УДК 519.688
DOI: 10.25559/SITITO.18.202201.28-38

Научная статья

Векторизация программного кода, содержащего маловероятные регионы, в задачах вычислительной геометрии

А. А. Рыбаков
ФГУ «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», г. Москва, Российская Федерация
Адрес: 117218, Российская Федерация, г. Москва, Нахимовский пр., д. 36, корп. 1
rybakov@jscc.ru

Аннотация

Повышение производительности приложений является важной практической задачей для проведения суперкомпьютерных расчетов. Наряду с распараллеливанием вычислений между узлами кластера (например, средствами MPI), а также с многопоточным программированием (например, с помощью OpenMP) используется векторизация программного кода, обеспечивающая параллелизм на уровне отдельных инструкций. Набор векторных инструкций AVX-512 микропроцессорной архитектуры Intel, обладает рядом уникальных возможностей, с помощью которых можно векторизовать программный код из очень широкого спектра задач. Использование специальных масочных регистров позволяет эффективно векторизовать код, содержащий условные операторы, а использование профильной информации о вероятности выполнения операций в исходной коде позволяет выполнять преобразования кода, приводящие к еще более эффективному применению автоматической векторизации. В работе рассматривается преобразование расщепления плоского цикла по условию, когда известно, что данное условие выполняется с высокой вероятностью. Рассматриваются практические задачи, в которых присутствует данный контекст. Для этих задач выделены условия для расщепления циклов для достижения эффективной векторизации. Описанные преобразования были выполнены, а эффективность результирующего кода проверена запусками на микропроцессорах семейств Intel Xeon Cascade Lake и Intel Xeon Phi Knights Landing, представлены результаты выполнения запусков.

Ключевые слова: векторизация, высокопроизводительные вычисления, геометрические примитивы, AVX-512

Финансирование: настоящая работа выполнена в Межведомственном суперкомпьютерном центре Российской академии наук – филиале ФГУ «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук» (МСЦ РАН) при финансовой поддержке Российского фонда фундаментальных исследований в рамках научного проекта № 20-07-00594 А «Методы векторизации приложений суперкомпьютерного моделирования газодинамических процессов, использующих метод погруженной границы». Разработанные в рамках данного проекта программные коды были использованы для запусков на суперкомпьютере МВС-10П ОП, находящемся в МСЦ РАН.

Автор заявляет об отсутствии конфликта интересов.

Для цитирования: Рыбаков А. А. Векторизация программного кода, содержащего маловероятные регионы, в задачах вычислительной геометрии // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 1. С. 28-38. doi: https://doi.org/10.25559/SITITO.18.202201.28-38

Original article

Vectorization of Program Code Containing Low Probability Regions in Computational Geometry Problems

A. A. Rybakov
Scientific Research Institute for System Analysis of the Russian Academy of Sciences, Moscow, Russian Federation
Address: 36, build. 1, Nakhimov Ave., Moscow 117218, Russian Federation
rybakov@jscc.ru

Abstract

Improving application performance is an important practical task for supercomputer calculations. Along with parallelization of calculations between cluster nodes (for example, using MPI tools), as well as multithreaded programming (for example, using OpenMP), program code vectorization is used, which provides parallelism at the level of individual instructions. The AVX-512 vector instruction set of the Intel microprocessor architecture has a number of unique features that can be used to vectorize program code from a very wide range of applications. The use of special mask registers makes it possible to effectively vectorize code containing conditional statements, and the use of profile information about the probability of performing operations in the source code allows performing code transformations that lead to even more efficient use of automatic vectorization. The paper considers the splitting transformation of a flat loop by a condition when we have knowledge about of a high probability of this condition. Practical problems in which this context is present are considered. For these problems, the conditions for splitting cycles to achieve efficient vectorization are highlighted. The described transformations were performed, and the effectiveness of the resulting code was verified by runs on microprocessors of the Intel Xeon Cascade Lake and Intel Xeon Phi Knights Landing families, the results of the runs are presented.

Keywords: vectorization, high performance calculations, geometric primitives, AVX-512

Funding: This work was carried out at the Interdepartmental Supercomputer Center of the Russian Academy of Sciences in the branch of the Federal State Institution “Federal Scientific Center Research Institute for System Research of the Russian Academy of Sciences” with the financial support of the Russian Foundation for Basic Research within the framework of scientific project No. 20-07-00594 A "Methods for vectorizing applications of supercomputer simulation of gas-dynamic processes using the immersed boundary method". The program codes developed within the framework of this work were used for launches on the МВС-10П ОП supercomputer located in the Interdepartmental Super-computer Center of the Russian Academy of Sciences.

The author declares no conflict of interest.

For citation: Rybakov A. A. Vectorization of Program Code Containing Low Probability Regions in Computational Geometry Problems. *Sovremennye informacionnye tehnologii i IT-obrazovanie = Modern Information Technologies and IT-Education.* 2022; 18(1):28-38. doi: https://doi.org/10.25559/SITITO.18.202201.28-38

Введение

Векторизация является низкоуровневой оптимизацией, с помощью которой можно добиться ускорения программного кода в несколько раз путем объединения однотипных скалярных инструкций в векторные операции [1]. Во многих современных микропроцессорных архитектурах присутствует поддержка векторных инструкций [2-4], однако в архитектуре Intel специальный набор инструкций AVX-512 предоставляет уникальные возможности векторизации программного кода, обусловленные множеством их особенностей [5-7]. Данный набор содержит векторные поэлементные арифметические операции, операции перестановки элементов векторов в произвольном порядке, комбинированные операции для поэлементного вычисления выражений вида $\pm a \cdot b \pm c$, векторные операции конвертации, сравнения, множественные обращения в память с произвольным набором смещений от базового адреса и многие другие операции. Важнейшей отличительной особенностью набора инструкций AVX-512 является наличие специальных масочных регистров, которые могут быть использованы большинством векторных операций. С помощью масок задаются индексы тех элементов, над которыми требуется произвести операцию. Остальные элементы векторов, индексы которых не попадают под маску в результирующем векторе либо обнуляются, либо сохраняют свое предыдущее значение (зависит от флага операции). Использованием масочных инструкций позволяет векторизовать программный код с условными операторами путем помещения операций из разных ветвей исполнения под свои предикаты, которые при векторизации сливаются в векторные маски.

С помощью специальных методов возможно произвести векторизацию программного контекста практически произвольного вида. Для выполнения векторизации хорошо подходят задачи по работе с матрицами [8], задачи сортировки и поиска [9], методы решения нелинейных уравнений и многие другие задачи [10]. В данной статье мы будем рассматривать программный контекст специального вида – плоский цикл – с помощью которого можно объединить и выполнять на одном микропроцессорном ядре сразу несколько копий мелкой процедуры с практически произвольным исходным кодом.

