УДК 004.93'1; 004.932

А.И. Кравчонок

АЛГОРИТМЫ МЕДИАННОЙ ФИЛЬТРАЦИИ С ОКНОМ 3×3 С ПРИМЕНЕНИЕМ ТЕХНОЛОГИЙ MMX И SSE2 ПРОЦЕССОРОВ СЕМЕЙСТВА x86

Предлагаются быстрые алгоритмы выполнения медианной фильтрации с окном 3×3 на персональном компьютере при помощи MMX- и SSE2-команд. Описываются различные неполные сортирующие сети для поиска медианы окна фильтра, допускающие реализацию при помощи MMX- и SSE2-команд. Алгоритмы позволяют значительно ускорить выполнение медианной фильтрации изображений, что делает возможным ее применение в режиме реального времени в программах обработки изображений.

Введение

При решении задач обработки видеопоследовательностей в режиме реального времени часто применяются различные типы фильтрации для улучшения качества обрабатываемых данных. Среди применяемых фильтров можно назвать усредняющий, гауссовский и др. Они являются линейными, и как следствие время их работы невелико. Медианный фильтр обладает хорошим сглаживающим эффектом без существенного размытия границ на изображении [1]. Перспективным направлением развития данного метода фильтрации является использование взвешенного и адаптивного медианного фильтра. Однако этот фильтр нелинейный и поэтому по скорости работы значительно уступает другим, более простым. Медианный фильтр редко применяется для задач обработки изображений в реальном времени, например при отслеживании объектов на видеопоследовательностях, хотя его применение могло бы значительно улучшить качество решения подобных задач [2, 3]. Для ускорения работы фильтра зачастую его реализуют аппаратно либо в параллельном режиме для многопроцессорных систем. Известны также реализации медианного фильтра с небольшим размером окна (3×3, 5×5) при помощи SIMD(Single Instruction, Multiple Data)-команд. Окон такого размера достаточно, чтобы удалить большую часть шумов на кадрах видеопоследовательностей. Можно также заметить, что увеличение размеров окна фильтрации приведет к тому, что будут потеряны более крупные детали изображения, так как медианный фильтр удаляет с изображения все детали, размер которых составляет менее половины окна фильтра. Учитывая, что при обработке видеопоследовательностей размер изображений относительно невелик в связи с невысоким разрешением видеокамер, подобное удаление достаточно крупных деталей может привести к потере информации. Следовательно, полезным был бы алгоритм, реализующий быструю медианную фильтрацию изображений с небольшим окном фильтра, например 3×3.

Ускорить выполнение медианной фильтрации позволяют следующие наиболее эффективные алгоритмы:

– быстрой медианной фильтрации Хуанга [4];
– слияния упорядоченных столбцов [5, 6];
– медианной фильтрации Паеса [7];
– неполной сортировкой прямым выбором [8];
– Кучеренко – Очина [9, 10];
– на основе неполных сортирующих сетей [11];
– Колта [12];
– Дифендорфа [13].

Значительного ускорения работы медианного фильтра можно добиться при помощи его реализации без использования условных операторов. Осуществить это можно путем замены условных переходов арифметическими действиями [8, 11, 14–18].

Хотя указанные алгоритмы и позволяют выполнять обработку изображений в режиме реального времени, однако их применение все же ограничено вследствие недостаточного быстродействия.

MMX (Multimedia Extensions) – дополнительный набор инструкций, позволяющий ускорить обработку мультимедиаданных (звука, видеопотоков, изображений). В настоящее время он поддерживается практически всеми процессорами как фирмы Intel, так и AMD. MMX включает в себя набор 64-битных регистров, а также дополнительные команды, которые позволяют выполнять некоторые арифметические операции параллельно. Например, за одну операцию можно параллельно складывать, вычитать либо выполнять другие операции над восемью байтами, четырьмя словами либо двумя двойными словами [15–19]. Технология SSE2 (Streaming SIMD Extensions 2) является развитием технологии MMX и позволяет работать со 128-битовыми данными, в том числе и с целочисленными [15–17].

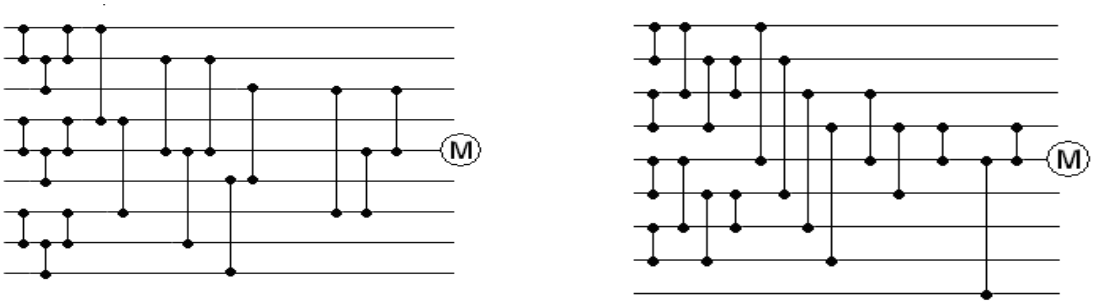
Применение MMX, а также SSE2 для осуществления медианной фильтрации значительно ускоряет ее выполнение, расширяя возможности и сферы использования подобного типа фильтрации для множества задач, включая обработку изображений в реальном времени.

Реализации медианного фильтра при помощи SIMD-инструкций (к такому типу инструкций относятся команды MMX и SSE2) существуют [12, 13, 20], однако можно предложить алгоритмы, более эффективные по сравнению с известными. Реализация медианного фильтра посредством SIMD-инструкций является наиболее предпочтительной, так как данный тип инструкций присутствует в настоящее время практически по всех известных процессорах, при этом использование данных инструкций позволяет значительно – в разы – ускорить выполнение медианной фильтрации.

1. Алгоритмы медианной фильтрации на основе сортирующих сетей

Алгоритмы медианной фильтрации на основе неполных сортирующих сетей выгодно отличаются от других возможностью легко реализовать в них замену условных операторов арифметическими действиями [11]. В основе сортирующих сетей лежит операция сравнения-обмена (компаратор), которая выполняется посредством сравнивающего устройства, или компаратора. Компаратор, имея два входа x и y и два выхода x' и y', выполняет операцию $x' = \min(x, y)$, $y' = \max(x, y)$, если задан порядок по возрастанию, или $x' = \max(x, y)$, $y' = \min(x, y)$, если задан порядок по убыванию. Вся сортирующая сеть составляется из подобных компараторов и соединяющих их проводов [21–24]. В дальнейшем будем рассматривать компараторы, для которых задан порядок по возрастанию.

Поиск медианы последовательности из девяти элементов можно осуществлять при помощи различных сортирующих сетей, при этом 19 компараторов – это необходимый минимум, который требуется для построения таких сетей [7, 25]. На рис. 1 изображены две неполные сортирующие сети, являющиеся модифицированными сортирующими сетями Флойда [21] и Бэтчера [23], которые позволяют выполнять поиск медианы девяти элементов за 19 операций сравнения-обмена [11].

