РЕАЛИЗАЦИЯ ДВУМЕРНЫХ БИХ-ФИЛЬТРОВ НА ГПУ С ИСПОЛЬЗОВАНИЕМ МЕТОДА НЕЖЕСТКОГО РАЗМЕЩЕНИЯ В МОДЕЛИ ОДНОРОДНЫХ РЕКУРРЕНТНЫХ УРАВНЕНИЙ

А.В. Никоноров, В.А. Фурсов, П.Ю. Якимов

В данной работе рассматривается ГПУ реализация двумерного фильтра с бесконечной импульсной характеристикой. Предложена декомпозиция БИХ-фильтра в форму, которая позволяет применить модель однородных рекуррентных уравнений для параллельной реализации. Представленный подход нежесткого размещения позволяет получить эффективную реализацию в одной функции-ядре на ГПУ. В статье приведены теоретические и экспериментальные оценки производительности предложенного алгоритма нежесткого размещения.

Введение. Одномерные и двумерные БИХ фильтры достаточно широко используются в задачах обработки изображений [1]. В работе [2] описано применение двумерного (2D) БИХ-фильтра для коррекции изображений, в системе дистанционного зондирования Земли (ДЗЗ). В указанной работе для решения задачи идентификации параметров фильтра предложено использовать малые тестовые фрагменты, которые формируются из искаженного изображения с использованием априорной информации о геометрической форме регистрируемых объектов. При этом двумерный БИХ-фильтр, обеспечивающий компенсацию сильных искажений с использованием опорной области небольших размеров, строится в виде параллельного соединения четырех физически реализуемых фильтров, которые используют соответствующий квадрант в опорной области. На рисунке 1 показаны результаты обработки размытого изображения с использованием двумерного БИХ-фильтра, построенного по технологии, описанной в [2].

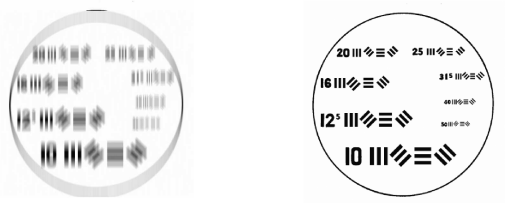


Рис. 1. Оригинальное и обработанное изображения

Представляется целесообразным использовать ГПУ-реализации БИХ-фильтров для повышения производительности обработки больших изображений в системах ДЗЗ и обработки видео в реальном времени. Вопросы повышения эффективности обработки сигналов с помощью одномерных БИХ-фильтров за счет распараллеливания достаточно хорошо изучены. Параллельная реализация двумерных БИХ-фильтров представляет собой частный случай модели вложенных циклов. Эта задача обычно решается с помощью, так называемой *модели многогранников (Polytope Model)* [3-5]. В общем случае подход к распараллеливанию вложенных циклов основывается на *теореме Лэмпорта о гиперплоскостях* [7] и модели распараллеливания рекуррентных уравнений [3].

В настоящей работе рассматривается реализация 2D БИХ-фильтра на ГПУ на основе модифицированной модели многогранников, приводятся следующие основные результаты исследований:

- Предложена структура БИХ-фильтра, максимально приспособленная для ГПУ реализации. Предложенная структура предполагает декомпозицию БИХ-фильтра на четыре квадрантных фильтра. Такой подход позволяет обеспечить физическую реализуемость. Далее каждый из квадрантных фильтров разделяется на КИХ и БИХ составляющие.
- КИХ составляющая вычисляется по схеме параллельной редукции [17], БИХ составляющая фильтра, следуя [16], представляет собой систему а*ффинных рекуррентных уравнений*. В настоящей работе показано, что БИХ составляющая может быть преобразована к системе однородных рекуррентных уравнений [16].
- При распараллеливании рекуррентных алгоритмов для ГПУ с использованием *модели многогранников* [18], используется последовательный запуск CUDA ядер с ЦПУ. Таким образом, исходная структура вложенных циклов преобразуется в плоский цикл запуска ядер на ГПУ. В настоящей работе предлагается модификация алгоритма размещения вычислительных элементов в модели многогранников, позволяющая реализовать рекуррентную обработку в

рамках одного CUDA ядра. Показано, что такой подход более эффективен для ГПУ реализации 2D БИХ-фильтра.

