УДК 004.4.23 Научная статья

https://doi.org/10.32603/2071-8985-2023-16-4-46-53

Анализ наборов векторизирующих опций компилятора GCC для алгоритмов сжатия видеостандарта H.264, реализованных на архитектуре MIPS SIMD

Д. В. Богаевский✉, С. Н. Ежов, Д. И. Каплун

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина), Санкт-Петербург, Россия

✉ dan4ezz94@gmail.com

Аннотация. В рамках статьи было проведено исследование влияния оптимизирующих, в том числе и векторизирующих, опций компилятора на производительность системы визуализации для перспективных систем на кристалле с векторным процессором.

На основе анализа оптимизирующих опций компилятора GCC (GNU Compiler Collection) для архитектуры MIPS SIMD (Microprocessor without Interlocked Pipeline Stages Single Instruction Multiple Data) были определены тестовые наборы параметров и опций компилятора, с использованием которых было проведено профилирование четырех тестовых алгоритмов, используемых в стандарте сжатия видео H.264.

Ключевые слова: архитектура MIPS SIMD, аппаратный эмулятор QEMU, оптимизация компилятора GCC, алгоритм сжатия H.264

Для цитирования: Богаевский Д. В., Ежов С. Н., Каплун Д. И. Анализ наборов векторизирующих опций компилятора GCC для алгоритмов сжатия видеостандарта H.264, реализованных на архитектуре MIPS SIMD // Изв. СПбГЭТУ «ЛЭТИ». 2023. Т. 16, № 4. С. 46–53. doi: 10.32603/2071-8985-2023-16-4-46-53.

Конфликт интересов. Авторы заявляют об отсутствии конфликта интересов.

Original article

Analysis of GCC Compiler Vectorizing Option Sets for H.264 Video Compression Algorithms Implemented on the MIPS SIMD Architecture

D. V. Bogaevskiy✉, S. N. Ezhov, D. I. Kaplun

Saint Petersburg Electrotechnical University, Saint Petersburg, Russia

✉ dan4ezz94@gmail.com

Abstract. As part of the article, a study was made of the influence of optimizing, including vectorizing, compiler options on the performance of the visualization system for promising systems on a chip with a vector processor.

Based on the analysis of the optimizing options of the GCC compiler for the MIPS SIMD architecture, test sets of parameters and compiler options were determined, using which the profiling of 4 test algorithms used in the H.264 video compression standard was carried out.

Keywords: MIPS SIMD architecture, hardware emulator QEMU, GCC compiler optimization, compressing algorithm H.264

For citation: Bogaevskiy D. V., Ezhov S. N., Kaplun D. I. Analysis of GCC Compiler Vectorizing Option Sets for H.264 Video Compression Algorithms Implemented on the MIPS SIMD Architecture // LETI Transactions on Electrical Engineering & Computer Science. 2023. Vol. 16, no. 4. P. 46–53. doi: 10.32603/2071-8985-2023-16-4-46-53.

Conflict of interest. The authors declare no conflicts of interest.

Введение. На сегодняшний день задачи обработки сигналов, изображений и видео занимают огромную часть в мире телекоммуникаций. Абсолютное большинство алгоритмов, используемых в этой области, хорошо распараллеливаются. В связи с этим к наиболее популярным способам

обработки изображений и видео относится применение GPGPU- (General Purpose Graphic Processor Unit) технологий (вычисления на графических процессорах) и векторных процессоров. Наиболее высокую производительность показывают GPGPU-технологии [1], однако они имеют ряд ограничений и не всегда подходят для решения некоторых задач обработки изображений [2]. Векторные вычисления (SIMD (Single Instruction Multiple Data)) тоже дают значительный прирост производительности [3] и менее требовательны, чем технологии GPGPU. Наиболее популярны векторные архитектуры Intel MMX/SSE [4] и ARM NEON [5].

В 2017 г. компания «Wave Computing», владеющая патентами и лицензиями MIPS Technologies, заявила о намерении сделать архитектуру MIPS открытой (Open Source) с целью привлечения разработчиков [6]. Данная архитектура имеет расширение для решения векторных задач [7], которое называют MIPS SIMD или MSA (MIPS SIMD Architecture). Ввиду популяризации архитектуры MIPS и огромного количества исследований для более популярных архитектур (Intel SSE) в рамках данной статьи будем оценивать и повышать производительность на архитектуре MSA.