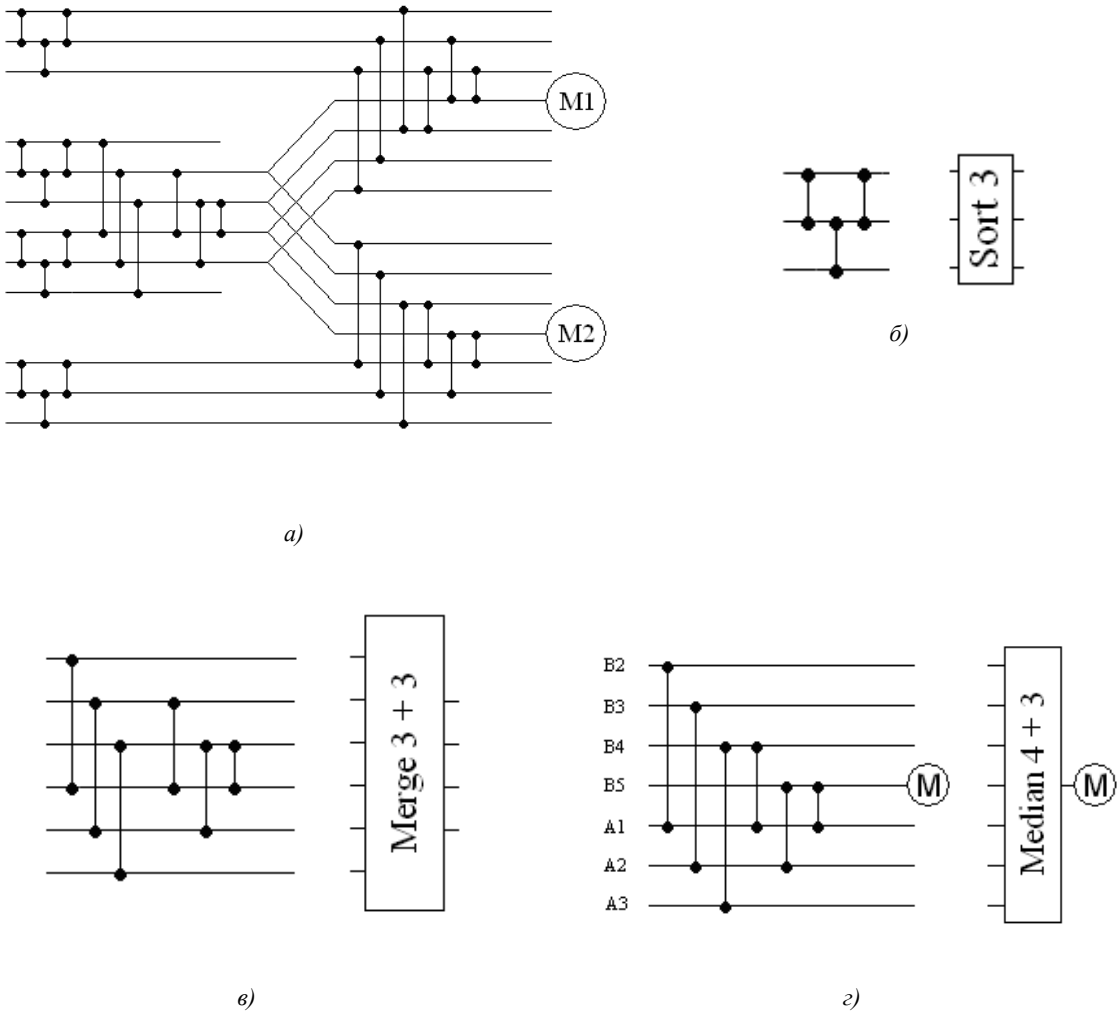


а) *б)*

Рис. 1. Сети поиска медианы девяти элементов: *а)* неполная сортирующая сеть для поиска медианы на основе сети Флойда; *б)* на основе сети Бэтчера

Если учесть тот факт, что при медианной фильтрации изображений соседние окна медианного фильтра пересекаются, можно построить более эффективные неполные сортирующие

сети для поиска медианы окна фильтра. На рис. 2, *а* изображена сортирующая сеть, позволяющая выполнять медианную фильтрацию изображений с окном фильтра 3×3 за 15 операций сравнения-обмена на поиск одной медианы окна фильтра. На вход сети поступают 12 элементов, образующих два пересекающихся окна фильтра, на выходе сети получаются две найденные медианы для каждого из окон фильтра. Сеть содержит 30 компараторов, поэтому на поиск одной медианы приходится 30/2 = 15 компараторов [11].



а) *б)* *в)* *г)*

Рис. 2. Неполная сортирующая сеть поиска медиан двух пересекающихся окон фильтра и ее элементы: *а)* сортирующая сеть; *б)* элемент сети, сортирующий тройку элементов, и его графическое обозначение; *в)* элемент сети, выполняющий слияние отсортированных троек, и его графическое обозначение; *г)* элемент сети, выполняющий поиск медианы на основе отсортированных четверки и тройки, и его графическое обозначение

Для упрощения графического отображения обозначим некоторые части сортирующей сети, изображенной на рис. 2, *а*, при помощи графических элементов. Элемент «Sort 3» (рис. 2, *б*) выполняет сортировку трех входящих элементов. Элемент «Merge 3+3» (рис. 2, *в*) выполняет слияние двух отсортированных троек, при этом первый и последний элементы из отсортированной шестерки нас не интересуют и дальше не рассматриваются. Элемент «Median 4+3» (рис. 2, *г*) выполняет поиск медианы окна фильтра на основе отсортированных четверки и тройки элементов. Рассматриваемая сортирующая сеть (рис. 2, *а*) изображена при помощи введенных графических обозначений на рис. 3.

Заметим, что у элемента «Median 4+3», обозначенного на рис. 3 буквой П, входы, предназначенные для отсортированных четверки и тройки элементов, перевернуты по отношению к исходному элементу «Median 4+3». Это сделано для простоты отображения сортирующей сети. Очевидно, что сеть можно построить и при помощи только исходных элементов, не переворачивая их входы.

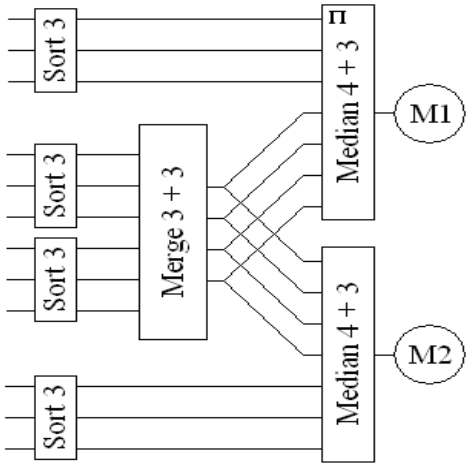


Рис. 3. Неполная сортирующая сеть поиска медиан двух пересекающихся окон фильтра, 30 компараторов

Применяя описанную выше сортирующую сеть, на каждом шаге выполнения фильтрации ведется поиск сразу двух медиан, а передвижение по изображению окон фильтра выполняется с шагом в два элемента. Поэтому на каждом следующем шаге можно не сортировать две первые тройки элементов, поступающие на вход сети, а сохранить результат их сортировки на предыдущем шаге алгоритма и использовать на последующем. Таким образом экономятся шесть операций сравнения-обмена и общее число компараторов в сети уменьшается до 24, при этом на одну медиану будет приходиться 24/2 = 12 операций сравнения-обмена.

