Программное обеспечение
процессора NM6403

Описание реализации алгоритма

Преобразование Адамара

Версия 2.01

Автор: Виталий Кашкаров

АКЦИОНЕРНОЕ ОБЩЕСТВО
НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

Оглавление

В данном документе рассматривается реализация алгоритма преобразования Адамара на процессоре NeuroMatrix® NM6403. В качестве параметров, определяющих преобразование, использованы следующие показатели:

- разрядность входных данных - 8 бит (числа со знаком),
- количество шагов - 8,
- разрядность выходных данных - 16 бит (числа со знаком),
- размер входного вектора - 256 байт;
- размер вектора результатов - 256 коротких слов (16 бит).

Программа была реализована на процессоре NM6403. Особенностями данного процессора являются наличие матричного операционного узла разрядностью 64x64 бита и возможность программно задавать разрядность обрабатываемых данных. Более подробно о процессоре см. [1,3].

Документ описывает подход к программированию преобразования Адамара на процессоре NM6403, а также содержит подробные комментарии к приемам программирования процессора на языке ассемблера. Особое внимание уделяется работе с векторным узлом процессора.

При подготовке этого раздела была использована книга [2].

Преобразование Адамара осуществляется над целыми числами. В качестве базисных функций используются двузначные функции Уолша, принимающие значения `1`, `-1`. Поэтому преобразование Адамара не имеет других операций кроме сложения и вычитания.

Преобразование Адамара можно выразить в виде произведения матрицы на вектор. Матрица преобразования - матрица Адамара - состоит из `+1` и `-1`. Строки и столбцы матрицы ортогональны. Матрицу Адамара можно определить рекурсивно:

$$H_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \text{ и } H_{2N} = \frac{1}{\sqrt{2}}\begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

Например, матрица восьмого порядка выглядит следующим образом:

Рис. 1 Матрица Адамара 8-го порядка.

$$H_8 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Если на входе имеется вектор $\boldsymbol{x}$ длиной N ($N=2^n$), то вектор y, полученный в результате преобразования Уолша-Адамара, равен $\boldsymbol{y} = H_N\boldsymbol{x}$.

Преобразование, описанное выше, называется дискретным (ДПА).

Однако на практике чаще используется быстрое преобразование Адамара (БПА). Его суть может быть описана таблицей:

Табл. 1 Шаги быстрого преобразования Адамара.

Входные данные	Первый шаг (1)	Второй шаг (2)	Третий шаг (3)
a_1	$b_1 = a_1 + a_2$	$c_1 = b_1 + b_3$	$d_1 = c_1 + c_5$
a_2	$b_2 = a_1 - a_2$	$c_2 = b_2 + b_4$	$d_1 = c_2 + c_6$
a_3	$b_3 = a_3 + a_4$	$c_3 = b_1 - b_3$	$d_1 = c_3 + c_7$
a_4	$b_4 = a_3 - a_4$	$c_4 = b_2 - b_4$	$d_1 = c_4 + c_8$
a_5	$b_5 = a_5 + a_6$	$c_5 = b_5 + b_7$	$d_1 = c_1 - c_5$

a_6	$b_6 = a_5 - a_6$	$c_6 = b_6 + b_8$	$d_1 = c_2 - c_6$
a_7	$b_7 = a_7 + a_8$	$c_5 = b_5 - b_7$	$d_1 = c_3 - c_7$
a_8	$b_8 = a_7 - a_8$	$c_6 = b_6 - b_8$	$d_1 = c_4 - c_8$

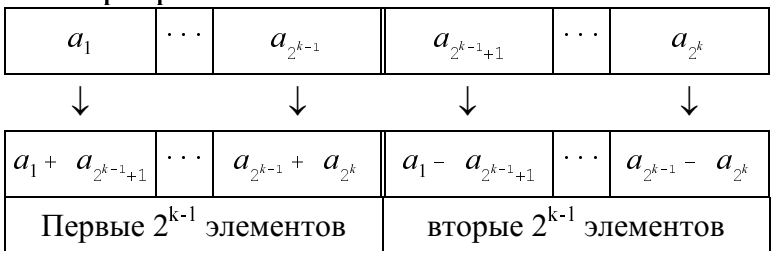
Из Табл. 1 видно, что элементы третьего шага преобразования могут быть выражены через элементы входного массива следующим образом:

$$d_1 = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8$$
$$d_2 = a_1 - a_2 + a_3 - a_4 + a_5 - a_6 + a_7 - a_8$$
$$d_3 = a_1 + a_2 - a_3 - a_4 + a_5 + a_6 - a_7 - a_8$$
$$d_4 = a_1 - a_2 - a_3 + a_4 + a_5 - a_6 - a_7 + a_8$$
$$d_5 = a_1 + a_2 + a_3 + a_4 - a_5 - a_6 - a_7 - a_8$$
$$d_6 = a_1 - a_2 + a_3 - a_4 - a_5 + a_6 - a_7 + a_8$$
$$d_7 = a_1 + a_2 - a_3 - a_4 - a_5 - a_6 + a_7 + a_8$$
$$d_8 = a_1 - a_2 - a_3 + a_4 - a_5 + a_6 + a_7 - a_8$$

Если обратиться к Рис. 1,то станет очевидно, что коэффициенты 1 и -1 перед элементами входного вектора в точности соответствуют матрице Адамара восьмого порядка.