Для получения наибольшей производительности помимо выбора подходящей архитектуры нужно обратить внимание и на саму реализацию программ. Существует несколько способов для повышения производительности программ, которые привязаны к конкретной архитектуре. Из [8] можно выделить следующие способы повышения производительности на примере векторной архитектуры Intel SSE: улучшение алгоритма для конкретной задачи, модификация кода (замена используемых функций на более быстрые) и выбор компилятора и оптимальных ключей для него. Главный недостаток способа модификации алгоритма заключается в его неуниверсальности (для каждого алгоритма нужно реализовывать все заново) и сложности реализации (сложно придумать что-то новое, когда до тебя множество людей пытались сделать тоже самое). К недостаткам способа модификации кода относятся также неуниверсальность и высокое требование к квалификации разработчика.

Поскольку данная статья – общее исследование, в ее рамках будут исследованы наборы ключей для компилятора, влияющих на затраты ресурсов (памяти и т. д.), и производительность (время выполнения программы). В [9] на примере SSE (Streaming SIMD Extensions) показано, как с помощью препроцессорных директив и ключей компилятора повысить производительность программы в несколько раз. Для оценки производительности в качестве метрики в статье использовалось время выполнения программ. Однако не всегда можно корректно оценить время выполнения программы на какой-либо архитектуре при наличии некоторых ограничений. Например, если бюджет разработчиков ограничен (нет средств на покупку железа), то они используют эмуляторы (VMware, Oracle VM VirtualBox, QEMU и др.). На эмуляторах невозможно корректно оценить время работы той или иной программы, поэтому возможен выбор других метрик для оценки производительности.

В данной статье мы предлагаем метрики, основанные на работе с памятью, для оценки эффективности векторизации при использовании различных ключей оптимизации компилятора. В нашей предыдущей публикации [10] в эмулятор QEMU [11] был встроен счетчик команд, позволяющий получить последовательность выполнения инструкций процессора, и реализованы некоторые алгоритмы из стандарта сжатия видео h264 на архитектуре MSA, что способствовало наличию исходного кода и инструмента профилирования для текущих исследований.

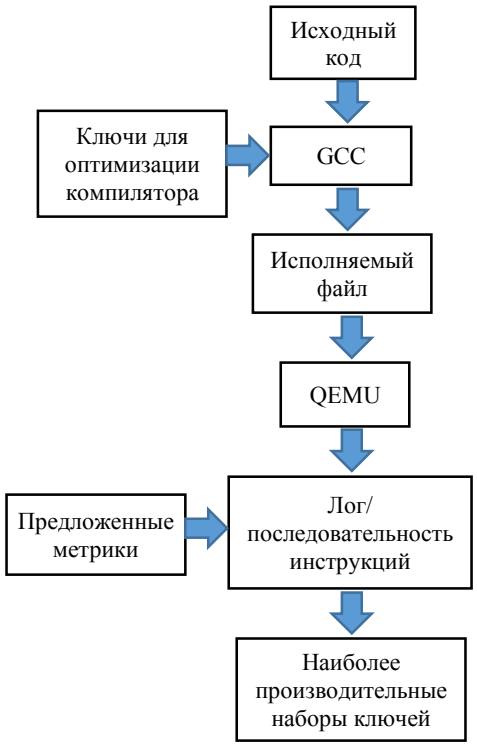

Рис. 1. Общая схема исследования
Fig. 1. General scheme of the study

В качестве тестовых алгоритмов были выбраны четыре алгоритма, используемых в стандарте сжатия видео h264 [12]. В качестве компилятора был выбран GCC [13], так как он – рекомендуемый (поставляется в MIPS SDK) [14] и наиболее популярный компилятор среди аналогов для архитектуры MSA.

На рис. 1 представлена общая схема исследования.

Выбор оптимизирующих преобразований компилятора GCC. Компилятор GCC (GNU Compiler Collection) поддерживает большой набор высокоуровневых оптимизирующих преобразований (ключей) [13]. Для уменьшения сложности исследования на основании документации к компилятору было уменьшено количество рассматриваемых ключей.

