Реализация функции пространственной фильтрации изображения на векторном сопроцессоре

С. И. Аряшев[1], П. А. Чибисов[2], В. В. Цветков[3], Д. А. Трубицын[4], К. А. Петров[5]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, aserg@cs.niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, chibisov@cs.niisi.ras.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, tsvetkov@cs.niisi.ras.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, trubitsyn@cs.niisi.ras.ru;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, petrovk@cs.niisi.ras.ru

Аннотация. Для увеличения быстродействия универсальных микропроцессоров MIPS-подобной архитектуры в НИИСИ РАН был разработан специализированный сопроцессор, позволяющий ускорять операции с комплексными и вещественными числами одинарной и двойной точности. В статье представлены результаты применения 128-го разрядного векторного сопроцессора для задачи фильтрации изображения. На примере двух вариантов векторизации показано повышение эффективности выполнения вычислений при решении этой задачи.

Ключевые слова: векторный сопроцессор, сопроцессор вещественной арифметики, CPV, фильтрация изображений.

1. Введение

В ФГУ ФНЦ НИИСИ РАН разрабатываются универсальные и специализированные процессоры на базе архитектуры MIPS. Специализированные микропроцессоры оснащаются математическим сопроцессором цифровой обработки CPV, использование которого может дать существенный прирост производительности [1] по сравнению с вычислениями на управляющем ядре и на других вычислительных алгоритмах, связанных с массивно-параллельной обработкой больших объёмов данных.

Одной из актуальных задач для сопроцессора CPV является задача фильтрации изображений [2]. Фильтрация изображений стала важной составляющей обработки графической информации [3]. Одним из методов фильтрации изображений является пространственная фильтрация – метод фильтрации изображения, при котором обработка изображения происходит с помощью выбранного оператора последовательно к каждой точке изображения [4]. В данной работе рассматриваются два варианта реализации алгоритма пространственной фильтрации изображения.

2. Постановка задачи

Алгоритм функции пространственной фильтрации представлен в референсном си-коде, который выполняется на целочисленных исполнительных устройствах процессора. Фильтрация выполняется на монохромном изображении, представленным массивом положительных 16-ти разрядных чисел. Для каждой точки изображения вычисляется свертка окрестности этой точки размером 5x5 элементов с ядром фильтра, представляющего собой матрицу целых чисел размером 5x5 элементов. Особенностью алгоритма является то, что обрабатываемая точка находится в левом верхнем углу окрестности.

Было представлено два варианта алгоритма фильтрации изображения на векторном сопроцессоре CPV вариантах. В первом варианте, назовем его «векторизация вширь», элементы векторов собирались из соседних элементов изображения или ядра фильтра. Во втором варианте, назовем его «векторизация вглубь», изображение в горизонтальном направлении делилось на четыре фрагмента, и вектора собирались из соответствующих элементов из каждого фрагмента. Элементы векторов представлены вещественными числами одинарной точности. Размерность векторов равна четырем.

3. Первый вариант - векторизация «вширь»

В первом варианте векторизации окрестность обрабатываемой точки, включающая 25

элементов изображения, представлялась блоком из семи векторов (рис.1). Пять из них объединяют четыре верхних элемента в каждом из 5 столбцов окрестности, шестой вектор собирается из первых четырех элементов в нижней строке окрестности. Седьмой вектор включает один элемент из нижней строки окрестности. Аналогичным образом в векторном виде представляется ядро фильтра. Загрузка элементов изображения из памяти в вектора CPV выполняется в ассемблерном коде через регистры процессора с использованием команд загрузки из памяти в регистры процессора половинных слов *lhu*, команд упаковки *dins* и команд *vmtl(h)* загрузки 64 разрядного слова из регистров процессора в младшую и старшую половину регистров CPV. Преобразование данных в регистрах векторного сопроцессора в формат *float* выполняется с помощью векторной команды преобразования целых чисел в вещественные *chw*. Вычисление свертки элементов окрестности с элементами ядра фильтра выполняется с помощью команд *mvmadd*, реализующих умножение матрицы 2x2 элемента на вектор, команд суммирования элементов вектора *vsum* и команд сборки векторов *muvehl*. За один цикл загрузки элементов изображения в блок регистров CPV свертка вычисляется для 4-х элементов в строке изображения. Запись результата осуществляется с использование команд *vsdm*. В памяти результирующее изображение преобразуется в целочисленный формат.

