МЕТОД НЕЖЁСТКОГО РАЗМЕЩЕНИЯ В МОДЕЛИ МНОГОГРАННИКОВ ДЛЯ ПОСТРОЕНИЯ ЭФФЕКТИВНЫХ АЛГОРИТМОВ ДВУМЕРНОЙ РЕКУРСИВНОЙ ОБРАБОТКИ ИЗОБРАЖЕНИЙ НА GPU

Никоноров А.В., Фурсов В.А., Якимов П.Ю.
Институт систем обработки изображений РАН,
Самарский государственный аэрокосмический университет имени академика С.П. Королёва (национальный исследовательский университет)

Аннотация

В настоящей работе рассматривается GPU-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 нежёсткого размещения позволяет получить эффективную реализацию в одной kernel-функции на GPU. В статье приведены теоретические и экспериментальные оценки производительности предложенного алгоритма нежёсткого размещения.

Ключевые слова: обработка изображений, модель многогранников, нежёсткое размещение, графические процессоры, CUDA, БИХ-фильтры.

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 искажённого 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 четырёх физически реализуемых фильтров, которые используют соответствующий квадрант в опорной области [9]. На рис. 1 показаны результаты обработки размытого изображения с использованием двумерного БИХ-фильтра, построенного по технологии, описанной в [2].

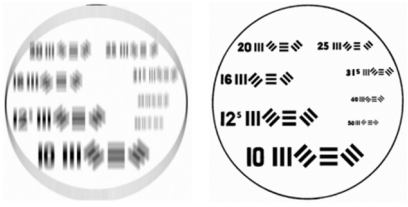


Рис. 1. Оригинальное и обработанное изображения

Представляется целесообразным использовать GPU-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 счёт 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, 4, 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 GPU на основе модифицированной модели многогранников, приводятся следующие основные результаты исследований:

- Предложена структура БИХ-фильтра, максимально приспособленная для GPU-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 [6], БИХ- составляющая фильтра, следуя [7], представляет собой систему *аффинных рекуррентных уравнений*. В настоящей работе показано, что БИХ- составляющая может быть преобразована к системе однородных рекуррентных уравнений [7]. Для полученной системы строится граф зависимостей, временное планирование и размещение вычислительных элементов [7], позволяющее построить эффективный параллельный алгоритм с учётом специфики архитектуры GPU.
- При распараллеливании рекуррентных алгоритмов для GPU с использованием *модели многогранников* [5] используется последовательный запуск CUDA ядер с CPU. Таким образом, исходная структура вложенных циклов преобразуется в плоский цикл запуска ядер на GPU. 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 GPU- реализации 2D БИХ-фильтра.

1. Нежёсткое размещение в одной функции-ядре на CUDA

В настоящей статье рассматривается основанная на модели многогранников [3] параллельная реализация описанного в [2] квадрантного фильтра на

GPU. В этом разделе рассмотрены необходимые сведения о модели и основных характеристиках архитектуры графического процессора. Далее описывается предложенный метод нежёсткого размещения для эффективной реализации БИХ-фильтра на GPU в рамках модели многогранников.

Построение БИХ-фильтра с использованием модели многогранников

Введём необходимые определения модели многогранников согласно работам [3] и [4]. В модели многогранников алгоритм с одним присвоением описывается системой рекуррентных уравнений, каждое из которых представляется в следующей форме:

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

где [.] – это оператор индексации, v и w – это индексированные переменные (массивы), F_v – функция с произвольным фиксированным количеством аргументов, аналогичных первому. 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 Zr, 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 рёбер, определяющих зависимости.

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

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

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

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

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 рёбрам 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:

$$x_1(n_1, n_2) = \sum_{r_1=0}^{m} \sum_{r_2=0}^{m} a(n_1 - r_1, n_2 - r_2) x(r_1, r_2), \quad (4)$$

$$y_1(n_1, n_2) = \sum_{k_1=1}^{m} \sum_{k_2=1}^{m} b(n_1 - k_1, n_2 - k_2) y(k_1, k_2), \quad (5)$$

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

Выражение (4) – это КИХ- компонента БИХ-фильтра. N – это размер 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, 2]*y1[i – 1, j – 2] + … + b[m, m]*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 $(m^2 - 1)$ рёбер в каждой вершине. Такое суммирование может быть реализовано на GPU с помощью параллельной редукции [6].