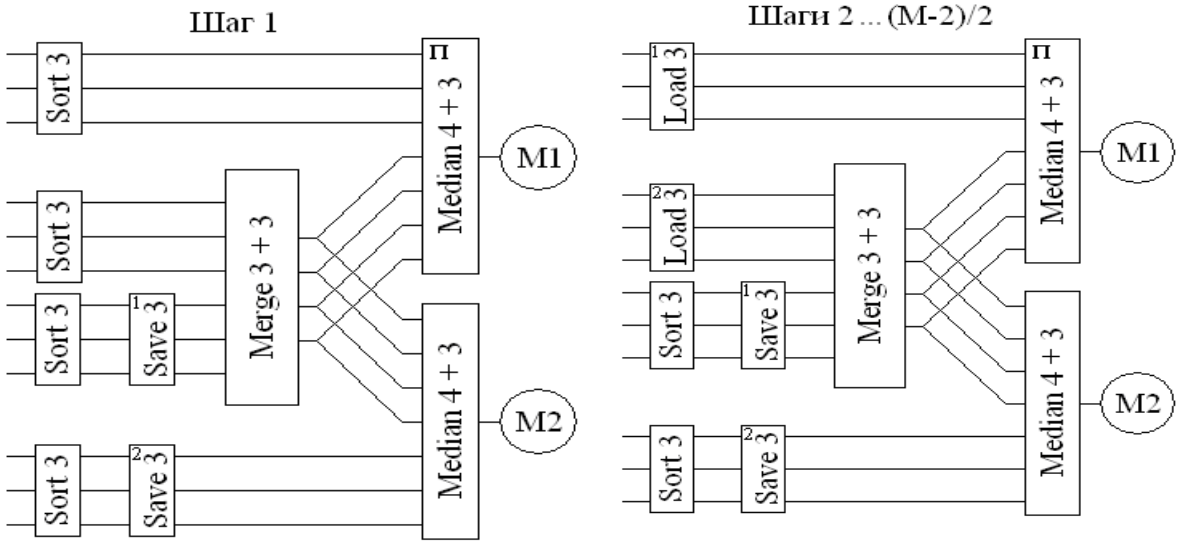
Введем два дополнительных графических элемента «Save 3» и «Load 3» (рис. 4, *а, б*). Элемент «Save 3» позволяет сохранять тройку элементов в некоторой области памяти, а элемент «Load 3» позволяет загружать ранее сохраненную тройку из памяти и использовать в сортирующей сети. Элементы имеют индексы, при этом элемент «Load 3» с некоторым индексом загружает только тройку, сохраненную последний раз при помощи элемента «Save 3» с таким же индексом, что и у него. Таким образом, элементы «Save 3» и «Load 3» с одинаковыми индексами соответствуют командам помещения в стек и извлечения из него, при этом стек вмещает в себя не более одной тройки элементов. Индексы соответствующих друг другу элементов отображаются в верхнем левом углу их графического отображения.

а) *б)*

Рис. 4. Графические элементы для сохранения и загрузки тройки элементов сортирующей сети:
а) элемент, сохраняющий тройку в память; *б)* элемент, загружающий тройку из памяти

Применяя введенные графические элементы, построим неполную сортирующую сеть, позволяющую сохранять отсортированные на некотором шаге работы алгоритма две тройки элементов, чтобы использовать их на следующем шаге алгоритма. Необходимо учесть также, что на первом шаге работы алгоритма негде взять уже отсортированные тройки и поэтому их необ-

ходимо сортировать. На рис. 5, *а* показана сортирующая сеть, которую необходимо применить на первом шаге работы алгоритма.



а) *б)*

Рис. 5. Сети поиска медиан окон фильтра: *а)* сеть, используемая на первом шаге выполнения фильтрации; *б)* сеть, используемая на последующих шагах выполнения фильтрации

Сеть позволяет получить две медианы для двух соседних пересекающихся окон фильтра за 30 операций сравнения-обмена и при этом сохраняет две отсортированные тройки элементов окна при помощи элементов «Save 3» для дальнейшего использования. На последующих шагах работы алгоритма применяется сортирующая сеть, изображенная на рис. 5, *б*. При помощи элементов «Load 3» она загружает сохраненные на предыдущих шагах алгоритма уже отсортированные две тройки элементов и также сохраняет две новые отсортированные тройки для последующего использования. Сеть выполняет поиск двух медиан соседних окон фильтра за 24 операции сравнения-обмена. Так как сеть, содержащая 30 компараторов, используется только один раз в начале работы алгоритма, а для всех дальнейших окон применяется сеть, содержащая 24 компаратора, получаем приблизительно 12 операций сравнения-обмена на поиск медианы окна фильтра.

2. Параллельная программная реализация компаратора при помощи MMX и SSE2

Будем рассматривать в дальнейшем компараторы, для которых задан порядок по возрастанию. Программно реализовать компаратор можно, например, следующим образом:

```
if ( a > b )
{
      v = a ; a = b ; b = v ;
}
```

где a и b – входные значения; v – временная переменная.

Заметим, что при обмене значений переменных a и b можно обойтись без временной переменной v, например, таким образом:

```
if ( a > b )
{
      a = a – b ; b = b + a ; a = b – a ;
}
```

Однако подобная реализация компаратора неэффективна, так как использует оператор условного перехода, который значительно замедляет выполнение программы. Существует несколько возможностей программно реализовать компаратор без использования условных операторов [6, 8, 11]. Среди них можно выделить следующие методы:

– операции XOR, метод предложен Паесом [7];
– табличный метод, реализованный в OpenCV [26];
– команды ассемблера [14–18];
– SIMD(MMX и SSE2)-команды [12, 13, 20].

Преимуществом метода, реализуемого посредством MMX- и SSE2-команд, по сравнению с другими является то, что помимо замены условных операторов арифметическими действиями он позволяет находить параллельно сразу несколько медиан, что приводит к еще более значительному увеличению быстродействия.

MMX и SSE2 включают в себя команды, поддерживающие арифметику с насыщением и позволяющие эффективно реализовать компаратор. Пусть $\overline{R}$ – некоторая переменная, значение которой будем вычислять следующим образом:

$$\overline{R} = \begin{cases} a-b, a > b; \\ 0, a \le b. \end{cases}$$

Получим, что $\overline{R}$ принимает значение разности переменных a и b в случае, если порядок следования переменных нарушен, и нулевое значение в случае, если порядок верен. Тогда компаратор можно реализовать следующим образом. При рассчитанном для переменных a и b значении $\overline{R}$ необходимо выполнить следующие две операции:

$$a = a - \overline{R};$$
$$b = b + \overline{R}.$$

В случае, если $a > b$:

$$\overline{R} = a - b;$$
$$a = a - \overline{R} = a - (a - b) = b;$$
$$b = b + \overline{R} = b + (a - b) = a,$$

получим обмен значений a и b. Если же $a \le b$:

$$\overline{R} = 0;$$
$$a = a - \overline{R} = a;$$
$$b = b + \overline{R} = b,$$

значения переменных a и b остаются прежними.

Вычисление $\overline{R}$ можно осуществить при помощи операции разности a и b, используя арифметику с насыщением. Это означает, что, в случае, если результат операции не вмещается в разрядной сетке, вместо переполнения устанавливается максимально возможное либо минимально возможное значение числа в зависимости от границы разрядной сетки (верхняя или нижняя), где произошло переполнение. Тогда любое отрицательное значение $R = a - b$ при вычитании беззнаковых байтовых чисел принимает значение ноль. MMX и SSE2 поддерживают арифметику с насыщением, поэтому позволяют эффективно программно реализовать компаратор описанным выше способом. При этом подобная реализация компаратора позволяет выполнять сразу 8 или 16 операций сравнения-обмена в параллельном режиме, что значительно ускоряет медианную фильтрацию.

Пусть (x_i, y_i), $0 \le x_i \le 255$, $0 \le y_i \le 255$, $i = \overline{1,8}$, – восемь пар беззнаковых элементов размерностью в байт и для каждой пары необходимо выполнить операцию упорядочивания элементов пары – операцию сравнения-обмена. Пусть mm0 – MMX-регистр, в который запишем последовательно x_i – первые элементы каждой пары, а mm1 – MMX-регистр, в который занесем последовательно y_i – вторые элементы каждой пары чисел (рис. 6). При помощи MMX-операции вычитания для упакованных чисел с использованием арифметики с насыщением получим значения r_i, которые поместим в регистр mm2. Теперь выполнить сравнение и возможный обмен элементов можно при помощи операций

$$x_i = x_i - r_i,\ i = \overline{1,8};$$

$$y_i = y_i + r_i,\ i = \overline{1,8}.$$

Заметим, что MMX позволяет выполнять каждую арифметическую операцию для всех $i = \overline{1,8}$ в параллельном режиме, вычисляя за одну команду сразу все необходимые значения.

mm0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
mm1	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
mm2	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
mm0	$x_1 - r_1$	$x_2 - r_2$	$x_3 - r_3$	$x_4 - r_4$	$x_5 - r_5$	$x_6 - r_6$	$x_7 - r_7$	$x_8 - r_8$
mm1	$y_1 + r_1$	$y_2 + r_2$	$y_3 + r_3$	$y_4 + r_4$	$y_5 + r_5$	$y_6 + r_6$	$y_7 + r_7$	$y_8 + r_8$

Рис. 6. Реализация операции сравнения-обмена при помощи MMX

Аналогичным образом компаратор реализуется при помощи SSE2-команд, которые в отличие от MMX позволяют выполнять параллельно 16 операций сравнения-обмена. Код, реализующий параллельный компаратор при помощи SSE2, выглядит следующим образом:

```
movdqa xmm2,xmm0;
psubusb xmm2,xmm1;
psubusb xmm0,xmm2;
paddusb xmm1,xmm2;
```

Подобная реализация операции сравнения-обмена при помощи MMX описана в [20].