В общем виде преобразование на *k*-ом шаге определяет взаимодействие 2^k элементов входного вектора. При этом первые 2^{k-1} элементов заполняются суммами, значений элемента текущей ячейки и ячейки, отстоящей от него на 2^{k-1} элементов, в вторые 2^{k-1} элементов разностями (см. Рис. 2):

Рис. 2 Преобразование на k-ом шаге



В результате преобразования каждый элемент выходного вектора выражается через все элементы входного. Однако, если отступить на один шаг преобразования назад, то легко заметить, что существует два блока данных, в каждом из которых наблюдается полная взаимосвязь между элементами, тогда как в формуле преобразования элементов одного блока не участвует ни один элемент другого, то есть блоки являются абсолютно независимыми друг от друга. Например, на первом шаге каждые два элемента являются независимыми от соседей, поскольку в формулу их вычисления входят только их значения:

```
b1 = a1 + a2
b2 = a1 - a2
```

поэтому, например, вычисление значений пар элементов (b_1, b_2), (b_3, b_4), ...(b_{2^k-1} , b_{2^k}) можно вести независимо. И так на каждом шаге вплоть до последнего можно выделить независимо вычисляемые блоки.

Преобразование Адамара на языке Си может быть записано следующим образом:

```
void Adamar( int    AStepsNum,  // количество шагов
             int*   AData)      // массив данных
{
  int    DataSz, i, j, jj, Block, Pair, Ind0, Ind1;
  int    Item0, Item1;

  /* подготовка */
  DataSz = 1 << StepsNum;     // вычисление размера массива

  /* преобразование Адамара */
  Block = 1;
  for( i = 0; i < AStepsNum; i++ ) {

    Pair = Block;           // расстояние между парами( a1 , a2k+1 )и тд.
    Block += Block;         // размер независимого блока на данном шаге
    for( j = 0; j < DataSz; j += Block) {
      for( jj = 0; jj < Pair; jj++ ) {
        Ind0 = j + jj;      // вычисляем индекс первого элемента
        Ind1 = ind0 + Pair;    // индекс второго элемента
        Item0 = AData[Ind0];   // считываем старые значения
        Item1 = AData[Ind1];
        AData[ind0] = Item0 + Item1; //заменяем старые значения новыми
        AData[ind1] = Item0 - Item1;
      }
    }
  }
}
```

Пример рассчитан на обработку 32-х разрядных команд. Входной параметр `AStepNum` задает количество шагов преобразования (`AStepNum = 8`). Параметр `AData` получает указатель на массив входных данных размером 256 байт.

Пример реализации функции Адамара на языке Си рассчитан на последовательное выполнение команд, которое характерно для процессоров типа Pentium. В примере нет никакого распараллеливания, поэтому компиляция этого текста в коды процессора NM6403 не приведет к удовлетворительным результатам.

Процессор NM6403 позволяет программно изменять разрядность обрабатываемых данных. Это значит, что задав соответствующее разбиение его рабочей матрицы, участвующей в вычислениях, можно в течении нескольких шагов выполнять преобразования над восьмиразрядными данными, а когда теоретически рассчитанная разрядность результатов потребует выхода за 8 бит, преобразовать данные к 16-ти разрядному виду и продолжить вычисления, и т.д. Максимально возможная разрядность накопителя, реализованная в процессоре NM6403, составляет 64 бита. То есть при разрядности начальных данных 8 бит возможно выполнить 56 шагов преобразования Адамара (при этом потребуется 2^{56}(~10^{17}) 64-ти разрядных слов памяти для размещения вектора результата).

Наличие у процессора матричного вычислительного узла позволяет вести вычисление результата сразу пяти шагов преобразования. Конечно, каждый шаг преобразования может быть реализован отдельно, однако эффективность параллельной обработки нескольких слоев будет несравнимо выше.

Концепция вычислений

Задача преобразования Адамара распадается на две функции. Первая вычисляет параллельно первые три шага преобразования с одновременной конвертацией разрядности данных из 8 в 16 бит. Вторая осуществляет параллельное вычисление следующих 5-ти шагов.

Обе функции написаны на ассемблере и имеют интерфейс, позволяющий вызвать их из программы на Си.

Главная программа на Си заполняет входной буфер начальными значениями. После этого она запускает счетчик тактов и вызывает написанные на ассемблере функции, которые выполняют преобразования над массивом входных данных. После того, как вычисления завершены, программа определяет количество процессорных тактов, затраченных на преобразование, и проверяет контрольную сумму получившегося результата. Если контрольная сумма равна ожидаемой, то программа возвращает количество тактов, затраченных на вычисления, иначе код ошибки (-1).

