

# К вычислению квадратов и абсолютных значений на процессоре NM6403(Л1879ВМ1)

Мушкаев С.В.

Москва, НТЦ "Модуль", e-mail: mushkaev@module.ru

## Аннотация

В данной статье рассматриваются вопросы реализации базовых операций: модуля и квадрата целых чисел при векторной обработке массивов данных на процессоре NM6403(Л1879ВМ1). Предлагаются некоторые нестандартные способы и подходы к их вычислению. Приводится пример быстрого варианта взятия модуля, в котором время вычисления модуля на процессоре NM6403 достигает минимального предела и составляет 1 такт на 8 байт. К рассматриваемым вариантам приводятся короткие поясняющие фрагменты программ с векторными инструкциями процессора. В приведенных примерах демонстрируются возможности процессора при работе с данными малой разрядности: 1, 2 и 4 бита.

## Введение

В цифровой обработке сигналов и изображений очень часто возникает задача вычисления квадрата или модуля над элементами массива. И проблема эффективной реализации даже таких элементарных, с точки зрения математики, операций в прикладных задачах остается довольно актуальной. Связано это прежде всего с тем, что, как правило, эти операции сопряжены с рядом других действий, и в совокупности задача распараллеливания и распределения аппаратных ресурсов процессора, может иметь несколько вариантов решений, выбор которых не всегда является тривиальным. Цель данной статьи состоит в том, чтобы по возможности расширить спектр таких решений, а так же рассмотреть подходы, позволяющие на их базе строить собственные быстрые алгоритмы.

## 1 Алгоритмы вычисления модуля на процессоре NM6403

В данном разделе предлагается несколько вариантов вычисления модуля на векторном процессоре. Каждый вариант задействуют различные ресурсы

векторного сопроцессора и требует разного времени вычисления.

Очень часто на практике вычисление модуля входит в более сложное арифметическое выражение, например такое, как мера сходства  $SAD = \sum_{i=0}^{n-1} |x_i - y_i|$  и в зависимости от требуемых ресурсов векторного узла и специфики задачи бывает удобно использовать либо тот, либо другой вариант вычисления модуля. Ниже рассматриваются преимущества и недостатки трех вариантов вычисления модуля.

### 1.1 Вариант 1

Наиболее простой способ и наиболее часто используемый на практике способ вычисления модуля состоит из 4 векторных инструкций:

```
rep 1 data=[ar0] with 0-data;
rep 1 [ar6],ram=afifo with activate afifo;
rep 1 data=[ar0] with mask afifo,data,ram;
rep 1 [ar6]=afifo;
```

Смысл этих инструкций состоит в определении знака аргумента, с помощью которого маскируется либо аргумент либо минус аргумент. В математическом виде это означает :

$abs(x) = (sign(-x) \wedge x) \vee (\overline{sign(-x) \wedge x})$ , где  $sign()$  - логическая функция активации( activate):

$$sign(x) = \begin{cases} 0 & \text{если } x \geq 0 \\ -1 & \text{если } x < 0 \end{cases} \quad (1)$$

Если входные и выходные данные располагаются на разных шинах памяти процессора (Local и Global Bus) [2], то в цикле первая и последняя инструкция блока будут выполняются параллельно и время обработки одного 64р. слова составит 3 такта. Заметим, что в каждом 64р. слове может быть упаковано несколько чисел меньшей разрядности, для которых модуль вычисляется одновременно. Так в случае байтовой упаковки время вычисления модуля в пересчете на один байтовый элемент составляет  $3/8 = 0.475 \simeq 0.5$  такта.