Начиная с процессоров Pentium III, в состав MMX вошли дополнительные команды. Среди них PMINUB и PMAXUB, которые позволяют извлекать максимальное значение из каждой пары упакованных элементов в выходном и входном операндах. В качестве операндов выступают MMX-регистры (либо ячейка памяти для входного операнда) [16]. При использовании MMX данные команды позволяют за одну операцию над двумя MMX-регистрами найти восемь минимумов или максимумов из восьми пар беззнаковых байтов. При использовании SSE2 команды позволяют найти 16 минимумов или максимумов за одну операцию. Используя описанные выше команды, легко реализовать компаратор, позволяющий упорядочивать сразу 8 для MMX и 16 для SSE2 пар входящих значений. Допустим, что в регистре xmm0 и xmm1 находятся беззнаковые байтовые числа, тогда реализация компаратора будет выглядеть так:

```
movdqa xmm2,xmm0;
pminub xmm0,xmm1;
pmaxub xmm1,xmm2;
```

Используя регистр xmm2 в качестве временной переменной, после выполнения указанного выше кода получим в регистре xmm0 минимумы 16 пар чисел, а в регистре xmm1 – максимумы.

Реализация компаратора при помощи указанных PMINUB- и PMAXUB-инструкций работает быстрее, так как требует меньшего количества MMX- или SSE2-команд.

Необходимо заметить, что похожая реализация медианной фильтрации при помощи поиска минимумов и максимумов пар чисел на основе инструкций AltiVec, описанная в статье Колта [12], несколько отличается от подобной реализации на SSE2, так как команды AltiVec имеют более широкие возможности по сравнению с SSE2. Набор команд AltiVec содержит недеструктивные операторы с тремя и четырьмя операндами, которые позволяют сохранять результат в новый регистр без потери данных в исходных регистрах [13]. Таким образом, при реализации на AltiVec нет необходимости использовать дополнительные инструкции пересылки данных при реализации компаратора, как на SSE2. Компаратор на AvtiVec реализуется посредством только двух команд в отличие от трех на SSE2. Существует и другое отличие AltiVec от SSE2, которое облегчает реализацию медианной фильтрации. AltiVec содержит 32 регистра в отличие от 8 в SSE2, поэтому все необходимые обрабатываемые 12 векторов без затруднений помещаются в отдельные регистры и легко обрабатываются. При использовании же SSE2 приходится сначала обрабатывать часть векторов, потом считывать в освободившиеся после обработки предыдущей части регистры новую порцию векторов и обрабатывать их отдельно. Следовательно, необходимо иметь алгоритм, который допускает подобную обработку векторов по частям.

Отметим, что в статье Колта [12] не описана реализация компараторов, так как автор разрабатывал алгоритм в терминах операций MIN и MAX. Однако не всегда пара операций MIN и MAX может реализовать компаратор (пример указан выше – в SSE2 и MMX для реализации компаратора нужна еще и пересылка) и не всегда пара таких операций работает так же быстро, как отдельная реализация компаратора. Примером может являться табличная реализация компаратора на CPU, которая работает быстрее, чем пара табличных операций поиска минимума и максимума пары чисел [8, 11].

Описанный выше алгоритм можно усовершенствовать, если заменить некоторые компараторы, как в алгоритме Колта, операциями MIN и MAX. Однако, применяя команды SSE2, подобную замену нужно делать осмотрительно, так как компаратор не заменяется в SSE2 двумя последовательными операциями MIN и MAX.

3. Параллельная реализация алгоритма медианной фильтрации на основе неполных сортирующих сетей при помощи MMX и SSE2

Алгоритм медианной фильтрации при помощи MMX реализуется посредством сортирующих сетей с компараторами, выполняющими параллельно восемь операций сравнения-обмена.

Рассмотрим два соседних окна медианного фильтра. Второе окно будет отличаться от первого только тремя элементами, т. е. во второе окно входят шесть элементов предыдущего окна.

Пусть изображение I имеет размеры $M \times N$, где M – ширина изображения, а N – его высота. Рассмотрим первые три строки изображения – A, B, C (для последующих строк фильтрация выполняется аналогично). Пусть a_i, b_i, c_i, $i = \overline{1, M}$, – элементы строк A, B, C соответственно. Рассмотрим первые 10 элементов каждой строки:

$$\begin{array}{cccccccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} \\ c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} \end{array}$$

В первое окно фильтра входят элементы a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3. Запишем их в виде столбца (рис. 7). Аналогичным образом запишем в виде столбцов элементы следующих окон фильтра, полученных путем сдвига вправо.

Заметим, что после расположения элементов окон фильтра в определенном порядке в виде столбцов порядок следования элементов в строках получившейся матрицы будет повторять порядок следования элементов в строках изображения. Имея определенное количество 8-байтных регистров, можно заполнять их значениями яркостей пикселей изображения, копируя за одну операцию сразу восемь значений (команда MMX movq), при этом нет необходимости менять порядок следования элементов изображения, формировать некие дополнительные структуры данных и использовать для этого дополнительные пересылки.

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9
c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}

Рис. 7. Элементы оконных фильтров, расположенные в виде столбцов

Скопировав необходимые данные в MMX-регистры, можно выполнять над ними операции сравнения-обмена, при этом параллельно будет обрабатываться восемь соседних окон медианного фильтра. Чтобы скопировать данные в MMX-регистры в таком порядке, как указано на рис. 7 (чтобы в первый байт каждого регистра попали первые из девяти элементов соседних окон медианного фильтра, во вторые байты – вторые элементы и т. д.), необходимо последовательно выполнять чтение восьми подряд идущих элементов из трех строк изображения, начиная с элементов a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3 (напоминаем, что команда MMX movq позволяет считывать восемь последовательно идущих байтовых элементов из массива).

Аналогичный подход используется при применении SSE2-команд с тем лишь отличием, что нужно считывать сразу по 16, а не по 8 байт из изображения. Это позволит вести поиск сразу 16 медиан. В SSE2 также используются другие команды чтения в регистры из оперативной памяти.

Реализация подобного механизма чтения данных рассмотрена в [13]. В статье Колта [12] описана несколько иная процедура чтения данных вследствие того, что команды AltiVec не допускают чтения невыровненных данных, однако содержат специальную команду формирования вектора данных с необходимым смещением из двух других векторов. Необходимо заметить, что в SSE2 есть две разные команды для чтения выровненных и невыровненных данных, при этом невыровненные данные читаются несколько медленнее, чем выровненные. Поэтому необходимо, где возможно, использовать команду чтения выровненных данных, в остальных случаях применять команду чтения невыровненных. При реализации компаратора используется пересылка из регистра в регистр, для которой можно использовать команду пересылки выровненных данных. Однако при чтении данных из оперативной памяти такой командой необходимо пользоваться осмотрительно. Вероятно, может быть разработан алгоритм, основанный на получении новых векторов данных из ранее считанных за счет сдвига, так как команды, аналогичной команде AltiVec, для формирования вектора данных на основе двух других в SSE2 нет.

