Интеллектуальные системы и технологии

Быстрая реализация расстояния Хэминга на VLIW-архитектурах на примере платформы Эльбрус*

Е.Е. Лимонова[I], Н.Л. Рженев[II], А.В. Усков[I], М.И. Нейман-заде[III]

[I] Институт системного анализа Федерального исследовательского центра «Информатика и управление» Российской академии наук, г. Москва, Россия
[II] ООО «Смарт Энджинс Сервис», г. Москва, Россия
[III] АО «МЦСТ», г. Москва, Россия

Аннотация. В работе показана быстрая реализация расстояния Хэмминга на платформе Эльбрус, обладающей VLIW-архитектурой. Приведены и экспериментально исследованы ключевые особенности архитектуры Эльбрус, позволяющие значительно повысить быстродействие вычисления расстояния Хэмминга, такие как наличие нескольких арифметико-логических устройств, работающих параллельно, особенности доступа в память, наличие буфера подкачки массивов и другие возможности параллельного исполнения команд. Реализация с учетом этих особенностей оказалась в 11,5 раз быстрее базовой реализации для достаточно больших длин массивов. Более того, исследованные техники повышения быстродействия для процессоров Эльбрус не являются специфичными для рассмотренного алгоритма и могут применяться для ряда других алгоритмов обработки данных.

Ключевые слова: *расстояние Хэмминга, VLIW-архитектура, процессор Эльбрус.*
DOI: 10.14357/20790279180507

Введение

На сегодняшний день технологии компьютерного зрения активно развиваются и находят широкое практическое применение. В связи с этим большое значение приобретает оценка и, при необходимости, повышение быстродействия как отдельных алгоритмов, так и систем распознавания в целом.

Расстоянием Хэмминга называется метрика, вычисляемая как число несовпадающих символов на соответствующих позициях в двух словах одинаковой длины:

$$d_H(u,s) = \sum_{k=1}^{L} I_{u_k \neq s_k} \ .$$

где u, s – слова одинаковой длины L, u_k, s_k – k-ые символы в словах u, s.

Расстояние Хэмминга часто применяется в теории кодирования при передаче данных по каналу с шумом [1,2] и при обработке изображений или распознавании образов. В последнем случае чаще всего речь идет о сравнении дескрипторов, отвечающих некоторым признакам изображений [3–5]. Также оно может использоваться при измерении расстояний между клеточными автоматами [6] и во многих других областях: в физике [7], лингвистике [8], анализе данных [9,10], генетике [11], биологии [12].

Однако задачам, связанным с вычислением расстояния Хэмминга, часто сопутствует большое количество операций. К примеру, чтобы сравнить два генома бактерий, необходимо произвести от

* Работа выполнена при частичной финансовой поддержке грантов РФФИ 17-29-03297 и 17-29-03263.

10^8 до 10^{10} вычислений расстояния Хэмминга между парой элементов [13]. В алгоритмах, предназначенных для работы в режиме реального времени (например, [14-16]), любой низкоуровневый алгоритм может составлять заметную долю вычислений, в связи с чем остро встает вопрос о его вычислительной оптимизации. С одной стороны, можно уменьшить число непосредственных вызовов вычисления расстояния Хэмминга, однако такой подход предполагает изменение алгоритма и может повлечь за собой изменение его работы, что не всегда приемлемо. С другой стороны, можно повышать быстродействие алгоритма с учетом особенностей архитектуры вычислительного устройства, как например в [17, 18]. В этой работе рассматривается именно последний подход.