Векторизация плоских циклов

Одним из наиболее удобных для векторизации видов контекста программного кода является плоский цикл [11]. Под плоским циклом в общем виде будем понимать следующую конструкцию:

```
for (int w = 0; w < WIDTH; w++)
{
block(w);
}
```

При этом будем считать, что плоский цикл удовлетворяет следующим соглашениям. Во-первых, будем считать, что внутри плоского цикла ведется работа с элементами данных одного размера, и количество итераций цикла в точности совпадает с количеством элементов в векторе (например, при использовании 512-битных векторных регистров и 32-битных элементов данных количество итераций плоского цикла должно быть ровно 16). Данное требование не ограничивает общности, так как цикл с большим количеством итераций можно разрезать на более короткие плоские циклы (при этом возможно возникновение цикла со слишком маленьким числом итераций – эпилога цикла – но в плане анализа производительности им можно пренебречь). Вторым требованием будем считать специальные ограничения на обращения в память внутри цикла. Так, условимся считать, что на итерации плоского цикла с номером i возможны обращения в память только вида x[i]. Данное требование является более жестким, однако в большинстве случаев исходный код может быть скорректирован для его удовлетворения. В дополнение к этому, для векторизации цикла с помощью инструкций AVX-512 будем считать, что все используемые массивы данных выровнены в памяти на 512 бит. Наконец, в качестве третьего требования зафиксируем отсутствие зависимостей между итерациями цикла. Это означает, что итерации плоского цикла могут выполняться в произвольном порядке и даже параллельно.

На первый взгляд может показаться, что накладываемые на плоский цикл требования сильно сужают круг программного контекста, пригодного для векторизации. Однако на практике многие расчетные задачи могут быть представлены в виде совокупности плоских циклов. Организация плоских циклов в вычислительных задачах может быть выполнена следующим образом. Допустим, что в некоторой практической задаче многократно требуется производить одинаковый набор вычислений над фиксированным набором входных данных и получать в результате него фиксированных набор выходных данных. Условно обозначим данную процедуру как fun(a, b, ...) ⟶ (x, y, ...), то есть некая функция f получает на вход набор скалярных значений a, b и т. д., а результатом ее является набор выходных значений x, y и т. д. Вычисления такого рода встречаются довольно часто, например, при решении задач компьютерного моделирования с использованием разностных схем и расчетных сеток (тогда в роли функции fun могут выступать различные действия по обработке данных отдельных расчетных ячеек). Если требуется выполнить несколько экземпляров функции fun над различными наборами данных, то эти наборы входных данных можно объединить в соответствующие массивы данных, и сами вычисления сформировать в виде цикла:

```
for (int w = 0; w < WIDTH; w++)
{
fun(a[w], b[w], ...) ⟶ (x[w], y[w], ...)
}
```

Выполнив такие несложные преобразования, мы получили плоский цикл, для которого можно добиться всех необходимых требований, предъявляемым к плоским циклам (его размер, выравнивание данных, отсутствие зависимостей между итерациями). Для упрощения векторизации можно выполнить inline-подстановку тела функции fun в место вызова (хотя вызовы функций также успешно векторизуются с использованием особенностей инструкций AVX-512).

Если тело плоского цикла содержит только простые арифме-

тические операции без условных операторов и операторов переходов (if, switch, continue, break и так далее), то векторизация осуществляется просто заменой скалярных инструкций их векторными аналогами – компилятор без труда справляется с данными преобразованиями (см. рис. 1).

Скалярный код

```
for (int i = 0; i < N; i++)
{
    d[i] = a[i] * b[i] + c[i];
}
```

Векторный код

```
vmovups a(,%rax,8), %zmm1
vmovups b(,%rax,8), %zmm0
vfmadd213pd c(,%rax,8), %zmm0, %zmm1
vmovupd %zmm1, d(,%rax,8)
```

Р и с. 1. Иллюстрация автоматической векторизации программного кода, не содержащего условий

F i g. 1. Illustration of automatic vectorization of program code without conditions

Сложности векторизации плоских циклов возникают в тех случаях, когда их тело содержит разветвленное управление (в этом случае компилятор может предсказать потери производительности при выполнении векторизации с помощью полного слияния в один линейный участок), гнезда циклов, вызовы функций, операторы передачи управления, работу с исключениями [12,13]. Между тем, часто встречается программный контекст, в котором тело плоского цикла содержит некоторую основную ветвь исполнения (наиболее вероятную), а также маловероятный код. Ниже мы рассмотрим примеры практических задач, реализация решения которых приводит к такому коду. А сейчас рассмотрим модельную запись такого плоского цикла в следующем виде:

```
for (int w = 0; w < WIDTH; w++)
{
        <block(w)>;
        if (<cond(w)>)
        {
        <block_true(w)>; // prob. ~ 100%
        }
        else
        {
        <block_false(w)>; // prob. ~ 0%
        }
}
```

В приведенном плоском цикле внутри его тела мы видим три блока с кодом. Блок кода <block(w)> выполнятся безусловно в начале тела цикла. А далее в зависимости от вычисляемого условия <cond(w)> (условие зависит от номера итерации, иначе можно было бы выполнить статическое расщепление цикла по условию) выполняется либо блок кода <block_true(w)> (с вероятностью, близкой к ста процентам), либо блок <block_false(w)> (с вероятностью, близкой к нулю). Будем считать, что блок <block(w)>, а также вероятный код является простым по структуре и пригодным для векторизации, тогда как код из блока <block_false(w)> должен выполняться редко, обрабатывает некоторые исключительные случаи и содержит крайне разветвленный и непригодный для векторизации контекст. В исходном виде компилятор, как правило, не справляется с векторизацией данного цикла, поэтому для его автоматической векторизации можно использовать преобразование исходного кода, связанное с расщеплением данного цикла по условию.

Для выполнения расщепления по условию создадим вместо одного цикла три новых. В первом цикле будем выполняться только блок <block(w)>, после которого все условия <cond(w)> записываются в локальный массив булевых значений (в маску). Во втором цикле на каждой итерации при истинном значении элемента tmp[w] выполняется ветка код из блока <block_true(w)>. Запись <block_true(w) ? tmp[w]> означает выполнение блока под предикатом. Третий же цикл инкапсулирует в себе выполнение маловероятного кода при условии ложности соответствующих значений tmp[w].

```
for (int w = 0; w < WIDTH; w++)
{
        <block(w)>;
        tmp[w] = <cond(w)>;
}

for (int w = 0; w < WIDTH; w++)
{
        <block_true(w) ? tmp[w]>;
}

if (<tmp - неполная маска>)
{
        for (int w = 0; w < WIDTH; w++)
        {
                if (!tmp[w])
                {
                        block_false(w);
                }
        }
}
```

В результате выполненных преобразований первый цикл содержит только векторизуемый безусловный код, второй цикл также содержит векторизуемый код, но с использованием предиката, который при векторизации преобразуется в векторную маску, накладываемую на все операции данного цикла. Третий код является маловероятным и его вообще не требуется оптимизировать (дополнительно перед его выполнением следует проверить маску на наличие в ней ложных элементов, что также в большинстве случаев избавит от необходимости выполнять все итерации цикла). В результате схематично векторизованная версия всех трех циклов будет выглядеть следующим образом:

```
<BLOCK();>
<tmp = COND();>

<BLOCK_TRUE() ? tmp;>

if (~tmp != 0x0)
{
        for (int w = 0; w < WIDTH; w++)
        {
```