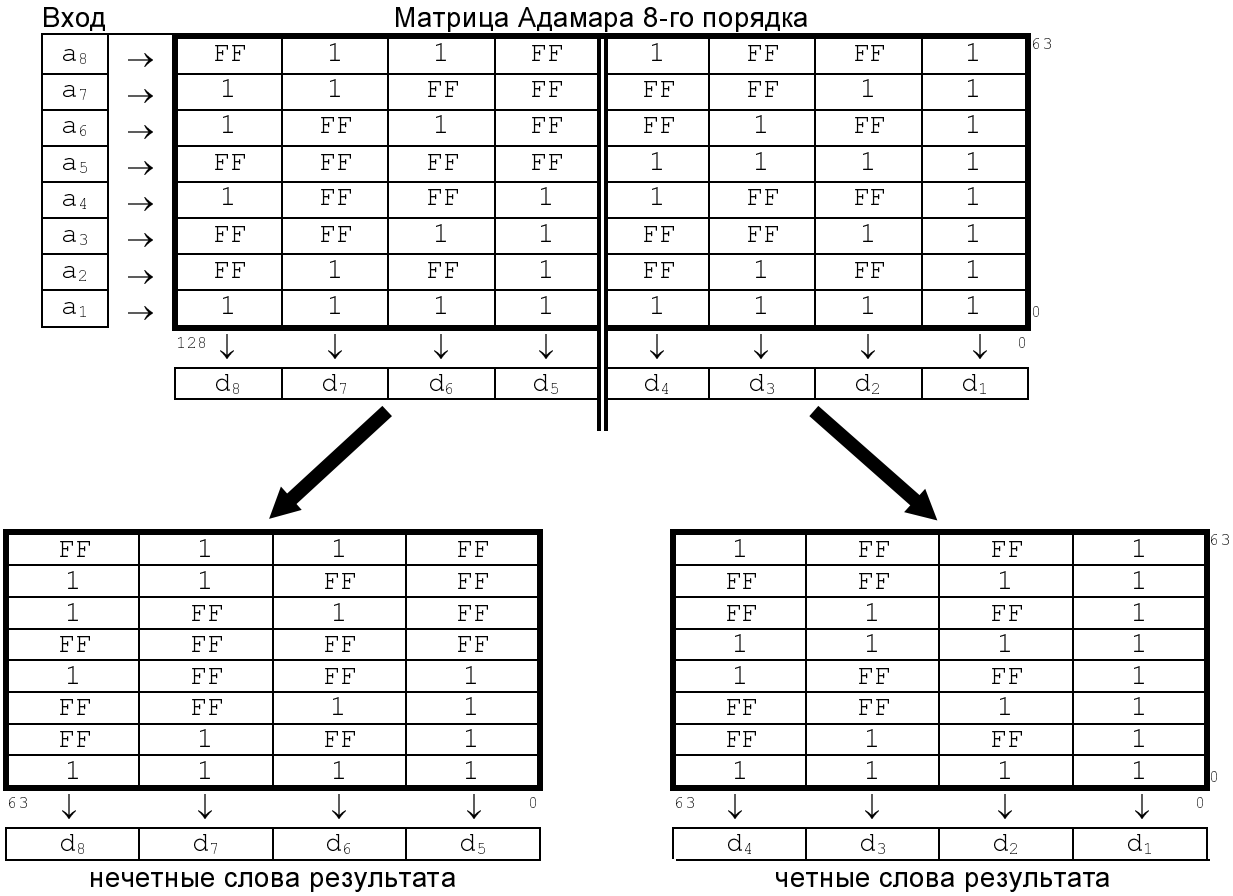
Реализация первых трех шагов преобразования

Поскольку разрядность входных данных 8 бит, а результаты вычислений хранятся в 16-битных словах, не требуется никаких

начальных преобразований над данными, то есть входные данные обрабатываются "как есть". Предполагается, что они упакованы по 8 8-ми разрядных элементов в 64-х разрядном слове. Таким образом, процессор NeuroMatrix® NM6403 обрабатывает параллельно 8 элементов входных данных. Результаты вычислений накапливаются в 16-ти разрядных накопителях, упакованных по 4 в 64-х разрядные слова.

Рассмотрим матрицу Адамара 8-го порядка (Рис. 1, Рис. 3). Она состоит из 8 строк и 8 столбцов. Каждый 8-ми разрядный элемент длинного входного слова умножается на каждую ячейку соответствующей строки матрицы Адамара. Результаты умножений суммируются в пределах каждого столбца. Для того чтобы предотвратить переполнение и потерю данных, необходимо аккумулировать результаты в 16-ти разрядных ячейках. Поэтому общая длина аккумулятора составляет 128 бит.

Рис. 3 Разделение матрицы Адамара 8-го порядка на подматрицы.



Из Рис. 3 видно, что матрица Адамара может быть разделена на две подматрицы. Первая подматрица используется для вычисления четных слов результата, вторая для нечетных.

Каждая подматрица содержит 32 ячейки, поэтому каждый такт процессор NeuroMatrix® NM6403 выполняет 32 MAC (умножения с накоплением).

Вычисления в рассматриваемой функции выполняются в следующей последовательности:

- В Рабочую Матрицу процессора NeuroMatrix® NM6403 загружаются весовые коэффициенты первой подматрицы Адамара;
- Вычисляются четные слова результата. Они накапливаются в `afifo`, глубина которого составляет 32 длинных слова;
- В то же самое время происходит загрузка весовых коэффициентов второй подматрицы Адамара;
- Вычисляется вторая порция результата (нечетные слова) и сохраняется во внешней памяти.

Ниже в данном документе приводится описание каждого шага вычислений из перечисленных выше.

Загрузка рабочей матрицы

Схема вычислений на Матричном Операционном Узле (МОУ) процессора NeuroMatrix® MN6403 приведена на Рис. 4.

Рис. 4 Обработка данных в Рабочей матрице.

a) Первая стадия вычислений (нечетные слова результата)

b) Вторая стадия вычислений (четные слова результата)

В узлах матрицы расположены весовые коэффициенты, предварительно загруженные из памяти. Обрабатываемый поток данных поступает на вход **X**. Входом и выходом при обработке данных являются два буфера, организованных по принципу FIFO, каждый глубиной в 32 64-х разрядных слова. В качестве источника **X** могут использоваться: `ram`, `data`, `afifo`. В качестве приемника результата всегда выступает `afifo`.