Практически все современные процессоры поддерживают SIMD (Single Instruction Multiple Data) расширения. Эти расширения позволяют выполнять одну и ту же операцию над несколькими элементами данных, помещающимися в регистр, реализуя таким образом параллелизм на уровне данных. Один из эффективных методов точного подсчета расстояния Хэмминга между двумя массивами таков: сравнение символов на соответствующих местах, представление результата сравнения в виде двоичной маски и последующее применение команды для подсчета единиц к элементам маски (см. пример на рис. 1). При этом операции сравнения символов и подсчета единиц могут быть реализованы в виде SIMD-команд [19]. Есть отдельные реализации для конкретных SIMD-расширений, например, с использованием AVX2, которое позволяет оперировать 256-битными регистрами. Можно добиться двукратного роста производительности по сравнению с оптимизированной для 64-битных регистров версией [20]. Всего лишь двукратный рост производительности связан с отсутствием команд, выполняющих необходимые действия непосредственно.

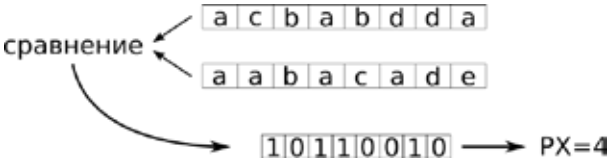

Рис. 1. Пример работы алгоритма вычисления расстояния Хэмминга (PX) между двумя векторами длины 8

Также стоит упомянуть о существовании оптимизированных методов вычисления расстояния Хэмминга на графических процессорах и программируемых логических интегральных схемах. Так, например, с использованием программируемых логических интегральных схем можно добиться ускорения вычислений в 4 раза по сравнению с графическим процессором и почти в 50 раз по сравнению с центральным процессором [13]. Однако следует отметить, что область применения логических интегральных схем весьма специфична – это разработка программно-аппаратных платформ. На практике же разработчики часто ограничены в выборе оборудования и должны обеспечить эффективную работу на центральных или графических процессорах.

1. VLIW-архитектура Эльбрус

Архитектура процессоров Эльбрус использует принцип широкого командного слова (Very Long Instruction Word, VLIW). Это означает, что процессор исполняет команды группами, причем внутри каждой группы отсутствуют зависимости и эти команды исполняются параллельно. Каждая такая группа называется широким командным словом. Формирование широких командных слов выполняет оптимизирующий компилятор, благодаря чему возможен более подробный анализ исходного кода, приводящий к более эффективному распараллеливанию [21].

Особенностью архитектуры Эльбрус являются методы работы с памятью. Помимо наличия кэша, позволяющего оптимизировать время доступа в память, процессоры Эльбрус поддерживают программно-аппаратный метод предварительной подкачки данных. Этот метод позволяет прогнозировать обращения в память и производить подкачку данных в буфер предварительной подкачки данных. Аппаратная часть процессора включает в себя специальное устройство для обращения к массивам (Array Access Unit, AAU), но необходимость подкачки определяется компилятором, который генерирует специальные инструкции для AAU. Использование устройства подкачки является более эффективным, чем помещение элементов массива в кэш, поскольку они чаще всего обрабатываются последовательно и редко используются более одного раза [22]. Однако необходимо отметить, что использование буфера предварительной подкачки на Эльбрусе возможно только при работе с выровненными данными. Благодаря этому чтение/запись выровненных данных происходят заметно быстрее, чем соответствующие операции для невыровненных данных.

Кроме того, процессоры Эльбрус поддерживают несколько видов параллелизма помимо параллелизма на уровне команд: SIMD-параллелизм, параллелизм потоков управления, параллелизм за-

дач в многомашинном комплексе. Особый интерес для нас представляет SIMD-параллелизм.

2. Особенности использования SIMD-расширения на платформе Эльбрус

Использование SIMD-расширений может осуществляться в двух режимах: автоматическом и непосредственном. В первом случае распараллеливание операций полностью выполняется компилятором без участия разработчика. Этот режим ограничен, так как оптимизированный код должен полностью повторять поведение исходного кода, в том числе и поведения при переполнении, округлении и т.д. При этом поведение команд SIMD-расширения может отличаться в этих аспектах от команд процессора. Кроме того, алгоритмы, применяющиеся в компиляторах, несовершенны и не всегда способны выполнить эффективное распараллеливание [23].

