https://doi.org/10.25559/SITITO.020.202403.678-686
УДК 004.4'422

Оригинальная статья

Применение векторных операций с динамической длиной для эффективной эмуляции векторных операций с фиксированной длиной

К. И. Владимиров, И. А. Тетюшкин*
ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)», г. Долгопрудный, Российская Федерация
Адрес: 141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9
* tetyushkin.ia@phystech.edu

Аннотация

Использование векторных регистров – один из наиболее эффективных способов повышения производительности процессора, особенно в задачах, связанных с параллельной обработкой данных. Для обеспечения переносимости кода между различными архитектурами разработчики часто прибегают к использованию сторонних библиотек, которые абстрагируют архитектурно-зависимые векторные операции через высокоуровневые конструкции языка программирования. Однако создание прозрачных обёрток над базовыми типами данных требует глубокого понимания как особенностей библиотеки, так и архитектурных отличий векторных расширений. Это особенно важно при портировании на новые платформы, такие как RISC-V, где принципы работы с векторами могут существенно отличаться от других архитектур. В статье рассматриваются существующие библиотеки для создания обобщённых векторных алгоритмов и предлагаются решения для добавления поддержки векторного расширения RISC-V в активно развивающуюся библиотеку EVE. Несмотря на наличие в EVE поддержки масштабируемой векторизации через SVE, интеграция RISC-V требует решения ряда дополнительных задач. Основной вызов – эффективное использование уникальных особенностей RISC-V, таких как группировка нескольких векторных регистров, а также адаптация существующих возможностей библиотеки для работы с новыми аппаратными особенностями, включая ограниченные операции над векторами, которые недоступны на уровне архитектуры.

Ключевые слова: RISC-V, векторные регистры, масштабируемая векторизация, EVE, обобщённые векторные алгоритмы

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Владимиров К. И., Тетюшкин И. А. Применение векторных операций с динамической длиной для эффективной эмуляции векторных операций с фиксированной длиной // Современные информационные технологии и ИТ-образование. 2024. Т. 20, № 3. С. 678-686. https://doi.org/10.25559/SITITO.020.202403.678-686

Original article

Using Dynamic-Length Vector Operations to Efficiently Emulate Fixed-Length Vector Operations

K. I. Vladimirov, I. A. Tetyushkin*
Moscow Institute of Physics and Technology (National Research University), Dolgoprudny, Russian Federation
Address: 9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation
* tetyushkin.ia@phystech.edu

Abstract
Using vector registers can significantly enhance processor performance, especially in tasks that involve parallel data processing. To ensure code portability across different architectures, developers often use third-party libraries. These libraries abstract architecture-specific vector operations using high-level programming constructs. However, creating transparent wrappers over the base data types requires a deep understanding of both the library's features and the architectural differences between the vector extensions. This is especially important when porting to new platforms such as RISC-V, where the principles of working with vectors may differ significantly from other architectures. This article reviews existing libraries for creating generalized vector algorithms and proposes solutions for adding support for the RISC-V vector extension to the EVE library, which is currently in active development. Although EVE supports scalable vectorization through SVE, integrating with RISC-V presents a number of additional challenges that need to be addressed. The main challenge is to efficiently utilize the unique features of RISC-V, such as the grouping of multiple vector registers. Additionally, we need to adapt existing libraries to work with the new hardware features. These features include limited vector operations that are not available at the architectural level.

Keywords: RISC-V, vector registers, scalable vectorization, EVE, generalized vector algorithms

Conflict of interests: The authors declares no conflict of interest.

For citation: Vladimirov K.I., Tetyushkin I.A. Using Dynamic-Length Vector Operations to Efficiently Emulate Fixed-Length Vector Operations. *Modern Information Technologies and IT-Education.* 2024;20(3):678-686. https://doi.org/10.25559/SITITO.020.202403.678-686

Введение

С развитием электронных вычислительных систем их производительность постоянно растёт [1]. Это достигается разными способами. Одним из таких способов является распараллеливание вычислений, при котором часть независимых задач выполняется на дополнительных устройствах, а результаты используются на основном вычислительном блоке. Этот подход реализован в таких фреймворках, как OpenMP [2], OpenCL [3] и MPI [4].

