УДК 004.383.3 / 519.684.6

ВЫЧИСЛИТЕЛЬНЫЕ РЕСУРСЫ ПРОЦЕССОРОВ NEUROMATRIX С ПЛАВАЮЩЕЙ ТОЧКОЙ В ЗАДАЧАХ ОБРАБОТКИ БОЛЬШИХ ПОТОКОВ ДАННЫХ

Мушкаев Сергей Викторович, Бродяженко Андрей Владимирович,
Болотников Александр Александрович

ЗАО НТЦ "Модуль", Москва

Аннотация:

Данная статья посвящена демонстрации принципов распараллеливания вычислений на процессоре NM6407 при работе с большими потоками данных. В вводной части доклада рассматривается структура векторного узла процессора NeuroMatrix NM6407 с плавающей точкой. В схематичной форме производится обзор вычислительных ресурсов и режимов работы векторного процессора с данными разного формата. Рассматривается распределенная структура памяти и шин данных, которая обеспечивает параллелизм и высокий темп загрузки вычислительных узлов.

В основной части доклада, на примере простейших базовых задач линейной алгебры, демонстрируется поведение процессора в динамике. Приводятся временные диаграммы и пошаговые алгоритмы действий. Раскрывается ряд особенностей, которые необходимо учитывать для сбалансированной и эффективной загрузки вычислительных ячеек во времени. В частности, раскрывается порядок обращения к памяти, работа с векторными регистрами и взаимодействие между ячейками.

В заключении демонстрируется подход к реализации более сложных алгоритмов, таких как быстрое преобразование Фурье. Изучается производительность и эффективность процессора для разного класса задач.

Ключевые слова: поточная обработка данных, параллельные вычисления, БПФ, BLAS, векторно-матричное умножение, процессорное ядро цифровой обработки сигналов NeuroMatrix, NM6407, NM6408.

COMPUTING RESOURCES OF THE FLOATING POINT NEUROMATRIX PROCESSORS IN PROCESSING BIG DATA STREAMS.

Sergey Mushkaev, Andrew Brodyazhenko,
Alexander Bolotnikov

Research Centre “Module”, Moscow

Abstract:

This article is devoted to demonstrating the principles of parallel computing on the processor NM6407 when operation with large data streams. In the introductory part of the report, the structure of the vector node of the NeuroMatrix NM6407 processor with a floating point is considered. In a schematic form, the computing resources and operating modes of the vector processor with data of different formats are reviewed. A distributed memory and data bus structure is considered, which ensures the parallelism and high rate of loading of computing nodes.

In the main part of the report, on the example of the simplest basic problems of linear algebra, the behavior of the processor in dynamics is demonstrated. Time diagrams and step-by-step action algorithms are given. A number of features that need to be considered for a balanced and efficient loading of computational cells in time are revealed. In particular, the procedure for accessing memory, working with vector registers and interaction between cells is disclosed.

The conclusion demonstrates an approach to the implementation of more complex algorithms, such as fast Fourier transform. The processor's performance and efficiency are studied for a different class of tasks.

Keywords: stream processing, parallel computation, FFT, BLAS, vector-matrix multiplication, processor core of digital signal processing (DSP) NeuroMatrix, NM6407, NM6408.

Введение

Современные архитектуры процессоров обладают сложной распределенной системой памяти, могут иметь гетерогенную структуру с несколькими процессорами. Каждый из процессоров может включать несколько сложно-функциональных вычислительных узлов. Для эффективного программирования специализированных процессоров необходимо учитывать структуру и принципы взаимодействия отдельных узлов.

В данной статье будут рассматриваться вопросы программирования задач по обработке больших потоков данных с плавающей точкой на процессоре NM6407 производства НТЦ Модуль. Будут описаны основные

вычислительные узлы, режимы и особенности их использования в наиболее типовых задачах, а также принципы распараллеливания, как по вычислительной нагрузке, так и по взаимодействию с памятью.

Архитектура процессора

Рассмотрим процессор, начиная от обобщенной структуры и заканчивая уровнем элементарных базовых операций. На рис.1 представлена структурная модель процессора: два независимых процессорных узла с плавающей и фиксированной точкой NMPU0 и NMPU1. Каждый узел имеет собственную накристалльную однотактовую память, DMA-контроллер и DDR-интерфейс с внешней памятью. Доступ в память осуществляется по 64-разрядным шинам. Обмен данным между процессорными узлами может осуществляться через общую разделяемую память (Shared Memory 0, 1).

Процессорный узел NMPU0 с плавающей точкой состоит из управляющего RISC-ядра и 4-ых вычислительных ячеек (FPU Cell0-Cell3). Целочисленный процессорный узел NMPU1 также включает свое RISC ядро и матрично-векторный умножитель [1].

На рис.2 показана схема процессорного узла NMPU0. Внутренняя память разделена на 8 банков по 128 кБ псевдо-двухпортовой памяти, к которым может быть осуществлен одновременный внешний доступ (например, со стороны DMA или коммуникационных портов) и обращение со стороны процессорного ядра. Память и процессорные ячейки объединены между собой коммутатором (switch 9→11), обеспечивающим параллельное обращение сразу к нескольким банкам памяти. Как видно из рисунка 2, каждая процессорная ячейка имеет две входных шины и одну выходную, что позволяет ячейке параллельно принимать два потока данных и отдавать один. Процессорные ячейки осуществляют доступ в память через шинный коммутатор 9→11. В общей сложности на все 4 ячейки (Cell0-Cell3), коммутатор обеспечивает максимум 4 одновременных доступа на чтение из памяти и 2 на запись.