## 1.2 Вариант 2

Данный вариант реализуется с помощью векторного множителя, что позволяет записать действие в 3 инструкции:

```
rep 1 ram=[ar0] with activate data;
rep 1      with vsum afifo,ram,ram;
rep 1 [ar6]=afifo;
```

(При этом в матрицу весов векторного множителя предварительно должен быть загружен инвертирующий множитель -1)

Здесь также сначала производится выделение знака аргумента, в зависимости от которого аргумент пройдет либо через умножающий вход с коэффициентом  $-1$ , либо без изменений через суммирующий. В математическом виде это означает:  $abs(x) = ((sign(-x) \wedge x) \cdot (-1)) + (\overline{sign(-x)} \wedge x)$

Если входные и выходные данные располагаются на разных шинах памяти процессора (Local и Global Bus), то в цикле первая и последняя инструкция блока будут выполняться параллельно и время обработки одного 64р. слова составит 2 такта.

## 1.3 Вариант 3

Вычисление абсолютного значения без операций ветвления можно также вычислить на базе операции "Исключающего ИЛИ" с помощью следующих выражений [1]:

$$|x| = (x \oplus sign(x)) - sign(x) \quad (2)$$

$$|x| = (x + sign(x)) \oplus sign(x) \quad (3)$$

$$|x| = x - (2x \wedge sign(x)) \quad (4)$$

По скорости все они уступают предыдущему варианту, поскольку требуют не менее 3 тактов.

## 1.4 Вариант 4

Формулу 2 можно упростить, удалив из нее последнее слагаемое. Полученная в результате формула  $|x| = sign(x) \oplus x$  математически уже не соответствует определению модуля. Для отрицательных чисел она отличается от модуля на единицу:

$$abs'(x) = \begin{cases} x & \text{если } x \geq 0 \\ -x - 1 & \text{если } x < 0 \end{cases}$$

Однако, данное обстоятельство на практике имеет большие преимущества:

1. Не возникает ошибка переполнения при взятии модуля от наименьшего отрицательного числа. Так, для байтовых чисел  $abs'(-128) = 127$ , а не

$-128$  как это происходило бы во всех предыдущих вариантах. Во многих задачах, например, по обработке изображений, ошибка в модуле на единицу не имеет существенного значения и ей можно пренебречь. В результате, при программировании исчезает необходимость предотвращать возможное переполнение, что позволяет работать в более узкой разрядной сетке.

2. Модуль вычисляется максимально быстро – за 1 процессорный такт на одно 64р. слово. Для реализации данного метода достаточно двух векторных инструкций, выполняющихся параллельно:

```
rep 1 data=[ar0] with activate data xor data;
rep 1 [ar6]=afifo;
```

3. Не требуется изменять, или использовать матрицу весовых коэффициентов. Это особенно важно в случае дальнейшего суммирования модулей.

## 2 Алгоритмы возведения в квадрат элементов вектора

Приведем два варианта возможного вычисления квадратов элементов вектора. Для простоты в качестве примера разрядность исходных примем равной 4 битам, а для результата – 8 битам.

### 2.1 Способ 1 - "через диагональную загрузку"

Данный способ является наиболее очевидным и состоит в загрузке элементов вектора в матрицу весовых коэффициентов по диагонали как показано на рис. 1. Подготовку диагональной матрицы наиболее выгодно производить на скалярном ядре. В этом случае подготовку можно осуществлять на фоне загрузки предыдущей порции диагональной матрицы в теневую матрицу весовых коэффициентов и практически не требует дополнительного времени. Так как сам процесс умножения вектора на весовую матрицу происходит за один такт, основное время вычисления будет определяться именно временем загрузки матрицы коэффициентов.

### 2.2 Алгоритм быстрого вычисления квадрата

Предлагаемый алгоритм не требует диагонального преобразования данных и загрузки их в матрицу весовых коэффициентов и может успешно использоваться при вычисления квадратов чисел малой разрядности или сумм квадратов на процессоре NM6403.