Данные в рабочую матрицу заносятся из внешней памяти, доступной процессору. В памяти они хранятся в виде массива 64-х разрядных слов. При загрузке матрицы в нее попадет столько слов, каково ее разбиение по строкам, заданное в регистре `sb2`. Каждая строка задается отдельным 64-х разрядным словом. Слово массива с нулевым

индексом попадает в нулевую строку матрицы. Как видно из Рис. 4, строки рабочей матрицы нумеруются снизу вверх. Такой способ нумерации обусловлен нумерацией битов в слове (нумерация ведется справа налево, младший бит находится справа). В том же направлении ведется увеличение адресов памяти.

Итак, приведем запись массива данных на языке ассемблера, соответствующую блоку коэффициентов, загружаемых в матрицу для выполнения первых трех шагов преобразования Адамара.

```
data ".data"
    M_1_3: long[16] = ( 0000100010001000lhl, // нулевая строка
                        0FFFF0001FFFF0001hl, //младший бит слова
                        0FFFFFFFF0001000lhl, // вторая строка
                        00001FFFFFFFF0001hl,
                        0000100010001000lhl,
                        0FFFF0001FFFF0001hl,
                        0FFFFFFFF0001000lhl,
                        00001FFFFFFFF0001hl, // седьмая строка

                        0000100010001000lhl, // нулевая строка второй
                        0FFFF0001FFFF0001hl, // части весовых кэфф.
                        0FFFFFFFF0001000lhl,
                        00001FFFFFFFF0001hl,
                        0FFFFFFFFFFFFFFFhl,
                        00001FFFF0001FFFFhl,
                        0000100001FFFFFFFhl,
                        0FFFF0001000FFFFhl); // седьмая строка
end ".data";
```

Поскольку рабочая матрица процессора NM6403 разбивается регистром `nb2` на 4 столбца, младшие 16 разрядов нулевого слова массива попадают в нулевой столбец, следующие 16 бит в первый столбец, и тд.

Примечание

Стоит обратить внимание на то, что хотя строки рабочей матрицы нумеруются снизу вверх, в нулевую строку попадет нулевой элемент массива, который при записи массива в столбик оказывается самым верхним.

Весовые коэффициенты читаются из памяти в специальную внутреннюю память нейропроцессора, называемую `wfifo`. Этот блок внутренней памяти также работает по принципу FIFO. Он используется только для загрузки данных в матрицу. `wfifo` обладает одной особенностью в сравнении с остальными внутренними блоками памяти. Он может заполняться постепенно, за несколько операций.

Загрузка данных из памяти в рабочую матрицу описывается следующими командами на языке ассемблера:

```
begin ".text"
<_Steps_1_3>
.branch; // установка бита, разрешающего параллельное выполнение
```

```
                // векторных команд.
            nb1 = 80808080h; // разбиение матрицы на столбцы.
            sb  = 03030303h; // разбиение матрицы на строки.

            ar0 = M_1_3;     // адрес массива весов
            rep 8 wfifo = [ar0++], ftw, wtw; // загрузка весов
            ...
        end "text_0_2";
```

Сначала заполняются регистры `nb1` и `sb`, определяющие будущее теневой матрицы. Регистры связаны с теневой матрицей. Реально разбиение, задаваемое ими, вступит в силу не сразу после присваивания им новых значений, а только после выполнения команды `wtw`, которая скопирует содержимое теневой матрицы в рабочую.

Регистры `nb1` и `sb` являются 64-х битными. Если они инициализируются 32-х разрядной константой, то процессор копирует это значение в старшие 32 бита, то есть получается, что регистр nb1 инициализирован длинной константой `8080808080808080h1`. То же верно и для регистра `sb`. Более подробно работа с регистрами `nb1` и `sb(sb1, sb2)` описана в [3].

Итак, определено будущее разбиение рабочей матрицы.

В адресный регистр заносится адрес массива весовых коэффициентов, а затем данные загружаются из памяти в `wfifo`.

По команде `ftw` данные из `wfifo` попадают в теневую матрицу. Эта передача занимает всегда 32 такта, независимо от того, сколько слов загружается в матрицу. Например, в случае загрузки матрицы Адамара восьмого порядка требуется загрузка восьми слов, но их перекодировка во внутреннее представление независимо от этого длится 32 такта. Однако, перекодировка ведется параллельно с закачкой весов и начинается с момента появления первого слова в `wfifo`.

После того, как закачка весов в теневую матрицу завершена, выполняется команда `wtw`. Она за один такт переписывает содержимое теневой матрицы в рабочую. При этом значения регистров `nb1` и `sb1` копируются в `nb2` и `sb2`. Загрузка рабочей матрицы завершена.

