УДК 004.428.2 DOI: 10.31114/2078-7707-2020-4-181-186

# Реализация функций библиотеки подпрограмм линейной алгебры на векторном сопроцессоре для не выравненных массивов

С.И. Аряшев, П.С. Зубковский, В.В. Цветков

ФГУ ФНЦ НИИСИ РАН, г. Москва, aserg@cs.niisi.ras.ru, zubkovsky@cs.niisi.ras.ru, vasily2002@mail.ru

***Аннотация* — Рассматриваются подходы к повышению производительности выполнения подпрограмм библиотеки линейной алгебры на векторном сопроцессоре. Предложены алгоритмы для реализации функций первого уровня с использованием эффективных команд загрузки/сохранения двух векторов при работе с невыровненными массивами.**

***Ключевые слова* — векторный сопроцессор, сопроцессор вещественной арифметики, коэффициент ускорения, команды загрузки, команды сохранения.**

## I. Введение

Для ускорения работы процессоров семейства КОМДИВ при решении задач линейной алгебры и задач с комплексными данными в архитектуру процессоров добавлен векторный сопроцессор (CPV) [1]. Векторный сопроцессор включает регистровый файл на 64 128-разрядных регистра. Максимальная ширина вектора 128 бит.

Эффективность применения векторного сопроцессора можно оценить по величине ускорения выполнения программ, реализующих функции на векторном сопроцессоре, по отношению к исполнению их на вещественном сопроцессоре. Время выполнения программ определяется временем загрузки данных в регистры, временем обработки данных и временем, затраченным на сохранение результатов в памяти. Получение более высоких значений коэффициента ускорения возможно за счет уменьшения времени на каждом этапе обработки и совмещения выполнения этапов. Для чего программисту предлагается набор векторных команд и эффективные команды загрузки/сохранения данных через кэш память 1-го уровня или пары векторов через кэш память 2-го уровня.

Возможность применения тех или иных команд загрузки/сохранения связана с особенностями выравнивания используемых массивов. Так, наиболее производительные команды VLDQ/VSDQ, выполняющие загрузку/сохранение двух 128 разрядных векторов требует выравнивания массивов по границе 32-х байтного слова (align32). Команды, выполняющие загрузку/сохранение одного вектора (VLDM/VSDM), можно использовать при работе с массивами, уровень выравнивания которых не ниже уровня выравнивания по границе 16-ти байтного слова (align16). Для массивов с уровнем выравнивания по границе 8 байтного слова (align8) и выше можно применять команды, выполняющие загрузку/сохранение данных в верхнюю (VLDH/VSDH) половину регистра и команды (VLD/VSDH) для загрузки/сохранения одних и тех же данных в обе половины регистра.

Более высокие значения коэффициента ускорения можно достигнуть, если массивы, с которыми работает программа выравнены по границе 32-х байтного слова. При уровнях выравнивания align16 или align8 время выполнения этапа загрузки/сохранения увеличивается, поскольку в этих случаях можно использовать менее производительные команды и эффект ускорения выполнения функций на векторном сопроцессоре снижается.

Проблемы возникают при работе с float массивами, выравненными по границе 4-х байтного слова (align4). Поскольку векторные команды загрузки/сохранения данных в регистры CPV для float массивов в системе команд векторного сопроцессора отсутствуют, то загрузку/сохранение векторных регистров приходится выполнять через GPR регистры процессора, что является затратным по времени и может нивелировать преимущества векторного сопроцессора.

В данной работе рассматриваются подходы, позволяющие минимизировать время загрузки/сохранения данных в векторные регистры сопроцессора за счет использования команд VLDQ/VSDQ при работе с массивами, уровень выравнивания которых не обязательно равен align32, а может быть и ниже. Например, align16/align8 для double/float массивов или align4 для float массивов. Предложенные подходы рассматриваются применительно к функциям первого уровня библиотеки BLAS.

## II. Реализация функций первого уровня библиотеки BLAS на векторном сопроцессоре

### *A. Функции первого уровня библиотеки BLAS*

На базе известной библиотеки GOTOBLAS [2] в нашем институте разработана версия библиотеки gotoblas+ адаптированная к векторному сопроцессору. В работах [3-4] изложены основные подходы, использованные при разработке библиотеки gotoblas+ и

примеры адаптации отдельных функций на различных уровнях библиотеки к исполнению на векторном сопроцессоре.