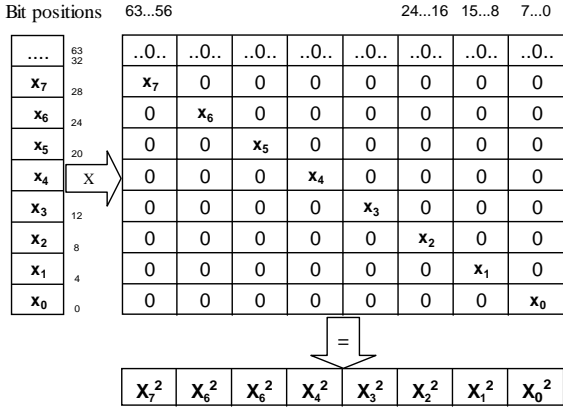


Рис. 1: Алгоритм вычисления квадратов через диагональную загрузку элементов вектора в матрицу весовых коэффициентов

Допустим имеется 4-х разрядное число со знаком:  $x$ , представляемое в бинарной записи как  $x = scba$ , где  $a, b, c$ -биты, а  $s$ -знак. Тогда  $x = -8s + 4c + 2b + a$ . Представим  $x^2$  следующим образом:

$$\begin{aligned}
 x^2 &= -(x \cdot (-x)) = -(scba \cdot (-scba)) = \\
 &= -(scba \cdot (\overline{scba} + 1)) = \\
 &= -(scba + scba \cdot \overline{scba})
 \end{aligned} \tag{5}$$

$$\begin{aligned}
 scba \cdot \overline{scba} &= \\
 &= (-8s + cba) \cdot (-\overline{8s} + \overline{cba}) = \\
 &= -8s \cdot \overline{cba} - \overline{8s} \cdot cba + \overline{cba} \cdot cba = \\
 &= -8(sss \oplus cba) + cba \cdot \overline{cba}
 \end{aligned} \tag{6}$$

$$\begin{aligned}
 cba \cdot \overline{cba} &= (4c + ba)(\overline{4c} + \overline{ba}) = \\
 &= 4c \cdot \overline{ba} + \overline{4c} \cdot ba + \overline{ba} \cdot ba = \\
 &= 4(cc \oplus ba) + \overline{ba} \cdot ba
 \end{aligned} \tag{7}$$

$$\begin{aligned}
 \overline{ba} \cdot ba &= (2\overline{b} + \overline{a})(2b + a) = \\
 &= 2\overline{b} \cdot a + 2b \cdot \overline{a} = 2(b \oplus a)
 \end{aligned} \tag{8}$$

Выполняя последовательно обратную подстановку получим

$$\begin{aligned}
 -x^2 &= x \cdot (-x) = scba \cdot (-scba) = \\
 &= scba - 8(sss \oplus cba) + 4(cc \oplus ba) + 2(b \oplus a)
 \end{aligned} \tag{9}$$

или

$$\begin{aligned}
 x^2 &= 8(sss \oplus cba) - \\
 &\quad -4(cc \oplus ba) - 2(b \oplus a) - scba
 \end{aligned} \tag{10}$$

Подобные рассуждения можно использовать для разложения квадратов чисел большой разрядности. В случае 5-разрядного  $x = sdcb a$  выражения для  $x^2$  выглядит следующим образом:

$$\begin{aligned}
 x^2 &= 16(ssss \oplus dcba) - sdcb a - \\
 &\quad -8(ddd \oplus cba) - 4(cc \oplus ba) - 2(b \oplus a)
 \end{aligned} \tag{11}$$

Из формулы (11) можно получить выражения для квадрата положительного 4-х разрядного числа  $x = dcba$ , подставляя в (11)  $s = 0$ :

$$\begin{aligned}
 x^2 &= 15dcba - \\
 &\quad -8(ddd \oplus cba) - 4(cc \oplus ba) - 2(b \oplus a)
 \end{aligned} \tag{12}$$

### 2.2.1 Реализация быстрого алгоритма. Вариант А

Непосредственная запись формулы (10) терминах операций, аппаратно поддерживаемых процессором NM6403, дает следующее выражение :

$$\begin{aligned}
 x^2 &= 8(0sss \oplus 0cba) - 4(00cc \oplus 00ba) - \\
 &\quad -2(000b \oplus 000a) - scba = \\
 &= 8((sign(scba) \oplus scba)) - \\
 &\quad -4((sign(cba) \oplus scba) \wedge 0011) - \\
 &\quad -2((sign(ba) \oplus scba) \wedge 0001) - scba,
 \end{aligned} \tag{13}$$

где согласно (1)  $sign(sbca) = ssss$ ,  $sign(bca) = 0bbb$  и т.д.