При использовании процессоров семейства x86 существует проблема: для параллельной обработки окон медианного фильтра требуется десять регистров – девять для хранения значений окон фильтра и один дополнительный для выполнения операции сравнения-обмена. Однако в процессорах семейства x86 MMX-регистров только восемь (так же, как и SSE2-регистров). Поэтому загрузить сразу все данные по восьми соседним окнам фильтра в MMX-регистры нет возможности и обрабатывать данные придется по частям. Для этого может быть использована,

например, сортирующая сеть, изображенная на рис. 8, в которой сначала сортируются первые шесть элементов. Вначале данные из изображения считываются в шесть MMX-регистров и седьмой регистр используется в качестве временной переменной. Заметим, что из шести отсортированных элементов первый и последний элементы не могут быть медианой окна фильтра, так как для первого элемента существует минимум пять элементов, больших, чем он, а для последнего – минимум пять меньших. Следовательно, первый и последний элементы отсортированной шестерки можно далее не рассматривать. Затем в MMX-регистры, в которых находятся первый и шестой элементы, загружаются следующие строки матрицы – седьмая и восьмая. В один свободный оставшийся регистр загружается последняя строка матрицы. Далее продолжаем выполнение операций сравнения-обмена, указанных в сортирующей сети. После выполнения всех операций медиана будет находиться в пятом регистре. Таким образом, отбросив некоторые данные на определенном шаге работы сети, можно реализовать указанную сортирующую сеть при помощи только восьми MMX-регистров.

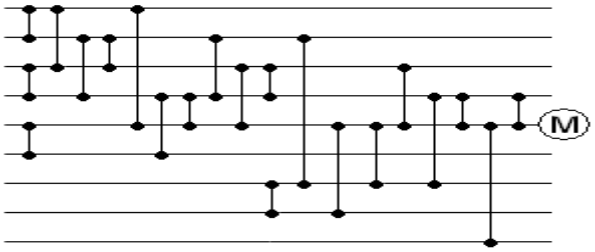


Рис. 8. Неполная сортирующая сеть поиска медианы, 21 компаратор

Остается лишь сохранить результат – восемь найденных медиан восьми соседних окон медианного фильтра. На следующем шаге можно снова считывать в регистры элементы следующих восьми окон медианного фильтра и продолжить поиск медиан аналогичным образом.

Путем небольшой модификации неполной сортирующей сети, построенной на основе сети Флойда (см. рис. 1, *а*), можно получить более эффективную сеть поиска медианы окна фильтра (рис. 9). Указанная сеть также может быть реализована посредством только восьми MMX-регистров.

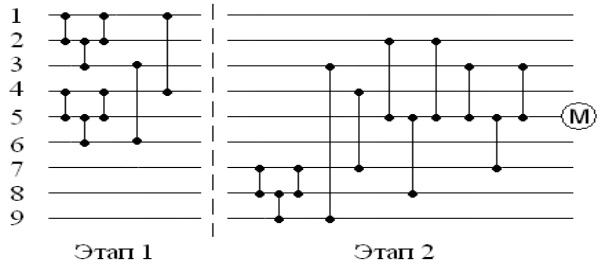


Рис. 9. Неполная сортирующая сеть поиска медианы на основе сети Флойда, 19 компараторов

Однако сеть, изображенную на рис. 5, реализовать подобным образом, используя только восемь MMX(SSE2)-регистров, сложно. Поэтому необходимо применить вариант с сохранением данных в отдельных массивах. Таким образом, элементы «Save 3» и «Load 3» получают конкретное наполнение в виде команд копирования трех MMX(SSE2)-регистров в некоторый массив и считывания ранее сохраненных регистров из этого массива. Подобную функцию сохранения придется также применять и для отсортированной четверки, полученной после слияния двух отсортированных троек, так как в процессе подсчета медианы при слиянии четверки и отсортированной тройки значения отсортированной четверки искажаются. Хотя копирование данных в массивы и из массивов замедляет скорость работы программы, можно заметить, что, так как для временного хранения значений MMX(SSE2)-регистров используется один и тот же

массив небольшого размера – шесть векторов по 8 байт (шесть векторов по 16 байт для SSE2), вследствие эффективного кэширования скорость выполнения данных операций достаточно высока. При подобном копировании данных можно использовать SSE2-команды для выровненных данных, что ускоряет работу операций сохранения и чтения.

Для того чтобы эффективно вычислять сразу 16 медиан для 16 окон фильтра при помощи SSE2-команд и описанных выше сортирующих сетей (21 и 19 компараторов), необходимо в SSE2-регистры помещать значения 16 окон фильтра, пересекающихся по горизонтали. Описанный же ранее в разд. 1 алгоритм, использующий пересечение двух соседних окон фильтра (12 компараторов), требуется применять для окон, пересекающихся по вертикали. В этом случае можно будет получить выигрыш и за счет параллельного вычисления 16 медиан окон, располагающихся по горизонтали, и за счет эффективного вычисления сразу двух медиан для окон, располагающихся по вертикали. Таким образом, на каждом шаге алгоритма вычисляется сразу 32 медианы, при этом для вычисления одной медианы требуется в среднем 12 операций сравнения-обмена, что на одну операцию превосходит алгоритм Кучеренко – Очина и на восемь операций – алгоритм Паеса [7].

Необходимо заметить, что для процессоров семейства x86-64, содержащих 16 MMX(SSE2)-регистров, описанный алгоритм реализуется без дополнительных операций сохранения элементов в массивы при помощи хранения необходимых данных в регистрах.

4. Оптимизированный алгоритм медианной фильтрации на основе неполных сортирующих сетей

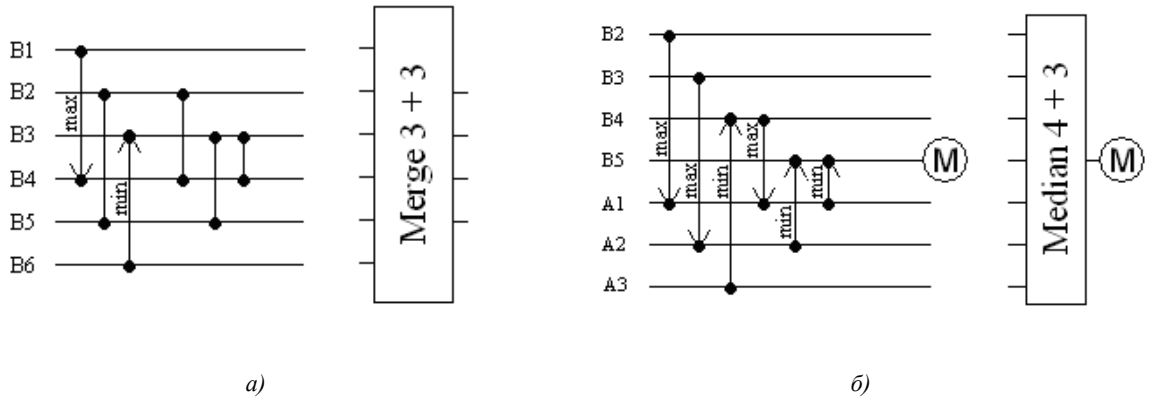
В статье Колта [12] алгоритм медианной фильтрации описан в терминах экстремумов. Компаратор, как известно, упорядочивает пару элементов. Будем называть термином «экстремум» операцию поиска минимума либо максимума пары элементов [12]. Тогда компаратор можно реализовать через две операции поиска минимума и максимума. Обозначим эти операции как MIN и MAX. В описанном выше алгоритме на основе неполных сортирующих сетей некоторые компараторы можно разбить на операции MIN и MAX. При этом появляется возможность удалить некоторые ненужные операции поиска экстремумов и получить более эффективный алгоритм, так как операция MIN или MAX реализуется на MMX или SSE2 проще, чем операция сравнения-обмена (компаратор). Однако следует помнить об ограничениях в реализации сортирующих сетей при помощи операций MIN и MAX, описанных в разд. 2 настоящей статьи.