Нежесткое размещение в одной функции-ядре на CUDA. В настоящей статье рассматривается основанная на модели многогранников [3] параллельная реализация описанного в [2] квадрантного фильтра на ГПУ. В этом разделе рассмотрены необходимые сведения о модели и основных характеристиках архитектуры графического процессора. Далее описывается предложенный метод нежесткого размещения для эффективной реализации БИХ-фильтра на ГПУ в рамках модели многогранников.

Построение БИХ-фильтра с использованием модели многогранников. В модели многогранников [3] и [4] алгоритм с одним присвоением описывается системой рекуррентных уравнений, каждое из которых представляется в следующей форме:

$$\forall x \in IS : v[f()x] = F_v(w[g(x)],...), \qquad (1)$$

где [.] – это оператор индексации. Ограничения, накладываемые на форму *индексных функций* f и g и на форму и свойства *индексного пространства IS*, могут различаться. Зависимости в индексных функциях модели (1) формируют граф отношений в пространстве индексов.

В данной работе мы используем два важных частных случая общей формы (1).

1. *Однородные рекуррентные уравнения* (*ОРУ*). В этой основной форме модели многогранников пространство индексов *IS* является пересечением многогранников в Z^r, где *r*-число рекуррентных соотношений (или вложенных циклов), а индексные функции $f(x)$ и $g(x)$ имеют вид $x + d$, где r – некоторый постоянный вектор [4].
2. *Аффинные рекуррентные уравнения* (*АРУ*). Для этой формы общего соотношения (1) функции индекса имеют более общий вид – $Ax + d$, где A - постоянная матрица а r - постоянный вектор. В такой форме, при невырожденной матрице A, становится возможным разделение данных.

Пусть теперь индексное пространство *IS* - это *r*-мерный многогранник. Рассмотрим граф отношений (*IS, E*), где *E* – это набор ребер, определяющих зависимости.

Функция $t:IS \rightarrow Z$ называется *планированием*, если она сохраняет зависимости между данными:

$$\forall x, x' : x, x' \in IS \wedge (x, x') \in E : t(x) < t(x') \cdot \qquad (2)$$

Функция $a:IS \rightarrow Z^{r-1}$ называется *размещением* по отношению к планированию t, если каждый процесс, описываемый этой функцией, является внутренне последовательным:

$$\forall x, x' : x, x' \in IS : t(x) < t(x') \Rightarrow a(x) \neq a(x') \cdot \qquad (3)$$

Планирование разделяет индексное пространство на параллельные гиперплоскости, т. е. подпространства размерности на единицу меньше, чем у индексного пространства. Каждая гиперплоскость представляет собой один срез времени, т.е. множество точек пространства индексов вычисления в которых могут быть выполнены одновременно. Гиперплоскости планирования не должны быть параллельны ребрам графа отношений. На основе данного требования как правило и выполняется построение планирования [4].

Размещение сегментирует индексное пространство на параллельные линии, т.е. подпространства размерности 1. Каждая такая линия содержит точки, обрабатываемые конкретным процессором. Каждому процессору соответствует одна такая линия.

Чтобы привести БИХ-фильтр к форме (1), необходимо проделать несколько преобразований. Выражение для вычисления БИХ фильтра может быть представлено следующим образом:

$$x_1(n_1, n_2) = \sum_{r_1} \sum_{r_2} a(n_1 - r_1, n_2 - r_2) x(r_1, r_2), \quad (4)$$

$$y_1(n_1, n_2) = \sum_{\substack{k_1 \\ (k_1 k_2 \neq n_1 n_2)}} \sum_{k_2} b(n_1 - k_1, n_2 - k_2) y(k_1, k_2), \qquad (5)$$

$$y(n_1, n_2) = y_1(n_1, n_2) + x_1(n_1, n_2) \cdot \quad (6)$$

Выражение (4) – это КИХ компонента БИХ-фильтра. Значение (2) можно вычислить, используя параллельную редукцию [6]. Выражение (5) – это чисто рекурсивная часть БИХ-фильтра. Несложно заметить, что (5) и (6) – это система АРУ. В самом деле, (5) и (6) могут быть представлены в виде вложенных циклов. Выражения $n_1 - k_1$ и $n_2 - k_2$ используются при индексации в выражении для y_1. Таким образом выражение (5) является АРУ.