На процессоре это выражение вычисляется за 11 процессорных тактов на 1 длинное слово. Т.е. каждое 4-х разрядное число возводится в квадрат и сохраняется в 8р. слове результата за 11/16 такта.

```

f1cr=08888888h;
rep 1 data=[ar0++] with activate data xor data;
rep 1 [ar6++]=afifo;

f1cr=0cccccccch;
rep 1 data=[ar0++] with activate data xor data;
rep 1 vsum ram, afifo;

rep 1 data=[ar0++] with vsum ram, shift data, data;
rep 1 [ar6++]= afifo;

rep 2 data=[ar6++] with vsum ,ram, data,0;
rep 2 data=[ar6++] with data + afifo;
rep 2 data=[ar6++] with vsum ,data,afifo;
rep 2 data=[ar6++] with vsum ,data,afifo;
rep 2 [ar5++]=afifo;

```

### 2.2.2 Реализация быстрого алгоритма. Вариант В

Данный вариант не использует операцию "исключающего ИЛИ" (XOR) в явном виде, а основывается на том свойстве, что побитовое сложение двух чисел эквивалентно операции XOR .

Архитектура процессора NM6403 позволяет выбрать произвольное разбиение 64р. слова на составляющие элементы вплоть до 1-разрядных слов, что позволяет выполнять функцию XOR через суммирование в каждом разряде.

Имея матрицу с 1-разрядными столбцами и 1-разрядными строками и загружая единичные биты в соответствующие ячейки мы получаем возможность суммировать биты из разных разрядов по функции XOR. На рис. 2 показан пример вычисления "Исключающего ИЛИ" бита  $b_0$  с каждым из битов  $b_1, b_2$  и  $b_3$



тов, выводя их по вышеприведенной схеме и используя только наиболее значимые слагаемые.

Также для приближенных расчетов квадратов чисел, корней и тригонометрических функций удачное применение на процессоре NM6403 находят методы из семейства алгоритмов под общим названием "CORDIC" или иначе известное под названием "цифра за цифрой".

### 3 Выводы

Рассмотренные выше способы демонстрируют высокую эффективность процессора NM6403 при вычислении модуля над элементами массива. Приведенный способ "неполного" взятия модуля позволяет достичь предельно возможной скорости его вычисления на процессоре NM6403 равной 1 такту на одно 64-разрядное слово. При работе с упакованными данными меньшей разрядности производительность повышается в соответствующее число раз. При этом гарантируется отсутствие переполнения, что позволяет дополнительно сужать разрядную сетку. Данные обстоятельства делают этот способ особенно привлекательным при обработке изображений, где погрешность младшего бита не играет особой роли.

Способ точного вычисления модуля уступает по скорости и составляет два такта на одно 64-разрядное слово.

Способы поэлементного возведения в квадрат значительно более трудоемкие чем модуль, однако в ряде задач их можно заменить на приближенные методы. В статье рассмотрен один из таких способов, позволяющий возводить в квадрат числа малой разрядности (8 и меньше) менее чем за такт.

Предложенные способы позволяют более эффективно использовать ресурсы процессора NM6403 и могут найти успешное применение в задачах корреляционного анализа, фильтрации, поиска соответствия, сжатия изображений и т.п.

### Список литературы

- [1] Генри Уорен. "Алгоритмические трюки для программистов"
- [2] НТЦ Модуль. "NeuroMatrix NM6403. Описание языка ассемблера"