Ниже представлено описание алгоритма в точности соответствующему референсному коду, но оптимизированного для выполнения на векторном сопроцессоре.

Из маски размером 5x5 выбирается 7 областей, как показано на рис. 1.

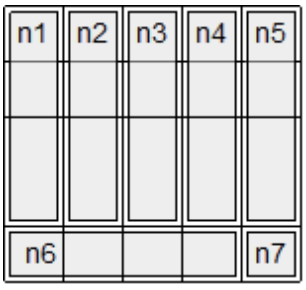

Рис. 1. 7 областей из маски 5x5

Из областей n1-n7 формируются векторы по 4 элемента (рис.2).

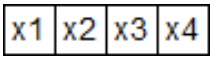

Рис. 2. Формируемый вектор

где x1, x2, x3, x4 – соответствующие значения элементов маски из покрываемой области. Для области n7 x1=x2=x3=x4 и все они соответствуют правому нижнему элементу маски. В итоге получается 7 векторов маски.

Входные данные размером WxH разбиваются на матрицы 8x5, которые состоят из четырёх подматриц 5x5 и которые разбиваются на 13 областей (рис. 3).

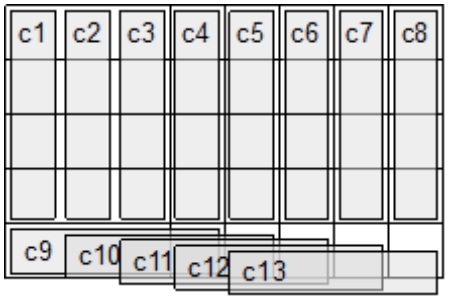

Рис. 3. 13 областей из маски 5x5

Из всех областей c1-c13 аналогичным образом формируются векторы по 4 элемента.

После формирования векторов производится расчёт одновременно для 2-х соседних результирующих точек. Расчёт проводится путём перемножения двух матриц входных данных, как показано на рис. 4, с использованием вектора маски как показано на рис. 5.

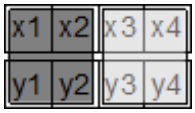

Рис. 4. Перемножение двух матриц входных данных

Рис. 5. Векторы маски для перемножения

с помощью команды векторного сопроцессора умножение матрицы на вектор с накоплением *mvmadd*: матрицы из c1 и c2 (x1/x2/y1/y2 и x3/x4/y3/y4) умножаются на векторы из n1 (z1/z2 и z3/z4). Эта операция производится для векторов из областей c1-c12 и n1-n6 с накоплением результата в два вектора R1 (первые две результирующие точки) и R2 (вторые две результирующие точки). Затем, после редукции векторов R1 и R2 в вектор R, производится финальная операция умножения с накоплением *vmadd* двух векторов c13 и вектора из области n7 с накоплением результата в вектор R. В итоге в векторе R находится результат наложения маски для 4-х соседних точек.

Код расчёта был написан на ассемблере. Изначально входные данные загружаются в стандартные регистры процессора с использованием команд *lhu* и семейства команд *dins* для полного

заполнения 64-х разрядных регистров. Затем происходит перенос данных в регистры векторного сопроцессора с помощью команд *vmtl* и *vmth*. После чего производится преобразование целых чисел в числа с плавающей точкой.

Оптимизация ассемблерного кода была направлена как на разрешение зависимостей по регистрам, так и на максимальное задействование суперскалярности. Для отслеживания условий выполнения инструкций суперскалярности и зависимостей по регистрам был написан анализатор бинарного кода. С помощью него, например, удалось добиться выполнение всего цикла расчёта без остановов конвейера процессора по причине зависимых команд.

В таблице 1 ниже представлен результат теста производительности (в тактах процессора) референсного кода и с использованием векторного сопроцессора для данных размером 1024x1024.

Таблица 1. Результаты теста производительности

Размер входных данных	1024x1024
Референсный код, такты	$116{,}5*10^6$
Векторный код, такты	$30{,}8*10^6$
Прирост, %	377

Наложение маски 5x5: 49 операций сложений и умножений => 196 операций для 4-х точек. Расчёт 4-х соседних точек на векторном сопроцессоре (одинарная точность): 16 инструкций (12 mvmadd + 3 vsum (редукция) + 1 vmadd) => 212 операций.

Для входных данных 1024x1024 пиковая производительность равна

1024 * 1024 / 4 * 16 = 4 194 304.

Таким образом при реализации первого варианта векторизации достигается производительность

4 194 304 / 30 887 459 ≈ 0,136.