Тем не менее, два внутренних цикла, для вычисления (5), можно разложить в прямую сумму. В этом случае выражения (5) и (6) принимают следующий вид ОРУ:

Листинг 1 - ОРУ реализация БИХ-фильтра:

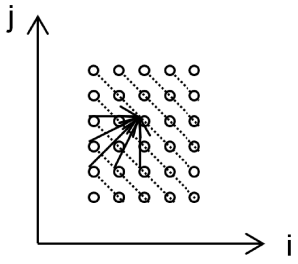
```
1 for(i = 0; i< N; i++ ){
2  for(j = 0; j < N; j++){
3   y1[i,j] = b[1,1]*y1[i - 1,j - 1] + …      + b[1,1]*y1[i - m, j – m];
4    y[i,j] = y1[i,j] + x1[i,j];
5  }
6 }
```

Строка 3 Листинга 1 состоит из (m^2–1) членов. Таким образом граф зависимостей содержит (m^2–1) ребер в каждой вершине. Такое суммирование может быть реализовано на ГПУ с помощью параллельной редукции [6].

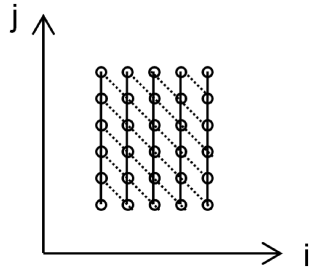
Граф отношений, для $m = 3$ показан на Рисунке 2.1. Даже для $m=3$ граф достаточно громоздкий, поэтому показана только одна вершину графа и восемь соответствующих ей ребер. Угол между ребрами и осью i - это значение в пределах от 180 до 270 градусов. Этот диапазон плотно заполнен на высоких значениях m.

Согласно [3] и [4], идея построения планирования и размещения имеет простой геометрический смысл, но ее алгебраическое описание гораздо сложнее. Согласно [4] гиперплоскости планирования должны быть не параллельны всем ребрам графа зависимостей. Мы будем использовать планирование определяемое функцией планирования [1 -1], которая удовлетворяет этим требованиям. Плоскости, соответствующие этой функции, показаны на Рисунке 2.1 пунктиром.

В соответствие с требованием о непараллельности планирования и размещения [4] в настоящей работе предлагается размещение [0 1] (Рисунок 2.2), которое является параллельным к оси j.



2.1 Граф отношений и планирование [1 -1]



2.2 Планирование и размещение[0 1]

Рис. 2. Граф зависимостей, планирование и размещение для БИХ-фильтра

Архитектурные особенности ГПУ. Особенностью архитектуры ГПУ является то, что вычисления выполняются блоками потоков, в рамках каждого блока потоки взаимодействуют между собой посредством разделяемой памяти. Связь между блоками реализуется только посредством глобальной памяти, средства взаимодействия и синхронизации крайне ограничены.

Взаимодействие между потоками одного блока является максимально быстрым, поскольку осуществляется посредством разделяемой памяти, расположенной непосредственно на ядре ГПУ, и имеет механизмы синхронизации. Взаимодействие между потоками разных блоков возможно лишь через глобальную память, что гораздо медленнее. Специальные примитивы синхронизация потоков разных блоков отсутствуют, синхронизация с данном случае может быть реализована через использование атомарных операций при работе с глобальной памятью.

Метод нежесткого размещения в модели многогранников. В работе [5] рассматривается технология автоматизированного построения CUDA-кода с распараллеливанием циклов. Основной цикл выполняется на ЦПУ, а вложенные циклы – на ГПУ. В Листинге 2 представлен фрагмент кода, который демонстрирует такой гибридный подход. Для краткости содержимое CUDA-ядра опущено.