Однако разработчики могут обращаться также к командам SIMD-расширений непосредственно, используя интринсики. Интринсики – функции, вызовы которых заменяются компилятором на высокоэффективный код для данной платформы, в частности, на команды SIMD-расширения. Процессоры Эльбрус-4С и Эльбрус-8С поддерживают набор интринсиков, для которого размер регистра составляет 64 бита. Он включает в себя операции для преобразования данных, инициализации элементов вектора, арифметических операций, побитовых логических операций, перестановки элементов вектора и др.

При использовании интринсиков на платформе Эльбрус следует уделять особое внимание доступу в память, поскольку в практических задачах, например, задачах обработки изображений, часто требуется невыровненное чтение данных в 64-битный регистр. Такое чтение само по себе неэффективно, так как требует пары команд чтения и последующей команды формирования блока данных, но, что еще важнее, при этом не может использоваться буфер подкачки массивов, повышающий скорость доступа к данным. Однако стоит отметить, что проблема неэффективного доступа к невыровненным данным актуальна для процессоров Эльбрус-4С и Эльбрус-8С, в то время как для более нового Эльбрус-8СВ с 5 версией системы команд она частично решена. Ожидается, что в процессорах Эльбрус с 6 версией системы команд она будет решена полностью. Поэтому низкоуровневую обработку данных следует выполнять с учетом выравнивания. Например, для числовых массивов, она может состоять из нескольких этапов: обработка начальной части (до границы выравнивания на 64-бита одного из массивов), обработка основной части, использующая выровненный доступ к памяти, и обработка оставшихся элементов массива. Поскольку анализ указателей во время компиляции является нетривиальной задачей, можно использовать флаг компилятора –faligned, с которым все операции доступа к памяти выполняются выровненным образом.

Следующая особенность использования интринсиков на платформе Эльбрус связана непосредственно с его VLIW-архитектурой. Благодаря наличию нескольких арифметико-логических устройств (АЛУ), которые работают параллельно и загружаются при формировании широких командных слов, несколько команд могут исполняться одновременно. Всего в процессорах Эльбрус-4С и Эльбрус-8С шесть АЛУ, которые можно задействовать в рамках одной широкой команды, однако каждое АЛУ поддерживает свой набор интринсиков. Простые операции, например, сложение или умножение элементов в 64-битных регистрах, как правило поддерживают два АЛУ. Это означает, что процессор Эльбрус может исполнить по две таких инструкции за один такт. Для этого в исполняемом коде следует использовать развертывание циклов. Оптимизирующий компилятор для платформы Эльбрус поддерживает прагму #pragma unroll(n), которая позволяет выполнить развертывание n итераций цикла.

Таким образом, для эффективного использования интринсиков на платформе Эльбрус необходимо обеспечить выровненный доступ в память и обеспечить использование всех доступных АЛУ путем развертывания циклов.

3. Быстрая реализация расстояния Хэмминга на основе SIMD-интринсиков

Для построения быстрой реализации вычисления расстояния Хэмминга на процессорах Эльбрус были последовательно рассмотрены особенности, описанные выше. В экспериментах была рассмотрена реализация сравнения двух битовых векторов данных. Двоичные значения были упакованы в массивы 8-битных целых чисел. Для простоты будем считать, что длины исходных векторов были кратны 8. В качестве базовой реализации был рассмотрен цикл в котором между каждой парой 8-битных элементов вычислялось побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ, а затем вычислялось число единиц результата с помощью таблицы предподсчитанных значений. Далее были рассмотрены следующие реализации:

1. Используется ИСКЛЮЧАЮЩЕЕ ИЛИ между 8-битными целыми числами и таблица предподсчитанных значений.
2. Используется ИСКЛЮЧАЮЩЕЕ ИЛИ между 32-битными целыми числами и интринсик для вычисления числа единиц в 32-битном целом числе – popcnt32. Выравнивания данных по границе в 32-бита не выполнялось.
3. Используется ИСКЛЮЧАЮЩЕЕ ИЛИ между 64-битными целыми числами и интринсик для вычисления числа единиц в 64-битном целом числе – popcnt64. Выравнивания данных по границе в 64-бита не выполнялось.
4. Используется ИСКЛЮЧАЮЩЕЕ ИЛИ между 64-битными целыми числами и интринсик для вычисления числа единиц в 64-битном целом числе – popcnt64. Доступ в память осуществляется выровненным образом. Поскольку начальные адреса массивов могут иметь разное выравнивание, при чтении одного из массивов выполняется чтение двух соседних 64-битных блоков и формирование из них необходимого 64-битного блока.
5. Используется ИСКЛЮЧАЮЩЕЕ ИЛИ между 64-битными целыми числами и интринсик для вычисления числа единиц в 64-битном целом числе – popcnt64. Доступ в память осуществляется выровненным образом. Поскольку начальные адреса массивов могут иметь разное выравнивание, при чтении одного из массивов выполняется чтение двух соседних 64-битных блоков и формирование из них необходимого 64-битного блока. Дополнительно использован флаг компилятора -faligned.
6. Используется ИСКЛЮЧАЮЩЕЕ ИЛИ между 64-битными целыми числами и интринсик для вычисления числа единиц в 64-битном целом числе – popcnt64. Доступ в память осуществляется выровненным образом. Поскольку начальные адреса массивов могут иметь разное выравнивание, при чтении одного из массивов выполняется чтение двух соседних 64-битных блоков и формирование из них необходимого 64-битного блока. Дополнительно использован флаг компилятора -faligned и прагмы компилятора #pragma unroll(2) (чтобы использовать для вычисления popcnt64 оба доступных АЛУ) и #pragma loop count(1000) (чтобы включить дополнительные цикловые оптимизации).

Для перечисленных вариантов реализации использовались язык программирования C++, компиляция выполнялась с помощью lcc 1.21.24 – оптимизирующего компилятора для платформы Эльбрус. Результаты замеров времени приведены в табл. 1.

Табл. 1

Сравнение различных реализаций расстояния Хэмминга между битовыми массивами 8-битных целых чисел длины 10^5 на Эльбрус-401PC. Времена были усреднены по 10^5 итерациям

Метод вычисления	**Время, мс**
Таблица предподсчитанных значений	141,18
popcnt32, без выравнивания	125,54
popcnt64, без выравнивания	58,00
popcnt64, выравнивание	17,36
popcnt64, выравнивание, -faligned	17,15
popcnt64, выравнивание, -faligned, развертывание цикла	12,23

Можно видеть, что все рассмотренные оптимизации привели к снижению времени работы в 11,5 раз. При этом стоит отметить, что использование интринсиков при невыровненном доступе в память показало ускорение лишь в 1,13 раза (для popcnt32) и в 2,4 раза (для popcnt64), в то время как учет выравнивания данных привел к использованию буфера подкачки массивов APB, с помощью которого удалось достичь ускорения еще в 3,4 раза (58 мс против 17,15 мс). Несмотря на то, что в рассмотренном примере использование флага -faligned не показало значительного прироста производительности, в более сложных алгоритмах возможна ситуация, когда компилятор не сможет проанализировать исходный код достаточно глубоко, чтобы сгенерировать команды для APB. Учет актуального количества специализированных АЛУ позволил ускорить вычисления еще в 1,4 раза.

Приведем псевдокод финального варианта реализации.

***Вход*:** *массивы целых 8-битных чисел A и B, массив T, в котором i-й элемент содержит число единиц в числе i.*

***Выход*:** *целое число res, равное расстоянию Хэмминга между A и B.*

```
offset ← число байтов до границы выравнивания массива A на 64-бита
effective_len ← максимальная длина массива, которую можно обработать блоками по 64-бита
для i от 1 до offset:
res ← res + T[xor(A_i, B_i)] // xor – операция вычисления ИСКЛЮЧАЮЩЕГО ИЛИ
v_a ← 64-битный блок данных, начиная с A_offset+1
v_b1 ← 64-битный блок данных, включающий элемент B_offset
```

```
v_b2 ← 64-битный блок данных, следующий за v_b1
// включение развертывания цикла и цикловых оптимизаций для всех 64-битных блоков данных от offset до effective_len:
v_b ← align(v_b1, v_b2) // создание 64-битного блока данных, соответствующего v_a
res ← res + popcnt64(xor(v_a, v_b))
v_a ← следующий 64-битный блок данных
v_b1 ← v_b2
v_b2 ← следующий 64-битный блок данных
// Обработать оставшиеся элементы, используя массив T
```