В данной работе рассматриваются особенности реализации на векторном процессоре функций первого уровня с учетом возможности работы с невыровненными массивами. Работа с невыровненными массивами подразумевает использование алгоритмов, допускающих применение для загрузки/сохранения команд, не согласованных с уровнем выравнивания исходных массивов.

На рис. 1 приводится список функций первого уровня библиотеки BLAS, указаны названия функций, функциональность и типы входных данных.

| | | prefixes |
|---|---|---|
| xROTG | Generate plane rotation | S, D |
| xROTMG | Generate modified plane rotation | S, D |
| xROT | Apply plane rotation | S, D |
| xROTM | Apply modified plane rotation | S, D |
| xSWAP | $x \leftrightarrow y$ | S, D, C, Z |
| xSCAL | $x \leftarrow \alpha x$ | S, D, C, Z, CS, ZD |
| xCOPY | $y \leftarrow x$ | S, D, C, Z |
| xAXPY | $y \leftarrow \alpha x + y$ | S, D, C, Z |
| xDOT | $dot \leftarrow x^T y$ | S, D, DS |
| xDOTU | $dot \leftarrow x^T y$ | C, Z |
| xDOTC | $dot \leftarrow x^H y$ | C, Z |
| xxDOT | $dot \leftarrow \alpha + x^T y$ | SDS |
| xNRM2 | $nrm2 \leftarrow \|\|x\|\|_2$ | S, D, SC, DZ |
| xASUM | $asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$ | S, D, SC, DZ |
| IxAMAX | $amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$ | S, D, C, Z |

**Рис. 1. Функции первого уровня BLAS**

В функциях первого уровня в качестве входных данных используются одномерные массивы - вектора. На данном этапе в библиотеке gotoblas+ поддерживаются вещественные входные данные двойной и одинарной точности, представленные префиксами S и D.

*B. Описание алгоритмов.*

При разработке алгоритмов программ для векторного сопроцессора существенным становятся особенности использования массивов данных. А именно, количество массивов, наличие процедуры сохранения данных, уровни выравнивания массивов. На рис. 2 представлена таблица, поясняющая особенности использования массивов в функциях первого уровня BLAS.

| Использование массивов | | f/s | функция |
|---|---|---|---|
| X | Y | | |
| | load | f | amax |
| | load | f | asum |
| load | load | f | dot |
| | load | f | iamax |
| | load | f | nrm2 |
| load | load/store | s | axpy |
| load | store | s | copy |
| load/store | load/store | s | rot |
| | load/store | s | scal |
| load/store | load/store | s | swap |

**Рис. 2. Особенности использования массивов в функциях первого уровня BLAS**

Результатом выполнения функций могут быть как вектора, в этом случае функция обозначается как подпрограмма (s), так и возвращаемые значения функций, в этом случае функция обозначается префиксом (f). Массивы данных, с которыми работает функция, обозначены через X и Y. Процедура загрузки входных данных в регистры сопроцессора обозначена как «load». Результирующие данные, если результатом работы функции является вектора, сохраняются («store») в одном из массивов или в обоих массивах, как, например, в функции SWAP. Для функций, возвращающих вычисленные значения, сохранения данных в массивы не происходит.

Рассмотрим вначале алгоритмы, в которых для загрузки/сохранения данных применяются команды согласованные с уровнем выравнивания используемых в программе массивов. Программам, использующим такие алгоритмы, будем приписывать префикс **kern_xx.**

В этих случаях, как отмечалось выше, более высокие значения коэффициента ускорения можно получить на выровненных массивах align32 за счет использования команд VLDQ/VSDQ, которые позволяют осуществить загрузку/сохранение сразу двух 128 разрядных векторов.