Листинг 2. Многоядерный подход.

```
for (T=...) {
  for (x 2CA(T)) buffer[sA(x,T)] = A[x];
  copy to device(buffer);
  dim3 blocks(Pu(T)-Pl(T)+1,1), threads(512,1,1);
  unsigned sharedSize = maxSharedSize A(T) + ...;
  kernel0<<<blocks,threads,sharedSize>>>(buffer, T, n);
  copy from device(buffer);
}
```

Как видно из листинга, для каждой плоскости планирования, соответствующей некоторому срезу времени, запускается новое ядро с различным числом блоков. Каждый блок отвечает одному процессорному элементу из размещения (3). Обрабатываемые данные при таком подходе расположены в глобальной памяти ГПУ, и на каждом шаге по времени копируются в разделяемую память каждого блока. Для нашей задачи, это означает $(m^2 - 1)$ чтений из глобальной памяти на каждом шаге. При больших значениях m это приведет к значительному замедлению выполнения алгоритма. Кроме того, дополнительные задержки возникают при вызове новых ядер.

В настоящей работе предлагается реализовать обработку полностью на ГПУ. При этим, каждый блок выполняет роль процессорного элемента. Различное количество блоков на разных временных срезах обеспечивается за счет использования блокировок в глобальной памяти ГПУ. На каждом шаге требуется в

среднем *(m + 4)* операций с глобальной памятью вместо *(m^2 – 1)*. Нити каждого блока подсчитывают сумму из 3 строки Листинга 1.

Обозначим через *G* количество параллельных процессорных элементов, то есть количество процессорных ядер на ГПУ. В размещении на Рисунке 2.2 константа *G* определяет максимальное число линий графа размещения. Далее принимаем по умолчанию такое размещение и будем рассматривать только полосы ширины *G*, т.е. пространство итераций $Z^g * Z^n$.

Таким образом, некоторый блок может начинать вычисление элемента $y_1(i_0,j_0)$, только если все элементы ($y_1(i,j)$: $i<i_0 \wedge j<j_0$) уже вычислены. Для некоторого размещении каждый процессорный элемент «закреплен» за некоторой линией в пространстве итераций. Для размещения [0 1] это означает, что некоторый блок закреплен за некоторым столбцом. Таким образом, для каждого столбца $i_0 = const$. Можно показать, что сформулированное требование эквивалентно следующему: вычисление элемента $y_1(i_0,j_0)$ возможно, если

$$j*(i_0-1) \geq j_0, \quad (7)$$

$$i_0 \equiv const, \quad (8)$$

где *j*(i)* – это максимальный индекс обработанного элемента в *i*-том столбце.

В современной ГПУ системе очередность запуска блоков непредсказуема. Таким образом, блок, закрепленный за некоторым столбцом и ожидающий выполнения условий (7)-(8) для фиксированного i_0 может оказаться в бесконечной блокировке.

Чтобы избежать блокировок, предлагается отказаться от жесткого закрепления обрабатываемого столбца за блоком. При выборе столбца обрабатываемого блоком предлагается следующее правило – блок обрабатывает ближайший столбец, для которого выполняется требование (7). Таким образом, требования принимают следующий вид: вычисление элемента $y_1(i_1,j_1)$ возможно, если

$$j*(i_1-1) \geq j_1, \quad (9)$$

$$i_1 = \arg\min_{i \notin L} |i_0 - i|, \quad (10)$$

здесь *L* – множество столбцов обрабатываемых в настоящий момент. Условие в (10) гарантирует, что никакие два блока не будут обрабатывать одновременно один столбец. Таким образом, в каждый момент времени должны обрабатываться все столбцы.

Алгоритм, соответствующий условиям (7), (8), реализует планирование и размещение с использованием классической моделью многогранников. Алгоритм, соответствующий условиям (9), (10), будем называть моделью многранников с *нежестким размещением*.

Принципиальный код, реализующий нежесткое размещение в рамках модели многогранников, приведен в листинге 3.