Однако, подобная реализация будет иметь преимущество лишь при достаточно больших длинах массивов. На рис. 2 представлено сравнение времени подсчета с использованием таблицы предподсчитанных значений и предложенной реализации с помощью интринсиков. Можно видеть, что версия с интринсиками начинает выигрывать только начиная с длин, превышающих 400 байт.

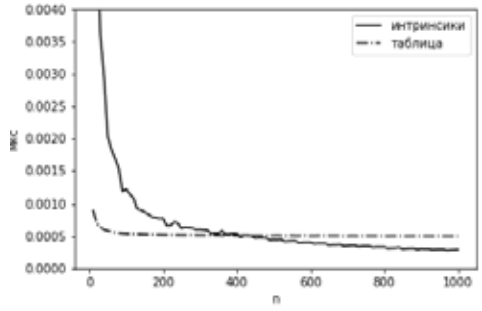

Рис. 2. Среднее время работы (на один элемент) расстояния Хэмминга между битовыми массивами 8-битных целых чисел длины n на Эльбрус-401РС

Заключение

В работе показана быстрая реализация расстояния Хэмминга на платформе Эльбрус, обладающей VLIW-архитектурой. Предложенная реализация работает в 11,5 раз быстрее, чем реализация с помощью таблицы предподсчитанных значений при достаточно большой длине входных векторов. Для достижения такого результата были использованы SIMD-расширения, а также учтены особенности архитектуры и доступа в память процессоров Эльбрус-4С и Эльбрус-8С. Был подробно рассмотрен процесс анализа исходного кода и поэтапный процесс его низкоуровневой оптимизации. Полученные результаты показывают, что процессоры Эльбрус содержат значительный ресурс для эффективной работы, с помощью которого можно добиться сокращения времени исполнения на порядок. Кроме того, ожидается, что на более новых процессорах Эльбрус будут усовершенствованы методы доступа в память, что позволит выполнять часть рассмотренных оптимизаций в автоматическом режиме.

Литература

1. *Ray J., Koopman P.* Efficient high Hamming distance CRCs for embedded networks // Dependable Systems and Networks, 2006. DSN 2006. International Conference on. – IEEE, 2006. – С. 3-12.
2. *Hamming R.W.* Error detecting and error correcting codes // Bell Labs Technical Journal. – 1950. – Т. 29. – №. 2. – С. 147-160.
3. *Rigas I., Economou G., Fotopoulos S.* Efficient modeling of visual saliency based on local sparse representation and the use of hamming distance // Computer Vision and Image Understanding. – 2015. – Т. 134. – С. 33-45.
4. *Revathy S., Ramasethu M.L.* Face and IRIS recognition in a video sequence using DBPNN and adaptive Hamming Distance. – 2016.
5. *Fakhfakh S., Tmar M., Mahdi W.* Image Retrieval Based on Using Hamming Distance // Procedia Computer Science. – 2015. – Т. 73. – С. 320-327.
6. *Li W., Packard N.* The structure of the elementary cellular automata rule space //Complex systems. – 1990. – Т. 4. – №. 3. – С. 281-297.
7. *Souza J.W.G. et al.* A new proposal for analyzing combustion process stability based on the Hamming distance // Physica A: Statistical Mechanics and its Applications. – 2014. – Т. 413. – С. 301-306.
8. *Ghani R.* Using error-correcting codes for text classification // ICML. – 2000. – С. 303-310.
9. *Ruan Y. et al.* Quantum Algorithm for K-Nearest Neighbors Classification Based on the Metric of Hamming Distance // International Journal of Theoretical Physics. – 2017. – Т. 56. – №. 11. – С. 3496-3507.
10. *Norouzi M., Fleet D.J., Salakhutdinov R.R.* Hamming distance metric learning // Advances in neural information processing systems. – 2012. – С. 1061-1069.
11. *Xin H. et al.* Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping // Bioinformatics. – 2015. – Т. 31. – №. 10. – С. 1553-1560.
12. S*ood S., Loguinov D.* Probabilistic near-duplicate detection using simhash // Proceedings of the 20th