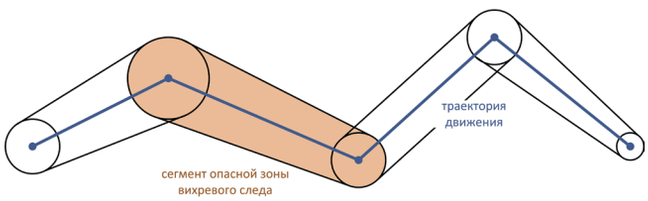
```
                if (!tmp[w])
                {
                        block_false(w);
                }
        }
}
```

Данное преобразования следует использовать при уверенности, что условие <cond(w)> является вероятным. При значении вероятности данного условия, близкой к 100%, теоретическое ускорение от применения векторизации будет близко к ширине векторизации (при условии, что <block(w)> и <block_true(w)> векторизуются идеально). Однако, если условие <cond(w)> не является вероятным, то выполнение данного преобразования приведет к деградации производительности. Ниже приведены примеры практических задач, контекст реализации которых хорошо подходит для выполнения данной оптимизации, после чего программный код успешно и автоматически векторизуется с помощью оптимизирующего компилятора (в данном случае использовался компилятор icc).

Численная задача обеспечения вихревой безопасности летательных аппаратов

В качестве первого примера рассмотрим задачу обеспечения вихревой безопасности летательного аппарата. Во время движения любой летательный аппарат создает турбулентные вихревые возмущения воздуха, которые остаются вдоль траектории его полета. Совокупность данных возмущений также называют вихревым следом. Интенсивность вихревого следа зависит от габаритов и скорости воздушного судна и других характеристик, а сам он может представлять серьезную опасность для других участников воздушного движения. Особенно это критично в местах скопления летательных аппаратов, где единовременно может находиться большое количество объектов. Со временем вихревой след эволюционирует, распадается на отдельные геометрические объекты и в конечном итоге пропадает. Для обеспечения безопасности полетов необходимо осуществлять мониторинг окружающего пространства и выполнять анализ собственной траектории полета на потенциальное пересечение с вихревыми следами всех окружающих объектов [14]. Положительный ответ на анализ конфликтов с вихревым следом какого-либо летательного аппарата требует принятия специальных мер реагирования, однако это крайне редкое событие относительно общего выполняемого объема мониторинга.

Для выполнения мониторинга требуется определить модель представления вихревого следа, а точнее опасной зоны, которую порождает данный вихревой след. Можно считать, что в фиксированный момент времени опасной зоной в некоторой точке траектории движения летательного аппарата является шар с центром в этой точке и радиусом, зависящим от интенсивности вихря. Траектория движения летательного аппарата представляется последовательностью точек в пространстве, в каждой из которых известен радиус опасной зоны вихревого следа (данная информация доступна участникам воздушного движения). Соединяя между собой сферы, соответствующие опасным зонам в соседних точках траектории движения, мы получим геометрическую модель опасной зоны вихревого следа летательного аппарата, как это показано на рис. 2.



Р и с. 2. Траектория движения летательного аппарата и опасная зона вихревого следа

F i g. 2. The trajectory of the aircraft and the danger zone of the vortex wake

Таким образом, для анализа конфликта траектории движения собственного летательного аппарата (которая в фиксированный момент времени представлена лучом) с опасной зоной вихревого следа какого-либо объекта (которая представлена в виде объединения отдельных сегментов) необходимо решить набор геометрических задач по анализу пересечений луча и сегмента опасной зоны вихревого следа. При этом собственная траектория движения может быть представлена в виде $\bar{P} = t\bar{V}$, где $\bar{P}$ – точка текущего положения собственного летательного аппарата, $\bar{V}$ – вектор его скорости, $t \geq 0$ – момент времени, начиная с текущего. Сегмент опасной зоны вихревого следа, с которым ищется пересечение, опирающийся на два шара $S_0(\bar{C}_0, R_0)$ и $S_1(\bar{C}_1, R_1)$ на концах соответствующего отрезка траектории движения, можно представить в виде семейства шаров, центры и радиусы которых задаются следующим образом:

$$\begin{cases} \overline{C(\alpha)} = \bar{C}_0 + \alpha(\bar{C}_1 - \bar{C}_0) = \bar{C}_0 + \alpha\overline{\Delta C} \\ R(\alpha) = R_0 + \alpha(R_1 - R_0) = R_0 + \alpha\Delta R \\ 0 \leq \alpha \leq 1 \end{cases}$$

Для анализа на пересечение с данным семейством шаров требуется проанализировать пересечение собственной траектории движения с поверхностью каждого шара из семейства (которая представляет собой сферу), то есть решить уравнение $|t\bar{V} - \overline{C(\alpha)}|^2 = R(\alpha)^2$. Решение данного уравнения существует при выполнении условия $\left(\overline{!(\alpha)}, \bar{V}\right)^2 - |\bar{V}|^2 \left(|\overline{C(\alpha)}|^2 - R(\alpha)^2\right) \geq 0$, так как это обычное квадратное уравнение относительно t. Для определения множества шаров из семейства, с которыми существует пересечение, нужно решить данное неравенство относительно α, что достигается прямой подстановкой выражений для $\overline{C(\alpha)}$ и $R(\alpha)$, после чего получится квадратное неравенство $A_2\alpha^2 + 2A_1\alpha + A_0 \geq 0$, в котором коэффициенты A_i вычисляются явно:

$$\begin{cases} A_2 = \left(\overline{\Delta C}, \bar{V}\right)^2 + |\bar{V}|^2 \left(\Delta R^2 - |\overline{\Delta C}|^2\right) \\ A_1 = \left(\bar{C}_0, \bar{V}\right)\left(\overline{\Delta C}, \bar{V}\right) + |\bar{V}|^2 \left(R_0\Delta R - \left(\bar{C}_0, \overline{\Delta C}\right)\right) \\ A_0 = \left(\bar{C}_0, \bar{V}\right)^2 + |\bar{V}|^2 \left(R_0^2 - |\bar{C}_0|^2\right) \end{cases}$$

Полученное неравенство относительно α требуется решить на отрезке $\alpha \in [0,1]$ в результате чего может получиться либо пустое множество, либо решение в виде $\alpha \in [\alpha_1, \alpha_2] \subseteq [0,1]$. В случае существования шаров, с поверхностью которых соб-