Рассмотрим сортирующую сеть слияния двух упорядоченных троек элементов (см. рис. 2, *в*). Если рассмотреть первый слева компаратор, то становится очевидным, что достаточно найти максимум пары элементов при помощи этого компаратора, минимум пары в дальнейшем не используется. То же можно сказать и о последнем компараторе, который должен найти лишь минимум пары элементов. В этом случае можно заменить два указанных компаратора операциями MAX и MIN соответственно. Таким образом, если считать одну операцию сравнения-обмена за две операции поиска экстремума, то общее количество экстремумов в данной сортирующей сети сократиться с 12 (6 операций сравнения-обмена) до 10. Полученная сортирующая сеть на основе экстремумов показана на рис. 10, *а*. На ней линиями со стрелкой обозначены операции поиска экстремума. Стрелкой указан провод, в котором сохранится результат операции, при этом значение на втором проводе не определено.

Если рассмотреть сортирующую сеть поиска медианы на основе отсортированных четверки и тройки элементов, то аналогичным образом можно сократить количество операций поиска экстремума с 12 до 6. Полученная сортирующая сеть показана на рис. 10, *б*.

Заменив в неполной сортирующей сети, изображенной на рис. 5, элементы слияния отсортированных троек и получения медианы на модифицированные, получим уменьшение количества операций поиска экстремумов с 24 до 17. Необходимо заметить, что при замене компаратора в сортирующей сети операцией поиска экстремума выигрыш в производительности будет несколько больше, чем ожидаемый из простой оценки уменьшения количества операций, так как компаратор на SSE2 реализуется при помощи трех операций: поиска MIN, MAX и опе-

рации пересылки, а операция поиска экстремума – при помощи одной. Таким образом, операция поиска экстремума выполняется при помощи SEE2-команд более чем в два раза эффективнее, чем операция сравнения-обмена.

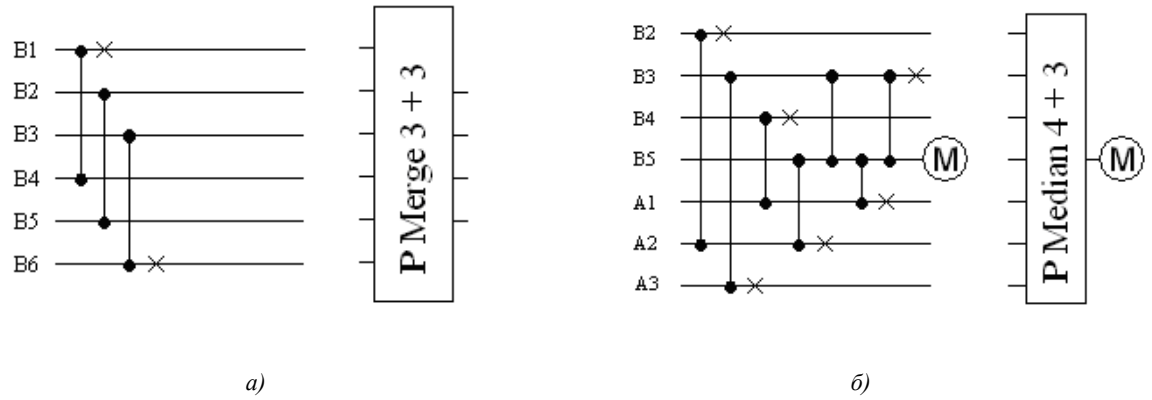


а) *б)*

Рис. 10. Неполные сортирующие сети на основе операций поиска экстремумов: *а)* элемент сети, выполняющий слияние отсортированных троек, и его графическое обозначение; *б)* элемент сети, выполняющий поиск медианы на основе отсортированных четверки и тройки, и его графическое обозначение

Полученная сортирующая сеть содержит 17 экстремумов, что на один экстремум меньше, чем в аналогичном алгоритме Колта. Заметим, что подобного количества операций поиска экстремумов алгоритм Колта достигает с помощью дополнительной оптимизации, описанной в статье [12]. Для этого необходимо применить сортировку двух пересекающихся столбцов-троек одновременно за 10 операций поиска экстремумов вместо 12 при раздельной сортировке столбцов.

В связи с тем что для эффективного поиска медианы при помощи отсортированных четверки и тройки элементов окна фильтра нет необходимости полностью сортировать четверку в процессе слияния отсортированных троек (см. рис. 2, *в*), сортирующую сеть поиска медиан двух соседних окон можно усовершенствовать. Модифицированные элементы неполного слияния двух отсортированных троек и поиска медианы изображены на рис. 11.



а) *б)*

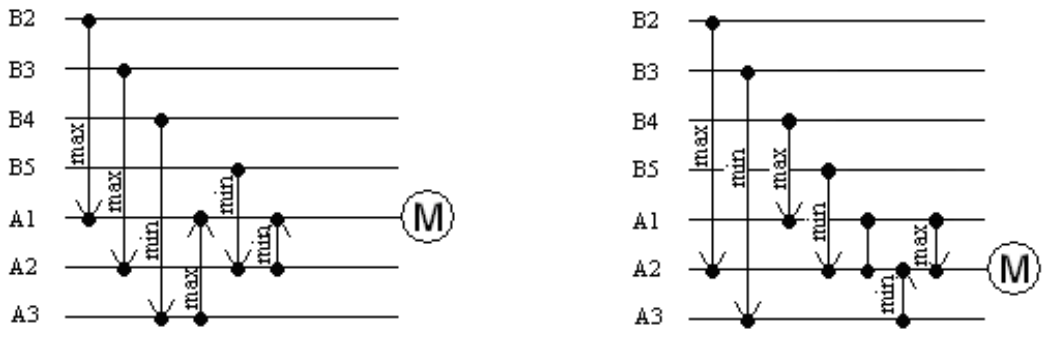
Рис. 11. Модифицированные элементы слияния отсортированных троек и поиска медианы: *а)* элемент сети, выполняющий неполное слияние отсортированных троек, и его графическое обозначение; *б)* элемент сети, выполняющий поиск медианы на основе неполностью отсортированной четверки и отсортированной тройки, и его графическое обозначение

Рассмотрим сеть, изображенную на рис. 11, *а*. Отличие данной сети от предыдущей (см. рис. 2, *в*) состоит в том, что она неполная. Результатом ее выполнения являются четыре элемента, связанные следующими соотношениями: B2<B3, B4<B5, B2<B5.

Рассмотрим сортирующую сеть, изображенную на рис. 11, *б*, и по порядку все компараторы этой сети. Элементы сети, которые отсеиваются после некоторой операции сравнения-обмена, так как не могут быть медианой, помечаются на изображении крестиками. После выполнения первой операции сравнения-обмена элемент на проводе B2 не может быть медианой, так как он является минимумом пяти элементов (B2<B3, B2<B5, B2<A2, B2<A3). После выполнения второй операции сравнения-обмена элемент на проводе A3 не может быть медианой как максимум пяти элементов (A3>A2, A3>A1, A3>B3, A3>B2). В связи с тем что были отсеяны два элемента из семи – один больше и один меньше медианы, любой элемент, являющийся максимумом или минимумом четырех элементов, если не рассматривать вычеркнутые, также не может быть медианой. Заметим, что после выполнения первых двух компараторов порядок элементов A1, A2 не нарушился, так как в провод A2 мог попасть только элемент, больший, чем A2, а следовательно, больший, чем A1. После выполнения третьей операции сравнения-обмена элемент B4 не может быть медианой как минимум четырех элементов: B4, B5, A1, A2. После выполнения четвертой операции сравнения-обмена элемент A2 не может быть медианой как максимум четырех элементов: A1, A2, B4, B5. Оставшиеся три элемента сортируются, при этом средний элемент будет медианой.

Таким образом, применяя модификацию элементов сети, изображенной на рис. 11, получаем сеть, требующую $(3+3+3+7\times 2)/2 = 11{,}5$ операций сравнения-обмена на одну медиану. Если преобразовать полученную сеть в сортирующую сеть на основе операций поиска экстремумов, то такая сеть будет содержать $(6+6+4+8\times 2)/2 = 16$ операций поиска экстремумов на одну медиану. Алгоритм на основе описанной сортирующей сети превосходит алгоритм Кучеренко – Очина на 1,5 операции сравнения-обмена и алгоритм Колта на одну операцию поиска экстремума.

