = \overline{0,n-1}$, требуется n векторных операций умножения длины n.

Доказательство. Рассмотрим свертку $E_n = C_{2n} \otimes D_{2n}$ на рис. 10.

c_0	d_0	d_1	…	d_{n-1}
…	…	…	...	...
c_{n-2}	d_{n-2}	d_{n-1}	…	d_{2n-3}
c_{n-1}	d_{n-1}	d_n	…	d_{2n-2}
…	…	…	…	…
0	d_{2n-2}	d_{2n-1}	…	d_{n-3}
0	d_{2n-1}	d_0	…	d_{n-2}
	e_0	e_1	⋯	e_{n-1}

РИС. 10. Вычисление свертки $E_n = C_{2n} \otimes D_{2n}$, где $c_i = 0,\ i = \overline{n,2n-1}$

Как было описано ранее (см. рис. 5), строки, выделенные серым цветом, не учитывать. Остается n^2 промежуточных значений свертки, что потребует n^2 однословных операций умножения. Значит достаточно выполнить n векторных операций умножения длины n. Лемма доказана.

Лемма 2. Для чисел A_N и B_N, представимых в виде $A_N = \sum_{i=0}^{N-1}(a_i \cdot 2^{\omega \cdot i})$, $B_N = \sum_{i=0}^{N-1}(b_i \cdot 2^{\omega \cdot i})$, где $0 \le a_i < 2^{\omega}$, $0 \le b_i < 2^{\omega}$, $i = \overline{0, N-1}$, $N = n \cdot k$, $n = 2^m$, $1 \le m \le 4$, при реализации операции умножения этих чисел $R_{2N} = \sum_{k=0}^{2N-1}(r_k \cdot 2^{\omega \cdot k})$, где $r_k = \sum_{k=i+j}(a_i \cdot b_j)$, $0 \le i < N$, $0 \le j < N$, $k = \overline{0, 2n-2}$, достаточно задействовать не более $P(k) = k \cdot (k+1)$ параллельных процессоров при добавлении $n-1$ младших и $n+1$ старших нулевых элементов к разрядности числа B_N. При этом потребуется $O(n \cdot k) = k \cdot (k+1) \cdot n$ векторных операций умножения длины n, где n – длина секции, приведенных в таблице.

ТАБЛИЦА. Зависимость количества векторных операций умножения $O(n \cdot k)$ длины n от длины N числа и числа k секций, на которое это число разбивается

N	k	n	P	$O(n \cdot k)$	N	k	n	P	$O(n \cdot k)$
4	1	4	2	8	32	1	32	2	64(128)
8	1	8	2	16	32	2	16	6	96
8	2	4	6	24	32	4	8	20	160
8	4	2	20	40	32	8	4	92	368
16	1	16	2	32	64	2	32	6	192(384)
16	2	8	6	48	64	4	16	20	320
16	4	4	20	80	64	8	8	72	576
16	8	2	72	144	64	16	4	272	1088

Доказательство. Разобъем многоразрядные числа длины $N = n \cdot k$ на k секций длины n. Покажем схему перераспределения вычислений на уровне секций, опустив элементы аналогично рис. 7.

Отметим клетки схемы, использующиеся при перераспределении вычислений. В каждой строке остается $k+1$ ячеек белого цвета. Всего таких строк k. Тогда рабочую группу процессоров можно представить в виде матрицы из k строк и $k+1$ столбцов. Что и требовалось доказать.

Рассмотрим схему перераспределения вычислений между параллельными процессорами при умножении чисел A_{32}, B_{32}, рис. 11. Разобъем их на 4-е секции $(k = 4)$.

0	(0,0)	…	$(0,k-1)$	$(0,k)$			
…		…	…	…	…		
$k-1$			$(k-1,0)$	$(k-1,1)$	$(k-1,2)$	…	$(k-1,k)$
	0	…	$k-1$	k	$k+1$	…	$2k-1$

РИС. 11. Схема перераспределения вычислений при умножении чисел, которые разделены на k секций

Тогда необходимо задействовать $4\cdot(4+1)=20$ параллельных процессоров, как показано на рис. 12. Рабочую группу из 20 процессоров можно интерпретировать в виде матрицы размерностью 4×5. Далее легко найти общее число векторных операций умножения длины 8. Так как для вычисления одной свертки длины 8 необходимо 8 векторных операции умножения, то для вычисления 20 сверток, выполняемых на 20 процессорах необходимо $4\cdot(4+1)\cdot 8=160$ векторных операций.