Выполнение вычислений

Далее приводится текст программы вычисления первых трех шагов преобразования Адамара:

```
// Секция исходного кода
begin ".text"
// Функция выполняет 3 шага преобразования Адамара.
<_Steps_1_3>
.branch; // установка бита параллельного выполнения векторных команд.
    ...
    // Загружается константа разбиения теневой матрицы на столбцы.
    gr0 = 80008000h;
```

```
    // Константа разбиения на столбцы копируется в nb1 | gr4 = 0.
    nb1 = gr0 with gr4 = false;
    // Загружается константа разбиения теневой матрицы на строки.
    sb  = 03030303h;
    // Загрузка адреса массива весовых коэффициентов | gr4 = 1;
    ar0 = M_1_3 with gr4++;
    // В wfifo загружаются 16 длинных слов, 8 из них перекодируются в
    // теневую матрицу, а затем содержимое теневой копируется в рабочую
    // матрицу.
    rep 16 wfifo = [ar0++], ftw, wtw;
    // Загрузка адреса входных данных | gr4 = 2;
    ar0 = [--ar5] with gr4 <<= 1;
    // Загрузка адреса буфера результатов | gr5 = 4; <- величина
    //                                                инкрементации адреса
    ar4 = [--ar5] with gr5 = gr4 << 1;
    // ar5 указывает на адрес буфера результатов + 2
    // (следующее длинное слово).
    ar5 = ar4 + gr4 with gr4 = gr5;
    //  Загрузка входных данных, выполнение вычислений и передача следующих
    // 8 длинных слов из wfifo в теневую матрицу.
    rep 32 ram = [ar0++],ftw with vsum , data, 0;
    //  Сохранение результатов в памяти. // Две последних инструкции
    rep 32 [ar4++gr4] = afifo;          // выполняются в параллель.

    // Эта часть кода используется для обхода процессорной ошибки.
.wait; // снятие бита параллельного выполнения векторных команд.
    // В регистр nb1 копируется его старое содержимое для блокировки
    // выполнения wtw пока ftw не завершит перекодировку весов.
    nb1 = gr0;
    // Копирование содержимого теневой матрицы в рабочую.
    wtw;
.branch; // установка бита параллельного выполнения векторных команд.
    // Выполнение второй части вычислений.
    rep 32 with vsum , ram, 0;          // Две векторных инструкции
    // Сохранение результата в памяти  // выполняются в параллель.
    rep 32 [ar5++gr5] = afifo;          //
    // Возвращение из подпрограммы.
    return;
end ".text";
```

После того, как рабочая матрица загружена, можно приступать непосредственно к вычислениям. Для этого необходимо задать адреса входного и выходного буферов. Желательно, чтобы эти буфера располагались в разных блоках памяти, находящихся на разных шинах

данных. В этом случае операции чтения входного буфера и запись выходного могут выполняться параллельно.

Каждая векторная инструкция содержит внутренний счетчик циклов, который количество выполняемых элементарных циклов в пределах от 1 до 32. Эти элементарные циклы описывают выполнение одной и той же операции над потоком данных. Максимальное количество данных, обрабатываемых одной векторной командой, определяется глубиной внутренних буферов памяти процессора NeuroMatrix® NM6403, которая равна 32 64-х разрядных слова.

Размер обрабатываемых буферов данных в рассматриваемой задаче достаточно мал для организации каких-либо внешних циклов.

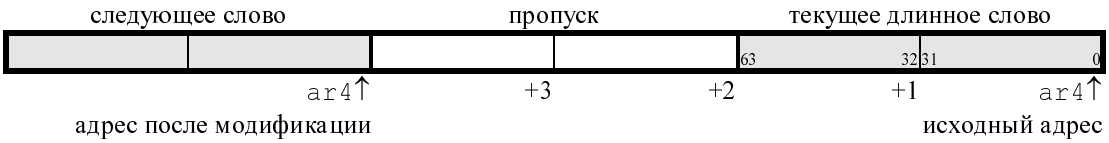
Некоторые строки ассемблерного текста, приведенного выше, требуют дополнительных комментариев.

Строка

```
rep 32 [ar4++gr4] = afifo;
```

задает сохранение массива результатов в памяти с одновременной модификацией адресного регистра. Ранее в программе регистру `gr4` было присвоено значение 4. Это значит, что каждый раз при модификации значение регистра `ar4` увеличивается на 4, что поясняется на Рис. 5.

Рис. 5 Модификация адресного регистра в векторной команде.

