УДК 004.42
DOI: 10.25559/SITITO.14.201803.594-602

ВЕКТОРИЗАЦИЯ ПЕРЕМНОЖЕНИЯ МАЛОРАЗМЕРНЫХ МАТРИЦ СПЕЦИАЛЬНОГО ВИДА С ИСПОЛЬЗОВАНИЕМ ИНСТРУКЦИЙ AVX-512

Л.А. Бендерский, А.А. Рыбаков, С.С. Шумилин
Научно-исследовательский институт системных исследований Российской академии наук, г. Москва, Россия

VECTORIZATION OF SMALL-SIZED SPECIAL-TYPE MATRICES MULTIPLICATION USING INSTRUCTIONS AVX-512

Leonid A. Benderskiy, Alexey A. Rybakov, Sergey S. Shumilin
Scientific Research Institute of System Analysys of the Russian Academy of Sciences, Moscow, Russia

Ключевые слова

Операции над матрицами; векторизация; KNL; AVX-512; функции-интринсики.

Аннотация

Современные расчетные пакеты для проведения суперкомпьютерных вычислений крайне требовательны к ресурсам, поэтому остро стоит вопрос повышения их производительности. В то же время появление новых аппаратных архитектур открывает новые возможности по оптимизации программного кода. Набор инструкций AVX-512, появившийся впервые в 2016-м году в микропроцессорах Intel Xeon Phi второго поколения, является мощным инструментом для создании высокопроизводительного параллельного кода. Набор инструкций AVX-512 обладает рядом особенностей, среди которых специальные регистры масок, позволяющие отбирать отдельные элементы векторов для обработки, комбинированные арифметические операции, операции чтения из памяти элементов данных с произвольными смещениями, векторные трансцендентные операции, многообразие операций перемешивания элементов векторов. Эти и многие другие уникальные возможности позволяют достигать кратного ускорения на наиболее важных участках расчетных кодов. Отдельные численные методы оперируют специальными объектами, эффективность обработки которых критически влияет на эффективность всего пакета. Для расчетных кодов RANS/ILES, предназначенных для расчета нестационарных турбулентных течений, такими специфическими объектами являются матрицы размера 5x5, расположенные внутри матриц размера 8x8 в качестве подматриц. Основной операцией для работы с такими матрицами является их перемножение. В статье рассматривается эффективный подход к векторизации перемножения матриц размера 8x8, а также проводится исследование, как понижение размера матриц влияет на эффективность векторизации. Предложенный подход реализован с помощью специальных функций-инстринсиков, и его эффективность проверена экспериментами, проведенными на суперкомпьютере, находящимся в МСЦ РАН.

Об авторах:

Бендерский Леонид Александрович, старший научный сотрудник, Межведомственный суперкомпьютерный центр Российской академии наук - филиал Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук» (119334, г. Москва, Ленинский проспект, 32а), ORCID: http://orcid.org/0000-0003-0529-3255, leosun.ben@gmail.com

Рыбаков Алексей Анатольевич, кандидат физико-математических наук, ведущий научный сотрудник, Межведомственный суперкомпьютерный центр Российской академии наук -филиал Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук» (119334, г. Москва, Ленинский проспект, 32а), ORCID: http://orcid.org/0000-0002-9755-8830, rybakov.aax@gmail.com, rybakov@jscc.ru

Шумилин Сергей Сергеевич, ведущий инженер, Межведомственный суперкомпьютерный центр Российской академии наук - филиал Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук» (119334, г. Москва, Ленинский проспект, 32а), ORCID: http://orcid.org/0000-0002-3953-7054, shumilin@jscc.ru

Keywords

Matrix operations; vectorization; KNL; AVX-512; intrinsic functions.

Abstract

Modern software packages for supercomputer calculations require a large amount of computing resources. At the same time there are new hardware architectures that open up new opportunities for program code optimizing. The AVX-512 instruction set is a unique tool with many useful features, allowing to create that allow creating high-performance parallel code for supercomputer calculations. The most striking features of AVX-512 instruction set are special mask registers that allow selecting the elements of vectors for processing, combined arithmetic instructions, vector transcendental instructions, operations of multiple memory access with different arbitrary offsets, and many others. Some numerical methods use special objects, the processing speed of which critically affects the speed of the entire calculation package. Matrices of size 5 by 5, represented as submatrices of 8-by-8 matrices, are such critical objects for the numerical RANS/ILES method, which is used for nonstationary turbulent flows calculating. The main operation for working with such matrices is multiplication. In this paper we consider an effective approach to vectorizing the multiplication of 8-by-8 matrices. Further, the effect of decreasing the dimension of the matrices on the efficiency of vectorization is estimated. The considered approach is realized with the help of special intrinsic functions for the AVX-512 instruction set and it is checked on a supercomputer located in JSCC RAS.

Введение

При реализации вычислительных методов часто приходится сталкиваться с объектами специального вида, обработка которых занимает львиную долю расчетного времени задачи. В данной статье будет рассматриваться специфика численных методов RANS/ILES [1, 2], от эффективной реализации которой существенным образом зависит скорость счета. Комбинированный RANS/ILES (Reynolds Averaged Navier-Stokes – RANS, Implicit Large Eddy Simulation – ILES) метод с явным разрешением турбулентных вихрей применяется для расчетов нестационарных турбулентных течений [3, 4, 5]. В данном методе около стенок для расчета течения решаются нестационарные уравнения Навье-Стокса с моделью турбулентности Спаларта-Аллмараса. Вдали от стенок течение описывается с помощью LES с неявной SGS-моделью – ILES. При таком подходе отсутствует явная SGS-модель турбулентности, а ее функцию выполняет схемная вязкость разностной схемы. В данном методе используется высокий порядок аппроксимации газодинамических параметров на гранях ячеек расчетной сетки. Интегрирование по времени выполняется с помощью технологии интегрирования по двойному времени. Интегрирование по физическому времени выполняется со вторым порядком точности, при этом используется достаточно малый шаг по времени. Для эффективного применения метода RANS/ILES требуется использование суперкомпьютера [6, 7].

При оптимизации программных кодов для использования на суперкомпьютере необходимо проведение оптимизации на разных уровнях: повышение эффективности использования MPI обменов между отдельными процессами [8, 9], использование многопоточного программирования путем применения OpenMP [10, 11], специальная подготовка и декомпозиция расчетной сетки [12, 13], балансировка вычислительной нагрузки между вычислительными узлами суперкомпьютера [14]. Кроме того, при использовании суперкомпьютера важно обеспечить отказоустойчивость исполняемых программных кодов, что выражается в разработке специальных алгоритмов организации системы контрольных точек, позволяющей восстанавливать прерванные вычисления с последней сохраненной позиции [15]. Наиболее низкоуровневой оптимизацией для ускорения расчетов является применение векторизации вычислений, с помощью которой возможно кратное ускорение отдельных участков программы за счет объединения скалярных вычислений в векторные [16, 17].