Если функция работает с двумя массивами, но уровни их выравнивания различные, то вначале определяется массив с минимальным уровнем выравнивания. Далее, загрузка/сохранение данных для обоих массивов выполняется с использованием команд, соответствующих этому минимальному уровню выравнивания. При этом мы частично проигрываем в производительности, отказавшись от применения более эффективных команд при работе с массивами, имеющими более высокий уровень выравнивания. Например, если один из массивов align32, а другой align8, то для обоих массивов мы вынуждены использовать команды VLD/VSD, загружающие/сохраняющие данные только из одной из половин регистра, отказавшись от использования команд VLDQ/VSDQ, выполняющих загрузку/сохранение сразу двух векторов при работе с массивом align32.

Проблемы при применении этой группы алгоритмов, как отмечалось выше, возникают при работе с float массивами align4, выровненными по границе 4-х байтного слова.

Другая группа алгоритмов, которые мы обозначаем как **kern_xq,** допускает возможность использовать команды VLDQ/VSDQ независимо от уровней выравнивания исходных массивов. Остановимся на некоторых особенностях таких алгоритмов.

Рассмотрим вначале случай, когда в функции используется только один массив.

В этом случае выделяется часть массива (будем называть телом массива), начальный адрес которого выравнен align32. Все, что находится перед телом массива, является головной частью. Количество

элементов (L) в головной части массива зависит от типа и уровня выравнивания массива.

Количество элементов в теле массива кратно количеству элементов, загружаемых при каждой загрузке в блок регистров CPV.

Оставшаяся часть массива представляет хвостовую часть массива.

Для загрузки данных из тела массива в блок регистров CPV и для сохранения данных результата из регистров CPV используются команды VLDQ/VSDQ. Обработка данных тела массива выполняется с использованием векторных команд. Головная часть массива и так называемый «хвост» обрабатывается на FPU.

Размер головной части массива зависит от уровня выравнивания массива и типа данных. На рис. 3 поясняется алгоритм для определения размера головной части массива.

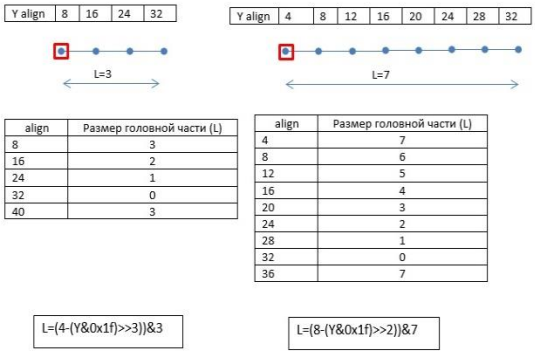


**Рис. 3. Определение размера головной части массива**

На рисунке приведены таблицы для вычисления размера головной части массива для двух типов массивов DOUBLE в левой половине рисунка и FLOAT – в правой половине. В верхней части показаны уровни выравнивания адресов указателей на начальные элементы массива. На рисунке для примера отмечены указатели на начало DOUBLE массива выровненного align8 и начало FLOAT массива выровненного align4. Длина головной части в каждом случае определяется количеством элементов от начального элемента массива до элемента с адресом, выровненным по границе 32-байтного слова. В нашем примере это составляет 3 для DOUBLE и 7 - для FLOAT массивов.

Блок схема программы, реализующей представленный алгоритм, приводится на рис. 4.

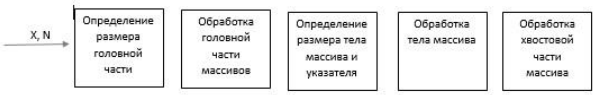


**Рис. 4. Блок схема программы для функции с одним массивом**

По значению указателя определяется уровень выравнивания массива, на основании которого вычисляется размер головной части. По завершению обработки головной части вычисляется размер тела массива и определяется указатель на начало тела массива.

Обработка тела массива выполняется с использованием векторных вычислительных команд, а для загрузки данных в блок регистров и для сохранения результата в памяти (если результатом работы программы является вектор) используются команды VLDQ/VSDQ.

Рассмотрим теперь случай, когда функция работает с двумя массивами. Возможны два варианта. Один из них, когда уровни выравнивания массивов X и Y совпадают, и другой, когда уровни выравнивания этих массивов различны.