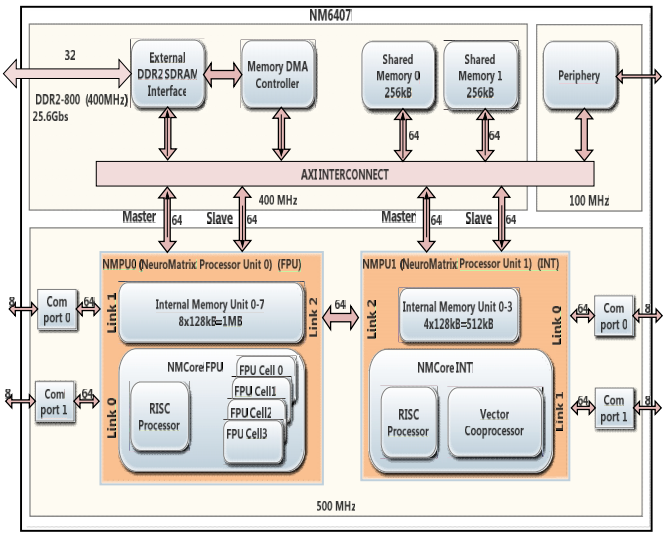

Рисунок 1. Программная модель процессора NM6407

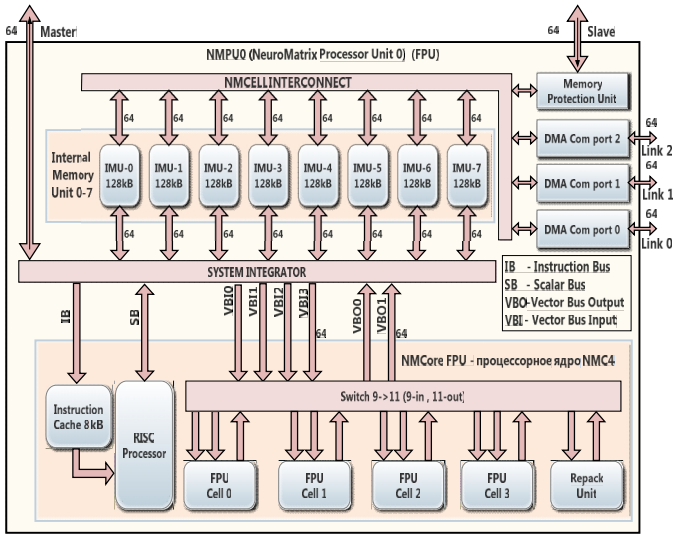

Рисунок 2. Процессорная система NMPU0

Каждая ячейка FPU Cell0–FPU Cell3 состоит из операционного узла и восьми векторных регистров вмещающих до 32-х 64-разрядных регистровых ячеек (см. рис. 3). В зависимости от режима работы операционного узла каждая регистровая ячейка (reg0, reg1…reg31) векторного регистра интерпретируется либо как 64-разрядное число с плавающей точкой двойной точности, либо как пара чисел одинарной точности, либо как комплексное число одинарной точности. На рис.3 показано, как векторные регистры выступают в качестве входных и выходных операндов для векторных инструкций операционного узла (Operation unit). Коммутатор 8→4 позволяет одновременно выбрать любые 3 регистра в качестве входных операндов, а из 4-го производить выгрузку в память. Коммутатор 3→8 обеспечивает на фоне вычислений параллельную загрузку двух регистров и одного регистра с выхода вычислителя по обратной связи новой порцией данных. Пересылки могут быть как между памятью и регистрами, так и между регистрами процессорных ячеек. Данные из ячеек (reg0, reg1.. reg31) векторных регистров выбираются последовательно с темпом в один такт и обрабатываются заданной инструкцией.

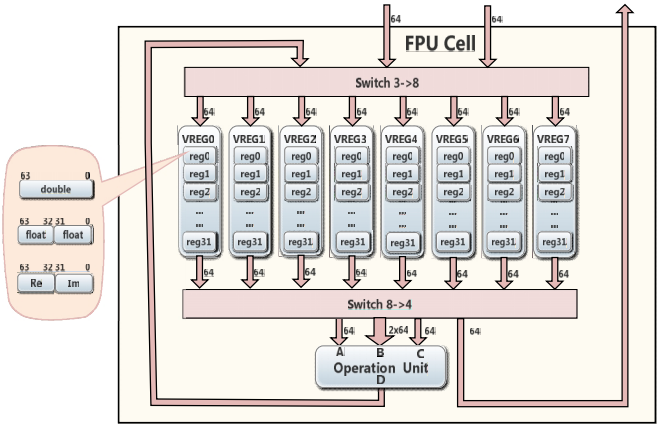

Рисунок 3. Процессорная ячейка FPU

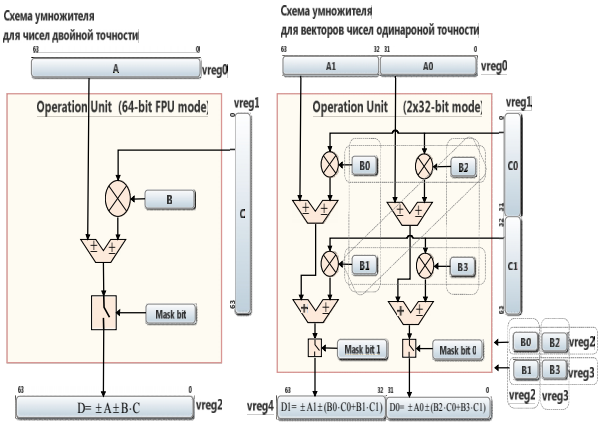

Рисунок 4. Структура операционного узла
а)- (слева) для чисел двойной то
б) – (справа) для чисел одинарной точности

Как показано на рис.4, операционный узел работает с векторами трех типов: данные двойной точности, комплексные числа одинарной точности и числа одинарной точности. Операционный узел выполняет над векторными регистрами SIMD операции сложения, вычитания, умножения, взятие модуля, выделение знака, маскирования, а также операцию вида A * x + b. В режиме комплексных чисел (мнимая и действительная часть по 32 бита) или в режиме 64-разрядных чисел двойной точности A, x, b – это обычные числа. В случае одинарной точности A является матрицей 2x2, x и b-вектора из двух 32-разрядных элементов с плавающей точкой. Выход с операционного вычислителя замкнут на вход через коммутатор 3→8, что позволяет накапливать результат вычислений в векторных регистрах.

В каждом процессорном такте происходит последовательная выборка данных из ячеек (reg0, reg1.. reg31) векторных регистров (vreg0..vreg8), между которыми производится заданное командой действие. Операции между векторными регистрами могут выполняться как поэлементно (например, при суммировании двух векторных регистров), так и выборочно, где в качестве второго операнда используется не весь векторный регистр, а только одна регистровая ячейка (например, при прибавлении к вектору числа).

В режиме двойной точности или в комплексном режиме в каждом такте производится операция над отдельными числами. В режиме одинарной точности операционный узел также может выполнять умножение с накоплением матрицы 2x2 на вектор из 2 элементов. Схематично процесс умножения двух чисел с накоплением показан на рис. 4a. На рисунке 4б показана схема умножителя в случае режима одинарной точности, где в весовые коэффициенты матрицы 2x2 (B0, B1, B2, B3) загружаются числа из пары векторных регистров [vreg2, verg3].