Применение операций MIN и MAX для поиска медиан окон фильтра позволяет выполнить усовершенствование, невозможное при использовании операций сравнения-обмена, а именно поиск медианы посредством операции слияния отсортированных четверки и тройки без сохранения и загрузки отсортированной четверки в отдельном массиве. Сортирующая сеть, позволяющая выполнить эту операцию, изображена на рис. 12, *а*. Заметим, что в процессе поиска медианы все промежуточные значения помещаются лишь в провода сортирующей сети, которые исходно содержат отсортированную тройку. Поэтому значения отсортированной четверки не искажаются и могут быть использованы далее при поиске второй медианы. Сортирующая сеть слияния (см. рис. 11, *б*) может быть преобразована таким образом, что позволит находить медиану без предварительного сохранения четверки элементов (рис. 12, *б*) аналогично сети, изображенной на рис. 12, *а*, что повысит быстродействие алгоритма.



а) *б)*

Рис. 12. Неполные сортирующие сети поиска медианы при сохранении отсортированной четверки в верхних проводах сети: *а)* сеть поиска медианы на основе отсортированных четверки и тройки; *б)* сеть поиска медианы на основе неполностью отсортированной четверки и отсортированной тройки

В статье Колта [12] приведено еще одно возможное усовершенствование алгоритма медианной фильтрации. Если использовать пересечение сортируемых столбцов-троек и выполнять сортировку сразу двух столбцов за пять операций сравнения-обмена в отличие от шести операций при отдельной сортировке столбцов, то количество экстремумов в алгоритме Колта уменьшается с 18 до 17. Применяя такую же оптимизацию в описанных в данной статье алгоритмах, можно уменьшить количество элементов сети. Для этого требуется начинать сортировку с двух средних элементов, по которым пересекаются столбцы, а затем сортировать оставшиеся элементы столбцов. Однако подобный алгоритм сложно реализовать при помощи MMX- и SSE2-команд.

Для сравнительной оценки описанных в данной статье алгоритмов были реализованы одни из лучших известных на данный момент алгоритмов медианной фильтрации для окна 3×3 на основе сортирующих сетей: алгоритм Кучеренко – Очина, алгоритм Паеса для сортирующих сетей на основе операции сравнения-обмена, алгоритм Колта в нескольких модификациях для сортирующих сетей на основе операций поиска экстремумов. Необходимо заметить, что алгоритм Кучеренко – Очина исходно разрабатывался для аппаратной реализации и его реализация при помощи SIMD не описывалась в оригинальной статье. То же самое можно сказать и об алгоритме Паеса. Реализация этих алгоритмов при помощи SIMD выполнялась по схеме, аналогичной описанной для алгоритмов, которые предлагаются в данной статье. Алгоритм Колта нельзя реализовать без некоторых модификаций при помощи MMX- или SSE2-команд в силу его ориентации на команды AltiVec. Для реализации недеструктивной операции поиска экстремума на MMX или SSE2 необходима дополнительная операция пересылки данных из регистра в регистр. Очевидно, при использовании такой реализации операции поиска экстремума алгоритм Колта будет содержать излишние пересылки для операций, в которых не требуется недеструктивность. Реализация любого компаратора сортирующей сети как пара операций поиска экстремума также будет содержать две пересылки, тогда как реализация компаратора на SSE2 содержит одну пересылку данных (в AltiVec оба варианта реализации идентичны – это две операции MIN и MAX без пересылок). Поэтому применение в некоторых местах компараторов, а в некоторых – операций поиска экстремумов более эффективно при реализации на SSE2. Необходимо заметить, что не во всех системах компаратор реализуется через операции MIN и MAX, поэтому иногда он бывает эффективнее, чем две последовательные операции MIN и MAX. Иногда скорость выполнения компаратора сопоставима со скоростью вычисления экстремума, например в табличной реализации компаратора при выполнении на процессоре. В таких случаях выгоднее применять алгоритмы с минимальным количеством компараторов, а не операций поиска экстремумов.

Исходя из сказанного выше, реализация алгоритма Колта на SSE2 была усовершенствована с помощью двух различных оптимизационных подходов:

– использования деструктивных операций поиска экстремума там, где это возможно;

– объединения пар операций поиска MIN и MAX в компараторы, где это возможно.

Подобная реализация позволила минимизировать количество дополнительных операций пересылки и ускорить работу алгоритма на SSE2. Полученный алгоритм основывается на сортирующей сети, построенной при помощи как операций сравнения-обмена, так и деструктивных операций поиска экстремума. Также можно заметить, что описанная в статье Колта схема получения векторов, содержащих отсортированные столбцы окон фильтра, была реализована на SSE2 при помощи сохранения двух векторов в массив и операций чтения невыровненных данных. Данная реализация, вероятно, является менее эффективной, чем схема из алгоритма Кучеренко – Очина с сохранением отсортированных на предыдущих шагах алгоритма столбцов, хотя и требует такого же количества операций сравнения-обмена. Таким образом, можно сказать, что реализация алгоритма Колта на SSE2 является несколько усовершенствованной модификацией исходной реализации Колта, хотя и требует такого же количества операций поиска экстремума для вычисления медианы окна фильтра.

Время выполнения различных алгоритмов медианной фильтрации для полутонового изображения 1024×768 на процессоре Intel Pentium 4 с тактовой частотой 3 ГГц приведено в таблице.

Быстродействие различных алгоритмов медианной фильтрации
для полутонового изображения размером 1024×768, мс

Алгоритмы медианной фильтрации	Условные операторы	Табличный метод	MMX	SSE2
Алгоритм Хуанга	78,7	62,0	–	–
Алгоритм слияния упорядоченных столбцов	50,9	20,0	–	–
Сортирующая сеть, 21 компаратор	92,0	25,9	2,7	2,17
Сортирующая сеть, 19 компараторов	83,71	25,17	2,54	2,29
Сортирующая сеть, 13 компараторов (алгоритм Кучеренко – Очина)	55,88	15,41	2,84	1,63
Сортирующая сеть, 12 компараторов (алгоритм на основе неполных сортирующих сетей)	54,5	15,3	2,6	1,53
Сортирующая сеть, 11,5 компараторов	–	–	**2,63**	**1,49**
Модифицированный алгоритм Колта, 18 экстремумов, 5 пересылок	–	–	3,55	1,94
Модифицированный алгоритм Колта, 17 экстремумов (пересечение столбцов), 5,5 пересылок	–	–	2,10	1,46
Неполная сортирующая сеть, 17 экстремумов	–	–	2,29	1,40
Неполная сортирующая сеть, 16 экстремумов	–	–	2,28	1,35
Неполная сортирующая сеть, 17 экстремумов без сохранения отсортированной четверки	–	–	2,04	1,34
Неполная сортирующая сеть, 16 экстремумов без сохранения отсортированной четверки	–	–	**2,09**	**1,27**

В названии алгоритмов указано количество пересылок без учета команд пересылок сохранения и загрузки отсортированных троек и четверок, а также команд чтения данных из изображения и их сохранения. Для сравнения скорости работы описанных в данной статье алгоритмов с использованием SIMD-команд и алгоритмов медианной фильтрации, в которых подобные команды не используются, в таблице приведено быстродействие алгоритмов медианной фильтрации Хуанга [4] и слияния упорядоченных столбцов [5, 6].

Из таблицы видно, что наилучшую производительность среди алгоритмов, основанных на применении компараторов, имеет алгоритм на основе сортирующей сети, который использует для вычисления медианы 11,5 операций сравнения-обмена. Скорость вычислений при помощи программы, реализующей данный алгоритм, приблизительно на 8,5 % выше, чем у программной реализации лучшего известного на данный момент алгоритма на основе компараторов – алгоритма Кучеренко – Очина. Этот алгоритм был реализован при помощи MMX- и SSE2-команд, что не предусматривали его авторы. Если рассматривать алгоритмы на основе экстремумов, то наилучшим по быстродействию является алгоритм на основе сортирующей сети, использующий 16 экстремумов на вычисление одной медианы окна фильтра. Алгоритм превосходит по быстродействию один из лучших известных на данный момент алгоритмов на основе экстремумов – алгоритм Колта – приблизительно на 13 %. При этом необходимо заметить, что алгоритм Колта был оптимизирован для выполнения при помощи MMX- и SSE2-команд. Также очевидно значительное превосходство с точки зрения быстродействия (приблизительно на порядок) алгоритмов, применяющих MMX- и SSE2-команды, перед алгоритмами, не использующими их.