Если разбить числа A_{32} и B_{32} на 2-е секции $(k=2)$, то схему (см. рис. 12) можно представить в виде циклических сверток длины 16 и их вычислением на $2\cdot(2+1)=6$ процессорах. Все процессоры выполнят $2\cdot(2+1)\cdot 16=96$ векторных операций умножения.

Хотя векторная операция длины 32 не поддерживается GPU, ее можно реализовать за счет выполнения подряд двух операций длины 16. Тогда числа A_{32} и B_{32} не нужно разбивать $(k=1)$ и вычисления можно представить в виде двух циклических сверток длины 32. Циклические свертки выполнятся на $1\cdot(1+1)=2$ процессорах, которые выполнят $1\cdot(1+1)\cdot 32=64$ векторные операции умножения длины 32, для реализации которых необходимо в два раза больше векторных операций длины 16, т. е. необходимо 128 векторных операций длины 16. Где P – число задействованных параллельных процессоров. 64(128) и 192(384) обозначают, что для реализации векторных операций длины 32 необходимо в два раза больше векторных операций длины 16.

0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)			
1		(1,0)	(1,1)	(1,2)	(1,3)	(1,4)		
2			(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	
3				(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
	0	1	2	3	4	5	6	7

РИС. 12. Схема перераспределения вычислений в рабочей группе из 20 процессоров

Далее представлен алгоритм 1, выполняющийся синхронно всеми процессорами в схеме перераспределения вычислений (см. рис. 11) при умножении чисел A_N и B_N, где $N=n\cdot k$, на основе циклической свертки размерностью $2n\times n$ в параллельной модели вычислений с использованием векторных операций.

Алгоритм (ядра) 1 использует входные значения V_N и W_{N+2n}, являющиеся преобразованиями A_N и B_N. Преобразование происходит на основе следующих выражений:

$$v_{i\cdot n+j} = a_{(i+1)\cdot n-1-j},\ w_{(i+1)\cdot n+j-1} = b_{i\cdot n+j},\quad j=\overline{0,n-1},\ i=\overline{0,k-1}.$$

$$w_j = 0,\quad j=\overline{0,n-2}.$$

$$w_{N+n-1+j} = 0,\quad j=\overline{0,n}.$$

Алгоритм 1. Вычисление значений одной секции из схемы перераспределения вычислений на основе циклической свертки (см. рис. 11) размерностью $2n\times n$ и добавление вычисленных значений к результату операции умножения.

Вход: $V_N =$

$$=\left(a_{n-1},a_{n-2},...,a_1,a_0 \mid a_{2n-1},a_{2n-2},...,a_{n+1},a_n \mid ... \mid a_{N-1},a_{N-2},...,a_{N-n+1},a_{N-n}\right),$$

$$W_{N+2n} =$$

$$=(0,0,...,0,b_0 \mid b_1,b_2,...,b_{n-3},b_{n-2} \mid ... \mid b_{N-n+1},b_{N-n+2},...,b_{N-2},b_{N-1},0 \mid 0,0,...,0,0),$$

i, j – индексы строки и столбца группы процессоров, интерпретируемой в виде матрицы, n – длина секции, на которые разбиты числа V_N и W_N, где $N=n\cdot k$.

Выход: R_{2N}, где $0\le r_i < 2^{\omega+\lceil\log_2 k\rceil}$, $i=\overline{0,2N-1}$.

1. $k\leftarrow i+j;\ v\_ix\leftarrow i\cdot n;\ w\_ix\leftarrow j\cdot n, r\_ix\leftarrow k\cdot n$.
2. X_n, Y_n, где $x_i\leftarrow v_{i+v\_ix}$, $y_i = w_{i+w\_ix}$, $i=\overline{0,n-1}$.
3. Для $i=0$ до $n-1$.
4. $Z_n\leftarrow X_n\cdot Y_n$ (векторная операция умножения).
5. $sum\leftarrow 0;\ sum\leftarrow sum+z_j;\ j=\overline{0,n-1};$
6. $r_{i+r\_ix}\leftarrow r_{i+r\_ix}+sum$ (вычисление результата).
7. Если $i<n-1$, то
8. $y_i\leftarrow y_{i+1},\ i=\overline{0,n-2};\ y_{n-1}\leftarrow w_{i+n+w\_ix}$.
9. Конец.
10. Конец цикла по i.

Примечание. На шаге 5 могут возникать небольшие блокировки из-за того, что несколько ядер могут добавлять свои значения sum в одни и те же ячейки результата r_{i+r_ix}. Реализация этого шага немного сложнее, но считаем, что синхронный переход на шаг 7 не происходит до тех пор, пока все процессоры не добавят значения к ячейкам результата. Шаги 5, 8 также могут быть реализованы на основе векторных операций сложения и циклического сдвига, но для упрощения изложения материала эта реализация опущена.