Существует четыре способа загрузки коэффициентов в матричный умножитель (см. рис.5):

- вертикальная загрузка, (коэффициенты [B0, B1] лежат смежно в vreg2 , [B2, B3] лежат смежно в vreg3)
- горизонтальная загрузка (коэффициенты [B0, B2] лежат смежно в vreg2 , [B1, B3] лежат смежно в vreg3)
- диагональная загрузка (коэффициенты [B1, B2] лежат смежно в vreg2)
- загрузка для комплексного умножения (коэффициенты [B.im,B.re] лежат смежно в vreg2)

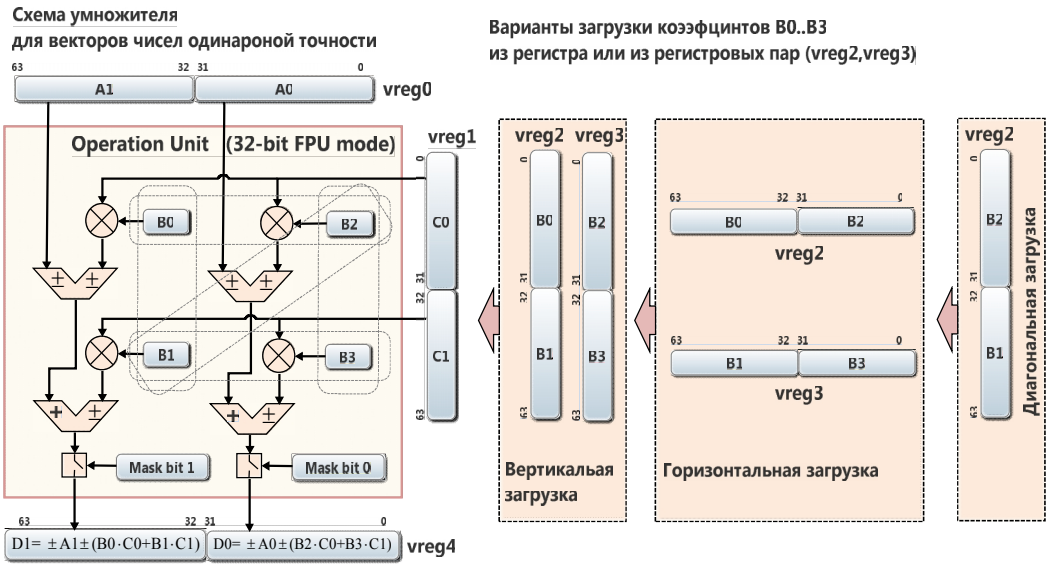

Рисунок 5. Способы загрузки умножителя

На рис 6. показаны варианты загрузки в весовые коэффициенты матричного умножителя и схема умножения. В зависимости от режима вычислитель осуществляет различные математические операции. Вертикальная загрузка позволяет выполнять транспонирование или умножать матрицу на вектор-столбец. Горизонтальная загрузка – умножение вектор-строку на матрицу. При диагональной загрузке осуществляется поэлементное умножение. Специальный инвертирующий вход сумматора на рис. 6в позволяет осуществлять комплексное умножение.

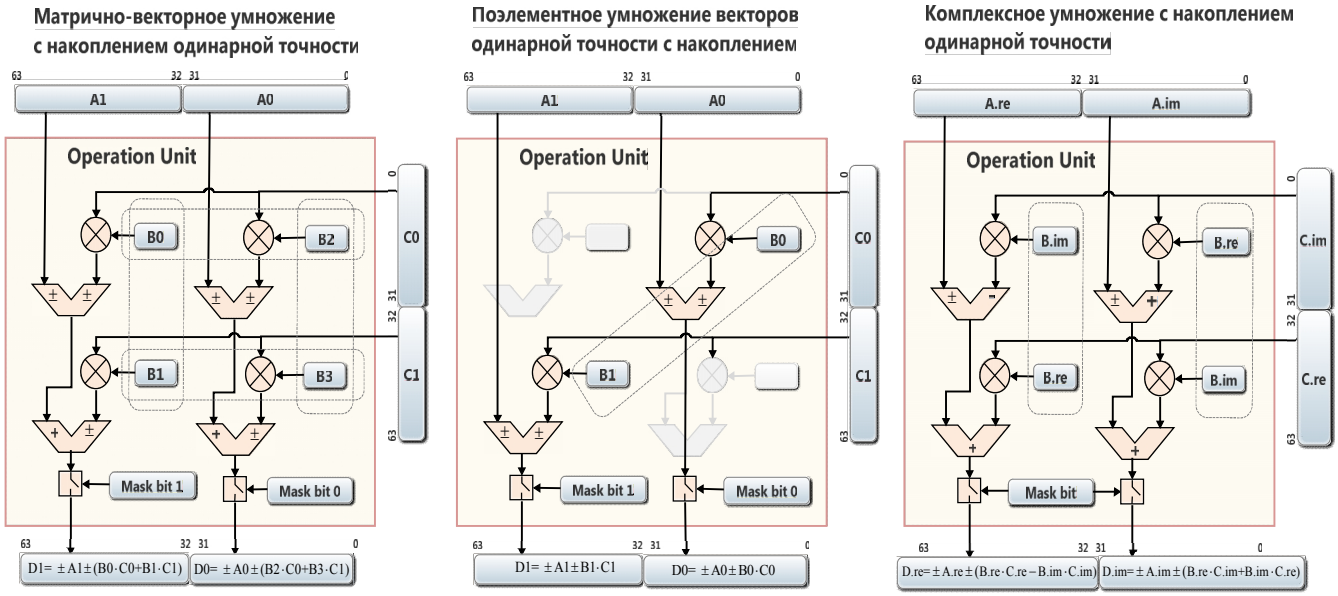

Рисунок 6. Операции умножения

Умножение матрицы на вектор-столбец

Рассмотрев элементарную команду умножения, распространим процесс до полного умножения матрицы на вектор произвольного размера. Не смотря на то, что процессор поддерживает SIMD инструкции с числом повторений до 32, для упрощения опишем процесс умножения, используя 4 векторных регистра глубиной только одно 64-разядное слово:

- Регистр, накапливающий результат (пара чисел [D_{00}, D_{01}] на рис.7)
- Регистр, содержащий данные вектор-столбца A ([A_{20}, A_{30}] на рис.7)
- Регистровая пара для хранения матрицы 2x2 коэффициентов B ([B_{02}, B_{03}] и [B_{12}, B_{13}])