При проведении вычислений с помощью метода RANS/ILES расчетные параметры компонуются внутри матриц размером 5x5, при этом данные матрицы являются подматрицами матриц размера 8x8 (это делается из соображений выровненности расположения в памяти). Наиболее частой операцией при проведении вычислений является перемножением таких матриц (данная функция занимает до 40% расчетного времени). Поэтому для повышения эффективности проведения расчетов с использованием метода RANS/ILES требуется эффективная реализация функций для работы с обозначенными объектами.

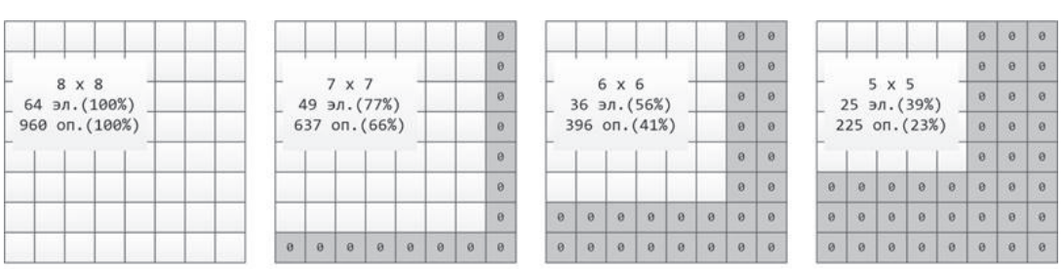

Рис. 1. Иллюстрация матриц размера 8x8, 7x7, 6x6, 5x5, представленных как подматрицы матрицы 8x8.
Fig 1. An illustration of 8x8, 7x7, 6x6, 5x5 matrices represented as submatrices of an 8x8 matrix

В данной статье мы рассмотрим матрицы размера 8x8 и эффективную реализацию перемножения таких матриц. После этого будет рассмотрена реализация перемножения для матриц размера 7x7, 6x6, 5x5 которые представлены как подматрицы матриц размера 8x8 (см. рис. 1). При этом элементами матриц являются вещественные 32-битные числа.

В качестве средства оптимизации программного кода использовался набор инструкций AVX-512. Данный набор инструкций представляет собой 512-битное расширение набора команд Intel x86, поддержанное в семействах микропроцессоров Intel Xeon Phi Knights Landing (KNL) и Intel Xeon Skylake. Инструкции из набора AVX-512 оперируют 512-битными векторами, способными вмещать по 16 элементов типа float. Важным преимуществом инструкций AVX-512 является возможность использования специальных регистров масок, определяющих, какие именно элементы векторов должны обрабатываться векторной операцией, а какие должны игнорироваться. Всего таких регистров 8 (k0-k7). Использование регистров масок по сути соответствует реализации предикатного исполнения инструкций и открывает широкие возможности по оптимизации кода. Так с момента появления первых микропроцессоров с поддержкой инструкций AVX-512 во всем мире ведутся исследования, направленные на повышение эффективности применения данных инструкций в суперкомпьютерных вычислениях. Можно отметить работы по ускорению вычислительного ядра программного пакета LAMMPS [18], по оптимизации операций для работы с разреженными матрицами высокой размерности [19], по ускорению расчетных кодов задач ядерной физики [20], по векторизации задачи о распаде произвольного разрыва и многие другие [21, 22, 23]. Так как набор инструкций AVX-512 будет расширяться в следующих версиях микропроцессоров Intel Xeon Skylake, то исследования в этой области на сегодняшний день не теряют своей актуальности.

Также следует отменить специализированные библиотеки Intel MKL [24] и libxsmm [25], в которых реализованы операции перемножения матриц, оптимизированные в частности для набора инструкций AVX 512, однако данные библиотеки нам не подходят по нескольким причинам. Во-первых, в них не учитываются нулевые элементы перемножаемых матриц, а также использование данных операций не позволяет оптимизировать последовательности операций по работе с матрицами. Например, в расчетных кодах RANS/ILES требуется эффективное выполнение комбинированной операции R=A×D×B, где A и B являются обычными матрицами, а D — вектором, представляющим собой диагональную матрицу. Библиотеки Intel MKL и libxsmm не поддерживают такой функционал, тогда как рассмотренные в данной статье реализации легко расширяются на более сложные цепочки часто выполняемых операций, эффективная реализация которых требуется в расчетном коде.

Теоретическая часть

В работе [26] был рассмотрен подход к перемножению матриц размера 8x8 и 16x16 с элементами типа float, основной идеей которого было вначале выполнить операции поэлементного перемножения строк первой матрицы на столбцы второй матрицы, а затем параллельно вычислить суммы элементов соответствующих векторов. В случае матриц 16x16 параллельно вычислялась сумма элементов 16 векторов, в случае матриц 8x8 параллельно вычислялась сумма элементов половинок 8 векторов. Достигалось это последовательным применением шаблонных связок, состоящих из операций perm, add и blend как показано ниже.

```
#define SWIZ_2_ADD_2_BLEND_1(X, Y, SWIZ_TYPE, BLEND_MASK) \
  _mm512_mask_blend_ps(BLEND_MASK, \
              ADD(X, _mm512_swizzle_ps(X, SWIZ_TYPE)), \
              ADD(Y, _mm512_swizzle_ps(Y, SWIZ_TYPE)))
#define PERM_2_ADD_2_BLEND_1(X, Y, PERM_TYPE, BLEND_MASK) \
  _mm512_mask_blend_ps(BLEND_MASK, \
              ADD(X, _mm512_permute4f128_ps(X, PERM_TYPE)), \
              ADD(Y, _mm512_permute4f128_ps(Y, PERM_TYPE)))
```