Рассмотрим описания опций компилятора GCC для архитектуры MIPS, которые использовались для генерации последовательностей ключей:

• Опции –EB (Big Endian) и –EL (Little Endian) генерируют код в порядке от старшего к младшему и от младшего к старшему соответственно. Так как после компиляции файла используется эмулятор qemu-mips64el, то необходимо использовать опцию – EL.

• Опция –march=arch генерирует код, который будет выполняться на архитектуре arch, которая может быть описана общими MIPS ISA или названием конкретного процессора. При компиляции использует процессор mips64r6.

• Опция –mtune=arch оптимизирует код под архитектуру arch. Данная опция контролирует алгоритм планировщика и «стоимость» арифметических операций. Целесообразно в качестве arch использовать процессор mips64r6.

• Опция –mshared установлена по умолчанию. Она позволяет генерировать полностью независимый от позиции код, который может быть связан с общими библиотеками. Данная опция влияет только на текущий ABI.

• Опция –mno-shared генерирует объекты, которые могут быть связаны только компоновщиком GNU. Опция влияет лишь на ABI перемещаемых объектов. Использование данной опции позволяет сделать исполняемый файл меньше и быстрее.

• Опция –mgp64 предполагает, что регистры общего назначения имеют ширину 64 бит.

• Опция – mfp64 предполагает, что регистры процессора с плавающей точкой имеют ширину 64 бита.

• Опция –mhard-float использует инструкции сопроцессора с плавающей точкой.

• Опция –mdouble-float предполагает, что сопроцессор с плавающей точкой поддерживает операции двойной точности.

• Опция –mabs=2008 управляет обработкой NaN в IEEE 754-2008 с помощью машинных инструкций abs.fmt и neg.fmt.

• Опция –mnan=2008 управляет кодированием NaN в IEEE754 с плавающей точкой. Выбирает кодировку IEEE754-2008.

• Опции –mllsc и –mno-llsc позволяют использовать или не использовать соответственно инструкции «ll», «sc», «sync» для реализации встроенных функций атомарной памяти.

• Опции –membedded-data и –mno-embedded-data, если это возможно, выделяют или запрещают выделять память соответственно под переменные в read-only разделе, далее по мере возможности выделяют или запрещают выделять соответственно память в секции small data, в противном случае в области данных. Опция –membedded-data уменьшает объем оперативной памяти, требуемой при выполнении программы.

• Опции –muninit-const-in-rodata и –mno-uninit-const-in-rodata помещает и запрещает помещать соответственно неинициализированные константы в разделе read-only. Опция –muninit-const-in-rodata работает лишь в сочетании с опцией –membedded-data.

• Опция –mmsa использует MIPS SIMD Architecture (MSA).

В силу своей неактуальности для поставленной задачи (противоречат используемой архитектуре, противоречат используемому ABI, исключены из Release 6 и пр.) оставшиеся опции MIPS, описанные в документации по компилятору GNU GCC [13], были исключены.

В результате работы алгоритма генерации было получено 372 уникальных сочетания ключей, к которым были добавлены 4 стандартных набора оптимизации $–O_0$, $–O_1$, $–O_2$, $–O_3$. Таким образом, суммарно было определено 1488 сочетаний.

Определение ключевых параметров выполнения программ. Проведение анализа производительности программ требует определения ряда ключевых параметров, наиболее полно описывающих процесс выполнения.

На производительность серьезно влияет эффективность взаимодействия с памятью, которая характеризуется двумя типами локальности – пространственной и временной [15].

Поскольку проводились исследования алгоритмов обработки изображений на векторной архитектуре, то подавляющее большинство операций – векторные. Соответственно, были предложены метрики для векторных операций, на основании которых можно было определить эффективность взаимодействия с памятью при выполнении программы.

Пространственная локальность описывает вероятность того, что в скором времени потребуется обратиться к ячейкам памяти, расположенным рядом с недавно вызванными ячейками. Пространственная локальность вычисляется по формуле:

$$L_{\text{Space}_i} = N_{C_i} / N_{LS},$$

где L_{Space_i} – пространственная локализация данных для i-го блока памяти; N_{C_i} – суммарное колво обращений по i-му блоку (128 бит); N_{LS} – общее число векторных операций чтения и записи.