ACM international conference on Information and knowledge management. – ACM, 2011. – С. 1117-1126.
13. *Kaiser M. et al.* Accelerating Hamming Distance Comparisons for Locality Sensitive Hashing (LSH) using FPGAs // 12th CeBiTec Symposium-Big Data in Medicine and Biotechnology-Abstract Book. – 2018. – Т. 12.
14. *Kuznetsova E. et al.* Viola-Jones based hybrid framework for real-time object detection in multispectral images // ICMV 2015, SPIE, vol. 9875, publ. code 98750N, pp. 1-6, 2015, DOI: 10.1117/12.2228707.
15. *Skoryukina N. et al.* Real Time Rectangular Document Detection on Mobile Devices // ICMV 2014, SPIE, 2015, vol. 9445, publ. code 94452A, pp. 1-6, 2015, DOI: 10.1117/12.2181377.
16. *Skoryukina N. et al.* Snapscreen: TV-stream frame search with projectively distorted and noisy query // Proc. SPIE 10341, ICMV 2016, 10341 ed., Antanas Verikas, Ed., Ninth International Conference on Machine Vision, 2017, vol. 10341, ISBN 978-15-10611-32-0, publ. code 103410Y, pp. 1-5, 2017, DOI: 10.1117/12.2268735.
17. *Бочаров Н.А., Лимонова Е.Е., Парамонов Н.Б. и др.* Оптимизация для вычислительной архитектуры Эльбрус модифицированного метода Виола и Джонса // Труды ИСА РАН. 2017. Т. 67. № 4. С. 10-21.
18. *Лимонова Е.Е., Бочаров Н.А., Парамонов Н.Б., Богданов Д.С., Арлазаров В.В., Славин О.А., Николаев Д.П.* Оценка быстродействия системы распознавания на VLIW архитектуре на примере платформы Эльбрус // Программирование (в печати).
19. *Hirvola T. et al.* Bit-parallel approximate string matching under Hamming distance. – 2016.
20. *Muła W., Kurz N., Lemire D.* Faster population counts using AVX2 instructions // The Computer Journal. 2017. Т. 61. №. 1. С. 111–120.
21. *Ким А.К., Бычков И.Н. и др.* Российские технологии «Эльбрус» для персональных компьютеров, серверов и суперкомпьютеров // Современные информационные технологии и ИТ-образование, М.: Фонд содействия развитию интернет-медиа, ИТ-образования, человеческого потенциала «Лига интернет-медиа», 2014, № 10, с. 39-50.
22. *Ким А.К., Перекатов В.И., Ермаков С.Г.* Микропроцессоры и вычислительные комплексы семейства «Эльбрус». – СПб.: Питер, 2013. – 272 С.
23. *Porpodas V., Jones T.M.* Throttling automatic vectorization: When less is more // Parallel Architecture and Compilation (PACT), 2015 International Conference on. – IEEE, 2015. – С. 432-444.