В результате получился довольно громоздкий код, со сложной схемой обращения в память. Некоторые недостатки кода удалось устранить, в результате чего было достигнуто ускорения более чем в два раза относительно неоптимизированной версии. Листинг кода, реализующий данный подход, показан ниже.

```
void mul_8x8_opt(float * __restrict a, float * __restrict b, float * __restrict r)
{
  ..........................................
  __m512 bj, bj2,
      m0, m1, m2, m3, m4, m5, m6, m7;

  // Индексы для работы со столбцами.
  __m512i ind_cc = _mm512_set_epi32(7 * V8 + 1, 6 * V8 + 1, 5 * V8 + 1, 4 * V8 + 1,
                    3 * V8 + 1, 2 * V8 + 1,    V8 + 1,        1,
                    7 * V8   , 6 * V8   , 5 * V8   , 4 * V8   ,
                    3 * V8   , 2 * V8   ,    V8   ,        0);
  __m512i ind_st = _mm512_set_epi32(7 * V8   , 7 * V8 + 1, 5 * V8   , 5 * V8 + 1,
                    3 * V8   , 3 * V8 + 1,    V8   ,    V8 + 1,
                    6 * V8 + 1, 6 * V8   , 4 * V8 + 1, 4 * V8   ,
                    2 * V8 + 1, 2 * V8   ,    1    ,        0);

  // Чтение всех строк матрицы a (по две в zmm регистр).
  __m512 a0 = _mm512_load_ps(&a[0]);
  ..........................................
  __m512 a3 = _mm512_load_ps(&a[6 * V8]);

  // Цикл по столбцам матрицы b.
  for (int j = 0; j < V8; j += 2)
  {
    bj = _mm512_i32gather_ps(ind_cc, &b[j], _MM_SCALE_4);
    bj2 = _mm512_permute4f128_ps(bj, _MM_PERM_BADC);

    // Поэлементное перемножение строк a и столбцов b.
    m0 = _mm512_mul_ps(a0, bj);
    m1 = _mm512_mul_ps(a0, bj2);
    ..........................................
    m6 = _mm512_mul_ps(a3, bj);
    m7 = _mm512_mul_ps(a3, bj2);

    // Параллельное выполнение суммирования подвекторов.
    m0 = SWIZ_2_ADD_2_BLEND_1(m0, m1, _MM_SWIZ_REG_CDAB, 0xAAAA);
    m1 = SWIZ_2_ADD_2_BLEND_1(m2, m3, _MM_SWIZ_REG_CDAB, 0xAAAA);
    m2 = SWIZ_2_ADD_2_BLEND_1(m4, m5, _MM_SWIZ_REG_CDAB, 0xAAAA);
    m3 = SWIZ_2_ADD_2_BLEND_1(m6, m7, _MM_SWIZ_REG_CDAB, 0xAAAA);
    m0 = SWIZ_2_ADD_2_BLEND_1(m0, m1, _MM_SWIZ_REG_BADC, 0xCCCC);
    m1 = SWIZ_2_ADD_2_BLEND_1(m2, m3, _MM_SWIZ_REG_BADC, 0xCCCC);
    m2 = PERM_2_ADD_2_BLEND_1(m0, m1, _MM_PERM_CDAB, 0xF0F0);

    // Сохранение результата.
    _mm512_i32scatter_ps(&r[j], ind_st, m2, _MM_SCALE_4);
  }
```

В приведенном листинге использованы следующие векторные примитивы, реализованные с помощью встроенных функций-инстринсиков (для работы с вещественными значениями используются векторы __m512, содержащие по 16 элементов типа float одинарной точности):

- _mm512_set_epi32 – инициализация целочисленного векторного регистра, содержащего 16 значений (в данном случае используется для задания смещений для доступа в память с помощью инструкций gather/scatter);
- _mm512_load_ps – чтение из памяти по выровненному адресу 512-битного вещественного вектора;
- _mm512_i32gather_ps – чтение из памяти 16 вещественных значений с произвольными смещениями относительно базового адреса и помещение их в один векторный регистр;
- _mm512_permute4f128_ps – перестановка 128-битных четвертей векторного регистра;
- _mm512_mul_ps – поэлементное перемножение двух вещественных векторов;
- _mm512_swizzle_ps – перестановка вещественных элементов внутри каждой из 128-битных четвертей векторного регистра;
- _mm512_mask_blend_ps – слияние двух вещественных векторов с помощью заданной маски, логика которого может быть записана в виде dst[i] = mask[i] ? src2[i] : src1[i];
- _mm512_i32scatter_ps – запись в память элементов векторного регистра с произвольными смещениями относительно базового адреса;

Данная реализация обладает существенными недостатками. Во-первых, в коде присутствуют медленные инструкции gather/scatter, читающие из памяти элементы данных с произвольными смещениями, которые выполняются существенно медленнее чтения из памяти последовательных данных [27]. Во-вторых, выполнение сначала поэлементного перемножения векторов, а затем параллельное нахождение сумм их элементов (в данном случае рассчитываются суммы элементов половинок векторов), делают невозможным использование эффективных комбинированных инструкций fmadd. Для того, чтобы избавиться от названных недостатков, запишем в явном виде значения элементов -й строки результирующей матрицы:

Данная реализация обладает существенными недостатками. Во-первых, в коде присутствуют медленные инструкции gather/scatter, читающие из памяти элементы данных с произвольными смещениями, которые выполняются существенно медленнее чтения из памяти последовательных данных [27]. Во-вторых, выполнение сначала поэлементного перемножения векторов, а затем параллельное нахождение сумм их элементов (в данном случае рассчитываются суммы элементов половинок векторов), делают невозможным использование эффективных комбинированных инструкций fmadd. Для того, чтобы избавиться от названных недостатков, запишем в явном виде значения элементов -й строки результирующей матрицы:

$$\begin{cases} r_{i0} = a_{i0}b_{00} + a_{i1}b_{10} + .. + a_{i7}b_{70} \\ .. \\ r_{i7} = a_{i0}b_{07} + a_{i1}b_{17} + .. + a_{i7}b_{77} \end{cases},$$

или в векторном виде

$$\overline{r_i} = a_{i0}\overline{b_0} + a_{i1}\overline{b_1} + .. + a_{i7}\overline{b_7}.$$

Аналогичные выражения можно записать для строки с номером $i + 1$:

$$\begin{cases} r_{i+1,0} = a_{i+1,0}b_{00} + a_{i+1,1}b_{10} + .. + a_{i+1,7}b_{70} \\ .. \\ r_{+1,i7} = a_{i+1,0}b_{07} + a_{i+1,1}b_{17} + .. + a_{+1,i7}b_{77} \end{cases},$$

или в векторном виде

$$\overline{r_{i+1}} = a_{i+1,0}\overline{b_0} + a_{i+1,1}\overline{b_1} + .. + a_{i+1,7}\overline{b_7}.$$

