# Дискретная математика и математическая кибернетика

# Discrete mathematics and mathematical cybernetics

*УДК 519.67*

## ОБОБЩЕННЫЙ БЛОЧНЫЙ АЛГОРИТМ ФЛОЙДА – УОРШЕЛЛА

***Н. А. ЛИХОДЕД***[1)]***, Д. С. СИПЕЙКО***[1)]

[1)]*Белорусский государственный университет, пр. Независимости, 4, 220030, г. Минск, Беларусь*

Одним из наиболее используемых на практике алгоритмов для поиска кратчайших путей между всеми парами вершин во взвешенных графах является алгоритм Флойда – Уоршелла. Блочная версия алгоритма служит основой для получения эффективных параллельных алгоритмов при реализации на многоядерных центральных процессорах, компьютерах с распределенной памятью, графических процессорах. Увеличение зернистости вычислений в блочных версиях алгоритмов приводит к более эффективному использованию кешей и более эффективной организации параллельных вычислений. В этой работе предложено обобщение блочного алгоритма Флойда – Уоршелла. Порядок выполнения блоков вычислений реорганизован таким образом, чтобы элементы массива, участвующие в коммуникационных операциях как чтения, так и записи, реже вытеснялись из памяти с быстрым доступом. Тогда при реализации алгоритма на графическом процессоре реже, по сравнению с исходным блочным алгоритмом, используется медленная глобальная память.

***Ключевые слова:*** параллельные алгоритмы; поиск кратчайших путей; графы; алгоритм Флойда – Уоршелла; блочный алгоритм; графический процессор; GPU.

***Благодарность.*** Работа выполнена в рамках государственной программы научных исследований Республики Беларусь «Конвергенция-2020» (подпрограмма «Методы математического моделирования сложных систем»).

**Образец цитирования:**
Лиходед НА, Сипейко ДС. Обобщенный блочный алгоритм Флойда – Уоршелла. *Журнал Белорусского государственного университета. Математика. Информатика.* 2019;3:84–92.
https://doi.org/10.33581/2520-6508-2019-3-84-92

**For citation:**
Likhoded NA, Sipeyko DS. Generalized blocked Floyd – Warshall algorithm. *Journal of the Belarusian State University. Mathematics and Informatics.* 2019;3:84–92. Russian.
https://doi.org/10.33581/2520-6508-2019-3-84-92

**Авторы:**
***Николай Александрович Лиходед*** – доктор физико-математических наук, профессор; профессор кафедры вычислительной математики факультета прикладной математики и информатики.
***Дмитрий Сергеевич Сипейко*** – магистрант факультета прикладной математики и информатики. Научный руководитель – Н. А. Лиходед.

**Authors:**
***Nikolai A. Likhoded***, doctor of science (physics and mathematics), full professor; professor at the department of computational mathematics, faculty of applied mathematics and computer science.
*likhoded@bsu.by*
***Dmitry S. Sipeyko***, master's degree student at the faculty of applied mathematics and computer science.
*mintaids@ya.ru*

# GENERALIZED BLOCKED FLOYD – WARSHALL ALGORITHM

***N. A. LIKHODED*[a], *D. S. SIPEYKO*[a]**

[a]*Belarusian State University, 4 Niezaliežnasci Avenue, Minsk 220030, Belarus*
*Corresponding author: N. A. Likhoded (likhoded@bsu.by)*

One of the most commonly used on practice all-pairs shortest paths algorithms on weighted graphs is Floyd – Warshall algorithm. Blocked version serves as a basis for obtaining effective parallel algorithms to be implemented on multi-core central processing units, on computers with distributed memory, on graphics processing units (GPU). Increasing computation granularity in blocked versions of algorithm leads to a more efficient usage of caches and more efficient organization of parallel computations. In this paper we introduce generalization of blocked Floyd – Warshall algorithm. Computing blocks execution order was reorganized in such a way that array elements which participate in communication operations, both reading and writing, are pushed out of fast-access memory less often. This means that in GPU implementation slow global memory is used less often, compared with the original blocked algorithm.

***Keywords:*** parallel algorithms; shortest paths; graphs; Floyd – Warshall algorithm; block algorithm; GPU.

***Acknowledgements.*** The prepared report was sponsored by the government program of scientific research of the Republic of Belarus «Convergence-2020» (subprogram «Methods of mathematical modeling of complex systems»).

## Введение

Поиск в графах играет важную роль при анализе больших наборов данных. Широкое применение в задачах маршрутизации и логистики имеют алгоритмы поиска кратчайших путей. Одним из наиболее используемых на практике алгоритмов для поиска кратчайших путей между всеми парами вершин во взвешенных графах является алгоритм Флойда – Уоршелла.