Другой метод повышения производительности заключается в выявлении часто используемых фрагментов кода и их оптимизации путем объединения в одну специализированную операцию. Этот подход лежит в основе расширений для битовых манипуляций, таких как x86 ABM, BMI1, BMI2, RISC-V bitmanip [5] и других.

Ещё одним способом ускорения вычислений являются векторные инструкции, предназначенные для одновременной обработки нескольких элементов данных [6]. Исторически они были разработаны для графических вычислений, поэтому первые реализации оперировали фиксированным размером вектора и могли обрабатывать только определенное количество элементов [19]. Эта особенность проявлялась в реализациях библиотек, абстрагирующих работу с конкретными векторными расширениями, таких как Boost.SIMD и EVE.

Недавно появились архитектуры, поддерживающие динамическую длину векторных операций, такие как ARM SVE [7] и RISC-V RVV [20]. Масштабируемая векторизация позволяет эффективнее использовать векторные регистры и сокращать размер кода за счёт устранения обработки данных, не кратных размеру векторного регистра (так называемых хвостов).

Для корректной работы с такими операциями необходимо заранее установить активную длину вектора [21], что усложняется в случае RISC-V из-за возможности использования до восьми базовых векторных регистров в одной операции.

При программировании на низком уровне с использованием векторных инструкций такой код становится плохо переносимым [22] между архитектурами с разными типами векторизации, или даже между разными архитектурами с масштабируемой векторизацией.

Поэтому важной задачей становится абстрагирование векторизации [23]. Существующие решения, такие как библиотека EVE, позволяют писать векторный код на высоком уровне, используя языковые и библиотечные абстракции, сохраняя при этом производительность кода на уровне, сравнимом с ручным использованием векторных инструкций [25].

Данная статья описывает реализацию поддержки RISC-V в библиотеке EVE [24]. Рассматриваются различные типы векторных вычислений и особенности масштабируемой векторизации в RISC-V. Проводится сравнительный анализ существующих высокоуровневых библиотек. Излагаются основные проблемы и решения, связанные с реализацией поддержки RVV в библиотеке EVE.

Векторные вычисления с фиксированной длиной

Исторически первые архитектуры с поддержкой SIMD *(single instruction-multiple data)* использовали фиксированное количество обрабатываемых элементов. Этот подход был удобен, так как позволял сделать векторный модуль ограниченно конфигурируемым, что значительно упрощало его разработку и интеграцию в вычислительные блоки.

При этом подходе возникают проблемы, когда необходимо выполнить операцию над группой элементов, если их число не кратно фиксированной длине вектора, заданной для конкретного расширения [15]. В таких случаях в коде появляются дополнительные блоки, отвечающие за обработку оставшихся элементов, так называемого «хвоста» [16]. Например, возьмём стандартное выставление определённого значения во все элементы массива. В алгоритме 1 показан данный метод с учётом обработки завершающих элементов. Как видно, обработка хвоста использует скалярный код, что может негативно сказаться на производительности из-за засорения кеша инструкций.

```
Input:  a[N] – массив, во все элементы которого
нужно выставить значение V
Input:  L – количество скалярных элементов, кото-
рые  может обработать векторный регистр.
1:  if N = 0 then
2:    return
3:  end if
4:  i   0
5:  while i < N do
6:    Vreg   V
7:    start   i
8:    end   i + L
9:    store(a, start, end, Vreg)
10:   i   i + L
11:  end while
12:  i   i – L
13:  while i < N do
14:    store(a, i, i + 1, 0)
15:    i   i + 1
16:  end while
```

Алгоритм 1. Операция выставления константы в элементах массива с помощью фиксированной векторизации

Algorithm 1. Operation of setting a constant in array elements using fixed vectorization

Кроме того, размеры регистров зафиксированы на уровне спецификации. В результате, если производитель хочет улучшить векторное расширение, увеличив количество обрабатываемых элементов за раз, ему придётся выпускать новое расширение, как это было в случае с Intel MMX, SSE, SSE2, SSE3, AVX и другими [17].

Векторные вычисления с динамической длиной

Проблема расширяемости векторных систем команд привела к разработке масштабируемой векторизации, использующей вектора динамической длины. В таких системах команд вводятся специальные инструкции, позволяющие получать информацию о доступном количестве элементов для выполнения операции [10].