Граф отношений для $m = 3$ показан на рис. 2*а*. Даже для $m = 3$ граф достаточно громоздкий, поэтому показана только одна вершину графа и восемь соответствующих ей рёбер. Угол между рёбрами 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 её алгебраическое описание гораздо сложнее. Построим планирование для задачи БИХ- фильтра, используя граф отношений. Согласно [4] гиперплоско-

сти планирования должны быть не параллельны всем рёбрам графа зависимостей. Мы будем использовать планирование, определяемое функцией планирования [1 –1], которая удовлетворяет этим требованиям. Плоскости, соответствующие этой функции, показаны на рис. 2*а* пунктиром.

В соответствии с требованием о непараллельности планирования и размещения [4] в настоящей работе предлагается размещение [0 1] (рис. 2*б*), которое является параллельным к оси *j*.

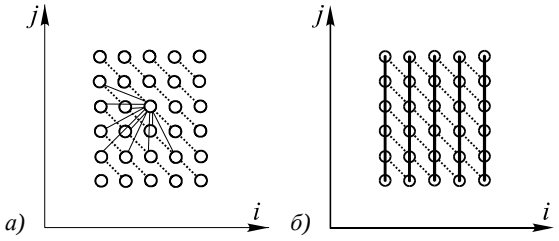


Рис. 2. Граф отношений и планирование [1 -1] (а); планирование и размещение [0 1] (б)

Архитектурные особенности GPU

Особенностью архитектуры GPU 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.

Такая архитектура может быть описана *двухуровневой массивно-многопоточной моделью* вычислений. Первый уровень модели составляют потоки (threads) в рамках одного блока (block). Второй уровень составляют блоки. Взаимодействие между потоками одного блока является максимально быстрым, поскольку осуществляется посредством разделяемой памяти, расположенной непосредственно на ядре GPU, и имеет механизмы синхронизации. Взаимодействие между потоками разных блоков возможно лишь через глобальную память, что гораздо медленнее. Специальные примитивы синхронизации потоков разных блоков отсутствуют, синхронизация в данном случае может быть реализована через использование атомарных операций при работе с глобальной памятью.

Метод нежёсткого размещения в модели многогранников

В работе [5] рассматривается технология автоматизированного построения CUDA-кода с распараллеливанием циклов. Основной цикл выполняется на CPU, а вложенные циклы – на GPU. В листинге 2 представлен фрагмент кода, который демонстрирует такой гибридный подход. Для краткости содержимое CUDA-ядра опущено.

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 GPU 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 приведёт к значительному замедлению выполнения алгоритма. Кроме того, дополнительные задержки возникают при вызове новых ядер.