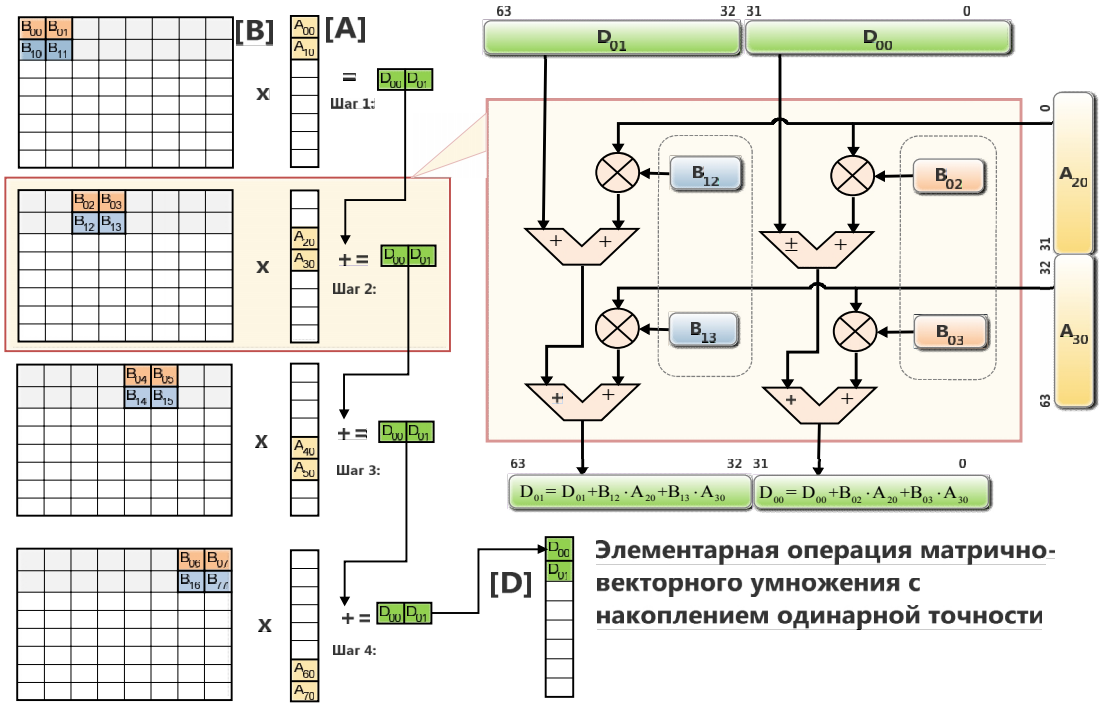

Рисунок 7. Умножение матрицы 8x2 на вектор-столбец

На рис. 7 показаны матрица B размером 8x8 и вектор A из 8 элементов в памяти и порядок загрузки данных в векторные регистры (регистры обозначены цветными овальными прямоугольниками) с последующим умножением и накоплением в выходной регистр. Процесс вычисления пары чисел D_{00}, D_{01} состоит из 4 шагов. На рисунке 7 справа развернута схема умножения второго шага. Если повторить данную операцию аналогично еще 3 раза, получится полное умножение матрицы на вектор. Отметим, что загрузка множителей B происходит вертикально (обозначено пунктиром). Другими словами, матрица коэффициентов 2x2 загружается в умножитель транспонированной, что математически позволяет реализовать умножение матрицы на вектор-столбец.

Умножение вектор-строки на матрицу.

Для осуществления умножения вектор-строки на матрицу загрузка коэффициентов матрицы B производится горизонтально по две пары чисел: [B_{20}, B_{21}] и [B_{31}, B_{30}] (объединены пунктиром на рис. 8). Процесс вычисления пары чисел [D_{00}, D_{01}] осуществляется также за четыре шага и показан на рис.8.