Это усложняет реализацию таких инструкций на уровне аппаратного обеспечения [11] и требует использования дополнительных команд для динамического определения количества элементов вектора, которые следует использовать в данный момент [18]. Однако, как результат, псевдокод (и результирующий код) для таких решений становится более компактным (см. алгоритм 2).

```
Input:  a[N] – массив,  во  все  элементы  которого
нужно  выставить значение V
1:   if N = 0 then
2:     return
3:   end if
4:   rest   N
5:   while rest - 0 do
6:     L   find_n(rest)
7:     Vreg   V
8:     start   N - rest
9:     end   start + L
10:    store(a, start, end, Vreg)
11:    rest   rest - L
12:  end while
```

Алгоритм 2. Операция выставления константы в элементах массива с помощью динамической векторизации

Algorithm 2. Operation of setting a constant in array elements using dynamic vectorization

Благодаря этому, последующие усовершенствования вычислительного устройства остаются незаметными для пользователей, которые продолжают работать с тем же интерфейсом.

Векторное расширение RISC-V и его особенности

Векторное расширение RISC-V обладает несколькими ключевыми особенностями. Одной из них является концепция группировки векторных регистров (LMUL). Эта функция позволяет гибко использовать несколько векторных регистров одновременно [11].

Благодаря этому пользователи могут за одну операцию обрабатывать сразу четверть (при LMUL = 8) векторного файла, что значительно ускоряет обработку данных. Даже при минимально доступной длине векторного регистра (VLEN), равной 64, пользователь может одновременно выполнить операцию над 64 элементами символьного типа *(char)*. Векторное расширение поддерживает использование 1, 2, 4 или 8 векторных регистров, что позволяет эффективно настраивать использование векторных регистров в зависимости от задач [12].

Кроме того, векторное расширение RISC-V поддерживает дробное использование векторных регистров. В этом случае LMUL называется дробным и позволяет использовать половину, четверть или восьмую часть векторного регистра, что особенно полезно для работы с малым количеством скалярных элементов.

Расширение также включает поддержку маскированных операций. В этом режиме часть элементов (не попадающих под маску) не обновляется. Важным аспектом является определение поведения этих не затронутых элементов. В расширении предусмотрена концепция агностик (обновляемых без учёта предыдущего состояния) либо нетронутых элементов в векторе. Если выбрано агностик-поведение, такие элементы либо заполняются единицами, либо сохраняют предыдущее значение, то есть остаются неопределёнными.

Эта особенность предоставляет гибкость при разработке вычислительного блока, что позволяет потенциально повысить его производительность. При использовании политики нетронутых элементов все не попадающие под маску элементы обязаны сохранять своё предыдущее состояние. Кроме того, термины «неизвестные» *(agnostic)* либо «нетронутые» *(undisturbed)* относятся к элементам, находящимся в хвосте вектора. Если пользователь при динамической настройке выбирает меньшее количество скалярных элементов, чем доступно в выбранной регистровой группе, все элементы, которые не могут быть использованы при установленной длине вектора, изменяются в соответствии с выбранным режимом.

Архитектурно-специфичные решения для написания векторного кода

Первым шагом при использовании новых расширений процессора обычно является работа с ассемблером или специальными встроенными функциями (интринсиками), которые непосредственно транслируются в ассемблерный код [9].

На этом этапе возникают определённые трудности при работе с векторными расширениями. Например, на листинге 1 приведены примеры векторной операции сложения нескольких элементов целочисленного типа для различных архитектур. Сверху использован ассемблерный синтаксис, снизу приведён интринсик.

```
# x86 syntax, SSE/AVX
paddd xmm0, xmm1
_m_paddd(lhs, rhs)
# risc-v, RVV
vadd.vv vd, vs2, vs1
__riscv_vadd(lhs, rhs)
# arm, NEON
add     v30.2d, v31.2d, v30.2d
vadd_s64(lhs, rhs)
# arm, SVE
add v4.d, v0.d, v1.d
svadd_x(svptrue_b32(), a, b)
```

Листинг 1. Операция сложения векторов на ассемблере и интринсиках

Listing 1. Vector addition operation in assembler and intrinsics

Отдельной проблемой, особенно при работе с ассемблером, является необходимость явного указания физических регистров в коде. Это ограничивает возможности оптимизации размещения регистров [13, 14], что может негативно сказаться на эффективности программы.
При работе с интринсиками, проблемой является то, что они часто не стандартизованы и их использование не позволяет изменить компилятор, а иногда даже конкретную версию компилятора.