ственная траектория движения пересекается, моменты времени пересечения вычисляются с помощью решения исходного квадратного уравнения относительно t, и корни этого уравнения выражаются по формулам $t_{1,2}(\alpha)=\left(\left(\overline{C(\alpha)},\overline{V}\right)\pm\sqrt{A_2\alpha^2+2A_1\alpha+A_0}\right)|\overline{V}|^{-2}$. Искомые моменты времени вхождения собственной траектории в семейство шаров и покидания этого семейства соответствуют минимальному и максимальному значению функции $t_{1,2}(\alpha)$ на отрезке $[\alpha_1,\alpha_2]$. Максимальное и минимальное значение функции $t_{1,2}(\alpha)$ достигаются либо на концах отрезка $[\alpha_1,\alpha_2]$, либо в точках локального экстремума $t_{1,2}(\alpha)'=0$, что в явном виде приводит к решению квадратного уравнения $A_2(q-A_2)\alpha^2+2A_1(q-A_2)\alpha+(qA_0-A_1^2)=0$ относительно α, где $q=\left(\overline{\Delta C},\overline{V}\right)^2$. Решая данное квадратное уравнение, получим потенциальные точки локальных экстремумов $\hat{\alpha}_1$, $\hat{\alpha}_2$, которые учитываются только в случае попадания их внутрь отрезка $[\alpha_1,\alpha_2]$. В общем случае моменты начала и окончания контакта собственной траектории с сегментом опасной зоны вихревого следа находятся по четырем значениям параметра α:

$$\begin{cases} t_1=\min\{t_1(\alpha_1),t_1(\alpha_2),t_1(\hat{\alpha}_1),t_1(\hat{\alpha}_2)\} \\ t_2=\max\{t_2(\alpha_1),t_2(\alpha_2),t_2(\hat{\alpha}_1),t_2(\hat{\alpha}_2)\} \end{cases}$$

Накладывая ограничение $t\ge 0$, получим окончательное решение для одного сегмента опасной зоны. Во время осуществления мониторинга данная локальная задача решается для большого количества сегментов опасных зон. При этом цикл обработки всех доступных сегментов является плоским и, вообще говоря, может быть векторизован. Однако логика нахождения значений $t_{1,2}$ достаточно разветвленная, и векторизация всего программного кода неэффективна. Анализ физического смысла решаемой задачи, а также сбор профиля исполнения невекторизованного кода показал, что в подавляющем большинстве случаев верно условие $(A_2<0)\wedge(m>0)\wedge(A_1^2-A_2A_0<m^2)$, где $m=\max(A_1+A_2,-A_1)$, что соответствует отсутствию пересечения прямой, содержащей собственную траекторию, с сегментом опасной зоны. Данный вывод позволяет локализовать маловероятный регион программного кода, в котором содержится основной объем вычислений со сложной логикой.

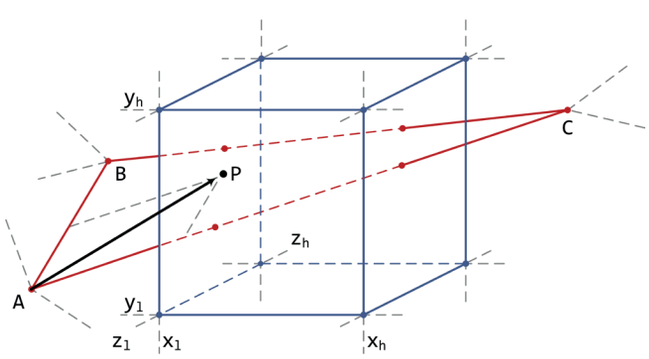
Численная задача классификации расчетных ячеек в методе погруженной границы

В качестве другого примера локализации маловероятных вычислений рассмотрим численное решение задач газовой динамики с помощью метода погруженной границы. Во время решения задач газовой динамики часто приходится сталкиваться с телами, которые обладают сложной или изменяющейся геометрией. Численное решение газодинамических задач вблизи поверхности таких тел стандартным способом сопряжены с проблемой построения расчетной сетки, согласованной с поверхностью. Метод погруженной границы позволяет использовать для расчетов простые и даже декартовы расчетные сетки, существенно снижая сложность организации вычислений [15,16]. Тонким местом применения метода является обработка граничных условий на сложной границе. Одним из подходов обработки граничных условий в данном методе является использование так называемых фиктивных ячеек [17]. Допустим, мы решаем задачу обтекания тела со сложной геометрией, при этом для расчетов будем использовать декартову сетку (ячейки которой имеют форму прямоугольного параллелепипеда), а поверхность представлена неструктурированной сеткой с треугольными ячейками. Идея метода погруженной границы базируется на разделении множества всех расчетных ячеек объемной сетки на 3 класса. Внешними ячейками назовем ячейки, целиком находящиеся вне обтекаемого тела. Внутренними, наоборот, назовем ячейки, которые целиком лежат внутри обтекаемого тела. Все остальные ячейки делятся поверхностью тела на внешнюю и внутреннюю области и называются граничными. Из граничных ячеек выделяются ячейки, у которых большая часть находится внутри тела, – они называются фиктивными. На каждой итерации расчетов для фиктивных ячеек требуется выполнять аппроксимацию газодинамических величин на основе данных окружающих ячеек, чтобы далее использовать эти величины для вычисления потоков между фиктивными ячейками и остальными ячейками, использующимися в расчетах.

Таким образом, важной составляющей реализации метода погруженной границы является поиск пересечения ячеек объемной сетки (прямоугольный параллелепипед) и ячеек неструктурированной поверхностной сетки (треугольник) [18]. Для определения факта пересечения будем представлять прямоугольный параллелепипед геометрическим местом точек $\overline{P}=(x,y,z)$, удовлетворяющих системе неравенств

$$\begin{cases} x_l\le x\le x_h \\ y_l\le y\le y_h \\ z_l\le z\le z_h \end{cases}$$

Треугольник в пространстве, вершинами которого являются точки $\overline{A}=(x_A,y_A,z_A)$, $\overline{B}=(x_B,y_B,z_B)$, $\overline{C}=(x_C,y_C,z_C)$ можно определить как геометрическое место точек $\overline{P}=\overline{A}+\alpha(\overline{B}-\overline{A})+\beta(\overline{C}-\overline{A})$, где $\alpha\ge 0$, $\beta\ge 0$, $\alpha+\beta\le 1$ (см. рис. 3).