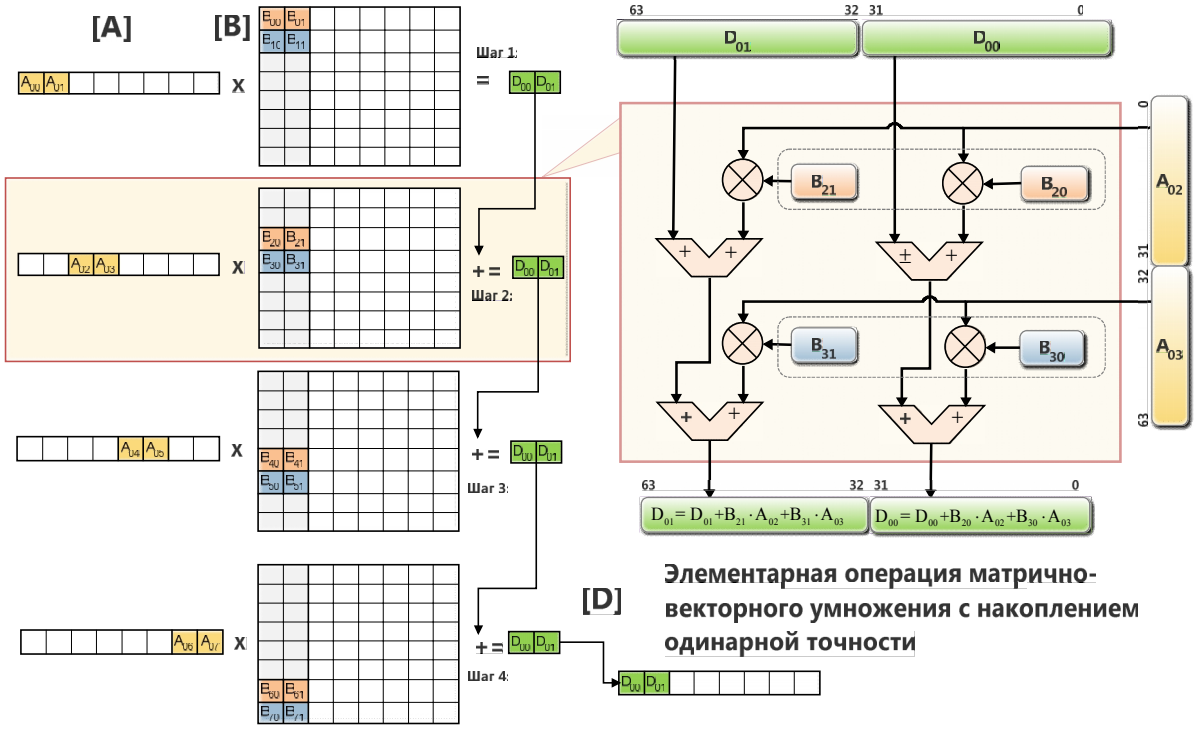

Рисунок 8. Умножение вектор-строки на матрицу 2x8 одинарной точности

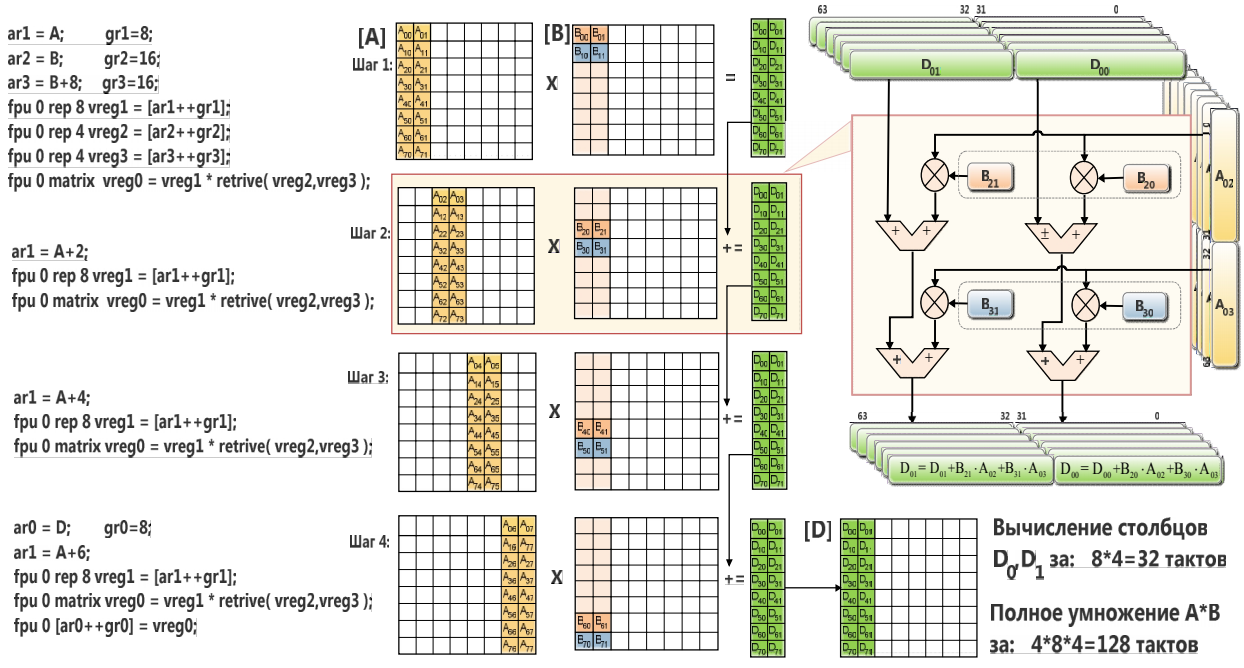

Рисунок 9. Последовательность операций матричного умножения

Оптимизация работы с памятью

В приведенных примерах присутствуют три потока данных: два входных для чтения A и B и один выходной для записи результата D . Поскольку на каждом шаге умножения требуется две пары чисел матрицы B, то на подкачку данных уходит 2 такта. Если данные массивов A, B и D размещены по разным банкам памяти, то обращение к ним будет параллельно и умножение займет 2 такта. Однако, если матрицу B разбить на четные и нечетные строки и хранить их в разных банках, то загрузку можно ускорить до одного такта.

В данных примерах было рассмотрено умножение векторных регистров глубиной в одной слово. SIMD инструкции с малым числом повторения (< 4) менее эффективны из-за задержек в конвейере, поэтому значения в тактах приведены условно, для наглядности. Однако, указанные цифры соответствуют действительности для SIMD инструкций с большим числом повторений. При таком допущении справедливо оценивать суммарную производительность, принимая время исполнения векторной команды за один такт. Далее рассмотрим использование SIMD инструкций с числом повторений 8 на примере перемножения матриц 8x8.

Перемножение матриц на одной ячейке.

Умножение двух матриц строится на принципах умножения вектор-строки на матрицу, описанного выше. Главным отличием является то, что теперь на первом шаге две пары чисел ([B_{00}, B_{01}] и [B_{10}, B_{11}]) матрицы B можно умножать сразу на два столбца ([A_{x0}, A_{x1}]) матрицы A, которые целиком помещаются в векторный регистр глубиной в 8 слов. На 2-ом шаге матрица ([B_{20}, B_{21}] и [B_{31}, B_{30}]) умножается на следующие два столбца ([A_{x2}, A_{x3}]) с прибавлением к вектору-результату от первого шага и так далее. За 4 шага полностью осуществляется процесс умножения двух матриц. В отличие от векторно-матричного умножения в данном случае не требуется непрерывная перезагрузка коэффициентов матричного умножителя. Так как она хранится в регистрах в течении 8 тактов умножения, то ее подкачка осуществляется на фоне непрерывного чтения матрицы A и не оказывает влияния на производительность.

На рисунке 10 показана временная диаграмма процессов умножения и обращения в память. Из рисунка видно, что если массивы расположены в разных банках, то доступ к ним осуществляется параллельно на фоне умножения. За счет так называемого «зацепления» производительность всей функции определяется лишь временем чтения матрицы A и временем разгона конвейера.

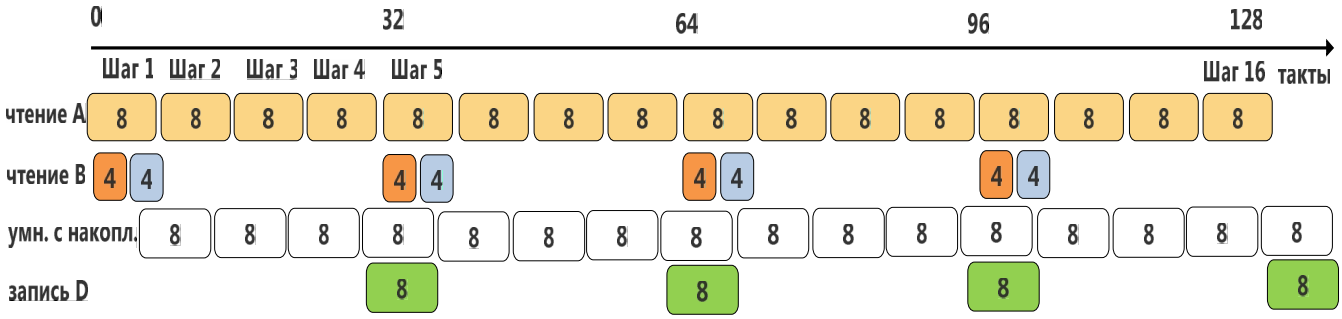

Рисунок 10. Временная диаграмма чтения-записи данных при умножении матриц 8x8

Распараллеливание матричного умножения на 4 ячейки

Сопроцессор с плавающей точкой содержит 4 вычислительных ячейки, поэтому предыдущую схему умножения распространим на все 4 ячейки следующим образом: матрица B вертикально разбивается на 4 равных подматрицы (см. рис. 11). Каждое умножение подматрицы производится согласно выше описанной схеме. При этом, так как во всех четырех ячейках участвуют одни и те же данные матрицы A, обращение к памяти в массив A можно сократить в 4 раза за счет регистровых пересылок между ячейками (см. рис. 11). При матрично-матричном умножении не требуется непрерывная подкачка данных матрицы B. Во время умножения в каждой ячейке образуются временные окна, которые могут использоваться для чтения данных B другими ячейками. На рис.12 показана временная диаграмма обращений в память каждой ячейкой. Видно, что благодаря «зацеплению» конфликтов по обращению в память не происходит. В результате, каждая ячейка совершает 4 умножения с накоплением (MAC) с темпом в один такт, а производительность всего процессора достигает 16 MAC или 32 FLOPs за такт. Следует отметить, что значение 16 MAC является пиковой производительностью. Реальные цифры меньше и зависят от размеров матриц, где сказываются потери, обусловленные расходами на вызов функции, инициализацию и разгон конвейера вычислений. При больших размерах матриц реальная производительность приближается к предельной и достигает 15.8 MAC за такт (31.6 FLOPS).