Языковая поддержка в C++

В настоящее время ведётся разработка поддержки абстракций для векторных инструкций на уровне стандарта языка C++. Эта поддержка уже доступна для некоторых архитектур, таких как x86, однако её использование в продуктовых решениях пока не рекомендуется, поскольку стандарт ещё не окончательно утверждён, и возможны изменения в паттерне использования. Тем не менее, результирующий код для векторного сложения остаётся обобщённым и понятным (см. листинг 2).

```
const stdx::native_simd<int> a;
const stdx::native_simd<int> b;
stdx::native_simd<int> c = a + b;
```

Листинг 2. Операция сложения векторов в библиотеке std::simd
Listing 2. Vector addition operation in the std::simd library

Библиотека Boost.SIMD

Исторически одной из первых библиотек для написания обобщённого векторного кода является Boost.SIMD. Эта библиотека предоставляет абстракции для работы с векторными инструкциями [8]. В настоящее время используется версия библиотеки под названием NSIMD, которая сохраняет аналогичный пользовательский интерфейс. В результате код для векторного сложения внешне выглядит схоже (листинг 2.3). Последнее обновление библиотеки было в 2021 году.

```
pack<int> x = 1;
pack<int> y = 2;
pack<int> z = x + y;
```

Листинг 3. Операция сложения векторов на NSIMD
Listing 3. Vector addition operation on NSIMD

Векторное сложение в этом случае выполняется аналогичным способом (листинг 3). Тут следует заметить, что поддерживать данную библиотеку сложно, так как в основе её работы лежат сложные преобразования макросов в языке C++.

Библиотека EVE

Известной библиотекой для построения векторного кода является EVE. Ключевым отличием от Boost является активное использование шаблонного кода и добавленной в C++20 функциональности, а именно концептов. Это предоставляет компилятору больше возможностей для оптимизации и позволяет организовать код с помощью пространств имен.
Ключевыми абстракциями, на которых основана внутренняя работа с векторными регистрами в EVE, являются типы `Wide` и `LogicalWide`. Эти типы в большинстве случаев являются прозрачными обертками над архитектурно-специфичными (и иногда компиляторно-специфичными) представлениями векторных регистров или битовых масок для векторных регистров, а также базовыми операциями над ними. Тип `Wide` параметризуется типом элемента (базовым типом языка C++) и количеством элементов, необходимых пользователю. В библиотеке EVE задаются ограничения на количество элементов, которые может выбрать пользователь, для максимальной эффективности использования векторного расширения.
Библиотека EVE включает группу дополнительных функций для выполнения более специфичных действий, таких как операции по битовой маске. Это позволяет пользователю избегать архитектурно-специфичных участков кода, поэтому итоговый код для простых случаев остаётся понятным и схожим (см. листинг 4).

```
eve::wide<int> a{1};
eve::wide<int> b{2};
eve::wide<int> c = a + b;
```

Листинг 4. Операция сложения векторов на EVE
Listing 4. Vector addition operation on EVE

Кроме базовых операций, данная библиотека предоставляет обширную коллекцию обобщенных алгоритмов для работы с типами `std::vector` и `std::array`. Эти алгоритмы включают перестановку значений в памяти, копирование с предусловием, сжатие данных по предикату, и другие операции, которые автоматически векторизуются. В сочетании с активно обновляющимся репозиторием, было решено использовать эту библиотеку для прототипирования поддержки векторного расширения RISC-V.

Вычисление количества доступных элементов

Самым простым решением кажется повторение подхода для SVE, при котором предполагается, что EVE может использовать только один векторный регистр для представления `Wide`. Однако это решение также имеет свои недостатки. Оно ограничивает возможности использования полного потенциала векторного расширения RISC-V и закладывает в само решение существенное ограничение, которое в будущем будет трудно обойти, поскольку потребует полного перепроектирования.
Другим вариантом является использование всегда максимально возможной группировки (LMUL=8). Это подход имеет свои плюсы и минусы: хотя операции над конкретными векторными регистрами будут оптимальными в терминах количества элементов, для сложных вычислений могут возникать проблемы с количеством доступных векторных регистров. Поскольку векторных регистров всего 32, это автоматически означает, что одновременно можно использовать в лучшем случае только 4 векторных регистра с группировкой по 8. Все остальные экземпляры типа `Wide` компилятор будет обязан сохранять в память и восстанавливать по мере необходимости. Это представляет собой существенную проблему для эффективности библиотеки.