Лимонова Елена Евгеньевна. Институт системного анализа Федерального исследовательского центра «Информатика и управление» Российской академии наук, г. Москва, Россия. Математик. Количество печатных работ: 16. Область научных интересов: обработка изображений, распознавание образов на мобильных устройствах. E-mail: limonova@smartengines.biz

Рженев Никита Леонидович. Московский физико-технический институт (государственный университет), г. Москва, Россия. Студент. Область научных интересов: алгоритмы обработки изображений, машинное обучение. E-mail: nikita.rzhenev@smartengines.ru

Усков Анатолий Васильевич. Институт системного анализа Федерального исследовательского центра «Информатика и управление» Российской академии наук, г. Москва, Россия. Зав. лабораторией. Кандидат физико-математических наук. Количество печатных работ: более 50 (в т. ч. 3 монографии). Область научных интересов: искусственный интеллект и системное программирование. E-mail: uskov@isa.ru

Нейман-заде Мурад Искендер-оглы. АО МЦСТ. Начальник отделения Систем Программирования. Кандидат физико-математических наук. Количество печатных работ: более 20. Область интересов: оптимизирующая компиляция, анализ производительности кода. E-mail: muradnz@mcst.ru

Fast implementation of Hamming distance on VLIW-architectures on the example of Elbrus platform

E.E. Limonova[I], N.L. Rzhenev[II], A.V. Uskov[I], M.I. Neiman-zade[III]

[I] Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia

[II] LLC "Smart Engines Service", Moscow, Russia

[III] AO "MCST", Moscow, Russia

Abstract. In this work we show fast implementation of Hamming Distance on Elbrus platform with VLIW-architecture. We introduce and examine experimentally key features of Elbrus architecture such as availability of several arithmetic and logic units working in parallel, special features of memory access, presence of array prefetch buffer and other opportunities for parallel execution. These features allow us to significantly rise computational performance of Hamming distance computation. The implementation considering the characteristics appeared to be 11.5 times faster than the basic implementation for large enough arrays. Besides, considered techniques of improving computational performance for Elbrus processors are not specific for this algorithm and can be used for a number of other data processing methods.

Keywords: *Hamming distance, VLIW-architecture, Elbrus processor.*

DOI: 10.14357/20790279180507

References

1. *Ray, J., & Koopman, P.* (2006, June). Efficient high Hamming distance CRCs for embedded networks. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on* (pp. 3-12). IEEE.
2. *Hamming, R.W.* (1950). Error detecting and error correcting codes. *Bell Labs Technical Journal*, *29*(2), 147-160.
3. *Rigas, I., Economou, G. and Fotopoulos, S.* (2015). Efficient modeling of visual saliency based on local sparse representation and the use of hamming distance. *Computer Vision and Image Understanding*, *134*, 33-45.
4. *Revathy, S. and Ramasethu, M.L.* (2016). Face and IRIS recognition in a video sequence using DBPNN and adaptive Hamming Distance.
5. *Fakhfakh, S., Tmar, M. and Mahdi, W.* (2015). Image Retrieval Based on Using Hamming Distance. *Procedia Computer Science*, *73*, 320-327.
6. *Li, W., & Packard, N.* (1990). The structure of the elementary cellular automata rule space. *Complex systems*, *4*(3), 281-297.
7. S*ouza, J.W.G., Pereira, H.B.B., Santos, A.A.B., Senna, V., & Moret, M.A.* (2014). A new proposal for analyzing combustion process stability based on the Hamming distance. *Physica A: Statistical Mechanics and its Applications*, *413*, 301-306.
8. *Ghani, R.* (2000, June). Using error-correcting codes for text classification. In *ICML* (pp. 303-310).
9. *Ruan, Y., Xue, X., Liu, H., Tan, J., & Li, X.* (2017). Quantum Algorithm for K-Nearest Neighbors Classification Based on the Metric of Hamming Distance. *International Journal of Theoretical Physics*, *56*(11), 3496-3507.
10. *Norouzi, M., Fleet, D.J., & Salakhutdinov, R.R.* (2012). Hamming distance metric learning. In *Advances in neural information processing systems* (pp. 1061-1069).
11. *Xin, H., Greth, J., Emmons, J., Pekhimenko, G., Kingsford, C., Alkan, C., & Mutlu, O.* (2015). Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping. *Bioinformatics*, *31*(10), 1553-1560.
12. *Sood, S., & Loguinov, D.* (2011, October). Probabilistic near-duplicate detection using simhash. In *Proceedings of the 20th ACM international conference on Information and knowledge management* (pp. 1117-1126). ACM.
13. *Kaiser, M., Pilz, S., Porrmann, F., Hagemeyer, J., & Porrmann, M.* (2018). Accelerating Hamming Distance Comparisons for Locality Sensitive Hashing (LSH) using FPGAs. In *12th CeBiTec Symposium-Big Data in Medicine and Biotechnology-Abstract Book*(Vol. 12).
14. *E. Kuznetsova et al.* Viola-Jones based hybrid framework for real-time object detection in multispectral images. In *Proceedings of ICMV 2015*, SPIE, vol. 9875, publ. code 98750N, pp. 1-6, 2015, DOI: 10.1117/12.2228707.
15. *N. Skoryukina et al.* Real Time Rectangular Document Detection on Mobile Devices. In *Proceedings ICMV 2014,* SPIE, 2015, vol. 9445, publ. code 94452A, pp. 1-6, 2015, DOI: 10.1117/12.2181377.
16. *N. Skoryukina et al.* Snapscreen: TV-stream frame search with projectively distorted and noisy query. Proc. SPIE 10341, ICMV 2016, 10341 ed., Antanas Verikas, Ed., Ninth International