В первом варианте мы поступаем аналогично действиям, описанным при работе с одним массивом. На примере любого из двух массивов мы выделяем тело массива, выравненного align32. Поскольку уровни выравнивания двух массивов одинаковые, то размеры головной части, тела массива и хвостовой части в каждом из массивов также будут одинаковые. Далее также как и в случае, когда в программе используется только один массив, программа поочередно выполняется на головной части массивов, затем на центральных частях массивов, представляющих тело массивов, и на «хвостах». Для обоих массивов соотношение размеров тела и самого массива одинаковые и, если это соотношение достаточно велико, то программа исполняется в основном с использованием для загрузки/сохранения данных из обоих массивов команд VLDQ/VSDQ, несмотря на то, что уровни выравнивания этих массивов могут отличается от align32.

Несколько иначе строится алгоритм, когда уровни выравнивания массивов различны. По-прежнему в каждом массиве выделяется головная часть, тело и хвостовая часть. Несмотря на то, что уровни выравнивания исходных массивов отличаются, размеры этих выделенных частей в обоих массивах одинаковые. Размер головной части определяется на примере анализа одного любого из массивов, или того массива, в который производится запись (если результатом работы функции является вектор). Этот массив будем называть реперным. Для конкретности будем считать, что это массив Y. В реперном массиве тело массива выравнено align32. В отличие от реперного массива уровень выравнивания тела во втором массиве отличается от align32, поскольку отличаются уровни выравнивания самих массивов. Уровень выравнивания тела во втором массиве можно определить, проанализировав указатель на его начало. Указатель на начало тела второго массива X_b определяется после вычисления размера головной части.

Далее, также как и в двух предыдущих случаях, программа поочередно выполняется на головной части массивов, затем на теле массивов и на «хвостах».

Отличие заключается в том, что при выполнении программы на теле массивов, загрузка/сохранение данных из тела реперного массива будет выполняться с использованием эффективных команд VLDQ/VSDQ, а из второго массива с использованием команд соответствующих уровню выравнивания тела этого массива, который в общем случае отличается от align32. Это приводит к снижению коэффициента ускорения, особенно, если тело второго массива окажется выравненным по границе 4-х байтного слова (align4).

## III. Результаты тестирования

В директории “kernel/mips64” в структуре библиотеки gotoblas+ находятся корневые функции, написанные на ассемблере и адаптированные к векторному сопроцессору. Корневые функции являются базовыми функциями при синтезе библиотечных функций. В этом разделе приведены результаты тестирования корневых функций первого уровня библиотеки gotoblas+. Для функций первого уровня каждой библиотечной функции соответствует одна корневая функция, которая совпадает по названию и по функциональности с библиотечной функцией.

Тестирование выполнялось при запуске корневых функций на процессоре, в архитектуре которого присутствуют векторный и вещественный сопроцессоры. Тестировались различные варианты корневых функций, в которых реализованы обе группы алгоритмов kern_xx и ***kern_xq*** для работы с выровненными и с невыровненными массивами.

### *A. Описание системы тестирования*

При тестировании, корневые функции, адаптированные к векторному сопроцессору, исполняются на CPV на массивах, имеющих различные уровни выравнивания, типы данных и размерности. Результаты выполнения функции на CPV и количество затраченных на выполнение тактов процессора сравнивается с эталонными значениями, полученными при запуске на FPU исходного (не адаптированного к CPV) варианта написанных на ассемблере базовых функций из библиотеки GOTOBLAS.

По результатам тестирования оценивается коэффициент ускорения выполнения функций на CPV по отношению к исполнению этих функций на FPU для различных вариантов, используемых при адаптации алгоритмов и различных типов массивов.

### *B. Результаты тестирования*

Для каждой функции по результатам тестирования построены диаграммы зависимости изменения коэффициента ускорения от размеров массива для различных типов и уровней выравнивания исходных массивов.

На рис. 5 в качестве примера приведены результаты тестирования функции AXPY_kern_xx на DOUBLE и FLOAT массивах. В этом тесте массивы X и Y имеют одинаковый уровень выравнивания. Для DOUBLE массивов тестирование выполнялось для трех значений уровня выравнивания: align32, align16 и align8, для FLOAT массивов тестирование дополнительно проводилось на массивах align4. Программы функций реализованы с использованием алгоритмов ***kern_xx*** для выравненных массивов.