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);
}
```

В настоящей работе предлагается реализовать обработку полностью на GPU. Основная идея предлагаемого подхода следующая. Каждый блок исполняет роль процессорного элемента. При этом различное количество блоков на разных временных срезах обеспечивается за счёт использования блокировок в глобальной памяти GPU. При этом на каждом шаге будет требоваться 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 GPU. В размещении на рис. 2*а* 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 $(y_1(i,j)$: $i < i_0 \otimes j < j_0)$ уже вычислены. Для некоторого размещения каждый процессорный элемент «закреплён» за некоторой линией в пространстве итераций. Для размещения [0 1] это означает, что некоторый блок закреплён за некоторым столбцом. Таким образом, для каждого столбца $i_0 = \text{const}$. Можно показать, что сформулированное требование эквивалентно следующему: вычисление элемента $y_1(i_0, j_0)$ возможно, если

$$j^*(i_0 - 1) \geq j_0, \tag{7}$$

$$i_0 \equiv const, \tag{8}$$

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

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

Чтобы избежать блокировок, предлагается отказаться от жёсткого 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

$$j^*(i_1 - 1) \geq j_1, \tag{9}$$

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

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 модели многогранников. Алгоритм, соответствующий условиям (9), (10), будем называть моделью многогранников с *нежёстким размещением*.

Принципиальный код, реализующий нежёсткое размещение в рамках модели многогранников, приведён в листинге 3.

Листинг 3. БИХ-фильтр с нежёстким 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 3 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. Причём 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 нежёсткого размещения в модели многогранников справедливо следующее:

Утверждение. Каскадно-рекурсивный алгоритм с нежёстким размещением для вычислительной модели CUDA, соответствующей архитектурным особенностям GPU, обладает следующими свойствами:
- ни один блок алгоритма не попадает в состояние бесконечной блокировки;
- после завершения работы алгоритма все элементы изображения обработаны.

Доказательство достаточно простое и в рамках настоящей работы опускается.

2. 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]. Эффективный алгоритм параллельной редукции на CUDA представлен в [5]. Сложность такого алгоритма составляет $[\log_2(m^2)] + 1$, где m^2 – это количество элементов, которые требуется сложить.

В предлагаемом алгоритме нежёсткого 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:

$$O = (2N-1)O_b = (2N-1)(\lceil 2\log_2 m \rceil + 1). \tag{11}$$

Рассмотрим теперь количество обращений к глобальной памяти GPU. Такие операции являются наиболее затратными по времени выполнения на GPU. 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 со множественным вызовом kernel-функций, понадобится m операций, чтобы считать область, соответствующую окну фильтра. В алгоритме с одной kernel-функцией 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 фиксированного размещения [5], использующий несколько kernel-функций, требует m операций с глобальной памятью на блок на каждой итерации. Таким образом, разницу в количестве требуемых операций с глобальной памятью между нежёстким и фиксированным размещением можно оценить как

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

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

3. Результаты экспериментов, выводы

Экспериментальные исследования проводились для задачи повышения качества изображения на основе двумерного БИХ- фильтра с окном размером $m \times m$ на квадратном изображении размера 2048×2048 пикселей.

Экспериментальное сравнение эффективности алгоритмов было проведено с использованием GPU NVIDIA GF 540M. Компиляция выполнялась с использованием compute capability 1.1, минимально необходимым для использования атомарных операций с глобальной памятью. Как для фиксированного, так и для нежёсткого размещения блок состоит из 32 потоков. Для алгоритма нежёсткого размещения использовалось 512 блоков. Притом, что блок обрабатывает один столбец изображения, обработка полного изображения выполнялась в четыре этапа. В алгоритме фиксированного размещения количество блоков меняется от 1 до 2048.

На рис. 3 приводится зависимость времени, необходимого для обработки всего изображения, от размера окна фильтра для алгоритма фиксированного размещения [5] и для предлагаемого алгоритма нежёсткого размещения. Предлагаемый алгоритм является эффективным для значений m больших, чем 5, 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 GPU NVIDIA GF 540M. Характер зависимости для алгоритма нежёсткого размещения близок к константной зависимости, а при фиксированном размещении зависимость близка к линейной.

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

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

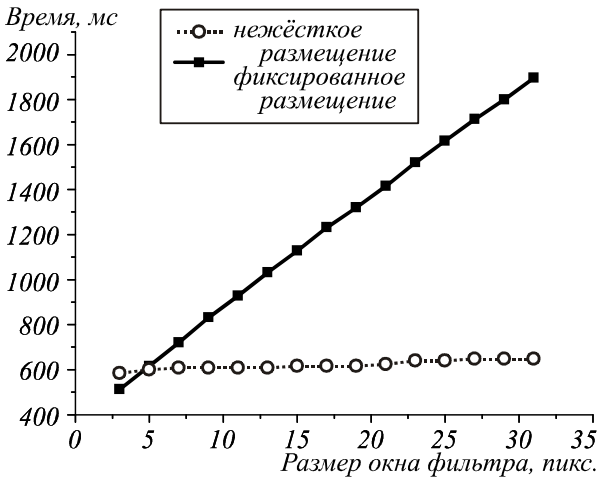


Рис. 3. Зависимость времени выполнения от размера опорного окна

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 (References)

1. **Soifer, V.** Computer Image Processing, Part II: Methods and algorithms / V. Soifer. – VDM Verlag, 2009. – 584 p.
2. **Fursov, V.** Stable IIR Filters Identification based on the Genetic Algorithms / V. Fursov, A. Nikonorov, P. Yakimov. - 8th Open German-Russian Workshop “Pattern recognition and image understanding”, workshop proceedings, November 21-26, 2011. – P. 71-74