Р и с. 3. Пересечение ячейки объемной сетки (прямоугольный параллелепипед) и ячейки неструктурированной сетки (треугольник)
F i g. 3. Intersection of a volume mesh cell (rectangular box) and an unstructured mesh cell (triangle)

Подставляя данное выражение в систему неравенств для прямоугольного параллелепипеда, получим следующую систему неравенств, решив которую, установим факт пересечения рассматриваемых объектов:

$$\begin{cases} x_l \le x_A + \alpha(x_B - x_A) + \beta(x_C - x_A) \le x_h \\ y_l \le y_A + \alpha(y_B - y_A) + \beta(y_C - y_A) \le y_h \\ z_l \le z_A + \alpha(z_B - z_A) + \beta(z_C - z_A) \le z_h \\ \alpha \ge 0 \\ \beta \ge 0 \\ \alpha + \beta \le 1 \end{cases}$$

Данная система может быть решена методом свертывания конечных систем линейных неравенств [19]. Для применения метода неравенства системы необходимо привести к виду $k_\alpha \alpha + k_\beta \beta + k \le 0$, после чего получим следующую систему:

$$\begin{cases} (x_B - x_A)\alpha + (x_C - x_A)\beta + (x_A - x_h) \le 0 \\ (x_A - x_B)\alpha + (x_A - x_C)\beta + (x_l - x_A) \le 0 \\ (y_B - y_A)\alpha + (y_C - y_A)\beta + (y_A - y_h) \le 0 \\ (y_A - y_B)\alpha + (y_A - y_C)\beta + (y_l - y_A) \le 0 \\ (z_B - z_A)\alpha + (z_C - z_A)\beta + (z_A - z_h) \le 0 \\ (z_A - z_B)\alpha + (z_A - z_C)\beta + (z_l - z_A) \le 0 \\ -1 \cdot \alpha + 0 \cdot \beta \le 0 \\ 0 \cdot \alpha + (-1) \cdot \beta \le 0 \\ \alpha + \beta + (-1) \le 0 \end{cases}$$

Данная система содержит всего две переменные, поэтому после выполнения одного шага свертывания она превратится в систему неравенств относительно одной переменной, решение такой системы не представляет сложности. Деформация системы для исключения из нее переменной α выполняется следующим образом. Составляется новая система, в которую войдут все неравенства исходной системы вида $k_\beta \beta + k \le 0$, а каждая пара неравенств

$$k_\alpha^1 \alpha + k_\beta^1 \beta + k^1 \le 0$$
$$k_\alpha^2 \alpha + k_\beta^2 \beta + k^2 \le 0$$

в которой коэффициенты при α удовлетворяют неравенствам $k_\alpha^1 < 0$, $k_\alpha^2 > 0$, войдет в деформированную систему в виде $(k_\beta^1 k_\beta^2 - k_\beta^2 k_\alpha^1)\beta + (k^1 k_\alpha^2 - k^2 k_\alpha^1) \le 0$. В результате выполнения деформации получим систему, состоящую не более чем из 17 неравенств (исходная система содержит 9 неравенств, из которых в одном коэффициент при α нулевой, в четырех – положительный, и в четырех – отрицательный).

Конечно, при нахождении пересечения пространственной и поверхностной сеток только малая доля пар ячеек действительно имеет пересечение. Нет необходимости перебирать все возможные пары объектов. Можно разбивать пространство на отдельные прямоугольные области, и если установлен факт, что две области не имеют пересечения, то и любые два объекта, целиком находящиеся в разных областях, также не пересекаются. Однако, даже при старте процедуры проверки пересечения с помощью решения системы неравенств, в подавляющем большинстве случаев выполняется условие

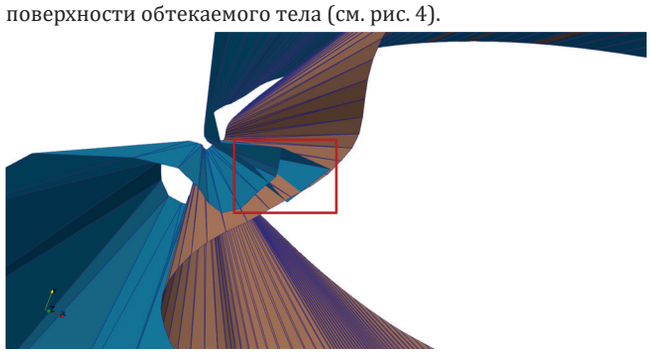
$$(x_{\max} < x_{lo}) \vee (y_{\max} < y_{lo}) \vee (z_{\max} < z_{lo}) \vee (x_{\min} > x_{hi}) \vee (y_{\min} > y_{hi}) \vee (z_{\min} > z_{hi})$$

где $x_{\max} = \max(x_A, x_B, x_C)$, $x_{\min} = \min(x_A, x_B, x_C)$, аналогично определяются $y_{\max}$, $y_{\min}$, $z_{\max}$, $z_{\min}$. Это условие гарантирует отсутствие пересечения между треугольником и прямоугольным параллелепипедом. Вычисление этого простого условия легко векторизуется и помогает избежать необходимости векторизации одновременного решения нескольких копий систем линейных неравенств (хотя данный код также можно векторизовать, но с некоторыми потерями в производительности из-за наличия в нем условных операторов и гнезда циклов).

Численная задача устранения самопересечений поверхностных сеток

В качестве третьего примера практического применения функционала, содержащего маловероятные ветви исполнения, рассмотрим задачу, возникающую при эволюции поверхности тела при моделировании формирования ледяного нароста на поверхности летательного аппарата при обдувании свободным потоком [20,21]. Поверхность обтекаемого тела моделируется с помощью неструктурированной расчетной сетки, ячейками которой являются треугольники. Для расчета обледенения в каждой расчетной ячейке на каждом шаге времени решается система уравнений массового и теплового баланса, из которой находится масса накопленного в ячейке льда. Во второй фазе решения задачи на основании информации о массе льда в ячейке (а значит и о высоте ледяного покрова) производится деформация расчетной сетки [22,23]. Во время деформации для каждой точки поверхностной сетки вычисляется вектор смещения (в простейшем случае считается, что во время деформации выполняется только перемещение узлов сетки без образования новых ячеек), и все узлы сетки одновременно перемещаются, образуя новую поверхность, которая передается на следующий шаг расчета обледенения.

Несмотря на комплексность и большое количество требуемых на перестроение поверхности ресурсов, нельзя гарантировать, что результирующая сетка останется корректной. Наиболее серьезной потенциальной проблемой в данном случае является возникновение самопересечений сетки, что приводит к аварийной остановке расчетов, так как при наличии самопересечений возникают сложности с идентификацией истинной поверхности обтекаемого тела (см. рис. 4).



Р и с. 4. Возникновение самопересечений поверхностной расчетной сетки при деформации (область конфликта выделена красным прямоугольником)

F i g. 4. The occurrence of self-intersections of the surface computational grid during deformation (the conflict area is highlighted with a red rectangle)

Таким образом, после перестроения поверхностной сетки необходимо выполнить проверку сетки на самопересечение и