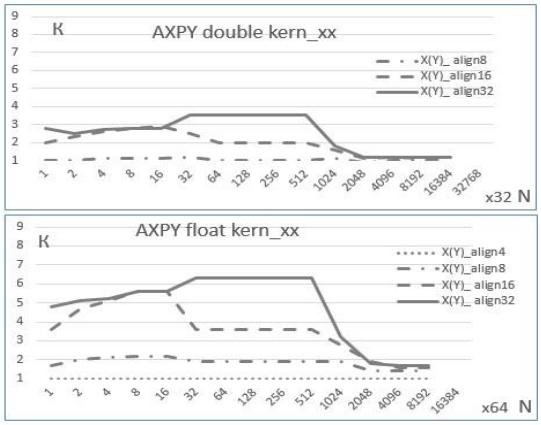


**Рис. 5. Зависимости коэффициента ускорения исполнения функции AXPY_kern_xx на векторном сопроцессоре в зависимости от количества элементов в массивах**

Как видно из приведенных зависимостей, значение коэффициента на FLOAT массивах примерно в два раза превышает значение коэффициента ускорения на DOUBLE массивах и достигает максимального значения K=6 на FLOAT массивах, выравненных align32, при размерах массивов, не превышающих размеры кэш2. Для загрузки/сохранения данных в регистры CPV на этих массивах используются команды VLDQ/VSDQ.

На массивах align16 загрузка/сохранение выполняется с использованием команд VLDM. При размерах массива не превышающих кэш1 коэффициент ускорения сравним с коэффициентом ускорения, достигнутым на массивах align32.

Для массивов align8, когда для загрузки/сохранения используются команды VLD/VLDH (VSD/VSDH), заметное ускорение наблюдается только на FLOAT массивах.

Ускорения не удается достигнуть на FLOAT массивах, выравненных align4. В этом случае программа функции исполняется на FPU.

Незначительное ускорение наблюдается как на FLOAT массивах, так и на DOUBLE массивах, при размерах массивов, превышающих кэш2.

Лучших результатов удается добиться, если при адаптации функций к векторному сопроцессору использовать алгоритмы ***kern_xq***, поддерживающие работу с невыровненными массивами. В качестве примера на рис. 6 приводятся результаты тестирования программы функции AXPY, написанной с использованием алгоритма для работы с невыровненными массивами.

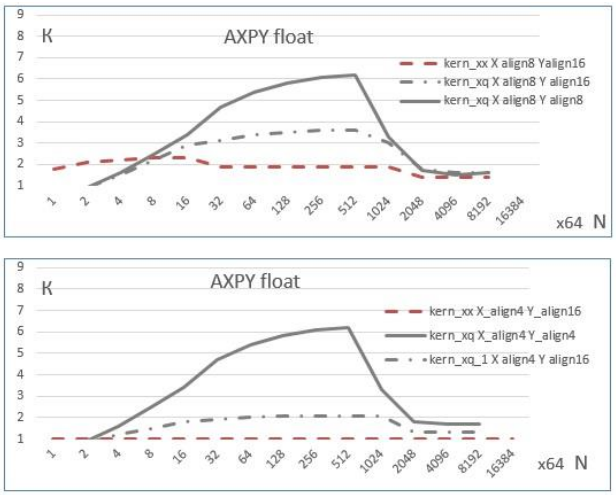


**Рис. 6. Зависимости коэффициента ускорения исполнения функции AXPY_kern_xq на векторном сопроцессоре от количества элементов в массивах**

На рисунке приведены результаты тестирования функции AXPY на FLOAT массивах. Массив X на верхнем рисунке выровнен по границе 8-х байтного слова, на нижнем рисунке - по границе 4-х байтного слова. Массив Y на обоих рисунках выровнен по границе 16-ти байтного слова при тестировании варианта с различными уровнями выравнивания массивов X и Y. При тестировании варианта с равнозначными уровнями выравнивания уровень выравнивания массива Y совпадает с уровнем выравнивания массива X.

Две верхние зависимости получены при тестировании функции, адаптированной с использованием алгоритма kern_xq для работы с невыровненными массивами. Одна из них, представленная сплошной линией, отображает результаты тестирования на массивах с равнозначными уровнями выравнивания.