3. **Lengauer, C.** Loop Parallelization in the Polytope Model / C. Lengauer. - Proceedings of the 4th International Conference on Concurrency Theory, 1993. - P. 398-416.
4. **Rajopadhye, S.** Synthesizing systolic arrays from recurrence equations / S. Rajopadhye, R.M. Fujimoto // Parallel Computing, 14(2), June 1990. - P. 163-189
5. **Baghdadi, S.** Putting Automatic Polyhedral Compilation for GPGPU to Work / S. Baghdadi, A. Groblinger, A. Cohen. - Proceedings of the 15th Workshop on Compilers for Parallel Computers, 2010.
6. **McCool, M.** Structured Parallel Programming with Deterministic Patterns / M. McCool. - HotPar’10, 2nd USENIX Workshop on Hot Topics in Parallelism, Berkeley, CA, June 2010. - P. 14-15.
7. **Lamport, L.** The parallel execution of DO loops / L. Lamport // ACM Communication. – 1974. – Vol. 17, issue 2. - P. 83-93.
8. **NVIDIA**, NVIDIA CUDA C Best Practices Guide // Santa Clara, 2012. – 75 p.
9. **Dudgeon**, **D.E.** Multidimensional digital signal processing / D.E. Dudgeon, R.M. Mersereau. – Prentice-Hall, Inc., Englewood Cliffs, 1984. – 400 p.

FLEXIBLE ALLOCATION METHOD IN POLYTOPE MODEL FOR DESIGNING OF EFFICIENT METHODS OF RECURSIVE TWO-DIMENSIONAL IMAGE PROCESSING USING GPU

A.V. Nikonorov, V.A. Fursov, P.Yu. Yakimov
Image Processing Systems Institute of the RAS,
S.P. Korolyov Samara State Aerospace University (National Research University)

Abstract

This paper considers a GPU implementation of a two-dimensional infinite impulse-response filter. We propose the decomposition of an IIR filter into the form which is applicable in the polytope model. The presented flexible allocation approach makes it possible to efficiently implement the polytope model in a single GPU kernel. Some theoretical performance estimations of the proposed flexible allocation algorithm are given in the paper.

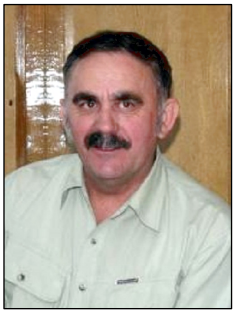
Key words: image processing, polytope model, GPU, fixed allocation, flexible allocation, CUDA, 2-D IIR Filters.

Сведения об авторах

Никоноров Артём Владимирович родился в 1979 году. В 2002 году окончил Самарский государственный аэрокосмический университет. Кандидат технических наук. В настоящее время работает научным сотрудником в Институте систем обработки изображений РАН и доцентом кафедры общей информатики СГАУ. Опубликовал более 30 работ. Область научных интересов: распознавание образов и анализ изображений, идентификация систем, параллельные и распределённые вычисления, вычисления с использованием графических процессоров.

E-mail: *admin@mcdk.com* .

Artem Vladimirovich Nikonorov (b. 1979). He graduated from SSAU in 2002 and became a PHD-student. After that, he got Candidate of Science (Engineering) degree in 2005. Now he works as an associated professor at General Informatics sub-department of SSAU. He has more than 30 publications. Field of scientific interest: pattern recognition and image analysis, system identification, parallel and distributed programming, GPGPU programming.



Фурсов Владимир Алексеевич, д.т.н., профессор, руководитель сектора в Институте систем обработки изображений РАН и заведующий кафедрой общей информатики в Самарском государственном аэрокосмическом университете. Область научных интересов: теория и методы оценивания по малому числу измерений, методы обработки и распознавания изображений, построение параллельных алгоритмов обработки и распознавания изображений, реализуемых с использованием многопроцессорных вычислительных систем.

E-mail: *fursov@smr.ru* .

Vladimir Alekseyevich Fursov is Doctor of Engineering Science, Professor, head of General Informatics sub-department of Samara State Aerospace University, leading researcher. Research interests are development of the theory of estimation on small number of observations, development of methods of image processing and training to pattern recognition, development of high-performance parallel methods both algorithms of image processing and pattern recognition oriented on application of multiprocessor computing systems.

Якимов Павел Юрьевич, родился в 1987 году, в 2011 году окончил магистратуру Самарского государственного аэрокосмического университета имени академика С.П. Королёва по специальности «Прикладная математика и информатика», В настоящее время работает стажёром-исследователем в Институте систем обработки изображений РАН и м.н.с. в СГАУ, имеет 25 опубликованных работ. Область научных интересов: распознавание образов и анализ изображений, параллельные и распределённые вычисления, вычисления с помощью графических процессоров.

E-mail: *pavel.yakimov@hotmail.com* .

Pavel Yurievich Yakimov, (b. 1987), graduated from SSAU in 2011, received Master's degree, majoring in Applied Mathematics and Informatics, currently studies Mastзer's Degree, works as an engineer in Samara State Aerospace University and Image Processing Systems Institute, has 25 scientific publications. Field of scientific interest: pattern recognition and image analysis, parallel and distributed programming, GPGPU programming.

Поступила в редакцию 6 июля 2012 г.

Дизайн: Я.Е. Тахтаров. Оформление и верстка: М.А. Вахе, С.В. Смагин и Я.Е. Тахтаров.

Подписано в печать 15.09.2012 г. Усл. печ. л. 19,40.
Отпечатано в типографии ООО «Предприятие «Новая техника».
Заказ № 11/3. Тираж 325 экз. Печать офсетная. Формат 62x84 1/8.