Необходимо заметить, что описанные выше алгоритмы медианной фильтрации могут быть реализованы посредством любых SIMD-команд, в том числе при помощи графических ускорителей (GPU).

Заключение

В статье предложены новые алгоритмы медианной фильтрации. Алгоритм на основе сортирующих сетей с операциями сравнения-обмена (компараторами) использует в среднем 11,5

таких операций на поиск одной медианы окна фильтра, что превосходит известные алгоритмы, основанные на применении компараторов, в том числе алгоритм Кучеренко – Очина. Алгоритм на основе сортирующих сетей с операциями поиска экстремумов использует в среднем 16 таких операций для поиска одной медианы окна фильтра, что также превосходит известные алгоритмы, в том числе алгоритм Колта. Проведен анализ различных неполных сортирующих сетей, допускающих удобную реализацию при помощи MMX- и SSE2-команд. Описана реализация этих алгоритмов при помощи MMX- и SSE2-команд, что значительно ускоряет их работу, позволяя вести поиск сразу нескольких медиан в параллельном режиме.

Рассмотрены также различные модификации известных алгоритмов медианной фильтрации на основе сортирующих сетей, в том числе алгоритма Кучеренко – Очина и алгоритма Колта, которые позволяют реализовать данные алгоритмы при помощи MMX- и SSE2-команд. Алгоритмы реализуются без применения условных операторов, что значительно повышает их производительность. Предложенные алгоритмы значительно превосходят по быстродействию известные алгоритмы медианной фильтрации, реализованные без применения MMX и SSE2, и позволяют производить медианную фильтрацию в режиме реального времени, применяя ее к широкому кругу задач обработки изображений.

Список литературы

1. Шапиро, Л. Компьютерное зрение / Л. Шапиро, Дж. Стокман. – М. : БИНОМ. Лаборатория знаний, 2006. – 752 с.

2. Залесский, Б.А. Отслеживание динамических объектов и их распознавание с помощью графовых алгоритмов / Б.А. Залесский, А.И. Кравчонок // Информатика. – 2006. – № 2 (10). – С. 17–26.

3. Залесский, Б.А. Отслеживание и распознавание движущихся объектов на основе их кластерного представления / Б.А. Залесский, А.И. Кравчонок // Информатика. – 2004. – № 2. – С. 68–78.

4. Быстрые алгоритмы в цифровой обработке изображений / Т.С. Хуанг [и др.]; под общ. ред. Т.С. Хуанга. – М. : Радио и связь, 1984. – 220 с.

5. Kopp, M. Efficient 3×3 Median Filter Computations / M. Kopp // Machine Graphics & Vision. – 1995. – Vol. 4, № 1/2. – P. 79–82.

6. Kravchonok, A. An Algorithm for Median Filtering on the Basis of Merging of Ordered Columns / A. Kravchonok, B. Zalesky, P. Lukashevich // Pattern Recognition and Image Analysis. – 2007. – Vol. 17, № 3. – P. 402–407.

7. Paeth, A. Median Finding of a 3×3 Grid / A. Paeth, W. Alan // Graphics Gems I. – Academic Press, 1990. – P. 171–175.

8. Кравчонок, А.И. Алгоритмы медианной фильтрации с окном 3×3 на основе неполной сортировки прямым выбором / А.И. Кравчонок // Информатика. – 2008. – № 1 (17). – С. 38–46.

9. Кучеренко, К.И. Двумерные медианные фильтры для обработки изображений / К.И. Кучеренко, Е.Ф. Очин // Зарубежная радиоэлектроника. – 1986. – № 6. – С. 50–61.

10. Кучеренко, К.И. Сортирующие сети двумерной медианной фильтрации полутоновых изображений / К.И. Кучеренко, Е.Ф. Очин // Радиотехника. – 1987. – № 7. – С. 36 – 38.

11. Кравчонок, А.И. Алгоритмы медианной фильтрации с окном 3×3 на основе неполных сортирующих сетей / А.И. Кравчонок // Информатика. – 2009. – № 1 (21). – С. 91–102.

12. Kolte, P. A Fast Median Filter Using AltiVec / P. Kolte, R. Smith, W. Su // International Conference on Computer Design, 1999 (ICCD '99). – Austin, TX, USA, 1999. – P. 384–391.

13. AltiVec extension to PowerPC accelerates media processing / K. Diefendorff [et al.] // IEEE Micro. – 2000. – Vol. 20, № 2. – P. 85–95.

14. Касперски, К. Техника оптимизации программ. Эффективное использование памяти / К. Касперски. – СПб. : БХВ – Петербург, 2003. – 464 с.

15. Магда, Ю.С. Аппаратное обеспечение и эффективное программирование / Ю.С. Магда. – СПб. : Питер, 2007. – 352 с.

16. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда. – СПб. : Питер, 2006. – 410 с.

17. Магда, Ю.С. Использование ассемблера для оптимизации программ на C++ / Ю.С. Магда. – СПб. : БХВ–Петербург, 2004. – 496 с.

18. Юров, В.И. Assembler. Практикум / В.И. Юров. – 2-е изд. – СПб. : Питер, 2006. – 399 с.

19. Зубков, С.В. Assembler для DOS, Windows и UNIX / С.В. Зубков. – 3-е изд. – М. : ДМК Пресс; СПб. : Питер, 2006. – 608 с.

20. Using MMX™ Instructions to Implement Median Filter // Intel® Software Network [Electronic resource]. – Mode of access : ftp://download.intel.com/ids/mmx/MMX_App_Filter_Median.pdf. – Date of access : 04.11.09.

21. Кнут, Д. Искусство программирования. Т. 3. Сортировка и поиск / Д. Кнут. – М. : Издательский дом «Вильямс», 2005. – 824 с.

22. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – 2-е изд. – М. : Издательский дом «Вильямс», 2007. – 1296 с.

23. Седжвик, Р. Фундаментальные алгоритмы на С. Анализ. Структуры данных. Сортировка. Поиск / Р. Седжвик ; пер. с англ. – СПб. : ООО «Диа Софт ЮП», 2003. – 672 с.

24. Миллер, Р. Последовательные и параллельные алгоритмы / Р. Миллер, Л. Боксер. – М. : БИНОМ. Лаборатория знаний, 2006. – 406 с.

25. Vasicek, Z. Novel Hardware Implementation of Adaptive Median Filters / Z. Vasicek, L. Sekanina // IEEE Workshop «Design and Diagnostics of Electronic Circuits and Systems» (DDECS'08). – Bratislava, 2008. – P. 1–6.

26. Open Computer Vision Library // Sourceforge.net. Open Source Software [Electronic resource]. – 1999. – Mode of access : http://sourceforge.net/projects/opencvlibrary/. – Date of access : 28.01.2009.

Поступила 04.11.09

Объединенный институт проблем
информатики НАН Беларуси,
Минск, Сурганова, 6
e-mail: alpha_storm@mail.ru

A.I. Kravchonok

ALGORITHMS FOR MEDIAN FILTERING WITH 3×3 WINDOW ON THE BASIS OF MMX AND SSE2 INSTRUCTIONS OF x86 PROCESSOR FAMILY

Fast algorithms of median filtering with the 3×3 window on the personal computer with the help of MMX- and SSE2-commands are suggested. Various incomplete sorting networks for searching a median of the filter window, supposing the implementation with the help of MMX- and SSE2-commands are described. These algorithms allow substantially accelerate the performance of median filtering when applied to digital images. It does possible the application of median filtering in real time in the image processing software.