В итоге, реализация приведённого алгоритма достигает 13.6% от пиковой.

4. Второй вариант - векторизация «вглубь»

Выполнение программы делится на несколько этапов (рис.6). На первом этапе изображение в горизонтальном направлении делится на 4 фрагмента и выполняется копирование исходного изображения в буфер с добавлением 4 граничных столбцов после каждого фрагмента, включающего W/4 столбцов.

Граничные столбцы дублируют первые четыре столбца в следующем фрагменте и предназначены для корректной обработки последних четырех столбцов в каждом предыдущем фрагменте. Назовем этот этап «pack1». Одновременно с добавлением граничных столбцов на этом этапе выполняется преобразование входных данных их формата *short* в формат *float*.

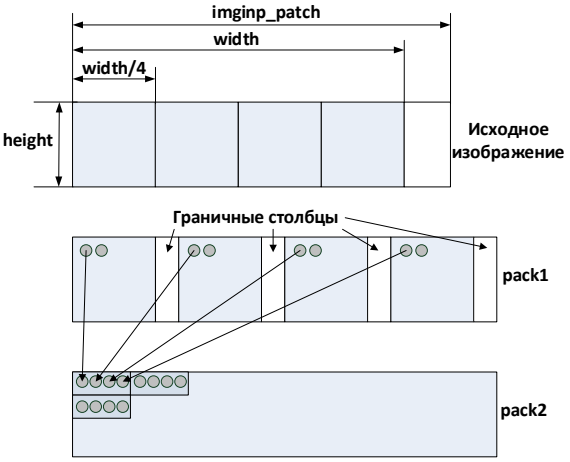

Рис. 6. Этапы pack1 и pack2

На следующем этапе выполняется упаковка этих данных. Четыре соответствующих элемента в памяти из каждого из четырех фрагментов объединяются группы по четыре элемента. Назовем этот этап «pack2». На этом этапе входные данные подготавливаются для загрузки их в регистры векторного процессора.

На следующем этапе путем выполнения векторных операций *vmadd* выполняется фильтрация одновременно во всех четырех фрагментах изображения и запись результата в формате *float* в память (рис. 7).

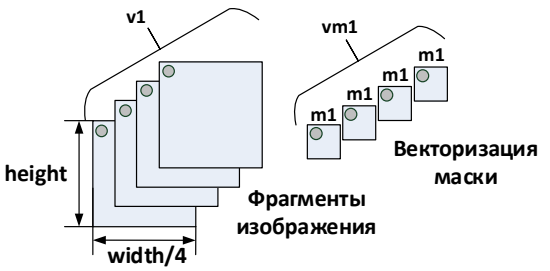

Рис. 7. Этап вычисления

На заключительном этапе – «pack3» выполняется распаковка результата и преобразование в формат *int.*

При векторизации «вглубь», как уже было сказано, изображение разбивается на четыре фрагмента и каждому фрагменту соответствует один элемент вектора. Все четыре фрагмента обрабатываются параллельно по одному и тому же

алгоритму с использованием векторных операций. Ядро фильтра представляется 25 векторами, у каждого из которых все элементы равны значению соответствующего коэффициента ядра. Элементы векторов маски могут загружаться из памяти как во время вычисления свертки, так и предварительно перед началом обработки. В первом случае под маску задействован только один регистр, оставляя доступными под загрузку элементов изображения 63 регистра. Загрузка регистров маски в реальном времени не приводит к увеличению общего времени обработки, поскольку основным фактором, определяющим время выполнения фильтрации при размерах фильтра 5x5 элементов, является время вычисления.

Цикл выполнения свертки для каждого элемента изображения разворачивается и выполняется на блоке регистров CPV. Сверка выполняется с использованием команды *vmadd* – умножение с накоплением вещественных чисел. Для каждого элемента результат накапливается в отдельном регистре. Задержка на выполнение одной команды *vmadd* составляет 5 тактов, поэтому для эффективной работы команды *vmadd* в конвейере с выдачей результата в каждом такте работы процессора необходимо выполнять одновременную обработку не менее 5 точек изображения. Одновременная обработка нескольких точек подразумевает обработку этих точек за одну загрузку блока регистров. Для вычисления свертки одного элемента изображения с фильтром 5x5 элементов требуется блок из 25 регистров. Имеющееся количество регистров позволяет выполнять одновременную обработку до 7 элементов в строке. В этом случае под загрузку элементов изображения задействуется 5x11=55 регистров CPV, семь регистров требуется для накопления результата и один регистр для маски. Общее количество задействованных регистров равно 63.