Учитывая то, что zmm регистры содержат по 16 элементов типа float, то целесообразно объединить приведенные выше формулы в одну, записанную в векторном виде следующим образом:

$$\begin{pmatrix} \overline{r_i} \\ \overline{r_{i+1}} \end{pmatrix} = \left(\begin{pmatrix} \overline{a_{i0}} \\ \overline{a_{i+1,0}} \end{pmatrix} \circ \begin{pmatrix} \overline{b_0} \\ \overline{b_0} \end{pmatrix} \right) + \left(\begin{pmatrix} \overline{a_{i1}} \\ \overline{a_{i+1,1}} \end{pmatrix} \circ \begin{pmatrix} \overline{b_1} \\ \overline{b_1} \end{pmatrix} \right) + .. + \left(\begin{pmatrix} \overline{a_{i7}} \\ \overline{a_{i+1,7}} \end{pmatrix} \circ \begin{pmatrix} \overline{b_7} \\ \overline{b_7} \end{pmatrix} \right),$$

где $\begin{pmatrix} \overline{r_i} \\ \overline{r_{i+1}} \end{pmatrix}$ обозначает комбинированный вектор состоящий из векторов $\overline{r_i}$ и $\overline{r_{i+1}}$, $\begin{pmatrix} \overline{b_j} \\ \overline{b_j} \end{pmatrix}$ – комбинированный вектор, состоящий из двух копий вектора $\overline{b_j}$, а выражение $\begin{pmatrix} \overline{a_{ij}} \\ \overline{a_{i+1,j}} \end{pmatrix}$ обозначает вектор, первые 8 элементов которого равны a_{ij}, а остальные 8 элементов – $a_{i+1,j}$ ($\circ$ – произведение Адамара, или поэлементное произведение векторов). Заметим, что получающийся по данной формуле комбинированный вектор $\begin{pmatrix} \overline{r_i} \\ \overline{r_{i+1}} \end{pmatrix}$ расположен в памяти последовательно, и записывать его в память можно с помощью интринсика _mm512_store_ps. При этом предполагаем, что значение i четно, то есть вектор $\begin{pmatrix} \overline{r_i} \\ \overline{r_{i+1}} \end{pmatrix}$ выровнен в памяти должным образом. Другие комбинированные векторы в данном выражении получаются с помощью инструкции perm (интринсик _mm512_permutexvar_ps), примененной к соответствующим загруженным соседним строкам матриц a и b. Таким образом, при реализации вышеприведенной формулы не требуется использование медленных инструкций gather/scatter, так как столбцы матриц не читаются и не записываются (работа ведется только со строками). После того, как требуемые векторы сформированы, нужно выполнить их попарное поэлементное перемножение, после чего сложить в один вектор (8 операций поэлементного умножения, 7 операций сложения). Эти действия можно выполнить, используя комбинированные операции fmadd.

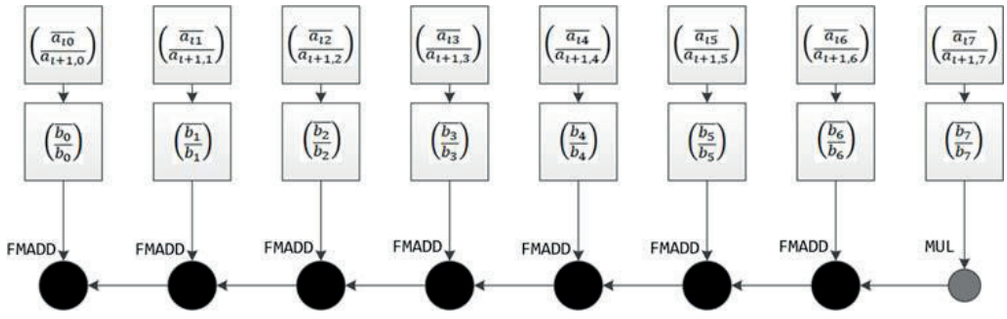

Рис. 2. Схема вычисления двух соседних строк результирующей матрицы путем последовательного сложения попарно перемноженных векторов.
Fig 2. The scheme for calculating the two adjacent rows of the resulting matrix by successive addition of pairwise multiplied vectors

Для вычисления значения $\left(\frac{\overline{r_i}}{\overline{r_{i+1}}}\right)$ потребуется 8 векторных операций (1 операция mul и 7 операций fmadd) (см. рис. 2). На данном рисунке для удобства представлена схема последовательного сложения перемноженных векторов. Возможен также другой вариант, реализующий вычисление $\left(\frac{\overline{r_i}}{\overline{r_{i+1}}}\right)$ с помощью балансировки дерева потока данных. Такой вариант потребует 4 операций mul, 4 операций fmadd и 3 операций add. В данном контексте вычислений длина критической цепочки не влияет на время работы кода, поэтому мы оставляем вариант с минимальным количеством инструкций (меньше 8 инструкций задействовать невозможно, так как нужно выполнить по крайней мере 8 операций умножения).

Реализация

Приведем функцию перемножения двух матриц размера 8x8, механизм которой был рассмотрен в предыдущем разделе. Для реализации выполним полную загрузку обеих матриц a и b. На это потребуется 8 операций обращения в память, так как за одну операцию загружаются две соседние строки матрицы. Далее требуется сформировать 8 векторов вида $\left(\frac{\overline{b_j}}{\overline{b_j}}\right)$, для чего потребуется еще 8 операций perm (для каждой пары загруженных строк матрицы b нужно выполнить дублирование первой строки и дублирование второй строки). Формирование векторов индексов для операций perm не требует вычислительного времени, так как индексы являются статическими и вычисляются на этапе компиляции.