Выводы. В работе рассмотрена реализация операции умножения многоразрядных чисел на основе циклической свертки в параллельной модели вычислений. Предложена схема перераспределения вычислений (см. рис. 11) на $P(k) = k \cdot (k+1)$ параллельных процессорах при реализации операции умножения *N*-разрядных чисел (где $N = n \cdot k$). В предложенной схеме N -разрядные числа разбиваются на k секций, каждая из них содержит по n элементов. Из леммы 1 следует, что для вычисления циклической свертки размерностью $2n \times n$, где $n = 2^m$ и $1 \le m \le 4,$ каждый задействованный процессор выполнит по n векторных операций умножения длины n. В лемме 2 показано, что при модификации каждого *N*-разрядного числа, задача реализации многоразрядной операции умножения может быть приведена к $P(k) = k \cdot (k+1)$ подзадачам вычисления циклических сверток меньшей разрядности на $P(k)$ процессорах параллельно. В работе приведена таблица зависимости количества векторных операций умножения $O(n \cdot k)$ длины n от длины N многоразрядного числа и числа k секций, на которое это число разбивается. Предложенная схема реализована на OpenCL 1.2 AMD-APP (938.2) [1, 7, 8].

Время выполнения “conv_column_uint_loc_size16.cl” ядра на GPU “AMD Radeon HD 7670M” составило 2,85 мс при умножении двух 496-байтных чисел при задействованной рабочей группе процессоров, которую можно интерпретировать в виде матрицы размерностью 31×32. Ядро базируется на векторных операциях длины 16. Знаки переноса учитываются при выполнении ядра. Результат вычисления предложенной схемы на GPU подтвердил корректность результатов.

А.М. Терещенко, В.К. Задірака

АНАЛІЗ СКЛАДНОСТІ ОПЕРАЦІЇ МНОЖЕННЯ БАГАТОРОЗРЯДНИХ ЧИСЕЛ У ПАРАЛЕЛЬНІЙ МОДЕЛІ ОБЧИСЛЕННЯ

Аналізується складність за кількістю векторних однослівних операцій множення при реалізації багаторозрядної операції множення в паралельній моделі. Пропонується ефективна схема перерозподілу обчислень на основі використання циклічних згорток меншої розрядності.

A.N. Tereshchenko, V.K. Zadiraka

COMPLEXITY ANALYSIS OF COMPUTATION OF MULTI-DIGIT MULTIPLICATION OPERATION IN PARALLEL MODEL

The complexity of number of vector multiplication operations in multi-digit multiplication operation computation is analyzed in parallel model. An effective scheme of balancing computations based on cyclic convolutions of smaller digit capacity is proposed.

1. *Задірака В., Олексюк О.* Комп'ютерна арифметика багаторозрядних чисел. – К.: Наук. думка, 2003. – 263 с.
2. *Bhattacharyya K., Banger R.* OpenCL Programming by Example // Packt. – 2013. – P. 304.

3. *Терещенко А.Н.* Оптимизация метода Питасси вычисления свертки // Искусственный интеллект. – 2009. – № 1. – С. 204 – 212.
4. *David A. Pitassi.* Fast convolution using the Walsh transform // Applications of Walsh Functions, 1971. – P. 13 – 133.
5. *Davis W.F.* A class of efficient convolution algorithms // Applicat. Walsh Functions. – 1972. – March. – P. 318 – 329.
6. *Терещенко А.Н., Мельникова С.С., Задирака В.К. и др.* Реализация операции умножения с использованием преобразования Уолша // Международный научно-технический журнал «Проблемы управления и информатики». – 2010. – № 2. – С. 102 – 126.
7. *Munshi A., Gaster B., Ginsburg D. et al.* OpenCL Programming Guide // Addison-Wesley. – 2011. – P. 648.
8. *Клюшин Д.А.* Полный курс С++. Профессиональная работа. – М.: Издательский дом “Вильямс”, 2004. – 672 с.

Получено 01.03.2016

Об авторах:

Терещенко Андрей Николаевич,
кандидат физико-математических наук, старший инженер-программист
Института кибернетики имени В.М. Глушкова НАН Украины,
E-mail:*teramidi@ukr.net*

Задирака Валерий Константинович,
доктор физико-математических наук, профессор, академик НАН Украины,
заведующий отделом Института кибернетики имени В.М. Глушкова НАН Украины.
E-mail:*zvk140@ukr.net*