Листинг 3. БИХ-фильтр с нежестким размещением

```
curRow = 0;
while(curRow < rowCount){
  curCol = blockIndex;
  while(columns[curCol - 1] < curRow
        && curCol > 0 && curRow > 0)
  {
    curCol = curCol – 1;
    curRow = columns[curCol] + 1;
  }
  lockState = atomicCAS(&locks[curCol], 0, 1);
  if(lockState) continue;

  doFilter(y1, curRow, curCol);
  columns[blockIndex] = curRow;
  atomicExch(&locks[curCol], 0);
  curRow = curRow + 1;
}
```

В листинге вычисления, выполняемые параллельно потоками блока, т.е. непосредственно вычисление $y_1(i_1,j_1)$ (5), обозначены функцией doFilter(). Массив *columns* реализует функцию $j^*(i)$. Множество *L* реализуется массивом блокировок *locks*.

Для обеспечения когерентности при работе с массивом блокировок в глобальной памяти - *locks*, используются атомарные операции CUDA/OpenCL. Причем для выполнения операции проверки и захвата блокировки столбца в одной транзакции целесообразно использовать функцию atomicCAS() [8], а для освобождения блокировки – atomicExch() [8]. Листинг 3 отражает лишь основную идею алгоритма, без таких частных моментов как, например, обработка начальных и граничных условий.

Для предлагаемого алгоритма нежесткого размещения в модели многогранников справедливо следующее Утверждение. Рекуррентный алгоритм с нежестким размещением для вычислительной модели CUDA, обладает следующими свойствами:

a) ни один блок алгоритма не попадает в состояние бесконечной блокировки;
b) после завершения работы алгоритма все элементы изображения обработаны.

Параллельная редукция и работа с shared памятью. Вычисление суммы в строке 3 Листинга 3, а также вычисление КИХ части фильтра согласно (4) выполняется по схеме редукции [6]. Сложность такого алгоритма составляет $[\log_2(m^2)]+1$, где m^2 – это количество элементов, которые требуется сложить.

В предлагаемом алгоритме нежесткого размещения блок с номером *N* должен ждать, пока блок с номером *N-1* закончит вычисления. Таким образом, для последнего блока, а, следовательно, и для всего алгоритма количество операций можно оценить как:

$$O = (2N-1)O_b = (2N-1)(\lceil 2\log_2 m \rceil + 1) \quad (11)$$

Рассмотрим теперь количество обращений к глобальной памяти ГПУ. Такие операции являются наиболее затратными по времени выполнения на ГПУ. Количество этих операций – m^2. При сдвиге блока вдоль столбца без изменения его номера блоку требуется прочесть *(m-1)* элементов массива y_1 и один элемент массива x_1. В этом случае требуется по две операции чтения/записи для работы с массивами *locks* и *columns*.

Операции считывания строк изображения выполняются нитями параллельно, поэтому, согласно алгоритму с множественным вызовом функций-ядер, понадобится *m* операций, чтобы считать область соответствующую окну фильтра. В алгоритме с одной функцией-ядром присутствует фиксированное количество операций с глобальной памятью – одна операция чтения изображения и 6 операций чтения/записи в массивы *locks*. Такое количество операций необходимо в случае, если блок не меняет номер обрабатываемого столбца данных, в случае же, когда блок меняет обрабатываемый столбец с i_0 на i_1, необходимо дополнительно прочитать i_0-i_1 столбцов, т.е. среднее количество столбцов, на которое должен сдвинуться блок.

Среднее количество операций с памятью может быть оценено как:

$$O_f = 5(1-p_l) + p_l c_l m ,$$

где p_l – это вероятность того, что какой-либо блок сменит свой столбец, а c_l – это среднее число столбцов, которые сменяются блоками.

Алгоритм жесткого размещения [5], использующий несколько функций-ядер, требует *m* операций с глобальной памятью на блок на каждой итерации. Таким образом, разница в колличестве требуемых операций с глобальной памятью между нежестким и фиксированным размещением можно оценить как:

$$d = m - 7 - p_l(c_l m - 5) \quad (12)$$

Значения p_l и c_l требуют экспериментальной оценки. Значение *d* меньше нуля, если $m<5$, поэтому использование нежесткого размещение оправдано, только для фильтров с $m>5$.

Результаты экспериментов. Экспериментальные исследования проводились для задачи повышения качества изображения на основе двумерного БИХ фильтра с окном размером *m*m* на квадратном изображении размера 2048*2048 пикселей.