Например, вектор (блок памяти) $\mathbf{V}_1$ имеет адрес 0x1000. В ходе анализа логирования было выявлено, что по адресу 0x1000 обращались с помощью векторных операций чтения и записи 100 раз. Общее количество операций чтения и записи составило 5000 раз. В этом случае пространственная локализация для блока по адресу 0x1000 высчитывается следующим образом:

$$L_{V_1} = 100/5000 = 0.02.$$

Временная локальность описывает временные диапазоны, после которых потребуются данные, хранящиеся по одному и тому же адресу. Временная локальность вычисляется как количество уникальных адресов, которые запрашиваются между двумя соседними запросами к адресу X, включая сам адрес X.

Например, в логе программы имеется следующая последовательность команд:

```
LOAD 0x1000
LOAD 0x3330
ADD
LOAD 0x4440
ADD
STORE 0x5550
LOAD 0x1000
```

Команды LOAD (чтение данных из памяти) запрашивают данные с некоторых адресов и помещают их в регистры, команда ADD производит сложение двух регистров, команда STORE записывает данные из регистра в память. В данном случае между двумя запросами данных по адресу 0x1000 находится 2 запроса к данным по другим адресам, поэтому временная локальность для адреса 0x1000 равна 3. Не исключена возможность, что один адрес может обладать несколькими значениями, поэтому в этом случае можно взять среднюю.

Применение компилятора, поддерживающего автовекторизацию данных, позволяет использовать объем векторизированных вычислений на единицу данных в качестве одного из ключевых параметров, описывающих выполнение программы.

Объем векторизированных вычислений на единицу данных описывает средний объем векторизованных вычислений, который приходится на единицу данных в приложении, и определяется по формуле:

$$V = N_V/N_{LS},$$

где V – среднее число векторных операций, приходящихся на один доступ к данным; N_V – общее число векторных операций; N_{LS} – общее число векторных операций чтения и записи.

Например, в программе выполнялось 120 000 векторных операций, из них 10 000 векторных. Среднее число векторных операций будет 12. Это число показывает, что с одним вектором в программе было проведено в среднем 12 операций.

Целевая архитектура не поддерживает DGMS-систему, способную прогнозировать степень детализации доступа к памяти [16]; таким образом, степень выравнивания данных также влияет на производительность исследуемых программ [17].

Выравнивание данных описывает количество обращений по адресам, которые не были выравнены по размеру используемого вектора (128 бит). Степень выравнивания данных оценивается по формуле:

$$A = N_{\text{Al}}/N_{\text{Unal}},$$

где A – оценка степени выравнивания данных; N_{Al} – количество обращений к выравненным адресам; N_{Unal} – количество обращений к невыравненным адресам.

В случае с архитектурой MIPS SIMD, вектор имеет размер 128 бит, поэтому будем считать выровненные адресами те, которые кратны 0x80, а не выравненными – все остальные. Например, в программе количество обращений к выравненным адресам равно 10 000, а к не выравненным – 1000. Соответственно, степень выравнивания будет 10.

Табл. 1. МПС для рассматриваемых ключевых параметров
Tab. 1. Pairwise comparison matrix for the considered key parameters

	Пространственная локализация	Временная локализация	Объем векторизированных вычислений	Выравнивание данных
Пространственная локализация	1	3	7	9
Временная локализация	1/3	1	3	3
Объем векторизированных вычислений	1/7	1/3	1/3	3
Выравнивание данных	1/9	1/3	1/3	1

Анализ полученных наборов ключей. Анализируемые данные были получены через скрипт, который перебирает всевозможные наборы выбранных в исследовании ключей (всего 1488 наборов), компилирует алгоритмы с этими ключами, запускает программы в эмуляторе QEMU и читает файлы с информацией о выполнении данных алгоритмов.

Для оценки полученных наборов по четырем метрикам (пространственная локализация, временная локализация, объем векторизованных вычислений и выравнивание данных) использовался метод взвешенной суммы [18]. При использовании данного подхода критерий полезности альтернативы определяется как сумма произведений весовых коэффициентов метрик и оценки этих метрик.

Метод взвешенной суммы использует весовые коэффициенты для каждой метрики. Для их вычисления использовался метод анализа иерархий, разработанный в 1970-х гг. американским ученым Саати [19]. В соответствии с методом была составлена матрица парных сравнений (далее – МПС) для вышеописанных ключевых параметров. Данная МПС представлена в табл. 1.