С учетом вышеописанных проблем было решено использовать гибридный подход, при котором количество векторных регистров группируется динамически в зависимости от требуемого пользователем количества элементов. Предлагаемый алгоритм 3 позволяет эффективно использовать векторный регистровый файл, учитывая начальное условие запроса константного количества элементов (равного степени двойки), а также ограничения на использование векторного регистра и реализации векторных интринсиков для RISC-V. Он возвращает отрицательное значение, если итоговый LMUL требуется сделать дробным.

```
Input: T – входной скалярный тип
Input: N – сколько элементов требуется хранить
Input: vec_size – битовый размер одного векторного регистра
Output: V – значение LMUL
1:   m1_len   vec_size
2:   min_len   m1_len * sizeof(T) / 8
3:   expected_len   sizeof(T) * N
4:   reg_len   std::max(min_len, expected_len)
5:   if reg_len >= m1_len then
6:        return reg_len / m1_len
7:   else
8:        return -m1_len / reg_len
9:   end if
```

Алгоритм 3. Реализация выбора группировки для векторного расширения
Algorithm 3. Implementation of group selection for vector expansion

Реализация получения и записи конкретного элемента

Одной из базовых операций для работы с векторными типами в библиотеке EVE является получение и изменение элемента в векторном регистре. К сожалению, в наборе инструкций для RISC-V отсутствует операция получения или вставки элемента в произвольную позицию в векторном регистре. Поэтому для обобщённой реализации требуется использовать промежуточный векторный регистр, полученный с помощью `riscv_vslidedown_tu`, который сдвигает векторный регистр так, чтобы интересуемый элемент оказался на нулевой позиции. После этого можно применить интринсики `riscv_vmv_x` или `riscv_vfmv_f` для получения значения элемента.

Заполнение элемента конкретным значением является немного более сложной задачей. Для этого предлагается использовать комбинацию концепции `eve::if_else`, результатом которой будет `Wide`, если соответствующее значение маски `true`. Для этой цели идеально подходит интринсик `__riscv_vmerge`. Таким образом, для вставки элемента в векторный регистр потребуется следующее:

- Создать новый векторный регистр, состоящий из одинаковых элементов, соответствующих тому, который надо вставить (`riscv_vfmv_v_f*` и `riscv_vmv_v_x*`)
- Создать логическую маску с единственным значением `true` в нужном месте
- Применить `eve::if_else`

Итоговые варианты представлены в алгоритмах 4 и 5.

```
Input:   a: wide<T, N> – Wide, куда нужно вставить элемент
Input:   V – значение, которое нужно вставить
Input:   L – позиция, на которую нужно вставить данный элемент
Output:   wide<T, N> – обновленный Wide
1:   mask   create_one_true_mask<T, N>(L)
2:   return if_else(mask, a, V)
```

Алгоритм 4. Реализация вставки элемента в `Wide`
Algorithm 4. Implementation of element insertion in `Wide`

```
Input:   a: wide<T, N> – Wide, откуда нужно получить элемент
Input:   L – позиция, откуда из Wide нужно получить элемент
Output: V – значение элемента
1:   on_first_pos   __riscv_vslidedown(a, L, N)
2:   return __riscv_vmv_x(on_first_pos)
```

Алгоритм 5. Реализация получения элемента из `Wide`
Algorithm 5. Implementation of getting an element from `Wide`

Реализация объединения и разделения векторных типов

Следующей базовой операцией для EVE является объединение векторных регистров в один — так называемая операция `combine`, а также разделение векторного типа на две части (операция `split`). Если для этих целей использовать ассемблер и физические регистры, то эти задачи в большинстве случаев (при группировке векторного регистра больше 2) тривиальны, так как можно точно определить, в каком физическом регистре находится нужная половина для «разделения» векторного регистра и куда нужно переместить векторные регистры для формирования регистровой группы. Соответственно, операции `split` и `combine` могут быть выполнены серией копирований векторов и изменением текущего векторного режима на увеличенный в два раза.