Нижняя зависимость представляет результаты тестирования функции, реализованной по алгоритму kern_xx. В этом случае используются команды загрузки/сохранения, согласованные с минимальным уровнем выравнивания между массивами X и Y.

Как видно из рисунков, использование алгоритма kern_xq, поддерживающего работу с невыровненными массивами, позволяет добиться наличия ускорения при выполнении на CPV функции на FLOAT массивах, выровненных по границе 4-х байтного слова. В общем случае применение алгоритма ***kern_xq*** позволяет получить более высокие значения коэффициента ускорения по сравнению с алгоритмом, использующим команды загрузки/сохранения, согласованные с уровнем выравнивания массивов. Наиболее высокие значения коэффициента ускорения при использовании алгоритма FLOAT массивах достигаются при равнозначных уровнях выравнивания массивов X и Y. Например, на FLOAT массивах с уровнем выравнивания X_align4 и Y_align4, при использовании алгоритма kern_xq максимальное значение коэффициента ускорения составляет примерно K=6. При использовании алгоритма для выравненных массивов ускорения в этом случае достигнуть не удается.

Сводная таблица значений коэффициента ускорения для функций первого уровня библиотеки gotoblas+ приведена на рис. 7. В таблице приведены значения коэффициента ускорения для различных функций, исполненных на DOUBLE и FLOAT массивах с различными уровнями выравнивания, указанными во второй строке таблицы. Программирование ядер этих функций для исполнения на векторном сопроцессоре выполнялось с использованием алгоритма kern***_xq***. Для функций, использующих два массива, предполагается равнозначные уровни выравнивания обоих массивов. Данные по значениям коэффициента ускорения приведены для массивов размером 4096 элементов.

| ФУНКЦИЯ | DOUBLE | | | FLOAT | | | |
|---|---|---|---|---|---|---|---|
| | align32 | align16 | align8 | align32 | align16 | align8 | align4 |
| amax | 1,2 | 1,2 | 1,2 | 2,1 | 2,1 | 2,1 | 2,1 |
| asum | 2,1 | 2,1 | 2,1 | 3,9 | 3,4 | 3,4 | 3,4 |
| dot | 6,2 | 5,7 | 5,7 | 11,9 | 9,0 | 9,0 | 9,0 |
| iamax | 1,4 | 1,4 | 1,4 | 2,3 | 2,3 | 2,3 | 2,3 |
| nrm2 | 10,5 | 9,9 | 9,9 | 10,5 | 8,4 | 8,4 | 8,4 |
| axpy | 3,5 | 3,4 | 3,4 | 6,2 | 5,6 | 5,6 | 5,6 |
| copy | 3,4 | 3,3 | 3,3 | 5,8 | 5,2 | 5,2 | 5,2 |
| rot | 0,8 | 0,8 | 0,8 | 1,9 | 1,9 | 1,9 | 1,9 |
| scal | 3,4 | 3,3 | 3,3 | 5,8 | 5,2 | 5,2 | 5,2 |
| swap | 3,8 | 3,7 | 3,7 | 6,3 | 6,0 | 6,0 | 6,0 |

**Рис. 7. Сводная таблица значений коэффициента ускорения для функций первого уровня библиотеки gotoblas+**

ЛИТЕРАТУРА


[1] Бобков С.Г., Аряшев С.И., Барских М.Е., Зубковский П.С., Ивасюк Е.В. Высокопроизводительные расширения архитектуры универсальных микропроцессоров для ускорения инженерных расчётов // Информационные технологии. 2014. № 6. С. 27-37.

[2] Kazushige Goto and Robert A. van de Geijn Anatomy of high-performance matrix multiplication // ACM Translation on mathematical Software. 2008. 34 (3). Pp. 1-25.

[3] Аряшев С.И., Зубковский П.С., Кулешов А.С., Цветков В.В. Адаптация библиотеки подпрограмм линейной алгебры GOTOBLAS к архитектуре векторного сопроцессора // Материалы 3-й Всероссийской научно-технической конференции «Суперкомпьютерные технологии». Дивноморское, Геленджик. 2014. Т. 1. С. 90-95.