После подготовки всех необходимых данных выполняется вычисление значений результирующей матрицы. Приведенный ниже блок операций (макрос BLOCK) осуществляет вычисление двух соседних строк результирующей матрицы. Реализация блока состоит из 8 операций perm, 1 операции mul и 7 операций fmadd, кроме того выполняется одна операция записи в память. Всего выполняется четыре таких блока, что в сумме и с учетом операций подготовки данных приводит к следующему итогу: 8 простых операций чтения из памяти, 40 операций perm, 4 операции mul, 28 операций fmadd, 4 простые операции записи в память. Итоговый код (без отображения некоторых повторяющихся участков) приведен на листинге ниже.

```
void mul_8x8_opt(float * __restrict a, float * __b, float * __restrict r)
{
  .....
  // Индексы для дублирования первой и второй половины zmm регистра.
  __m512i ind_df = _mm512_set_epi32( 7,  6,  5,  4,  3,  2, 1, 0,
                                     7,  6,  5,  4,  3,  2, 1, 0);
  __m512i ind_ds = _mm512_set_epi32(15, 14, 13, 12, 11, 10, 9, 8,
                                    15, 14, 13, 12, 11, 10, 9, 8);

  // Загрузка всех строк матрицы b (b0-b7) с дублированием.
  __m512 b0 = _mm512_load_ps(&b[0]);
  __m512 b1 = _mm512_permutexvar_ps(ind_ds, b0);
  b0 = _mm512_permutexvar_ps(ind_df, b0);
  .....
  __m512 b6 = _mm512_load_ps(&b[6 * V8]);
  __m512 b7 = _mm512_permutexvar_ps(ind_ds, b6);
  b6 = _mm512_permutexvar_ps(ind_df, b6);

  // Загрузка всех строк матрицы a (по две в zmm регистр).
  __m512 a0 = _mm512_load_ps(&a[0]);
  .....
  __m512 a6 = _mm512_load_ps(&a[6 * V8]);

  // Индексы для выбора элементов матрицы a.
  __m512i ind_0 = _mm512_set_epi32( 8,  8,  8,  8,  8,  8,  8,  8,
                                    0,  0,  0,  0,  0,  0,  0,  0);
  .....
  __m512i ind_7 = _mm512_set_epi32(15, 15, 15, 15, 15, 15, 15, 15,
                                    7,  7,  7,  7,  7,  7,  7,  7);

// Определение основного блока вычислений.
#define PERMXV _mm512_permutexvar_ps
#define MUL _mm512_mul_ps
#define FMADD _mm512_fmadd_ps
#define BLOCK(N, A)                    \
  _mm512_store_ps(&r[N * V8],          \
    FMADD(PERMXV(ind_0, A), b0,        \
      FMADD(PERMXV(ind_1, A), b1,      \
        FMADD(PERMXV(ind_2, A), b2,    \
          FMADD(PERMXV(ind_3, A), b3,  \
            FMADD(PERMXV(ind_4, A), b4,\
              FMADD(PERMXV(ind_5, A), b5,\
                FMADD(PERMXV(ind_6, A), b6,\
                  MUL(PERMXV(ind_7, A), b7)))))))));

  // Вычисление и сохранение результата.
  BLOCK(0, a0);
```

```
  BLOCK(2, a2);
  BLOCK(4, a4);
  BLOCK(6, a6);

#undef PERMXVAR
#undef MUL
#undef FMADD
#undef BLOCK

}
```

В приведенном листинге появляется функция _mm512_permutexvar_ps, позволяющая переставлять элементы вещественного векторного регистра в произвольном порядке, определенном с помощью целочисленного векторного регистра (dst[i] = src[idx[i]]).

Заметим, что 28 векторных операций fmadd и 4 векторные операции mul соответствуют (28·2+4)·16=960 скалярным операциям, что в точности совпадает с количеством скалярных операций, требуемых для выполнения перемножения двух матриц размера 8x8. Таким образом, в предложенной реализации нет лишних арифметических операций, и результат каждой выполненной операции влияет на конечный результат.

При реализации перемножения матриц размера 7x7, 6x6, 5x5 удаляются заведомо лишние векторные операции (например, умножение на вектор, все элементы которого равны нулю), однако все равно остаются элементы векторов, обработка которых избыточна, что приводит к снижению эффективности векторизации в этих случаях. Ниже приведем таблицу с точным подсчетом количества скалярных операций для невекторизованных функций и количества векторных операций для их векторизованных аналогов (см. табл. 1).

Таблица 1. Статистика по количеству операций в невекторизованном и векторизованном вариантах функций перемножения матриц размером 8x8, 7x7, 6x6, 5x5.
Table 1. Statistics on the number of operations in non-vectorized and vectorized variants of the functions of multiplication of 8x8, 7x7, 6x6, 5x5 matrices

	Невекторизованный вариант	**Векторизованный вариант**
8x8	512 mul, 448 add 960 арифметических операций	4 mul, 28 fmadd, 40 perm соответствует 960 арифметическим операциям
7x7	343 mul, 294 add 637 арифметических операций	4 mul, 24 fmadd, 35 perm соответствует 832 арифметическим операциям
6x6	216 mul, 180 add 396 арифметических операций	3 mul, 15 fmadd, 24 perm соответствует 528 арифметическим операциям
5x5	125 mul, 100 add 225 арифметических операций	3 mul, 12 fmadd, 20 perm соответсвует 432 арифметическим операциям

Из таблицы 1 видно, что с понижением размерности перемножаемых матриц возрастает избыточность вычислений (для размера 5x5 данная избыточность почти достигает двукратного размера), что сказывается отрицательно на эффективности векторизации.

Полученные результаты

Описанные в статье подходы к векторизации перемножения матриц были проверены на суперкомпьютере МВС-10П МСЦ РАН, на его вычислительном сегменте, содержащем микропроцессоры Intel Xeon Phi 7290 KNL. Были рассмотрены 4 функции: mul_8x8, mul_7x7, mul_6x6, mul_5x5, выполняющие перемножение матриц соответствующих размеров. Для каждой функции были рассмотрены 3 варианта реализации. В качестве первого варианта был использован старый способ векторизации с параллельным вычислением сумм элементов векторов (обозначен OLD VECT), в качестве второго варианта было взято прямое ручное вычисление каждого элемента результирующей матрицы, в котором все циклы были удалены и для каждого элемента путем многократного копирования был написан скалярный код (обозначен FULL UNROLL), в качестве третьего варианта взят рассмотренный подход, основанный на обращениях только к строкам матриц и использовании комбинированных операций fmadd (обозначен NEW VECT).

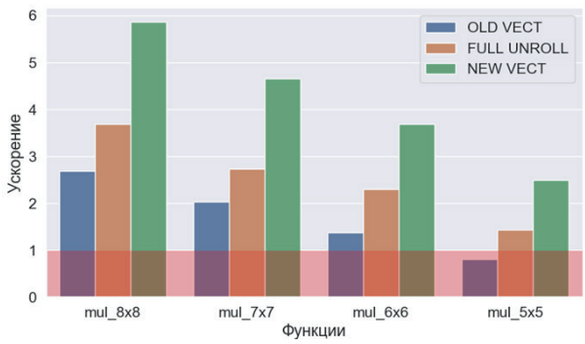

Рис.3. Ускорение функций mul_8x8, mul_7x7, mul_6x6, mul_5x5 после их оптимизации с помощью подходов OLD VECT, FULL UNROLL, NEW VECT.
Fig 3. Acceleration of mul_8x8, mul_7x7, mul_6x6, mul_5x5 functions after their optimization with the help of OLD VECT, FULL UNROLL, NEW VECT approaches

На рис. 3 показаны результаты тестирования описанных подходов на машине. Из рисунка следует, что OLD VECT оказался наименее эффективным подходом, его применение даже менее выгодно, чем прямое написание скалярного кода. Для матриц размера 5x5 данный метод оптимизации вовсе приводит к замедлению оригинальной неоптимизированной версии функции. Метод NEW VECT демонстрирует наилучшие результаты из описанных подходов, на матрицах размера 8x8 продемонстрировано ускорение почти в 6 раз по сравнению с оригинальным кодом. При понижении размерности матриц эффективность векторизации несколько снижается, однако даже для матриц размера 5x5 наблюдается ускорение примерно в 2,5 раза, что позволяет внедрять данный метод в промышленный код.