Однако для интерфейса векторных интринсиков эта возможность недоступна, так как нет доступа к физическим регистрам. Кроме того, даже с использованием физических регистров необходимо найти способ, который позволил бы эффективно соединять и разделять векторные типы, соответствующие части физического регистра.

Начнем с реализации функциональности для разделения векторного типа на две половины. Имплементация представлена в алгоритмах 6 и 7. Ключевой особенностью тут является вызовы `lower_lmul` и `get_calculate_lmul`, так как для возвращения правильного векторного типа требуется преобразовать внутренний тип, и вернуть тип с меньшей группировкой векторных регистров.

```
Input:   in: wide<T, N> – входной векторный тип
Output:   wide<T, N/2> – побитово равен “нижней”половине входного   векторного типа
1:   in_lmul   calculate_lmul_v < T,N >
2:   out_lmul   calculate_lmul_v < T,N/2 >
3:   if in_lmul  = out_lmul then
4:        return in
5:   else
```

```
6:        return lower_lmul(in)
7:    end if
```

Алгоритм 6. Реализация получения нижней половины векторного типа (split_lower)

Algorithm 6. Implementation of obtaining the lower half of a vector type (split_lower)

```
Input:   in: wide<T, N> – входной векторный тип
Output:   wide<T, N/2> – побитово равен "верхней"половине входного      векторного типа
1:    shift_size    N/2
2:    tmp    in ≫ shift_size
3:    return split_low(tmp)
```

Алгоритм 7. Реализация получения верхней половины векторного типа (split_upper)

Algorithm 7. Implementation of obtaining the upper half of a vector type (split_upper)

Перейдем к реализации операции «соединения» векторных типов (combine). Реализация представлена в алгоритме 8. Здесь ключевой является функция lmul_extend, задача которой — создать больший векторный тип, где нижняя половина равна входному Wide. Кроме того, необходима функция create_true_mask, которая создает правильный logical для обеспечения ожидаемого результата после применения if_else.

```
Input:   a,b - wide<T, N> – входные векторные типы
Output:   wide<T, 2N> – побитово  старшая  часть  равна a,  младшая часть равна b
1:    a_wider    lmul_extend(a)
2:    b_wider    lmul_extend(b)
3:    shift_size    N
4:    a_wider    a_wider ≪ shift_size
5:    mask    create_true_mask<T, 2N>(shift_size)
6:    return if_else(mask, wider_b, wider_a)
```

Алгоритм 8. Реализация объединения векторных регистров (combine)

Algorithm 8. Implementation of the vector register merger (combine)

Обсуждение и заключение

В данной работе проведён обзор существующих решений для построения обобщённого векторного кода и описаны ключевые особенности реализации поддержки векторного расширения RISC-V в активно развивающейся библиотеке EVE. Мы рассмотрели основные проблемы и подходы к интеграции нового векторного расширения, в том числе особенности работы с группировкой векторных регистров и маскированными операциями, а также их влияние на производительность и переносимость кода.

Обсуждены преимущества и недостатки существующих библиотек и подходов к реализации, таких как Boost.SIMD и NSIMD, с акцентом на их поддержку фиксированных и масштабируемых векторных регистров. Важно отметить, что выбор гибридного подхода для поддержки векторного расширения RISC-V в библиотеке EVE позволяет эффективно использовать возможности архитектуры и обеспечивает оптимальную производительность в широком спектре задач.

В качестве дальнейших шагов исследования рекомендуется изучить возможности оптимизации, специфичных для производителей конкретных процессоров RISC-V с векторным расширением. Также следует рассмотреть улучшения в генерации ассемблерного кода путём добавления RISC-V-специфичных оптимизаций в компилятор, что может значительно повысить производительность и эффективность использования векторных инструкций. В перспективе будет полезно провести дополнительные тесты и оценки производительности на реальных приложениях, чтобы выявить и устранить возможные узкие места и оптимизировать работу библиотеки в различных сценариях использования.

References

[1] Denning P.J., Lewis T.G. Exponential laws of computing growth. *Communications of the ACM.* 2016;60(1):54-65. https://doi.org/10.1145/2976758

[2] Gordon A. A quick overview of OpenMP for multi-core programming. *Journal of Computing Sciences in Colleges.* 2021;28(2):48.