На Рисунке 3 показана зависимость времени вычислений от размера окна для алгоритма фиксированного размещения [5] и предлагаемого алгоритма нежесткого размещения. Предлагаемый алгоритм является эффективным для значений *m* больших, чем 7, что очень близко к теоретической оценке (12). Для фильтра с размером окна в 31 пиксель предложенный алгоритм примерно в 2,4 раза быстрее. Результаты были получены с использованием ГПУ Nvidia GF 540m. Характер зависимости для алгоритма нежесткого размещения близок к константной зависимости, а при фиксированном размещении зависимость близка к линейной.

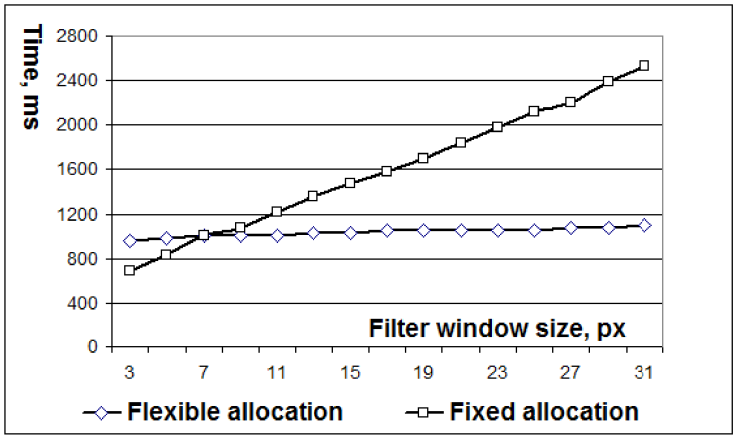


Рис. 3. Зависимость времени выполнения от размера опорного окна

Заключение. Модификация 2D БИХ-фильтров, представленная в данной работе, позволяет использовать модель однородных рекуррентных уравнений для описания фильтра. Предлагаемое нежесткое размещение для модели многогранников позволяет реализовать параллельные циклы для 2D-IIR фильтра в одной функции-ядра CUDA. В работе представлены оценки сложности и количества операций с глобальной памятью для предлагаемого одноядерного алгоритма нежесткого размещения.

Теоретическая оценка числа операций с глобальной памятью для нежесткого размещения меньше, чем для фиксированного размещения. Эта оценка подтверждается экспериментами. Оценка зависит от некоторых характеристик ГПУ. Детальное исследование этих характеристик является задачей для дальнейшего изучения. Интересно также исследовать применимость предлагаемого подхода нежесткого размещения для других вычислительных задач, отличных от задачи 2D БИХ-фильтрации.

Благодарности. Работа выполнена при поддержке Министерства образования и науки (ГК № 07.514.11.4105) и РФФИ (проекты № 11-07-12051-офи-м, № 12-07-00581-а).

ЛИТЕРАТУРА:

1. Soifer V.A. , Computer Image Processing, Part II: Methods and algorithms / VDM Verlag, 2009, 584 p.
2. Fursov V., Nikonorov A., Yakimov P., Stable IIR Filters Identification based on the Genetic Algorithms / 8th Open German-Russain Workshop "Pattern recognition and image understanding", workshop proceedings, November 21-26, 2011 pp.71-74.
3. Lengauer C., Loop Parallelization in the Polytope Model / Proceedings of the 4th International Conference on Concurrency Theory, 1993, pp. 398—416.
4. Rajopadhye S. V., Fujimoto R. M., Synthesizing systolic arrays from recurrence equations / Parallel Computing, 14(2) pp. 163-189, June 1990.
5. Baghdadi S., Groblinger A., Cohen A., Putting Automatic Polyhedral Compilation for GPGPU to Work / Proceedings of the 15th Workshop on Compilers for Parallel Computers, 2010.
6. McCool M.D., Structured Parallel Programming with Deterministic Patterns / HotPar ’10, 2nd USENIX Workshop on Hot Topics in Parallelism, 14-15 June 2010, Berkeley, CA.
7. Lamport L., The parallel execution of DO loops / ACM Communication, vol. 17 issue 2, 1974, pp. 83-93
8. NVIDIA CUDA C Best Practices Guide / Santa Clara, 2010, 75p.