Заключение

Продемонстрированные результаты тестирования двух разных подходов к векторизации перемножения матриц показывает гибкость и богатые возможности набора инструкций AVX-512 в оптимизации программного кода. В то же время стоит отметить, что такие на первый взгляд удобные инструкции gather/scatter на самом деле оказываются крайне неэффективными, и их следует избегать (именно использование этих инструкций определило низкую эффективность подхода OLD VECT). Применение нового подхода к векторизации перемножения матриц размера 5x5 позволило более чем в два раза ускорить этот важный горячий участок кода несмотря на наличие избыточности вычислений почти в двукратном размере (см. табл. 1).

Благодарности

Работа выполнена в МСЦ РАН в рамках государственного задания по теме 0065-2018-0409 «Разработка архитектур, системных решений и методов для создания вычислительных комплексов и распределенных сред мультипетафлопсного диапазона производительности, в том числе нетрадиционных архитектур микропроцессоров». Для расчетов при проведении исследований использовался суперкомпьютер МВС-10П, находящийся в МСЦ РАН.

Список использованных источников

[1] *Lyubimov D.A.* Development and Application of a High-Resolution Technique for Jet Flow Computation Using Large Eddy Simulation // High Temperature. 2012. Vol. 50, no. 3. Pp. 420-436. DOI: 10.1134/S0018151X12020101

[2] *Benderskii L.A., Lyubimov D.A., Chestnykh A.O., Shabanov B.M., Rybakov A.A.* The Use of the RANS/ILES Method to Study the Influence of Coflow Wind on the Flow in a Hot, Nonisobaric, Supersonic Airdrome Jet during Its Interaction with the Jet Blast Deflector // High Temperature. 2018. Vol. 56, no. 2. Pp. 247-254. DOI: 10.1134/S0018151X18020037

[3] *Lyubimov D.A.* Investigation of the Effect of a Pylon and a Wing with Flaps on the Flow within an Exhaust Jet of a Double-Flow Turbojet Engine by a Simulation Method for Large Eddies // High Temperature. 2013. Vol. 51, no. 1. Pp. 111-127. DOI: 10.1134/S0018151X12050100

[4] *Любимов Д.А., Потехина И.В.* Исследование нестационарных режимов работы сверхзвукового воздухозаборника RANS/ILES-методом // Теплофизика высоких температур. 2016. Т. 54, № 5. С. 784-791. DOI: 10.7868/S0040364416050185

[5] *Lyubimov D., Maslov V., Mironova A., Secundov A., Zakharov D.* Experimental and Numeric Investigation of Jet Flap Interaction Effects // International Journal of Aeroacoustics. 2014. Vol. 13, no. 3-4. Pp. 275-302. DOI: 10.1260/1475-472X.13.3-4.275

[6] *Бендерский Л.А., Любимов Д.А., Рыбаков А.А.* Анализ эффективности масштабирования при расчетах высокоскоростных турбулентных течений на суперкомпьютере RANS/ILES методов высокого разрешения // Труды НИИСИ РАН. 2017. Т. 7, № 4. С. 32-40. DOI: 10.25682/NIISI.2018.4.9975

[7] *Рыбаков А.А.* Внутреннее представление и механизм межпроцессного обмена для блочно-структурированной сетки при выполнении расчетов на суперкомпьютере // Программные системы: теория и приложения. 2017. Т. 8, № 1. С. 121-134. DOI: 10.25209/2079-3316-2017-8-1-121-134

[8] *Klenk B., Fröning H.* An Overview of MPI Characteristics of Exascale Proxy Applications / J. M. Kunkel et al. (Eds.) // ISC High Performance 2017. LNCS. Vol. 10266. 2017. Pp. 217–236. DOI: 10.1007/978-3-319-58667-0_12

[9] *Роганов В.А., Осипов В.И., Матвеев Г.А.* Решение задачи Дирихле для уравнения Пуассона методом Гаусса-Зейделя на языке параллельного программирования T++ // Программные системы: теория и приложения. 2016. Т. 7, № 3. С. 99-107. DOI: 10.25209/2079-3316-2016-7-3-99-107

[10] *Dorris J., Kurzak J., Luszczek P.* Task-Based Cholesky Decomposition on Knights Corner Using OpenMP / M. Taufer et al. (Eds.) // ISC High Performance Workshops 2016. LNCS. Vol. 9945. 2016. Pp. 544–562. DOI: 10.1007/978-3-319-46079-6_37

[11] *Krause M.J., Förster B., Mink A., Nirschl H.* Towards Solving Fluid Flow Domain Identification Problems with Adjoint Lattice Boltzmann Methods / W. E. Nagel et al. (Eds.) // High Performance Computing in Science and Engineering'16. Springer, Cham, 2016. Pp. 337-353. DOI 10.1007/978-3-319-47066-5_23

[12] *Golovchenko E., Dorofeeva E., Gasilova I., Boldareva A.* Numerical Experiments with New Algorithms for Parallel Decomposition of Large Computational Meshes // Advances in Parallel Computing. 2014. Vol. 25. Pp. 441-450. DOI: 10.3233/978-1-61499-381-0-441

[13] *Петров М.Н., Титарев В.А., Утюжников С.В., Чикиткин А.В.* Многопоточная OpenMP-реализация метода LU-SGS с использованием многоуровневой декомпозиции неструктурированной расчетной сетки // Журнал вычислительной математики и математической физики. 2017. Т. 57, № 11. С. 1895-1905. DOI: 10.7868/S0044466917110138

[14] *Рыбаков А.А.* Распределение вычислительной нагрузки между узлами гетерогенного вычислительного кластера // Программные продукты, системы и алгоритмы. 2018. № 1. С. 26-32. DOI: 10.15827/2311-6749.26.300