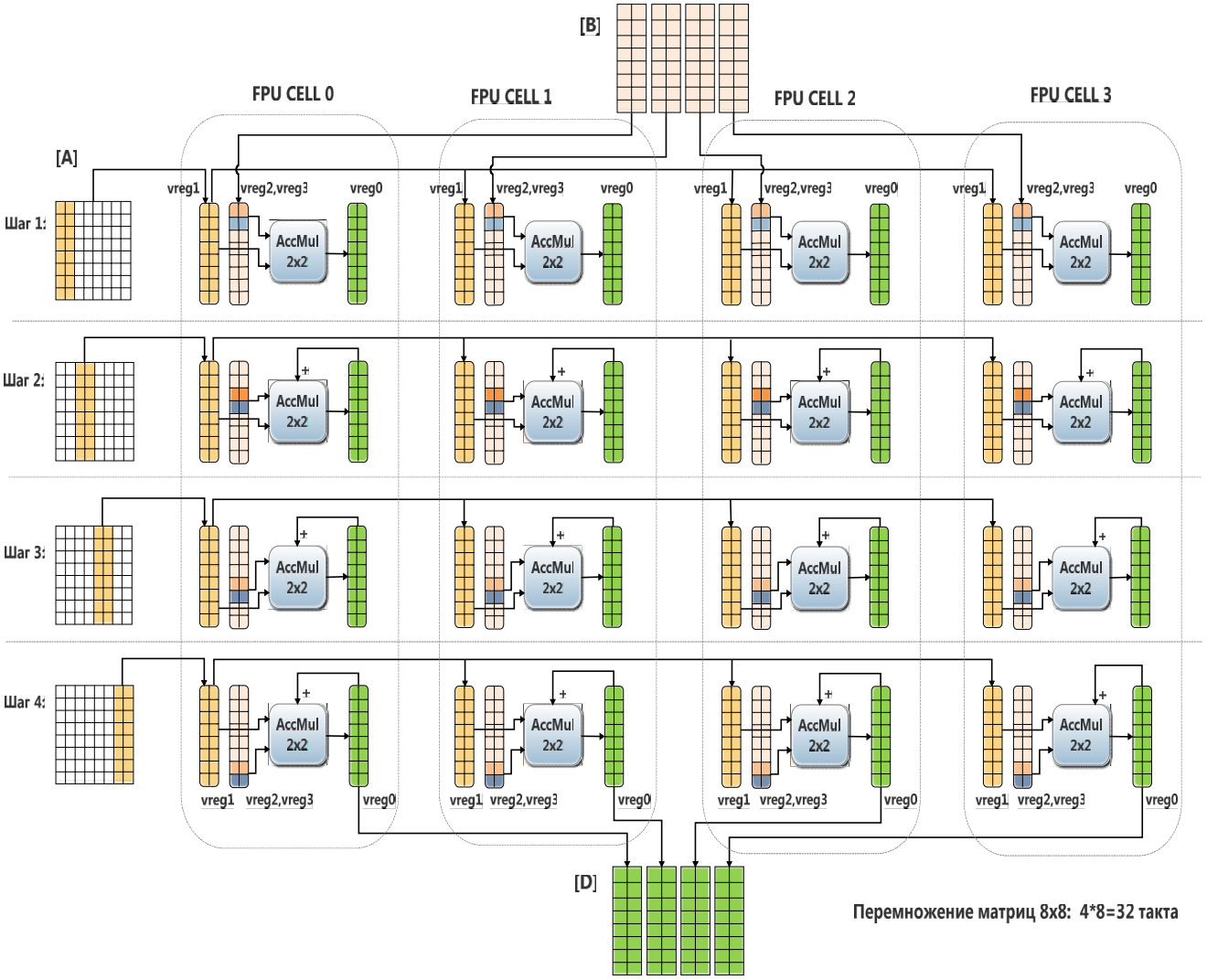

Рисунок 11. Последовательность операций матричного умножения в пакетном режиме на 4 ячейках

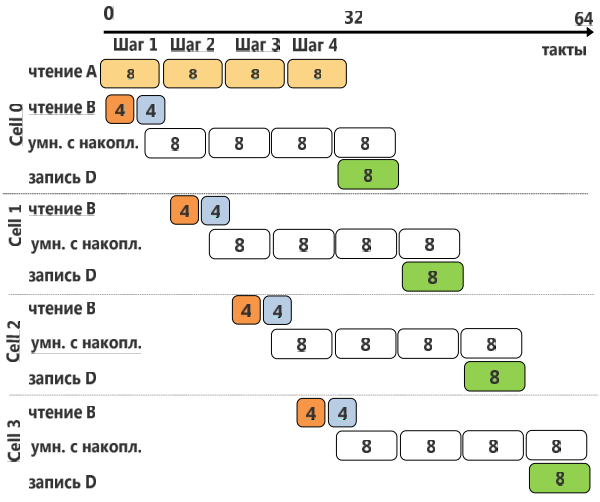

Рисунок 12. Временная диаграмма чтения-записи данных при умножении матриц 8x8

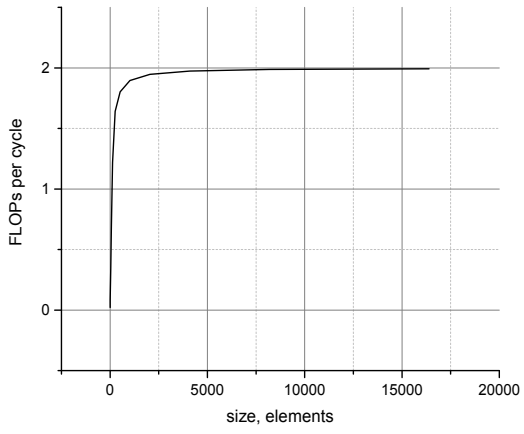

Рисунок 13. График зависимости среднего числа операций за такт от длины векторов для функций: $\vec{y} = \alpha\vec{x}$ и $\vec{z} = \vec{x}\vec{y}$

Анализ производительности от размера данных

Для процессора NM6407 были реализованы простейшие функции линейной алгебры из состава библиотеки BLAS . Производительность функций определяется тремя факторами:

1 – числом задействованных вычислительных ячеек в задаче

2 – распределением данных по разным банкам

3 – размером данных

Рассмотрим простейшие функции линейной алгебры из состава библиотеки BLAS: умножение вектора на скаляр $\vec{y} = \alpha\vec{x}$ и поэлементное перемножение векторов $\vec{z} = \vec{x}\vec{y}$. Данные функции являются достаточно простыми - всего с 1-ой операцией умножения на каждый элемент. Для умножения вектора на константу и поэлементного умножения чисел одинарной точности ячейка работает в режиме диагональной загрузки (рис 6.б) и способна осуществлять по 2 умножения за такт. В данном случае максимальная производительность ограничивается лишь скоростью обращения за данными и составляет 2 FLOPs за такт, остальные 3 ячейки из 4 остаются незадействованными. Для функций на рис. 13 приведен график зависимости усредненного числа операций за такт от длины векторов. Как видно из графика при большом объеме данных (более 1000 элементов) реальная производительность приближается к 2 FLOPs за такт. При малых размерах (менее 1000) сказывается влияние потерь на вызов С-функции, чтение параметров и прочих подготовительных операций.

В случае скалярного произведения $dot = (\vec{x} \cdot \vec{y})$ ячейка загружается вертикально (пара чисел вектора $\vec{x}$ загружается в правую половину и нули в левую, см. рис.6а). Ячейка производит 2 умножения с накоплением, что соответствует 4 FLOPS за такт.