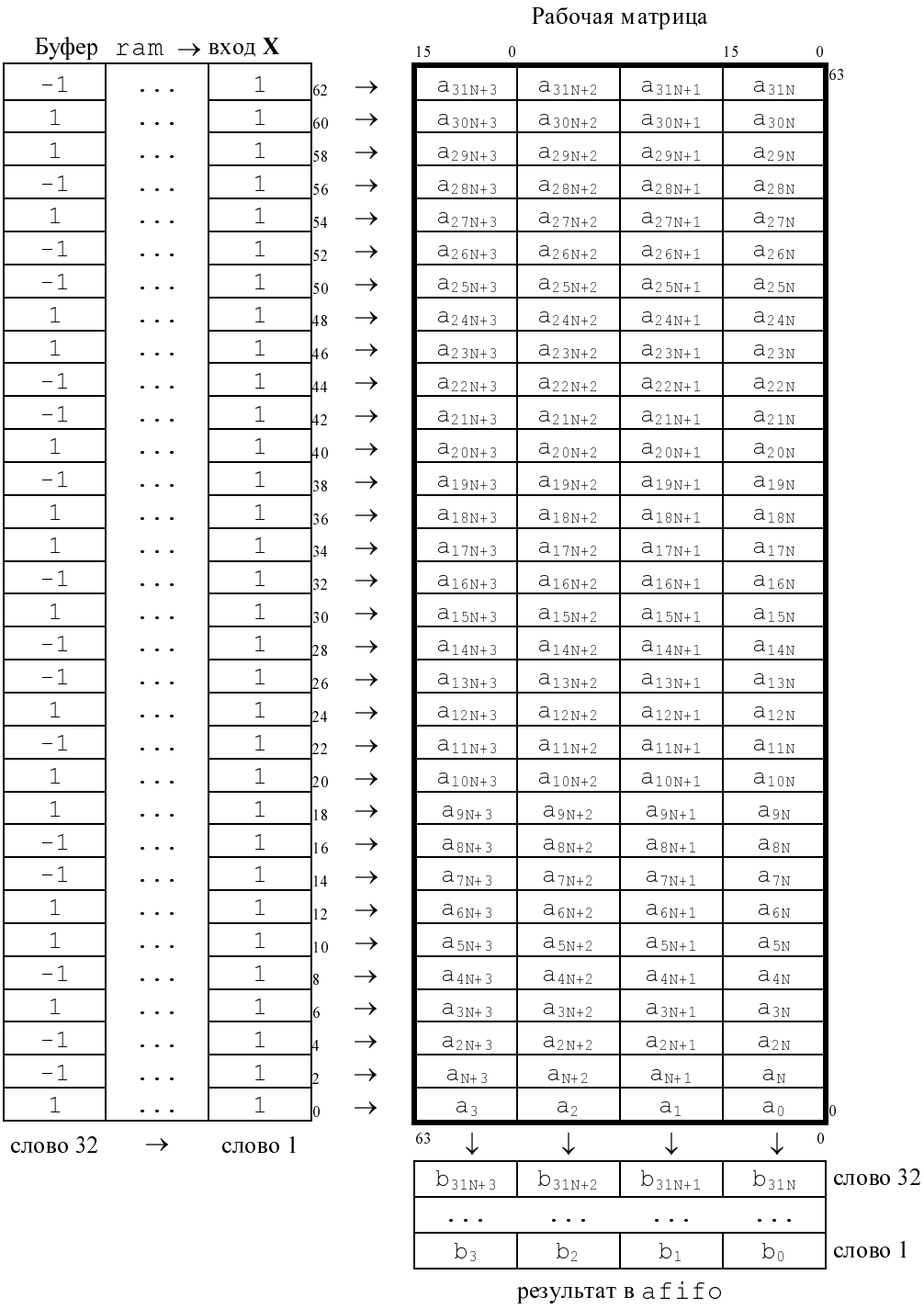


Процессор NeuroMatrix® NM6403 адресуется к 32-х разрядным и к 64-х разрядным словам. Для того, чтобы адресоваться к 64-х разрядным словам с модификацией адресного регистра, в процессоре существует два метода адресации. Если необходимо адресоваться к соседнему 64-х разрядному слову, то можно использовать адресацию с простой инкрементацией: `[ar0++]`. В векторных инструкциях это всегда обозначает увеличение значение регистра на 2, то есть к соседнему длинному слову. Если необходимо использовать другой регулярный способ адресации, то величину инкремента необходимо поместить в регистр общего назначения парный адресному: `[ar4++gr4]`. Регистр общего назначения должен иметь только четное значение, для того чтобы адресоваться по четным адресам. Например, если необходимо адресоваться к следующему слову, находящемуся через одно длинное слово, необходимо инициализировать gr4 числом 4. Этот метод адресации поясняется на Рис. 5.

Реализация следующих пяти шагов преобразования Адамара

Как уже упоминалось, преобразование Адамара содержит только операции сложения и вычитания. Таким образом, возможно разбить рабочую матрицу процессора на 32 строки, так чтобы разрядность данных, поступающих на вход **X** равнялась 2 битам. Этого достаточно для хранения чисел `-1`, `0` и `1`.

Рис. 6 Матричный узел процессора в режиме обработки 128 элементов.



Разбиение матрицы на 32 строки позволяет выполнять в каждом столбце по 32 операции сложения/вычитания. При обработке 16-ти

разрядных данных матрица разбивается на 4 столбца, поэтому можно за один процессорный такт выполнить 128 операций сложения/вычитания (см. Рис. 6).

Основной вопрос состоит в том, как обеспечить взаимодействие всех 32-х строк матрицы. Его решение содержится в следующей таблице:

Табл. 2 Связи между элементами пяти шагов преобразования Адамара.

Шаг 3	Шаг 4	Шаг 5	Шаг 6	Шаг 7	Шаг 8
1	1 + 2	1 + 3	1 + 5	1 + 9	1 + 17
2	1 - 2	2 + 4	2 + 6	2 + 10	2 + 18
3	3 + 4	1 - 3	3 + 7	3 + 11	3 + 19
4	3 - 4	2 - 4	4 + 8	4 + 12	4 + 20
5	5 + 6	5 + 7	1 - 5	5 + 13	5 + 21
6	5 - 6	6 + 8	2 - 6	6 + 14	6 + 22
7	7 + 8	5 - 7	3 - 7	7 + 15	7 + 23
8	7 - 8	6 - 8	4 - 8	8 + 16	8 + 24
9	9 + 10	9 + 11	9 + 13	1 - 9	9 + 25
10	9 - 10	10 + 12	10 + 16	2 - 10	10 + 26
11	11 + 12	9 - 11	11 + 15	3 - 11	11 + 27
12	11 - 12	10 - 12	12 + 16	4 - 12	12 + 28
13	13 + 14	13 + 15	9 - 13	5 - 13	13 + 29
14	13 - 14	14 + 16	10 - 16	6 - 14	14 + 30
15	15 + 16	13 - 15	11 - 15	7 - 15	15 + 31
16	15 - 16	14 - 16	12 - 16	8 - 16	16 + 32
17	17 + 18	17 + 19	17 + 21	17 + 25	1 - 17
18	17 - 18	18 + 20	18 + 22	18 + 26	2 - 18
19	19 + 20	17 - 19	19 + 23	19 + 27	3 - 19
20	19 - 20	18 - 20	20 + 24	20 + 28	4 - 20
21	21 + 22	21 + 23	17 - 21	21 + 29	5 - 21
22	21 - 22	22 + 24	18 - 22	22 + 30	6 - 22
23	23 + 24	21 - 23	19 - 23	23 + 31	7 - 23
24	23 - 24	22 - 24	20 - 24	24 + 32	8 - 24
25	25 + 26	25 + 27	25 + 29	17 - 25	9 - 25
26	25 - 26	26 + 28	26 + 30	18 - 26	10 - 26
27	27 + 28	25 - 27	27 + 31	19 - 27	11 - 27
28	27 - 28	26 - 28	28 + 32	20 - 28	12 - 28
29	29 + 30	29 + 31	25 - 29	21 - 29	13 - 29
30	29 - 30	30 + 32	26 - 30	22 - 30	14 - 30
31	31 + 32	29 - 31	27 - 31	23 - 31	15 - 31
32	31 - 32	30 - 32	28 - 32	24 - 32	16 - 32