[15] *Четверушкин Б.Н., Якобовский М.В.* Вычислительные алгоритмы и отказоустойчивость гиперэкзафлопсных вычислительных систем // Доклады Академии наук. 2017. Т. 472, № 1. С. 13-17. DOI: 10.7868/S0869565217010042

[16] *Krzikalla O., Wende F., Höhnerbach M.* Dynamic SIMD Vector Lane Scheduling / M. Taufer et al. (Eds.) // ISC High Performance Workshops 2016. LNCS. 2016. Vol. 9945. Pp. 354–365. DOI: 10.1007/978-3-319-46079-6_25

[17] *Дикарев Н.И., Шабанов Б.М., Шмелев А.С.* Использование «сдвоенного» умножителя и сумматора в векторном процессоре с архитектурой управления потоком данных // Программные системы: теория и приложения. 2015. Т. 6, № 4. С. 227-241. DOI: 10.25209/2079-3316-2015-6-4-227-241

[18] *McDoniel W., Höhnerbach M., Canales R.* et al. LAMMPS' PPPM Long-Range Solver for the Second Generation Xeon Phi / J. M. Kunkel et al. (Eds.) // ISC High Performance 2017. LNCS. 2017. Vol. 10266. Pp. 61–78. DOI: 10.1007/978-3-319-58667-0_4

[19] *Malas T., Kurth T., Deslippe J.* Optimization of the Sparse Matrix-Vector Products of an IDR Krylov Iterative Solver in EMGeo for the Intel KNL Manycore Processor / M. Taufer et al. (Eds.) // ISC High Performance Workshops 2016. LNCS. 2016. Vol. 9945. Pp. 378–389. DOI: 10.1007/978-3-319-46079-6_27

[20] *Cook B., Maris P., Shao M.* High Performance Optimizations for Nuclear Physics Code MFDn on KNL / M. Taufer et al. (Eds.) // ISC High Performance Workshops 2016. LNCS. 2016. Vol. 9945. Pp. 366–377. DOI: 10.1007/978-3-319-46079-6_26

[21] *Рыбаков А.А.* Оптимизация задачи об определении конфликтов с опасными зонами движения летательных аппаратов для выполнения на Intel Xeon Phi // Программные продукты и системы. 2017. Т. 30, № 3. С. 524-528. DOI: 10.15827/0236-235X.119.524-528

[22] *Bramas B.* A Novel Hybrid Quicksort Algorithm Vectorized using AVX-512 on Intel Skylake // International Journal of Advanced Computer Science and Applications. 2017. Vol. 8, no. 10. Pp. 337-344. DOI: 10.14569/IJACSA.2017.081044

[23] *Соколов А.П., Щетинин В.Н., Сапелкин А.С.* Параллельный алгоритм реконструкции поверхности прочности композиционных материалов для архитектуры Intel MIC (Intel Many Integrated Core Architecture) // Программные системы: теория и приложения. 2016. Т. 7, № 2. С. 3-25. DOI: 10.25209/2079-3316-2016-7-2-3-25

[24] Intel Math Kernel Library [Электронный ресурс]. URL: https://software.intel.com/en-us/mkl (дата обращения: 17.05.2018).

[25] Library targeting Intel Architecture for specialized dense and sparse matrix operations, and deep learning primitives [Электронный ресурс]. URL: https://github.com/hfp/libxsmm (дата обращения: 17.05.2018).

[26] *Бендерский Л.А., Лещев С.А., Рыбаков А.А.* Векторизация операций над матрицами малой размерности для процессора Intel Xeon Phi Knights Landing // Современные информационные технологии и ИТ-образование. 2018. Т. 14, № 1. С. 73-90. DOI: 10.25559/SITITO.14.201801.073-090

[27] *Рыбаков А.А., Телегин П.Н., Шабанов Б.М.* Проблемы векторизации гнезд циклов с использованием инструкций AVX-512 // Программные продукты, системы и алгоритмы. 2018. № 3. С. 1-11. DOI: 10.15827/2311-6749.28.314

Поступила 17.05.2018; принята в печать 10.08.2018;
опубликована онлайн 30.09.2018.

References

[1] Lyubimov D.A. Development and Application of a High-Resolution Technique for Jet Flow Computation Using Large Eddy Simulation. *High Temperature*. 2012; 50(3):420-436. DOI: 10.1134/S0018151X12020101

[2] Benderskii L.A., Lyubimov D.A., Chestnykh A.O., Shabanov B.M., Rybakov A.A. The Use of the RANS/ILES Method to Study the Influence of Coflow Wind on the Flow in a Hot, Nonisobaric, Supersonic Airdrome Jet during Its Interaction with the Jet Blast Deflector. *High Temperature*. 2018; 56(2):247-254. DOI: 10.1134/S0018151X18020037

[3] Lyubimov D.A. Investigation of the Effect of a Pylon and a Wing with Flaps on the Flow within an Exhaust Jet of a Double-Flow Turbojet Engine by a Simulation Method for Large Eddies. *High Temperature*. 2013; 51(1):111-127. DOI: 10.1134/S0018151X12050100

[4] Lyubimov D.A., Potekhina I.V. A study of unsteady-state operating conditions of a supersonic inlet by the RANS/ILES method. *High Temperature*. 2016; 54(5):737-744. DOI: 10.1134/S0018151X16050187

[5] Lyubimov D., Maslov V., Mironova A., Secundov A., Zakharov D. Experimental and Numeric Investigation of Jet Flap Interaction Effects. *International Journal of Aeroacoustics*. 2014; 13(3-4):275-302. DOI: 10.1260/1475-472X.13.3-4.275

[6] Benderskij L.A., Lyubimov D.A., Rybakov A.A. Scaling of fluid dynamic calculations using the RANS/ILES method on supercomputer. *Trudy NIISI RAN*. 2017; 7(4):32-40. DOI: 10.25682/NIISI.2018.4.9975 (In Russian).