[4] Аряшев С.И., Зубковский П.С., Цветков В.В. Реализация функции копирования массивов на векторном сопроцессоре // Проблемы разработки перспективных микро- и наноэлектронных систем. 2018. Вып. 3. С. 144-147.

# Implementation of Functions of the Linear Algebra Subroutines on a Vector Coprocessor for Unaligned Arrays

S.I. Aryashev, P.S. Zubkovsky, V.V. Tsvetkov

aserg@cs.niisi.ras.ru, zubkovsky@cs.niisi.ras.ru, vasily2002@mail.ru

*Abstract* — **The efficiency of using a vector coprocessor can be estimated by the amount of acceleration of the execution of programs that implement functions on a vector coprocessor, in relation to their execution on a real coprocessor. Program execution time is determined by the time data is loaded into registers, the time it takes to process data, and the time it takes to store results in memory. Getting higher values of the acceleration coefficient is possible by reducing the time at each stage of processing and combining the execution of stages. For this purpose, the programmer is offered a set of vector commands and effective commands for loading / saving data through the cache memory of the 1st level or pairs of vectors through the cache memory of the 2nd level.**

**The ability to use certain load/save commands is related to the alignment of the arrays used. For example, the most efficient VLDQ/VSDQ commands that load/save two 128-bit vectors require alignment of arrays along the border of a 32-byte word (align32). Commands that load/save a single vector (VLDM/VSDM) can be used when working with arrays whose alignment level is not lower than the alignment level on the border of a 16-byte word (align16). For arrays with an 8-byte word alignment level (align8) or higher, you can use commands that load/save data in the upper (VLDH/VSDH) half of the register, or commands (VLD/VSDH) to load/save the same data in both halves of the register.**

**Higher values of the acceleration coefficient can be achieved if the arrays that the program works with are aligned along the border of a 32-byte word. At align16 or align8 alignment levels, the load/save stage execution time increases, because in these cases, you can use less productive commands and the effect of speeding up the execution of functions on the vector coprocessor is reduced.**

**Problems occur when working with float arrays aligned along the border of a 4-byte word (align4). Since there are no vector commands for loading/saving data to CPV registers for float arrays in the vector coprocessor command system, the loading/saving of vector registers has to be performed via the processor's GPR registers, which is time-consuming and can offset the advantages of the vector coprocessor.**

**In this paper, we consider approaches that allow us to minimize the time of loading/saving data to vector registers of the coprocessor by using VLDQ/VSDQ commands when working with arrays whose alignment level is not necessarily equal to align32, but may be lower. For example, align16/align8 for double/float arrays, or align4 for float arrays. The proposed approaches are considered in relation to the functions of the first level of the BLAS library.**

**Keywords — vector coprocessor, coprocessor of real arithmetic, acceleration factor, loading instructions, save instructions.**

## REFERENCES

[1] Bobkov S.G., Aryashev S.I., Barshyn M.E., Zubkovsky P.S., Ivasyuk E.V. High-Performance Extensions of Microprocessor Architecture for Speeding-Up of Scientific and Engineering Calculations // Information technologies. 2014. № 6. P. 27-37 (in Russian).

[2] Kazushige Goto and Robert A. van de Geijn Anatomy of high-performance matrix multiplication // ACM Translation on mathematical Software. 2008. 34 (3). P. 1-25.

[3] Aryashev S.I., Zubkovsky P.S., Kuleshov A.S., Tsvetkov V.V. Adaptation of the library of subroutines of linear algebra GOTOBLAS to the vector coprocessor architecture // Materials of the 3rd All-Russian scientific and technical conference "Supercomputer Technologies" Divnomorskoe, Gelendzhik. Rostov-on-Don. 2014. V. 1. P. 90-95 (in Russian).

[4] Aryashev S.I., Zubkovsky P.S., Tsvetkov V.V. The Results of the Implementation of the Copy Function on a Vector Coprocessor // Problemy razrabotki perspektivnyh mikro- i nanojelektronnyh sistem - 2018. Sbornik trudov / pod obshh. red. akademika RAN A.L. Stempkovskogo. M.: IPPM RAN, 2018. Chast' 3. P. 144-147 (in Russian).