Conference on Machine Vision, 2017, vol. 10341, ISBN 978-15-10611-32-0, publ. code 103410Y, pp. 1-5, 2017, DOI: 10.1117/12.2268735.
17. *N.A. Bocharov et al.* Optimization for the Elbrus computing architecture of the modified method of Viola and Jones. Trudy ISA RAN, vol. 67, no 4, pp. 10-21, 2017.
18. *Limonova E.E., Bocharov N.A., Paramonov N.B., Bogdanov D.S., Arlazarov V.V., Slavin O.A., Nikolaev D.P.* Recognition system efficiency evaluation on VLIW architecture on the example of Elbrus platform // *Programming and Computer Software (in print)*
19. *Hirvola, T.* (2016). Bit-parallel approximate string matching under Hamming distance.
20. *Muła, W., Kurz, N., & Lemire, D.* (2017). Faster population counts using AVX2 instructions. *The Computer Journal*, *61*(1), 111-120.
21. *Kim A.K., Bychkov I.N.* Russian Technologies "Elbrus" for personal computers, servers and supercomputers // Modern Information Technologies and IT Education, Moscow: Foundation for the Promotion of Internet Media, IT Education, Human Capacity "League of Internet Media", 2014, No. 10. 39-50.
22. *Kim A.K., Perekatov V.I., Ermakov S.G.* Microprocessors and computing systems of the Elbrus family. – SPb: Peter, 2013. – 272 C.
23. *Porpodas, V., & Jones, T.M.* (2015, October). Throttling automatic vectorization: When less is more. In Parallel Architecture and Compilation (PACT), 2015 International Conference on (pp. 432-444). IEEE.

E.E. Limonova. Institute for Systems Analysis, Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia, mathematician, LLC "Smart Engines Service", programmer. Number of publications: 16. Research interests: image processing, pattern recognition on mobile devices.
E-mail: limonova@smartengines.biz

N.L. Rzhenev. Moscow Institute of Physics and Technology (State University), Institutsky per., 9, Moscow, region, Dolgoprudny, 141701, Russia. Student. Research interests: image processing algorithms, machine learning. E-mail: nikita.rzhenev@smartengines.ru

A.V. Uskov. Institute for Systems Analysis, Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia, head of laboratory. Ph.D. in Physics and Mathematics. Number of publications: more than 50 (including 3 monographs). Research interests: artificial intelligence and system programming. E-mail: uskov@isa.ru

M.I. Neiman-zade. Department of AO "MCST", head of Programming Systems. Ph.D. in Physics and Mathematics. Number of publications: more than 20. Research interests: optimizing compilation, performance analysis. E-mail: muradnz@mcst.ru