Рассмотрим более сложную функцию вида $\vec{y} = \alpha A\vec{x} + \beta\vec{y}$, где A матрица размера h на L α, β скаляры, а $\vec{y}$ и $\vec{x}$ векторы длиной h. Данную функцию можно реализовать, задействовав уже три ячейки. Первая вычисляет $A\vec{x}$, результат с помощью регистровой пересылки поступает на вторую, где умножается на α, а третья ячейка вычисляет $\beta\vec{y}$ и прибавляет к результату от второй. Первая ячейка выполняет матрично-векторное умножение, которое является самой длинной операцией. 2-ая и 3-ая ячейки загружаются лишь по мере готовности вектора результата от первой. Производительность, главным образом, определяется временем вычисления $A\vec{x}$. Для больших размеров A производительность можно приблизительно принять 4 FLOPs за такт.

Максимально задействовать все 4 ячейки удается при матричном умножении, где достигается максимально возможная пиковая производительность 32 FLOPs за такт.

В таблице 1 приведены значения предельной (расчетной) и реальной производительностей некоторых функций линейной алгебры из состава библиотеки BLAS для данных одинарной точности. Значения производительности были замерены на максимальных объемах данных, помещающихся во внутренних банках памяти процессора, что соответствует размеру векторов 32000 или матриц 180x180.

Таблица 1 – Производительность функций линейной алгебры (BLAS) на процессоре NM6407 для чисел с плавающей точкой одинарной точности

Функция	**Кол-во ячеек**	**Макс. (расчетная) производительность, FLOPs/cycle**	**Реальная производительность, умножений/такт**
$\vec{y} = \alpha\vec{x}$	1	2	1.89
$dot = (\vec{x}\vec{y})$	1	4	3.78
$\vec{z} = \vec{x}\vec{y}$	1	2	1.90
$\vec{y} = \alpha A\vec{x} + \beta\vec{y}$	3	~4	3.79
$\vec{y} = \alpha A^T\vec{x} + \beta\vec{y}$	3	~4	3.84
AB = C	4	32	31.6

Как видно из полученных результатов, реальная производительность достигает 95-98% от теоретически возможной. Приведенные функции жестко поддерживают интерфейс стандартной библиотеки BLAS (Basic Linear Algebra Subprograms), что в некоторых случаях при их реализации ограничивает программиста в распределении вычислений на несколько ячеек, а также не позволяет максимально распараллелить потоки входных и выходных данных. Во второй колонке таблицы указано количество задействованных ячеек для разных функций, где видно, что оно не везде равно максимальному числу. Максимальная производительность достигает предельного пикового значения – (32 FLOPs/cycle) для функций матричного умножения, где участвуют все 4 ячейки.

Однако, при реализации частной задачи, как правило, всегда есть возможность более эффективно распределить данные между банками памяти, распараллелив обмен с памятью. Также можно сгруппировать вычисления до более сложных выражений и задействовать, таким образом, большее число вычислительных ячеек. Так, в целях поддержки процессора математическими функциями, помимо библиотеки BLAS, разрабатываются аналогичные, либо более сложные функции с максимальной адаптацией под архитектуру процессора. Данные функции разрабатываются в составе библиотеки NMPP [2]. Интерфейс этих функций уже отличается от библиотеки BLAS, но в некоторых случаях позволяет значительно повысить производительность (в 2 и более раз).

Реализация быстрого преобразования Фурье

В заключении рассмотрим реализацию алгоритмически сложных потоковых вычислений на примере БПФ-256. Стандартным подходом для реализации БПФ является использование схемы Кули-Тьюки [3]. Однако, для процессора NM6407 она не совсем эффективна. Из-за сложного порядка выборки данных на первых слоях алгоритма не удается использовать SIMD инструкции с большим числом повторений. Кроме того, алгоритм необходимо адаптировать под 4 вычислительные ячейки и 8-ми банковую структуру внутренней памяти. Наиболее

оптимальным вариантом получилась реализация комбинированной схемы, где первый слой реализован через последовательность дискретных преобразований Фурье по основанию 8 см. рис. 14. Последующие слои представляют набор классических «бабочек» с комплексным умножением, сложением и вычитанием. Подробно алгоритм БПФ описан в статье [4]. В данном разделе наибольший интерес представляет блок ДПФ-8, поэтому разберем только его реализацию.

Блок ДПФ представляет собой умножение матрицы 8x8 на вектор, описанный ранее, однако, поскольку таких блоков в алгоритме БПФ-256 – 32, то появляется возможность более эффективно задействовать процессорные ячейки.

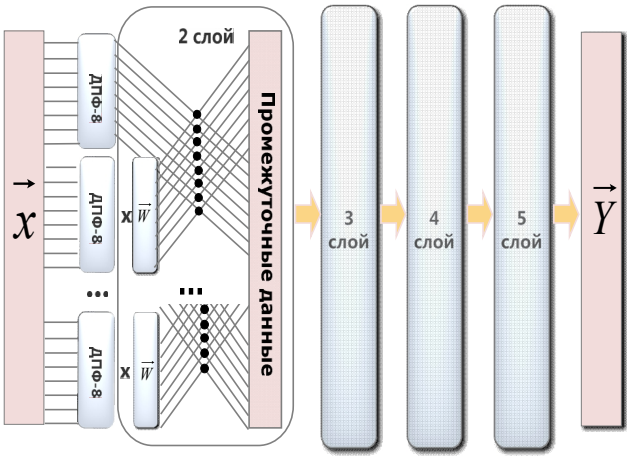

Рисунок 14. Блок-схема алгоритма БПФ-256

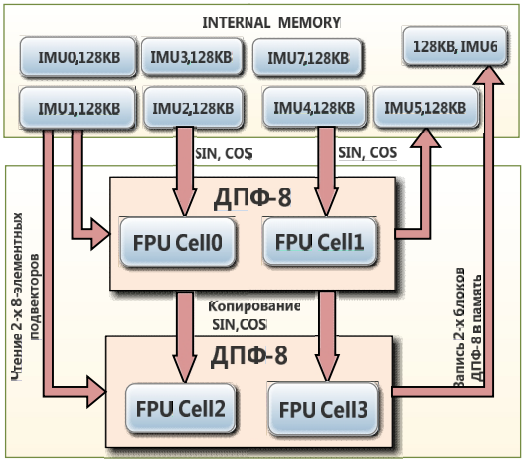

Рисунок 15. Схема вычисления 2-х блоков ДПФ-8

Как уже сказано, ДПФ-8 можно вычислить на NM6407 так же, как и умножение матрицы на вектор. На двух процессорных ячейках вычисляется один блок ДПФ-8. Соответственно 4 ячейки вычисляют сразу 2 блока ДПФ-8, а матрица коэффициентов W для каждого ДПФ-8 одна и та же и хранится в 2-х ячейках FPU Cell