[7] Rybakov A. Inner representation and crossprocess exchange mechanism for block-structured grid for supercomputer calculations. *Program systems: Theory and applications*. 2017; 8:1(32):121–134. (In Russian) DOI: 10.25209/2079-3316-2017-8-1-121-134

[8] Klenk B., Fröning H. An Overview of MPI Characteristics of Exascale Proxy Applications. J. M. Kunkel et al. (Eds.) *ISC High Performance 2017*. LNCS. Vol. 10266, pp. 217–236, 2017. DOI: 10.1007/978-3-319-58667-0_12

[9] Roganov V., Osipov V., Matveev G. Solving the 2D Poisson PDE by Gauss-Seidel method with parallel programming system OpenTS. *Program systems: theory and applications*. 2016; 7:3(30):99–107. (In Russian) DOI: 10.25209/2079-3316-2016-7-3-99-107

[10] Dorris J., Kurzak J., Luszczek P. Task-Based Cholesky Decomposition on Knights Corner Using OpenMP. M. Taufer et al. (Eds.) *ISC High Performance Workshops 2016*. LNCS. Vol. 9945, pp. 544–562, 2016. DOI: 10.1007/978-3-319-46079-6_37

[11] Krause M.J., Förster B., Mink A., Nirschl H. Towards Solving Fluid Flow Domain Identification Problems with Adjoint Lattice Boltzmann Methods. W. E. Nagel et al. (Eds.) *High Performance Computing in Science and Engineering'16*. Springer, Cham, pp. 337-353, 2016. DOI 10.1007/978-3-319-47066-5_23

[12] Golovchenko E., Dorofeeva E., Gasilova I., Boldareva A. Numerical Experiments with New Algorithms for Parallel Decomposition of Large Computational Meshes. *Advances in Parallel Computing*. 2014; 25:441-450. DOI: 10.3233/978-1-61499-381-0-441

[13] Petrov M.N., Titarev V.A., Utyuzhnikov S.V., Chikitkin A.V. A multithreaded OpenMP implementation of the LU-SGS method using the multilevel decomposition of the unstructured computational mesh. *Computational Mathematics and Mathematical Physics*. 2017; 57(11):1856-1865. DOI: 10.1134/S0965542517110124

[14] Rybakov A.A. Distribution of computational load between nodes of a heterogeneous computing cluster. *Programmnye produkty, sistemy i algoritmy*. 2018; 1:26-32. (In Russian) DOI: 10.15827/2311-6749.26.300

[15] Chetverushkin B.N., Yakobovskiy M.V. Numerical algorithms and fault tolerance of hyperexascale computer systems. *Doklady Mathematics*. 2017; 95(1):7-11. DOI: 10.1134/S1064562417010021

[16] Krzikalla O., Wende F., Höhnerbach M. Dynamic SIMD Vector Lane Scheduling. M. Taufer et al. (Eds.) *ISC High Performance Workshops 2016*. LNCS. Vol. 9945, pp. 354–365, 2016. DOI: 10.1007/978-3-319-46079-6_25

[17] Dikarev N., Shabanov B., Shmelv A. Fused MultiplyAdders Using in Vector Dataflow Processor. *Program systems: theory and applications*. 2015; 6:4(27):227–241. (In Russian) DOI: 10.25209/2079-3316-2015-6-4-227-241

[18] McDoniel W., Höhnerbach M., Canales R. et al. LAMMPS' PPPM Long-Range Solver for the Second Generation Xeon Phi. J. M. Kunkel et al. (Eds.) *ISC High Performance 2017*. LNCS. Vol. 10266, pp. 61–78, 2017. DOI: 10.1007/978-3-319-58667-0_4

[19] Malas T., Kurth T., Deslippe J. Optimization of the Sparse Matrix-Vector Products of an IDR Krylov Iterative Solver in EMGeo for the Intel KNL Manycore Processor. M. Taufer et al. (Eds.) *ISC High Performance Workshops 2016*. LNCS. Vol. 9945, pp. 378–389, 2016. DOI: 10.1007/978-3-319-46079-6_27

[20] Cook B., Maris P., Shao M. High Performance Optimizations for Nuclear Physics Code MFDn on KNL. M. Taufer et al. (Eds.) *ISC High Performance Workshops 2016*. LNCS. Vol. 9945, pp. 366–377, 2016. DOI: 10.1007/978-3-319-46079-6_26

[21] Rybakov A.A. Optimization of the problem of conflict detection with dangerous aircraft movement areas to execute on Intel Xeon Phi. *Programmnye produkty i sistemy* = Software & Systems. 2017; 30(3):524-528. (In Russian) DOI: 10.15827/0236-235.X.030.3.524-528

[22] Bramas B. A Novel Hybrid Quicksort Algorithm Vectorized using AVX-512 on Intel Skylake. *International Journal of Advanced Computer Science and Applications*. 2017; 8(10):337-344. DOI: 10.14569/IJACSA.2017.081044

[23] Sokolov A.P., Shchetinin V.N., Sapelkin A.S. Strength surface reconstruction using special parallel algorithm based on Intel MIC (Intel Many Integrated Core) architecture. *Program systems: theory and applications*. 2016; 7:2(29):3–25. (In Russian) DOI: 10.25209/2079-3316-2016-7-2-3-25

[24] Intel Math Kernel Library. Available at: https://software.intel.com/en-us/mkl (accessed 17.05.2018).

[25] Library targeting Intel Architecture for specialized dense and sparse matrix operations, and deep learning primitives. Available at: https://github.com/hfp/libxsmm (accessed 17.05.2018).

[26] Benderskij L.A., Leshchev S.A., Rybakov A.A. Vectorization of operations on small- dimensional matrices for intel xeon phi knights landing processor. *Modern Information Technology and IT-education*. 2018; 14(1):73-90. (In Russian) DOI: 10.25559/SITITO.14.201801.073-090

[27] Rybakov A.A., Telegin P.N., Shabanov B.M. Problems looping vectoring using instructions AVX-512. *Programmnye produkty, sistemy i algoritmy*. 2018(3):1-11. (In Russian) DOI: 10.15827/2311-6749.28.314

Submitted 17.05.2018; revised 10.08.2018;
published online 30.09.2018.

About the authors:

Leonid A. Benderskiy, Senior Researcher, Joint Supercomputer Center of the Russian Academy of Sciences – branch of Scientific Research Institute of System Analysis of the Russian Academy of Sciences (32a Leninsky Av., Moscow 119334, Russia), ORCID: http://orcid.org/0000-0003-0529-3255, leosun.ben@gmail.com

Alexey A. Rybakov, Candidate of Physical and Mathematical Sciences, Lead researcher, Joint Supercomputer Center of the Russian Academy of Sciences – branch of Scientific Research Institute of System Analysis of the Russian Academy of Sciences (32a Leninsky Av., Moscow 119334, Russia), ORCID: http://orcid.org/0000-0002-9755-8830,
rybakov.aax@gmail.com, rybakov@jscc.ru

Sergey S. Shumilin, Lead engineer, Joint Supercomputer Center of the Russian Academy of Sciences – branch of Scientific Research Institute of System Analysis of the Russian Academy of Sciences (32a Leninsky Av., Moscow 119334, Russia), ORCID: http://orcid.org/0000-0002-3953-7054, shumilin@jscc.ru