Примечание *В таблице цифры в каждом столбце обозначают номера ячеек предыдущего шага.*

Из Табл. 2 видно, что возможно выразить значения элементов, получаемых на 8-ом шаге через элементы, вычисленные на 3-ем шаге.

Например, рассмотрим, как значение 7-ой ячейки 8-го шага, выраженное через значения ячеек 3-го шага (см. выделенные ячейки).

Значение 7-ой ячейки 8-го шага записывается, как сумма значений 7-ой и 23-ей ячеек шага 7, каждая из которых в свою очередь выражается через значения ячеек 6-го шага, и т д.

В результате значение 7-ой ячейки 8-го шага выражается через значения ячеек 3-го шага следующим образом:

$$b_7 = a_1+a_2-a_3-a_4-a_5-a_6+a_7+a_8+a_9+a_{10}-a_{11}-a_{12}-a_{13}-a_{14}+a_{15}+a_{16}+ a_{17}+a_{18}-a_{19}-a_{20}-a_{21}-a_{22}+a_{23}+a_{24}+a_{25}+a_{26}-a_{27}-a_{28}-a_{29}-a_{30}+a_{31}+a_{32}$$

То же самое может быть выражено 64-х разрядной константой:

`05FF55FF55FF55FF5hl`

где под каждый знак отведено 2 бита.

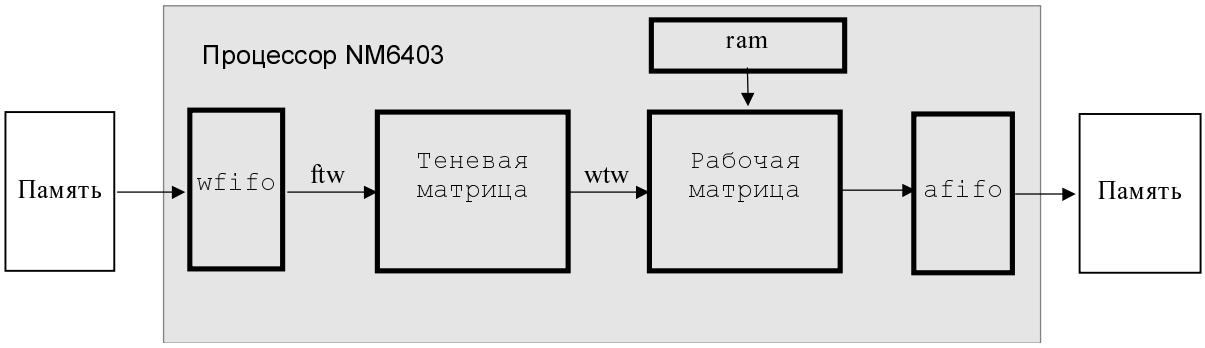
Схема вычислений, используемая в рассматриваемой функции отлична от той, что была использована при выполнении первых трех шагов. Основное отличие состоит в том, что в рабочую матрицу загружаются не весовые коэффициенты, а данные, тогда как веса подаются на вход **X**.

Выполнение параллельного вычисления 5-ти шагов преобразования Адамара осуществляется по следующему сценарию:

- Загрузка весовых коэффициентов из внешней памяти в блок `ram`;
- Загрузка четных слов входных данных в `wfifo` → Тен. Матр. → Раб. Матр.;
- Загрузка нечетных слов входных данных в `wfifo` → Тен. Матр. с одновременным выполнением вычислений над четными словами;
- Сохранение четных слов результата в памяти;
- Выполнение вычислений над нечетными словами и параллельное сохранение вычисленных слов результата в памяти.

Схема обработки данных, используемая в данной задаче, может быть представлена следующим образом:

Рис. 7 Схема обработки данных в процессоре NM6403 при параллельном вычислении пяти шагов преобразования Адамара.