[3] Burns R., Davidson C., Dodds A. Enabling OpenCL and SYCL for RISC-V processors. In: Proceedings of the 9th International Workshop on OpenCL (IWOCL '21). New York, NY, USA: Association for Computing Machinery; 2021. Article number: 15. https://doi.org/10.1145/3456669.3456687

[4] Hammond J., Dalcin L., Schnetter E., PéRache M., Besnard J.-B., Brown J., Gadeschi G.B., Byrne S., Schuchart J., Zhou H. MPI Application Binary Interface Standardization. In: Proceedings of the 30th European MPI Users' Group Meeting (EuroMPI '23). New York, NY, USA: Association for Computing Machinery; 2023. Article number: 1. https://doi.org/10.1145/3615318.3615319

[5] Ijaz M., Saleem F., Shahid U., Waheed S., Coulon J.-R. Implementation and Performance Evaluation of Bit Manipulation Extension on CVA6 RISC-V. In: Proceedings of the 20th ACM International Conference on Computing Frontiers (CF '23). New York, NY, USA: Association for Computing Machinery; 2023. p. 385-386. https://doi.org/10.1145/3587135.3591439

[6] Zhong D., Cao Q., Bosilca G., Dongarra J. Using Advanced Vector Extensions AVX-512 for MPI Reductions. In: Proceedings of the 27th European MPI Users' Group Meeting (EuroMPI/USA '20). New York, NY, USA: Association for Computing Machiner; 2020. p. 1-10. https://doi.org/10.1145/3416315.3416316

[7] Lin K.-K., et al. Rewriting and Optimizing Vector Length Agnostic Intrinsics from Arm SVE to RVV. In: Workshop Proceedings of the 53rd International Conference on Parallel Processing (ICPP Workshops '24). New York, NY, USA: Association for Computing Machinery; 2024. p. 38-47. https://doi.org/10.1145/3677333.3678151

[8] Estérie P., Falcou J., Gaunard M., Lapresté J.-T. Boost.SIMD: generic programming for portable SIMDization. In: Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing (WPMVP '14). New York, NY, USA: Association for Computing Machinery; 2014. p. 1-8. https://doi.org/10.1145/2568058.2568063

[9] Krzikalla O., Zitzlsberger G. Code vectorization using Intel Array Notation. In: Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing (WPMVP '16). New York, NY, USA: Association for Computing Machinery; 2016. Article number: 6. https://doi.org/10.1145/2870650.2870655

[10] Yzelman A.N. Generalised vectorisation for sparse matrix: vector multiplication. In: Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms (IA3 '15). New York, NY, USA: Association for Computing Machinery; 2015. Association for Computing Machinery, 6. https://doi.org/10.1145/2833179.2833185

[11] Islam M.A., Kise K. Resource-efficient RISC-V Vector Extension Architecture for FPGA-based Accelerators. In: Proceedings of the 13th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART '23). New York, NY, USA: Association for Computing Machinery; 2024. p. 78-85. https://doi.org/10.1145/3597031.3597047

[12] Gupta S.R., Papadopoulou N., Pericàs M. Challenges and Opportunities in the Co-design of Convolutions and RISC-V Vector Processors. In: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '23). New York, NY, USA: Association for Computing Machinery; 2023. p. 1550-1556. https://doi.org/10.1145/3624062.3624232

[13] Guan X., et al. PresCount: Effective Register Allocation for Bank Conflict Reduction. In: Proceedings of the 2024 IEEE/ACM International Symposium on Code Generation and Optimization (CGO '24). IEEE Press; 2024. p. 170-181. https://doi.org/10.1109/CGO57630.2024.10444841

[14] Krause P.K. The complexity of register allocation. *Discrete Applied Mathematics*. 2014;168:51-59. https://doi.org/10.1016/j.dam.2013.03.015

[15] Tian X., et al. Compiling C/C++ SIMD Extensions for Function and Loop Vectorizaion on Multicore-SIMD Processors. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum. Shanghai, China: IEEE Press; 2012. p. 2349-2358. https://doi.org/10.1109/IPDPSW.2012.292

[16] Brankovic S., Markovic A., Simic D., Rikalo A. Improving performance of sorting small arrays on MIPS CPUs using bitonic sort and SIMD instructions. In: 2019 27th Telecommunications Forum (TELFOR). Belgrade, Serbia: IEEE Press; 2019. p. 1-4. https://doi.org/10.1109/TELFOR48224.2019.8971325