удалить скрытые под поверхностью петли и складки при наличии таковых [24,25]. В общем случае для решения этой задачи нужно решить множество задач на определение пересечения в пространстве двух треугольников (с той оговоркой, что, как и в предыдущем примере, можно упростить задачу, разбивая пространство на непересекающиеся области). То есть поставленная задача сводится к геометрической задаче определения пересечения двух треугольников. Для решения этой задачи можно воспользоваться следующим простым фактом: если два треугольника в пространстве пересекаются, то хотя бы одна сторона хотя бы одного треугольника пересекает второй треугольник. То есть задача пересечения двух треугольников в пространстве сводится к решению шести задач пересечения треугольника с отрезком. Треугольник, как и ранее, будем представлять в виде геометрического места точек $\overline{P} = \overline{A} + \alpha(\overline{B} - \overline{A}) + \beta(\overline{C} - \overline{A})$, где $\alpha \geq 0$, $\beta \geq 0$, $\alpha + \beta \leq 1$, а отрезок – в виде геометрического места точек $\overline{P} = \overline{R} + \varphi(\overline{Q} - \overline{R})$, где $0 \leq \varphi \leq 1$. То есть решение задачи сводится к решению системы из трех линейных уравнений относительно переменных α, β, φ: $\alpha(\overline{B} - \overline{A}) + \beta(\overline{C} - \overline{A}) + \varphi(\overline{R} - \overline{Q}) = \overline{R} - \overline{A}$. Без учета ограничений на переменные α, β, φ данная система может иметь одно решение (если прямая RQ пересекает плоскость ABC), иметь бесконечное число решений (прямая лежит в плоскости) и не иметь решений (прямая параллельна плоскости). Наиболее частым случаем является вариант ровно одного решения данной системы, что соответствует условию неравенства нулю определителя

$$\Delta = \begin{vmatrix} x_B - x_A & x_C - x_A & x_R - x_Q \\ y_B - y_A & y_C - y_A & y_R - y_Q \\ z_B - z_A & z_C - z_A & z_R - z_Q \end{vmatrix}$$

Он же и является наиболее простым, так как в этом случае явно находится точка пересечения прямой и плоскости, после чего, применяя ограничения на переменные, устанавливается факт пересечения отрезка с треугольником. Случай же нахождения прямой в плоскости треугольника является наиболее сложным, так как содержит большое количество частных случаев, которые требуется обработать в коде. Поэтому условие $\Delta \neq 0$ может быть использовано для повышения эффективности векторизации.

Заключение

Рассмотренный в данной работе метод расщепления плоского цикла по вероятному условию был апробирован на наборе задач вычислительной геометрии: задача нахождения пересечения луча с семейством сфер (в рамках задачи мониторинга вихревой обстановки при движении летательного аппарата), задача установления факта пересечения в пространстве треугольника с прямоугольным параллелепипедом (в рамках реализации численных методов погруженной границы), задача нахождения точек пересечения двух треугольников в пространстве (в рамках нахождения самопересечений неструктурированной поверхностной сетки). Описанные задачи были реализованы на языке программирования C++, для них была выполнена векторизация кода и произведены запуски на вычислительных узлах на базе микропроцессоров Intel Xeon Gold 6248R (Cascade Lake) и Intel Xeon Phi 7290 (Knights Landing). При выполнении векторизации использовались вещественные данные одинарной точности (float размера 32 бита), таким образом идеальный теоретический показатель ускорения от векторизации с помощью 512-битных векторных инструкций равняется 16.

Задача нахождения пересечения луча с семейством сфер оказалась непригодной для векторизации в исходном виде ввиду слишком разветвленного управления и наличия тяжелых операций (извлечение квадратного корня). Однако после расщепления цикла по условию $(A_2 < 0) \wedge (m > 0) \wedge (A_1^2 - A_2 A_0 < m^2)$ компилятор icc успешно выполнил автоматическую векторизацию, при этом ускорение векторизованного кода на микропроцессоре Intel Xeon Phi Knights Landing составило 5,1 раза по сравнению со скалярной версией кода.

Для задачи установления факта пересечения в пространстве треугольника с прямоугольным параллелепипедом в рамках работы [18] была выполнена полная ручная векторизация исходного кода с использованием специальных функций-интринсиков, ускорение на микропроцессоре Intel Xeon Phi Knights Landing составило 6,7 раз. Использование расщепления плоского цикла по условию $(x_{\max} < x_{lo}) \vee (y_{\max} < y_{lo}) \vee (z_{\max} < z_{lo}) \vee (x_{\min} > x_{hi}) \vee (y_{\min} > y_{hi}) \vee (z_{\min} > z_{hi})$ и выполнение автоматической векторизации с помощью компилятора icc снизило этот показатель до 4,8 раз, однако исходный код стал существенно проще.

Задача нахождения точек пересечения двух треугольников в пространстве оказалась непригодной для векторизации в исходном виде. После выполнения расщепления основного плоского цикла по условию $\Delta \neq 0$, автоматическая векторизация успешно применилась, и результирующий код продемонстрировал ускорение 4,6 раз на микропроцессоре Intel Xeon Cascade Lake.

Описанный в работе прием по модификации исходного кода плоского цикла по вероятному условию позволяет компилятору автоматически применять векторизацию к получившимся циклам, что приводит к существенному ускорению приложений с минимальными затратами человеческих усилий.

Список использованных источников

[1] Effective SIMD Vectorization for Intel Xeon Phi Coprocessors / X. Tian, H. Saito, S. V. Preis, E. N. Garcia [и др.] // Scientific Programming. 2015. Vol. 2015. Article number: 269764. doi: https://doi.org/10.1155/2015/269764

[2] Pohl A., Cosenza B., Juurlink B. Control Flow Vectorization for ARM NEON // SCOPES'18: Proceedings of the 21th International Workshop on Software and Compilers for Embedded Systems. New York, NY, USA: ACM, 2018. P. 66-75. doi: https://doi.org/10.1145/3207719.3207721

[3] Vector Processing Unit: A RISC-V based SIMD Co-processor for Embedded Processing / M. Ali, M. von Ameln, D. Goehringer, F. Leporati [и др.] // Proceedings of the 24th Euromicro Conference on Digital System Design. Palermo, Italy: IEEE Press, 2021. P. 30-34. doi: https://doi.org/10.1109/DSD53832.2021.00014

[4] Методы распараллеливания программ в оптимизирующем компиляторе для ВК семейства Эльбрус / В. Ю. Волконский, А. В. Грабежной [и др.] // Современные информационные технологии и ИТ-образование. 2011. № 7. С. 46-59. URL: https://elibrary.ru/item.asp?id=23020730 (дата обращения: 17.01.2022).