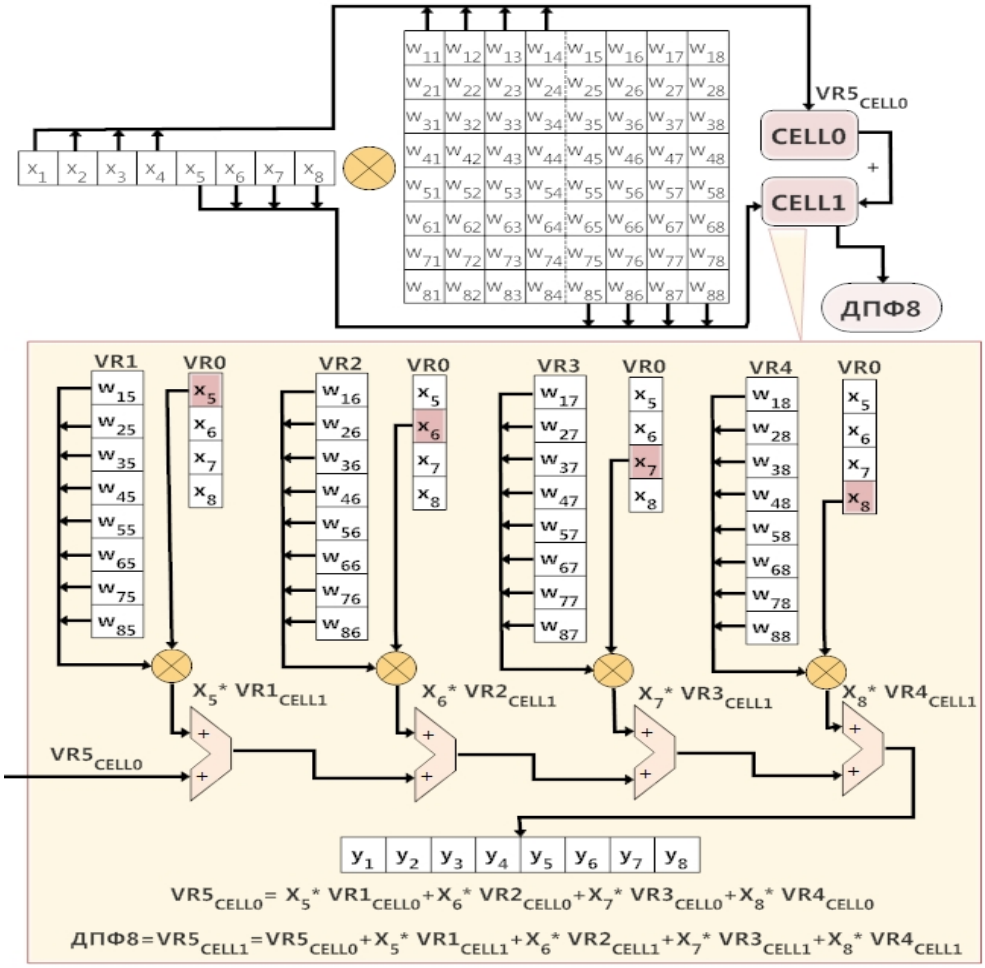

Рисунок 16. Подробная схема вычисления ДПФ-8

одновременно (рис. 15).

Рассмотрим более подробно вычисление блока ДПФ-8. Столбцы матрицы W: W(, 1), W(, 2), W(, 3) и W(, 4), загружаются из памяти в векторные регистры: VR1, VR2, VR3 и VR4 ячейки CELL0 (рис.16). Аналогично из памяти в регистр VR0 той же ячейки попадают первые 4-е элемента вектора $\vec{x}$. После чего, происходит поэлементное комплексное умножение с накоплением каждого столбца матрицы, находящегося в векторном регистре процессорной ячейки, на один из первых 4-х элементов вектора $\vec{x}$, лежащих в VR0. Накопление результата происходит в регистре VR5. Поочередный выбор действующего элемента регистра VR0 на каждом этапе вычислений показан розовым квадратом на рисунке 16 и производится автоматически с помощью специальной команды.

Такие же операции, как в CELL0, выполняются и в CELL1. Причем, 8-элементный вектор, полученный в ходе вычислений на CELL0, копируется в один из регистров CELL1 и затем суммируется с регистром VR5 ячейки CELL1. После этого в регистре VR5 находится 8-элементный вектор: ДПФ-8.

При вычислении серии блоков ДПФ-8 (от 100 блоков и больше) время, затраченное на вычисление одного такого блока равно 83 тактам. Причем коэффициенты матрицы W хранятся в векторных регистрах ячеек и повторно используются при вычислении каждого блока ДПФ-8 из всей серии. При однократном вычислении такого блока число тактов увеличивается до 127. Так получается, что векторный сопроцессор с плавающей точкой эффективнее работает с большими потоками данных, в данном случае с серией блоков ДПФ-8.

Таблица 2 – Сравнение производительности некоторых функций БПФ на разных процессорах

БПФ	**NM6407 (500МГц)**		**C674x DSPs**		**Pentium 4**	**ARM Cortex A-15**	
	Такты	**Время, мкс**	**Такты**	**Время, мкс**	**Такты**	**Такты**	**Время, мкс**
128	**1290**	2.58	-	-	1987	-	-
256	**2276**	**4.55**	**2401**	**5.27**	**4178**	**8644**	**8.64**
512	**5587**	11.17	-	-	9725	-	-
1024	**14523**	29.04	10950	24.01	22382	43916	43.92

В таблице 2 приведены значения производительности функций БПФ в тактах для процессоров NM6407, Texas Instruments C674x, Intel Pentium 4 и ARM Cortex A-15. На основе этих данных, можно сделать вывод, что NM6407 обгоняет все процессоры из таблицы при вычислении БПФ-256. Остальные функции БПФ NM6407 вычисляет быстрее Pentium 4 в среднем на 22%.

Заключение

При работе с большими массивами данных за счет распределенной структуры памяти на 8 банков и конвейерного выполнения команд процессор показывает высокую степень распараллеливания потоков данных, в которых реальная производительность достигает порядка 95-98% от расчетной. При максимальной загрузке процессорных ячеек, в таких операциях типа матричного умножения, производительность достигает 98% от пиковой. Анализ и сравнение производительностей представленных в таблице 1 и 2 показывают высокую эффективность процессора в задачах обработки больших потоков данных, однако, для достижения максимально высоких показателей необходимо учитывать все особенности процессора и адаптировать вычисления под его архитектуру.

Литература

1. «СБИС на базе ядра NMC3 для программного приемника навигационных сигналов», Косоруков Д.Е., Эйсымонт А.Л., Осипов В.Г., Панфилов А.П., Черников В.М., Виксне П.Е., Шелухин А.М., Насонов И.И., Сборник докладов Международной конференции "Микроэлектроника 2015"
2. URL: https://github.com/RC-MODULE/nmpp
3. Р.Отнес, Л.Эноксен “Прикладной Анализ Временных Рядов”,-М.:Мир,1982
4. «Адаптация алгоритма БПФ для матрично-векторного сопроцессора nm6407 с плавающей точкой» 19-ая Международная конференции “ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ И ЕЁ ПРИМЕНЕНИЕ”