В результате использования метода взвешенной суммы для МПС ключевых параметров (табл. 1) были получены нормированные весовые коэффициенты, представленные в табл. 2.

Табл. 2. Нормированные весовые коэффициенты
Tab. 2. Normalized weight coefficients

Метрика	Выравнивание данных
Пространственная локализация	0.614743
Временная локализация	0.222088
Объем векторизированных вычислений	0.106319
Выравнивание данных	0.056850

Индекс согласованности ИС МПС составил 0.042253.

Случайная согласованность СС = 0.9.

Отношение согласованности ОС составило ОС = ИС/СС = 0.046948.

Так как ОС < 0.1, уточнения экспертных оценок не требуется.

Целевые функции для каждого альтернативного варианта сочетаний рассчитывались по формуле

$$F_i = \sum \omega_j \times M_{ij},$$

где F_i – целевая функция i-го набора ключей; ω_j – весовая функция для j-й метрики; M_{ij} – нормированное значение j-й компонентой метрики i-го набора ключей.

В результате вычислений было выявлено 372 набора ключей, представленные в приложении А, целевые функции которых максимальны для каждого из алгоритмов и совпадают по каждому из алгоритмов соответственно.

На основании анализа полученных значений целевых функций, в качестве рекомендованного набора была выбрана следующая последовательность оптимизирующих опций компилятора:

–EL, –mabi=64, –march=mips64r6, –mmsa, –O0, –static, –mno-shared, –mgp64, –mfp64, –mhard-float

Данная последовательность обладает максимальным средним значением целевой функции по 4 рассмотренным алгоритмам.

Выводы. На сегодняшний день предлагается множество способов увеличить производительность приложений, использующих векторные архитектуры, с помощью компиляторов: использование машинного обучения [20] и генетических алгоритмов [21] для выбора ключей, разработка собственных алгоритмов автовекторизации [22] и пр.

В статье были предложены метрики для оценки повышения производительности программ обработки изображений и видео, реализованных на архитектуре MSA, также с помощью этих метрик были проанализированы и представлены наборы ключей для компилятора GCC. В результате профилирования, которое было разработано в рамках наших предыдущих публикаций [10], [23], были получены численные пред-

ставления каждого из параметров для алгоритмов, используемых в стандарте сжатия видео h264, и получены, соответственно, 5952 результата. Для каждого из ключевых параметров были определены весовые коэффициенты в соответствии с методикой анализа иерархий Саати, после чего при помощи весовых функций определены 372 рекомендуемых набора ключей, целевые функции которых максимальны для каждого из 4 рассмотренных алгоритмов.

Основным недостатком предложенных метрик состоит в невозможности оценить временной прирост производительности. Однако главное достоинство представляет возможность глубокого изучения поведения программы при различных изменениях данных, их объема, модификаций алгоритма и прочего.

В дальнейших исследованиях планируется расширить набор метрик и применить их к другим областям. Планируется разработать метрики для оценки эффективности работы кэша процессора, однако для этого не подойдет эмулятор QEMU, так как он не поддерживает возможность эмуляции кэша. В качестве примера рассматриваемых нами областей, где целесообразно применение векторных вычислений, можно привести библиотеки для 3D-графики. В графических библиотеках в вычислениях, происходящих на CPU, наиболее часто используются матричные вычисления, которые хорошо распараллеливаются.

Список литературы

1. Zhang N., Chen Y., Wang J. Image parallel processing based on GPU // 2010 2nd Intern. Conf. on Advanced Comp. Control. Shenyang, China. 2010. Vol. 3. P. 367–370. doi: 10.1109/ICACC.2010.5486836.

2. Saahithyan V., Suthakar S. Performance analysis of basic image processing algorithms on GPU // 2017 Intern. Conf. on Inventive Systems and Control (ICISC). Coimbatore, India. 2017. P. 1–6. doi: 10.1109/ICISC.2017.8068687.

3. A study of the use of SIMD instructions for two image processing algorithms / E. Welch, D. Patru, E. Saber, K. Bengtson // 2012 Western New York Image Proc. Workshop, Rochester, NY, USA. 2012. P. 21–24. doi: 10.1109/WNYIPW.2012.6466650.