[5] Cebrian J. M., Natvig L., Jahre M. Scalability Analysis of AVX-512 Extensions // Journal of Supercomputing. 2020. Vol. 76, issue 3. P. 2082-2097. doi: https://doi.org/10.1007/s11227-019-02840-7

[6] Shabanov B. M., Rybakov A. A., Shumilin S. S. Vectorization of High-performance Scientific Calculations Using AVX-512 Intruction Set // Lobachevskii Journal of Mathematics. 2019. Vol. 40, issue 5. P. 580-598. doi: https://doi.org/10.1134/S1995080219050196

[7] Hossain M. M., Saule E. Impact of AVX-512 Instructions on Graph Partitioning Problems // Proceedings of the 50th International Conference on Parallel Processing Workshop. New York, NY, USA: ACM, 2021. Article number: 33. P. 1-9. doi: https://doi.org/10.1145/3458744.3473362

[8] 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 // High Performance Computing. ISC High Performance 2016. Lecture Notes in Computer Science; M. Taufer, B. Mohr, J. Kunkel (eds.). Vol. 9945. Springer, Cham, 2016. P. 378-389. doi: https://doi.org/10.1007/978-3-319-46079-6_27

[9] 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, issue 10. P. 337-344. doi: http://dx.doi.org/10.14569/IJACSA.2017.081044

[10] LAMMPS' PPPM Long-Range Solver for the Second Generation Xeon Phi / W. McDoniel, M. Höhnerbach [и др.] // High Performance Computing. ISC High Performance 2017. Lecture Notes in Computer Science; J. M. Kunkel, R. Yokota, P. Balaji, D. Keyes (eds.). Vol. 10266. Springer, Cham, 2017. P. 61-78. doi: https://doi.org/10.1007/978-3-319-58667-0_4

[11] Vectorization of Flat Loops of Arbitrary Structure Using Instructions AVX-512 / G. I. Savin, B. M. Shabanov, A. A. Rybakov, S. S. Shumilin // Lobachevskii Journal of Mathematics. 2020. Vol. 41, issue 12. P. 2566-2574. doi: https://doi.org/10.1134/S1995080220120331

[12] Krzikalla O., Wende F., Höhnerbach M. Dynamic SIMD Vector Lane Scheduling // High Performance Computing. ISC High Performance 2016. Lecture Notes in Computer Science; M. Taufer, B. Mohr, J. Kunkel (eds.). Vol. 9945. Springer, Cham, 2016. P. 354-365. doi: https://doi.org/10.1007/978-3-319-46079-6_25

[13] Rybakov A. A., Shumilin S. S. Vectorization of the Riemann solver using the AVX-512 instruction set // Program Systems: Theory and Applications. 2019. Vol. 10, № 3(42). P. 41-58. doi: https://doi.org/10.25209/2079-3316-2019-10-3-41-58

[14] 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: https://doi.org/10.15827/0236-235X.119.3.524-528

[15] Абалакин И. В., Жданова Н. С., Козубская Т. К. Метод погруженных границ для численного моделирования невязких сжимаемых течений // Журнал вычислительной математики и математической физики. 2018. Т. 58, № 9. С. 1462 1471. doi: https://doi.org/10.31857/S004446690002525-8

[16] Mori Y., Peskin C. S. Implicit second-order immersed boundary methods with boundary mass // Computer Methods in Applied Mechanics and Engineering. 2008. Vol. 197, issues 25-28. P. 2049-2067. doi: https://doi.org/10.1016/j.cma.2007.05.028

[17] Peter S., De A. K. A parallel implementation of the ghost-cell immersed boundary method with application to stationary and moving boundary problems // Sadhana. 2016. Vol. 41, issue 4. P. 441-450. doi: https://doi.org/10.1007/s12046-016-0484-9

[18] Рыбаков А. А. Векторизация нахождения пересечения объемной и поверхностной сеток для микропроцессоров с поддержкой AVX-512 // Труды НИИСИ РАН. 2019. Т. 9, № 5. С. 5-14. URL: https://elibrary.ru/item.asp?id=41595664 (дата обращения: 17.01.2022).

[19] Черников С. Н. Свертывание конечных систем линейных неравенств // Доклады АН СССР. 1963. Т. 152, № 5. С. 1075-1078.

[20] Bourgault Y., Beaugendre H., Habashi W. Development of a shallow-water icing model in FENSAP-ICE // Journal of Aircraft. 2000. Vol. 37, issue 4. P. 640-646. doi: https://doi.org/10.2514/2.2646

[21] Fu P., Farzaneh M., Bouchard G. Modeling a Water Flow on an Icing Surface // Proceedings of the 11th International Workshop on Atmospheric Icing of Structures (IWAIS'2005). Montreal, Canada; 2005. URL: https://www.compusult.com/web/iwais/iwais-2005 (дата обращения: 17.01.2022).

[22] Discrete Surface Evolution and Mesh Deformation for Aircraft Icing Applications / Thompson D. [и др.] // Proceedings of the 5th AIAA Atmospheric and Space Environments Conference. San Diego, CA, 2013. doi: https://doi.org/10.2514/6.2013-2544

[23] Three-Dimensional Surface Evolution and Mesh Deformation for Aircraft Icing Application / X. Tong, D. Thompson, Q. Arnoldus, E. Collins, E. Luke // Journal of Aircraft. 2017. Vol. 54, no. 3. P. 1047-1063. doi: https://doi.org/10.2514/1.C033949

[24] Jung W., Shin H., Choi B. K. Self-intersection Removal in Triangular Mesh Offsetting // Computer-Aided Design and Applications. 2004. Vol. 1. P. 477-484. doi: https://doi.org/10.1080/16864360.2004.10738290

[25] Skorkovská V., Kolingerová I., Benes B. A Simple and Robust Approach to Computation of Meshes Intersection // Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications. Vol. 1. SciTePress, 2018. P. 175-182. doi: https://doi.org/10.5220/0006538401750182

Поступила 17.01.2022; одобрена после рецензирования 05.03.2022; принята к публикации 14.03.2022.

Об авторе:

Рыбаков Алексей Анатольевич, ведущий научный сотрудник Межведомственного суперкомпьютерного центра Российской академии наук – филиала Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук» (119334, Российская Федерация, г. Москва, Ленинский пр., д. 32а), кандидат физико-математических наук, **ORCID: https://orcid.org/0000-0002-9755-8830**, rybakov@jscc.ru

Автор прочитал и одобрил окончательный вариант рукописи.

References

[1] Tian X., Saito H., Preis S.V., Garcia E.N., Kozhukhov S.S. Effective SIMD Vectorization for Intel Xeon Phi Coprocessors. *Scientific Programming*. 2015; 2015:269764. (In Eng.) doi: https://doi.org/10.1155/2015/269764

[2] Pohl A., Cosenza B., Juurlink B. Control Flow Vectorization for ARM NEON. *SCOPES'18: Proceedings of the 21th International Workshop on Software and Compilers for Embedded Systems*. ACM, New York, NY, USA; 2018. p. 66-75. (In Eng.) doi: https://doi.org/10.1145/3207719.3207721

[3] Ali M., et al. Vector Processing Unit: A RISC-V based SIMD Co-processor for Embedded Processing. *Proceedings of the 24th Euromicro Conference on Digital System Design.* IEEE Press, Palermo, Italy; 2021. p. 30-34. (In Eng.) doi: https://doi.org/10.1109/DSD53832.2021.00014

[4] Volkonsky V.Yu., et al. *Metody rasparallelivaniya programm v optimiziruyushchem kompilyatore dlya VK semejstva El'brus* [Methods for parallelizing programs in an optimizing compiler for a computer complex of the Elbrus family]. *Sovremennye informacionnye tehnologii i IT*-obrazovanie = Modern Information Technologies and IT-Education. 2011; (7):46-59. Available at: https://elibrary.ru/item.asp?id=23020730 (accessed 17.01.2022). (In Russ., abstract in Eng.)