На рис. 8 представлены результаты оценки коэффициента ускорения выполнения фильтрации на этапе вычисления на векторном сопроцессоре по отношению ко времени выполнения референсного кода на целочисленных устройствах процессора при различном количестве одновременно обрабатываемых точек изображения. Фильтрация выполнялась на изображении размером 1000x1000 элемента. До значения количества обрабатываемых точек равном 4 под маску выделялось 25 регистров, и загрузка регистров маски выполнялась до начала вычислений. При дальнейшем увеличении количества обрабатываемых точек часть регистров маски загружались во время вычислений и количество выделяемых под маску регистров сокращалось вплоть до одного при количестве одновременно обрабатываемых точек равном 7. Видно, что коэффициент ускорения увеличивается с увеличением количества обрабатываемых точек. Максимальное значение достигается при 4 точках и превышает 10. При дальнейшем увеличении количества обрабатываемых точек коэффициент ускорения незначительно снижается.

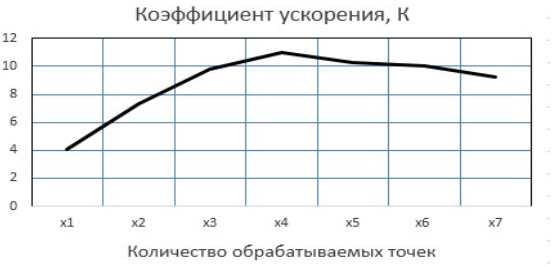

Рис. 8. Зависимость коэффициента ускорения от количества обрабатываемых точек

5. Сравнительная оценка двух вариантов реализации алгоритма фильтрации изображений

Для сравнения производительности двух алгоритмов время их выполнения сопоставлялось с временем выполнения референсного кода на целочисленных исполнительных устройствах процессора. Время выполнения отдельных этапов можно оценить исходя из коэффициента ускорения и измеренного времени выполнения референсного кода, которое при параметрах изображения width=1000, height=1000 составляет $116{,}5*10^6$ тактов.

Результаты оценки ускорения выполнения фильтрации приведены в таблице 2. Измерения коэффициентов ускорения выполнялось при моделировании работы процессора и векторного сопроцессора на ПЛИС ALTERA. Функция фильтра вызывалась со следующими параметрами:

width = 1000, height = 1000,
imginp_pitch = 1004, imgout_pith = 1000,
shift = 0, round = 0, thr = 1900000.

Таблица 2. Сравнение коэффициентов ускорения двух вариантов векторизации

Коэффициент	Вариант 1	Вариант 2
K	3.8	11
K1	-	6,1
K2	-	3,6
K3	-	2,1

Для второго варианта определены четыре коэффициента ускорения К, К1, К2 и К3. К – максимальный коэффициент ускорения только на этапе вычислений в ситуации, когда элементы исходного результирующего изображений представлены в формате *float* и упакованы в соответствии с концепцией векторизации вглубь. К1 – коэффициент ускорения вычислений с учетом времени потраченного на упаковку элементов входного изображения, которое уже представлено в формате *float* (этап «pack2»). К2 - коэффициент ускорения вычислений с учетом времени потраченного на преобразование входного изображения во *float* и упаковку (этапы pack1 и pack2**).** К3 - коэффициент ускорения с учетом всех преобразований, то есть когда входное изображение представлено в формате *short*, а результирующее в формате *int*. Такие форматы входного и результирующего изображения приняты в референсном коде и в первом варианте реализации алгоритма фильтрации на CPV. Оценка ускорения только на вычислительном этапе представляет интерес, поскольку преобразование и упаковка (этапы pack1 и pack2) могут быть выполнены на фоне загрузки изображения в память, а необходимость выполнения обратного преобразования (этапа pack3) определяется тем, каким образом используется результат фильтрации в дальнейших вычислениях.

Как видно из таблицы 2 на этапе вычисления ускорение выполнения фильтрации на CPV по отношению к выполнению референсного кода на целочисленных исполнительных устройствах процессора во втором варианте алгоритма равно 11. Время выполнения этапа вычислений во втором варианте, то есть без учета времени, потраченного на преобразования, составляет примерно 10^7 тактов и существенно меньше времени выполнения фильтрации в первом варианте реализации алгоритма. С учетом времени, потраченного на прямое и обратное преобразования массивов, коэффициент время выполнения фильтрации во втором варианте алгоритма примерно в два раза больше, чем в первом варианте. Следует отметить, что преобразования форматов и упаковка массивов реализована на уровне компиляции и дополнительных усилий по оптимизации этих процедур предпринято не было, так как данная задача уже решалась в рамках [5].