4. Conte G., Tommesani S., Zanichelli F. The long and winding road to high-performance image processing with MMX/SSE // Proc. Fifth IEEE Intern. Workshop on Comp. Architectures for Machine Perception. Padua, Italy. 2000. P. 302–310. doi: 10.1109/CAMP.2000.875989.

5. Prasannakumar N. A scheme for accelerating image processing algorithms using SIMD for ARM Cortex A based systems // 2017 Intern. Conf. on Innovations in Information, Embedded and Communication Systems (ICIIECS). Coimbatore, India. 2017. P. 1–5. doi: 10.1109/ICIIECS. 2017.8275887.

6. Wave Computing Launches the MIPS Open Initiative. URL: https://wavecomp.ai/wave-computing-launches-the-mips-open-initiative/ (дата обращения 25.04.2022).

7. MIPS SIMD. URL: https://www.mips.com/products/architectures/ase/simd/ (дата обращения 25.04.2022).

8. Yang C., Yang X., Xue J. Improving the performance of GCC by exploiting IA-64 architectural features // Proc. of the 10th Asia-Pacific conf. on Advances in Comp. Systems Architecture (ACSAC'05). Springer-Verlag, Berlin, Germany. 2005. P. 236–251. doi: 10.1007/11572961_20.

9. Image Processing Acceleration Techniques using Intel ® Streaming SIMD Extensions and Intel ® Advanced Extensions. URL: https://software.intel.com/content/dam/develop/external/us/en/documents/image-processing-whitepaper-100pct-ccereviewed-update-164443.pdf (дата обращения 25.04. 2022).

10. Study of vector processor architectures for image processing using model profiling / D. V. Bogayevskiy, S. N. Ezhov, D. I. Kaplun, M. V. Minenko, S. I. Aryashev, K. A. Petrov // 2019 8th Mediterranean Conf. on Embedded Comp. (MECO). Budva, Montenegro. 2019. P. 1–4. doi: 10.1109/MECO.2019.8760039.

11. Bellard F. QEMU, a fast and portable dynamic translator // Proc. of the Annual Conf. on USENIX Annual Technical Conf. ATEC '05. Berkley, USA: USENIX Association. 2005. P. 41–46.

12. Overview of the H.264/AVC video coding standard / T. Wiegand, G. J. Sullivan, G. Bjontegaard, A. Luthra // IEEE Transactions on Circuits and Systems for Video Technology. 2003. Vol. 13, no. 7. P. 560–576.

13. GCC, the GNU Compiler Collection. URL: https://gcc.gnu.org/ (дата обращения 25.04.2022).

14. MIPS Developer Tools. URL: https://www.mips.com/develop/tools/ (дата обращения 25.04.2022).

15. Harris S. L., Harris D. M. 8 – Memory Systems / In book: Digital Design and Computer Architecture. Morgan Kaufmann. 2016. P. 486–529. doi: 10.1016/B978-0-12-800056-4.00008-X.

16. Brodsky A., Wang X. S. Decision-Guidance Management Systems (DGMS): Seamless Integration of Data Acquisition, Learning, Prediction and Optimization // Proc. of the 41st Annual Hawaii Intern. Conf. on System Sciences (HICSS 2008). Waikoloa, HI, USA. 2008. P. 71–71. doi: 10.1109/HICSS.2008.114.

17. The dynamic granularity memory system / D. H. Yoon, M. K. Jeong, M. Sullivan, M. Erez // Proc. of the 39th Annual Intern. Symp. on Comp. Architecture, ISCA`12, Portland, Oregon. 2012. P. 548–559.

18. Marler R. T., Arora J. S. The weighted sum method for multi-objective optimization: new insights // Struct Multidisc Optim 41. 2010. P. 853–862. doi: 10.1007/s00158-009-0460-7.

19. Saaty T. L. Decision making for leaders: The Analytic Hierarchy Process for decision in a complex World. University of Pittsburgh. RWS Publications, Pittsburgh (USA). 1990. 292 p.

20. Liu H., Zhao R., Nie K. Using ensemble learning to improve automatic vectorization of tensor contraction program // IEEE Access. 2018. Vol. 6. P. 47112–47124. doi: 10.1109/ACCESS.2018.2867151.