[17] Keliris A., Maniatakos M. Investigating large integer arithmetic on Intel Xeon Phi SIMD extensions. In: 2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS). Santorini, Greece: IEEE Press; 2014. p. 1-6. https://doi.org/10.1109/DTIS.2014.6850661

[18] Edamatsu T., Takahashi D. Efficient Large Integer Multiplication with Arm SVE Instructions. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia '23). New York, NY, USA: Association for Computing Machinery; 2023. p. 9-17. https://doi.org/10.1145/3578178.3578193

[19] Wang J., Hu Y. Enabling Efficient SIMD Acceleration for Virtual Radio Access Network. In: Proceedings of the 50th International Conference on Parallel Processing (ICPP '21). New York, NY, USA: Association for Computing Machinery; 2021. Article number: 63. https://doi.org/10.1145/3472456.3472477

[20] Shih M.-S., et al. Register-Pressure Aware Predicator for Length Multiplier of RVV. In: Workshop Proceedings of the 51st International Conference on Parallel Processing (ICPP Workshops '22). New York, NY, USA: Association for Computing Machinery; 2023. Article number: 10. https://doi.org/10.1145/3547276.3548513

[21] Lai H.-M., Lee J.-K. Efficient Support of the Scan Vector Model for RISC-V Vector Extension. In: Workshop Proceedings of the 51st International Conference on Parallel Processing (ICPP Workshops '22). New York, NY, USA: Association for Computing Machinery; 2023. Article number: 15. https://doi.org/10.1145/3547276.3548518

[22] Hao X., et al. POPA: Expressing High and Portable Performance across Spatial and Vector Architectures for Tensor Computations. In: Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '24). New York, NY, USA: Association for Computing Machinery; 2024. p. 199-210. https://doi.org/10.1145/3626202.3637566

[23] Wang H., et al. Simple, portable and fast SIMD intrinsic programming: generic SIMD library. In: Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing (WPMVP '14). New York, NY, USA: 2014. p. 9-16. https://doi.org/10.1145/2568058.2568059

[24] Falcou J., Sérot J. EVE, an Object Oriented SIMD Library. In: Bubak M., van Albada G.D., Sloot P.M.A., Dongarra J. (Eds.) Computational Science – ICCS 2004. ICCS 2004. *Lecture Notes in Computer Science*. Vol. 3038. Berlin, Heidelberg: Springer; 2004. p. 314-321. https://doi.org/10.1007/978-3-540-24688-6_43

[25] Bramas B. Inastemp: A Novel Intrinsics-as-Template Library for Portable SIMD-Vectorization. *Scientific Programming*. 2017;2017(1):5482468. https://doi.org/10.1155/2017/5482468

Поступила 23.07.2024; одобрена после рецензирования 14.09.2024; принята к публикации 25.09.2024.
Submitted 23.07.2024; approved after reviewing 14.09.2024; accepted for publication 25.09.2024.

Об авторах:

Владимиров Константин Игоревич, старший преподаватель кафедры микропроцессорных технологий в интеллектуальных системах факультета радиотехники и кибернетики, ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)» (141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9), **ORCID: https://orcid.org/0000-0003-0925-1300,** konstantin.vladimirov@gmail.com

Тетюшкин Иван Андреевич, аспирант кафедры микропроцессорных технологий в интеллектуальных системах факультета радиотехники и кибернетики, ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)» (141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9), **ORCID: https://orcid.org/0009-0000-7584-8653**, tetyushkin.ia@phystech.edu

Все авторы прочитали и одобрили окончательный вариант рукописи.

About the authors:

Konstantin I. Vladimirov, Senior Lecturer of the Chair of Microprocessor Technologies in Intelligent Systems, Department of Radio Engineering and Cybernetics, Moscow Institute of Physics and Technology (National Research University) (9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation), **ORCID: https://orcid.org/0000-0003-0925-1300,** konstantin.vladimirov@gmail.com

Ivan A. Tetyushkin, Postgraduate Student of the Chair of Microprocessor Technologies in Intelligent Systems, Department of Radio Engineering and Cybernetics, Moscow Institute of Physics and Technology (National Research University) (9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation), **ORCID: https://orcid.org/0009-0000-7584-8653**, tetyushkin.ia@phystech.edu

All authors have read and approved the final manuscript.