Оценим эффективность выполнения вычислений на векторном сопроцессоре во втором варианте реализации алгоритма. Эффективность организации вычислений будем оценивать, как процентное отношение реально достигнутой производительности к максимально возможному (пиковому) значению. Это отношение можно вычислить как отношение реально выполненного количества вычислительных вещественных операций за такт к теоретически возможному максимальному количеству вычислительных вещественных операций за такт, которое можно реализовать при вычислениях.

При выполнении фильтрации монохромного изображения с размером фильтра m*m для каждой точки изображения выполняется m^2 операций умножения и m(m-1) операция сложения. Для изображения размером *height* * *width* необходимое количество вычислительных операций равно:

$$N \approx (2m^2)* \textbf{ height * width},$$

что при значении параметров: height = 1000, widths = 1000, m = 5 составляет $5*10^7$.

При выполнении фильтрации на векторном сопроцессоре во втором варианте реализации алгоритма при этих значениях параметров затрачивается, как было показано выше, примерно 10^7 тактов. То есть в этом варианте на вычислительном этапе выполняется 5 вычислительных вещественных операций за такт. Максимальное теоретически возможное (пиковое) количество операций за такт равно 4x2=8. Коэффициент 4 – это размерность вектора (*float*), а коэффициент 2 – вклад использования векторной команды *vmadd*. Таким образом, на вычислительном этапе во втором варианте достигается производительность 5/8*100% = 62,5% от пикового значения.

В итоге, реализация приведённого алгоритма фильтрации достигает 62,5% от пиковой.

6. Заключение

В статье были рассмотрены два варианта алгоритмов пространственной фильтрации изображения с помощью векторного сопроцессора. Была проведена сравнительная оценка каждого из вариантов фильтрации.

Предложенные алгоритмы позволили повысить производительность выбранной задачи пространственной фильтрации изображения. Первый вариант – векторизация «вширь» – позволил достичь ускорения выполнения фильтрации в 3,8 раза, а второй вариант – векторизация «вглубь» – в 2,1 раза при решении оригинальной задачи по сравнению с решением на управляющем ядре. При этом большая часть выигрыша в производительности теряется на переконвертацию входных и выходных данных, в то время как выполнение непосредственно вычислительной задачи ускоряется в 11 раз. Корректировка исходной задачи под выполнение на векторном сопроцессоре избавляет от необходимости конвертации данных и позволяет достичь максимальной производительности при использовании CPV.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2022-0004.

Implementation of Spatial Image Filtering Function on a Vector Coprocessor

S. I. Aryashev, P. A. Chibisov, V. V. Tsvetkov, D. A. Trubitsyn, K. A. Petrov

Abstract. To increase the performance of universal microprocessors of MIPS-like architecture, NIISI RAS developed a specialized coprocessor that allows accelerating operations with complex and real numbers of single and double precision. The article presents the results of using a 128-bit vector coprocessor for the image filtering task. Using the example of two vectorization options, the increase in the efficiency of calculations when solving this problem is show

Keywords: vector coprocessor, CPV, image filtering

Литература

1. П.Б. Богданов, О.Ю. Сударева. Применение отечественных специализированных процессоров семейства КОМДИВ в научных расчетах // Информационные технологии и вычислительные системы 3/2016.

2. S. Sadangi, S. Baraha, D. K. Satpathy and P. K. Biswal, "FPGA implementation of spatial filtering techniques for 2D images," 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2017, pp. 1213-1217, doi: 10.1109/RTEICT.2017.8256791.

3. D. H. Rao and P. P. Panduranga, "A Survey on Image Enhancement Techniques: Classical Spatial Filter, Neural Network, Cellular Neural Network, and Fuzzy Filter," 2006 IEEE International Conference on Industrial Technology, Mumbai, India, 2006, pp. 2821-2826, doi: 10.1109/ICIT.2006.372671.

4. Н.С. Сергеев. Пространственная фильтрация изображений в системах технического зрения.

5. П.Б. Богданов, О.Ю. Сударева. Декодирование изображений в формате JPEG на процессорах КОМДИВ // Информационные технологии и вычислительные системы 1/2019.