21. San-Chih Lin, Chi-Kuang Chang, Nai-Wei Lin. Automatic selection of GCC optimization options using a gene weighted genetic algorithm // 2008 13th Asia-Pacific Computer Systems Architecture Conf. Hsinchu, Taiwan. 2008. P. 1–8. doi: 10.1109/APCSAC.2008.4625477.

22. Nuzman D., Henderson R. Multi-platform auto-vectorization // Intern. Symp. on Code Generation and Optimization (CGO'06). New York, NY, USA. 2006. P. 281–294. doi: 10.1109/CGO.2006.25.

23. Development and Implementation of the H.264-Codec Deblocking Filter Based on the MIPS SIMD Architecture / D. Bogaevskiy, M. Minenko, S. Ezhov, D. I. Kaplun // 2021 IEEE Conf. of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). St. Petersburg, Moscow, Russia. 2021. P. 246–251. doi: 10.1109/ElConRus51938.2021.9396406.

Информация об авторах

Богаевский Данил Васильевич – канд. техн. наук, мл. науч. сотр. НОЦ ЦТТ СПбГЭТУ «ЛЭТИ».
E-mail: dan4ezz94@gmail.com

Ежов Сергей Николаевич – канд. техн. наук, доцент, зам. зав. по научной работе, кафедра систем автоматизированного проектирования СПбГЭТУ «ЛЭТИ».
E-mail: sn_ezhov@mail.ru

Каплун Дмитрий Ильич – д-р техн. наук, доцент, зам. зав. по научной работе, кафедра автоматики и процессов управления СПбГЭТУ «ЛЭТИ».
E-mail: dikaplun@etu.ru
https://orcid.org/0000-0003-2765-4509

References

1. Zhang N., Chen Y., Wang J. Image parallel processing based on GPU // 2010 2nd Intern. Conf. on Advanced Computer Control. Shenjang, China. 2010. Vol. 3. P. 367–370. doi: 10.1109/ICACC.2010.5486836.

2. Saahithyan V., Suthakar S. Performance analysis of basic image processing algorithms on GPU // 2017 Intern. Conf. on Inventive Systems and Control (ICISC). Coimbatore, India. 2017. P. 1–6. doi: 10.1109/ICISC.2017.8068687.

3. A study of the use of SIMD instructions for two image processing algorithms / E. Welch, D. Patru, E. Saber, K. Bengtson // 2012 Western New York Image Processing Workshop, Rochester, NY, USA. 2012. P. 21–24. doi: 10.1109/WNYIPW.2012.6466650.

4. Conte G., Tommesani S., Zanichelli F. The long and winding road to high-performance image processing with MMX/SSE // Proc. Fifth IEEE Intern. Workshop on Computer Architectures for Machine Perception. Padua, Italy. 2000. P. 302–310. doi: 10.1109/CAMP.2000.875989.

5. Prasannakumar N. A scheme for accelerating image processing algorithms using SIMD for ARM Cortex A based systems // 2017 Intern. Conf. on Innovations in Information, Embedded and Communication Systems (ICIIECS). Coimbatore, India. 2017. P. 1–5. doi: 10.1109/ICIIECS. 2017.8275887.

6. Wave Computing Launches the MIPS Open Initiative. URL: https://wavecomp.ai/wave-computing-launches-the-mips-open-initiative/ (data obrashhenija 25.04.2022).

7. MIPS SIMD. URL: https://www.mips.com/products/architectures/ase/simd/ (data obrashhenija 25.04.2022).

8. Yang C., Yang X., Xue J. Improving the performance of GCC by exploiting IA-64 architectural features // Proc. of the 10th Asia-Pacific conf. on Advances in Computer Systems Architecture (ACSAC'05). Springer-Verlag, Berlin, Germany. 2005. P. 236–251. doi: 10.1007/11572961_20.

9. Image Processing Acceleration Techniques using Intel ® Streaming SIMD Extensions and Intel ® Advanced Extensions. URL: https://software.intel.com/content/dam/develop/external/us/en/documents/image-processing-whitepaper-100pct-ccereviewed-update-164443.pdf (data obrashhenija 25.04. 2022).

10. Study of Vector Processor Architectures for Image Processing Using Model Profiling / D. V. Bogayevskiy, S. N. Ezhov, D. I. Kaplun, M. V. Minenko, S. I. Aryashev, K. A. Petrov // 2019 8th Mediterranean Conf. on Embedded Comp. (MECO). Budva, Montenegro. 2019. P. 1–4. doi: 10.1109/MECO.2019.8760039.