[5] Cebrian J.M., Natvig L., Jahre M. Scalability Analysis of AVX-512 Extensions. *Journal of Supercomputing*. 2020; 76(3):2082-2097. (In Eng.) doi: https://doi.org/10.1007/s11227-019-02840-7

[6] Shabanov B.M., Rybakov A.A., Shumilin S.S. Vectorization of High-performance Scientific Calculations Using AVX-512 Intruction Set. *Lobachevskii Journal of Mathematics*. 2019; 40(5):580-598. (In Eng.) doi: https://doi.org/10.1134/S1995080219050196

[7] Hossain M.M., Saule E. Impact of AVX-512 Instructions on Graph Partitioning Problems. *Proceedings of the 50th International Conference on Parallel Processing Workshop.* ACM, New York, NY, USA; 2021. Article number: 33. p. 1-9. (In Eng.) doi: https://doi.org/10.1145/3458744.3473362

[8] 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. In: Taufer M., Mohr B., Kunkel J. (eds.) *High Performance Computing. ISC High Performance 2016. Lecture Notes in Computer Science*. Vol. 9945. Springer, Cham; 2016. p. 378-389. (In Eng.) doi: https://doi.org/10.1007/978-3-319-46079-6_27

[9] 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. (In Eng.) doi: http://dx.doi.org/10.14569/IJACSA.2017.081044

[10] McDoniel W., Höhnerbach M., Canales R., Ismail A.E., Bientinesi P. LAMMPS' PPPM Long-Range Solver for the Second Generation Xeon Phi. In: Kunkel J.M., Yokota R., Balaji P., Keyes D. (eds.) *High Performance Computing. ISC High Performance 2017. Lecture Notes in Computer Science*. Vol. 10266. Springer, Cham; 2017. p. 61-78. (In Eng.) doi: https://doi.org/10.1007/978-3-319-58667-0_4

[11] Savin G.I., Shabanov B.M., Rybakov A.A., Shumilin S.S. Vectorization of Flat Loops of Arbitrary Structure Using Instructions AVX-512. *Lobachevskii Journal of Mathematics*. 2020; 41(12):2566-2574. (In Eng.) doi: https://doi.org/10.1134/S1995080220120331

[12] Krzikalla O., Wende F., Höhnerbach M. Dynamic SIMD Vector Lane Scheduling. In: Taufer M., Mohr B., Kunkel J. (eds.) *High Performance Computing. ISC High Performance 2016. Lecture Notes in Computer Science.* Vol. 9945. Springer, Cham; 2016. p. 354-365. (In Eng.) doi: https://doi.org/10.1007/978-3-319-46079-6_25

[13] Rybakov A.A., Shumilin S.S. Vectorization of the Riemann solver using the AVX-512 instruction set. *Program Systems: Theory and Applications*. 2019; 10(3):41-58. (In Eng.) doi: https://doi.org/10.25209/2079-3316-2019-10-3-41-58

[14] 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 Russ., abstract in Eng.) doi: https://doi.org/10.15827/0236-235X.030.3.524-528

[15] Abalakin I.V., Zhdanova N.S., Kozubskaya T.K. Immersed Boundary Method for Numerical Simulation of Inviscid Compressible Flows. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* = Computational Mathematics and Mathematical Physics. 2018; 58(9):1411-1419. (In Eng.) doi: https://doi.org/10.1134/S0965542518090026

[16] Mori Y., Peskin C.S. Implicit second-order immersed boundary methods with boundary mass. *Computer Methods in Applied Mechanics and Engineering*. 2008; 197(25-28):2049-2067. (In Eng.) doi: https://doi.org/10.1016/j.cma.2007.05.028

[17] Peter S., De A.K. A parallel implementation of the ghost-cell immersed boundary method with application to stationary and moving boundary problems. *Sadhana*. 2016; 41(4):441-450. (In Eng.) doi: https://doi.org/10.1007/s12046-016-0484-9

[18] Rybakov A.A. *Vektorizaciya nahozhdeniya peresecheniya ob"emnoj i poverhnostnoj setok dlya mikroprocessorov s podderzhkoj AVX-512* [Vectorization of finding the intersection of volume grid and surface grid for microprocessors with AVX-512 support]. *Trudy NIISI RAN* = Proceedings of NIISI RAS. 2019; 9(5):5-14. Available at: https://elibrary.ru/item.asp?id=41595664 (accessed 17.01.2022). (In Russ., abstract in Eng.)

[19] Chernikov S. N. *Svertyvanie konechnyh sistem linejnyh neravenstv* [Collapse of finite systems of linear inequalities]. *Doklady AN SSSR* = Doklady of the USSR Academy of Sciences. 1963; 152(5):1075-1078. (In Russ.)
[20] Bourgault Y., Beaugendre H., Habashi W. Development of a shallow-water icing model in FENSAP-ICE. *Journal of Aircraft*. 2000; 37(4):640-646. (In Eng.) doi: https://doi.org/10.2514/2.2646
[21] Fu P., Farzaneh M., Bouchard G. Modeling a Water Flow on an Icing Surface. *Proceedings of the 11th International Workshop on Atmospheric Icing of Structures (IWAIS'2005).* Montreal, Canada; 2005. Available at: https://www.compusult.com/web/iwais/iwais-2005 (accessed 17.01.2022). (In Eng.)
[22] Thompson D., et al. Discrete Surface Evolution and Mesh Deformation for Aircraft Icing Applications. *Proceedings of the 5th AIAA Atmospheric and Space Environments Conference*. San Diego, CA; 2013. (In Eng.) doi: https://doi.org/10.2514/6.2013-2544
[23] Tong X., Thompson D., Arnoldus Q., Collins E., Luke E. Three-Dimensional Surface Evolution and Mesh Deformation for Aircraft Icing Application. *Journal of Aircraft*. 2017; 54(3):1047-1063. (In Eng.) doi: https://doi.org/10.2514/1.C033949
[24] Jung W., Shin H., Choi B.K. Self-intersection Removal in Triangular Mesh Offsetting. *Computer-Aided Design and Applications*. 2004; 1:477-484. (In Eng.) doi: https://doi.org/10.1080/16864360.2004.10738290
[25] Skorkovská V., Kolingerová I., Benes B. A Simple and Robust Approach to Computation of Meshes Intersection. *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. Vol. 1. SciTePress; 2018. p. 175-182. (In Eng.) doi: https://doi.org/10.5220/0006538401750182

Submitted 17.01.2022; approved after reviewing 05.03.2022; accepted for publication 14.03.2022.

About the author:

Alexey A. Rybakov, Lead Researcher in Joint Supercomputer Center of the Russian Academy of Sciences – Branch of the Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences" (32a Leninsky Ave., Moscow 119334, Russian Federation), Cand.Sci. (Phys.-Math.), **ORCID: https://orcid.org/0000-0002-9755-8830,** rybakov@jscc.ru

The author has read and approved the final manuscript.