Выполнение вычислений

Далее приводится текст программы вычисления пяти шагов преобразования Адамара:

```
// Секция исходного кода
begin ".text"
<_Steps_4_8>
.branch; // установка бита параллельного выполнения векторных команд.
    // Загрузка адреса массива весовых коэффициентов | gr0 = 0.
    ar6 = M_4_8 with gr0 = false;
    // Загружается константа разбиения теневой матрицы на столбцы.
    gr2 = 80008000h;
    // Загружается константа разбиения теневой матрицы на строки.
    sb  = 0FFFFFFFFh;
    // Константа разбиения на столбцы копируется в nb1 | gr0 = 1;
    nb1 = gr2 with gr0++;
    // Загрузка адреса входных данных | gr0 = 2;
    ar0 = [--ar5] with gr0 <<= 1;
    // ar1 указывает на адрес буфера входных данных+2
    // (следующее длинное слово)| gr1 = 4
    ar1 = ar0 + gr0 with gr1 = gr0 << 1;
    // Загрузка адреса буфера результатов | gr4 = 2;
    ar4 = [--ar5] with gr4 = gr0;
    // ar5 указывает на адрес буфера результатов + 2
    // (следующее длинное слово) | gr5 = 4
    ar5 = ar4 + gr4 with gr5 = gr1;
    // gr0 = 4 | gr4 = 4 <- Инкремент для доступа к (не)четным словам.
    gr0 = gr1 with gr4 = gr1;

    // Загрузка входных данных в wfifo -> Тен.Матр. -> Раб.Матр.
    rep 32 wfifo = [ar0++gr0], ftw, wtw;
    // Загрузка весов ram;  // Эти две инструкции
    rep 32 ram = [ar0++];    // выполняются в параллель.

    // Загрузка следующей части входных данных в wfifo -> Тен.Матр.
    // и параллельное выполнение вычислений.
    rep 32 wfifo = [ar1++gr1], ftw with vsum , ram, 0;
    // Сохранение результатов в. // These two instructions
    rep 32 [ar4++gr4] = afifo;  // are executed in parallel.

// Эта часть кода используется для обхода процессорной ошибки.
.wait; // снятие бита параллельного выполнения векторных команд.
```

```
        // В регистр nb1 копируется его старое содержимое для блокировки
        // выполнения wtw пока ftw не завершит перекодировку весов.
        nb1 = gr2;
        // Копирование содержимого теневой матрицы в рабочую.
        wtw;
    .branch;  // установка бита параллельного выполнения векторных команд.

        // Выполнение вычислений.
        rep 32 with vsum , ram, 0;          // Две векторных инструкции
        // Сохранение результата в памяти   // выполняются в параллель.
        rep 32 [ar5++gr5] = afifo;          //

        // Возвращение из подпрограммы.
        return;
    .wait;
    end ".text";
```

Полный набор исходных текстов данного приложения и утилиты его сборки в каталоге NEURO\EXAMPLES\FHT. Этот каталог появляется при установке базового ПО процессора NeuroMatrix® NM6403.

Другой путь получения исходных текстов - обратиться на WEB-site НТЦ Модуль по адресу: `www.module.ru`.

Процессор NeuroMatrix® NM6403 имеет архитектуру, удобную для выполнения преобразования Адамара. На выполнение 256-ти точечного преобразования он расходует 349 тактов, включая затраты на вызовы Си-функций.

Для вычисления 256-ти точечного быстрого преобразования Адамара по основанию 2 требуется 2048 операций сложения/вычитания. Из этого следует, что эффективная скорость выполнения алгоритма на процессоре составляет:

(8 операций * 256 элементов) / 349 тактов = ~5.8 арифметических операции за один такт.

Однако, реальная картина вычислений значительно отличается от теории. И на первом шаге в силу архитектуры процессора быстрое преобразование выполнялось по основанию 8, а на втором по основанию 32, поэтому суммарно процессору пришлось сделать больше операций сложения/вычитания, нежели было подсчитано выше.

Если рассмотреть те операции, которые необходимо совершить описанном подходе, то в первой функции `Steps_1_3` на каждый элемент выполняется 8 сложений/вычитаний, а во второй функции `Steps_4_8` для получения элемента результата выполняется 32 сложения/вычитания. Итого, 8+32 = 40 операций сложения/вычитания на каждый из 256 элементов, что составляет:

(40 операций * 256 элементов) / 349 тактов = ~29.3 арифметических операции за один такт.

Табл. 3 Производительность NeuroMatrix® NM6403 на задаче 256-ти точечного быстрого преобразования Адамара.

Тип	Количество арифм. операций за такт	Количество арифм. операций в секунду (50МГц)
Эффективная производительность	5.8	$2,9*10^8$
Общая производительность	29.3	$1,4*10^9$

По результатам испытаний приводится сравнительная таблица времен, затраченных на выполнение 21-но шагового преобразования Адамара:

Процессор	Частота	Время выполнения
Pentium II	300 MHz	2.58 сек
NM6403	40 MHz	0.42 сек
Alfa 21164	533 MHz	0.33 сек

Литература

1. НТЦ Модуль. "Процессор NeuroMatrix® NM6403. Введение в архитектуру." http://www.module.ru/files/archover.pdf
2. С. Кун. "Матричные процессоры на СБИС". Москва, Мир, 1991г., стр. 86-87.
3. НТЦ Модуль. "ПО процессора NeuroMatrix® NM6403. Описание языка ассемблера".

АКЦИОНЕРНОЕ ОБЩЕСТВО
НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

Научно-технический центр Модуль
АЯ 166, Москва, 125190, Россия
Тел: +7 (095) 152-9335
Факс: +7 (095) 152-4661
E-Mail: postmast@module.ru
WWW: http://www.module.ru

Напечатано в России. Дата издания: 08.04.99