11. Bellard F. QEMU, a fast and portable dynamic translator // Proc. of the annual conf. on USENIX Annual Technical Conf. ATEC'05. Berkley, USA: USENIX Association. 2005. P. 41–46.

12. Overview of the H.264/AVC video coding standard / T. Wiegand, G. J. Sullivan, G. Bjontegaard, A. Luthra // IEEE Transactions on Circuits and Systems for Video Technology. 2003. Vol. 13, no. 7. P. 560–576.

13. GCC, the GNU Compiler Collection. URL: https://gcc.gnu.org/ (data obrashhenija 25.04.2022).

14. MIPS Developer Tools. URL: https://www.mips.com/develop/tools/ (data obrashhenija 25.04.2022).

15. Harris S. L., Harris D. M. 8 – Memory Systems // In: Digital Design and Computer Architecture. Morgan Kaufmann. 2016. P. 486–529. doi: 10.1016/B978-0-12-800056-4.00008-X.

16. Brodsky A., Wang X. S. Decision-Guidance Management Systems (DGMS): Seamless integration of data acquisition, learning, prediction and optimization // Proc. of the 41st Annual Hawaii Intern. Conf. on System Sci. (HICSS 2008). Waikoloa, HI, USA. 2008. P. 71–71. doi: 10.1109/HICSS.2008.114.

17. The dynamic granularity memory system / D. H. Yoon, M. K. Jeong, M. Sullivan, M. Erez // Proc. of the 39th Ann. Intern. Symp. on Comp. Architecture, ISCA`12, Portland, Oregon. 2012. P. 548–559.

18. Marler R. T., Arora J. S. The weighted sum method for multi-objective optimization: new insights // Struct Multidisc Optim. 41. 2010. P. 853–862. doi: 10.1007/s00158-009-0460-7.

19. Saaty T. L. Decision making for leaders: The Analytic Hierarchy Process for decision in a complex World. University of Pittsburgh. RWS Publications, Pittsburgh (USA). 1990. 292 p.

20. Liu H., Zhao R., Nie K. Using ensemble learning to improve automatic vectorization of tensor contraction program // IEEE Access. 2018. Vol. 6. P. 47112–47124. doi: 10.1109/ACCESS.2018.2867151.

21. San-Chih Lin, Chi-Kuang Chang, Nai-Wei Lin. Automatic selection of GCC optimization options using a gene weighted genetic algorithm // 2008 13th Asia-Pacific Computer Systems Architecture Conf. Hsinchu, Taiwan. 2008. P. 1–8. doi: 10.1109/APCSAC.2008.4625477.

22. Nuzman D., Henderson R. Multi-platform auto-vectorization // Intern. Symp. on Code Generation and Optimization (CGO'06). New York, NY, USA. 2006. P. 281–294. doi: 10.1109/CGO.2006.25.

23. Development and implementation of the H.264-Codec Deblocking Filter based on the MIPS SIMD Architecture / D. Bogaevskiy, M. Minenko, S. Ezhov, D. I. Kaplun // 2021 IEEE Conf. of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). St. Petersburg, Moscow, Russia. 2021. P. 246–251. doi: 10.1109/ElConRus51938.2021.9396406.

Information about the authors

Danil V. Bogaevskiy – Cand. Sci. (Eng.), junior researcher of the Scientific and Educational Center «Digital Telecommunication Technologies», Saint Petersburg Electrotechnical University.
E-mail: dan4ezz94@gmail.com

Sergei N. Ezhov – Cand. Sci. (Eng.), Associate Professor, Deputy Head on scientific work of the Department of Automated Design Systems, Saint Petersburg Electrotechnical University.
E-mail: sn_ezhov@mail.ru

Dmitry I. Kaplun – Dr Sci. (Eng.), Associate Professor, Deputy Head on scientific work of the Department of Automation and Control Processes, Saint Petersburg Electrotechnical University.
E-mail: dikaplun@etu.ru
https://orcid.org/0000-0003-2765-4509

Статья поступила в редакцию 17.12.2022; принята к публикации после рецензирования 13.01.2023; опубликована онлайн 25.04.2023.

Submitted 17.12.2022; accepted 13.01.2023; published online 25.04.2023.