Все алгоритмы решения задачи поиска кратчайших путей между всеми парами вершин графа имеют нелинейную трудоемкость, поэтому для решения задач большого размера требуются методы, хорошо приспособленные для реализации на современных вычислительных устройствах. Среди них – блочный алгоритм Флойда – Уоршелла [1; 2]. Он служит основой для получения эффективных параллельных версий алгоритма для реализации на многоядерных центральных процессорах, компьютерах с распределенной памятью, графических процессорах (GPU) [3–6]. Как и во многих других случаях применения блочных алгоритмов, увеличение зерна вычислений приводит к более эффективному использованию кешей и более эффективной организации параллельных вычислений.

Цель работы – построение модификации блочного алгоритма Флойда – Уоршелла, в которой порядок выполнения блоков вычислений реорганизован таким образом, чтобы реже вытеснялись из памяти с быстрым доступом (кеши, регистры) элементы массива, участвующие не только в коммуникационных операциях чтения, но и в операциях записи. Можно ожидать, что тогда при реализации алгоритма будет более эффективно, по сравнению с исходным блочным алгоритмом, использоваться память с быстрым доступом. Предлагаемая модификация задает параметрическое семейство блочных алгоритмов, которое включает в себя и исходный алгоритм.

Степень использования памяти с быстрым доступом отражает вычислительное свойство метода, называемое локальностью, которая при реализации алгоритмов на многопроцессорных вычислительных устройствах играет важнейшую роль в достижении высокой производительности [7–9]. В данной работе построенный обобщенный алгоритм Флойда – Уоршелла реализован на графическом процессоре, при вычислениях на котором быстрым является процесс обращения к регистрам, разделяемой памяти мультипроцессора и кешам, но не обращение к глобальной памяти GPU. Реализация на основе обобщенного алгоритма приводит к сокращению числа обращений к глобальной памяти и, как показали вычислительные эксперименты, к уменьшению времени выполнения.

## Точечный алгоритм. Зависимости алгоритма

Пусть $G(V, E)$ – некоторый граф, $V$ – множество вершин, $E$ – множество ребер. Будем считать, что вершины графа занумерованы последовательными целыми числами от 1 до $n$, а граф задан матрицей смежности $A$ размером $n \times n$.

Приведем основную часть последовательного точечного (т. е. не блочного) алгоритма Флойда – Уоршелла:

do $k = 1, n$
  do $i = 1, n$
    do $j = 1, n$
      $S_1(k, i, j)$: $a(i, j) = \min\left(a(i, j), a(i, k) + a(k, j)\right)$
    enddo
  enddo
enddo

Перед началом выполнения алгоритма матрица расстояний $A$ заполняется длинами ребер графа (или заведомо большим числом, если ребра нет). На каждом шаге $k$ матрица $A$ обновляется. По завершении работы алгоритма матрица $A$ содержит длины кратчайших путей между всеми вершинами графа.

В гнезде циклов имеется один выполняемый оператор $S_1$ и используется один массив $a$ размерности 2. Область изменения параметров циклов (область итераций) для оператора $S_1$ имеет размерность 3:

$$V_1 = \left\{(k, i, j) \in \mathbb{Z}^3 \,|\, 1 \le k \le n, 1 \le i \le n, 1 \le j \le n\right\}.$$

Формально между операциями одного слоя $k$ существуют информационные зависимости, которые связаны с обновлением или использованием элементов массивов $a(i, j)$ при $i = k$ или $j = k$. Устранить указанные зависимости можно введением дополнительных массивов [5]. Но непосредственно из записи алгоритма Флойда – Уоршелла следует, что данные $a(i, j)$ не обновляются, если $i = k$ или $j = k$. Поэтому для фиксированного $k$ все операции фактически используют данные, вычисленные на предыдущем $(k-1)$-м шаге. Можно допустить, что все операции при фиксированном $k$ не зависят друг от друга и возможен произвольный порядок их выполнения.

Рассмотрим зависимости и векторы зависимостей алгоритма с учетом сделанного допущения. Векторы зависимостей будем для наглядности помечать элементами матрицы, фигурирующими на порождающих зависимости вхождениях. Например, вектор $d^{a(i,j),a(i,k)}$ порождается зависимостью между данными $a(i, j)$ в левой части оператора $S_1$ и данными $a(i, k)$ в его правой части. Укажем итерации, порождающие зависимости, и векторы зависимостей:

• $S_1(k-1, i, j) \to S_1(k, i, j)$: данное $a(i, j)$, вычисленное на итерации $(k-1, i, j)$, является аргументом $a(i, j)$ для вычислений на итерации $(k, i, j)$; $d^{a(i,j),a(i,j)} = (1, 0, 0)$;

• $S_1(k-1, i, k) \to S_1(k, i, j)$: $a(i, k)$, вычисленное на итерации $(k-1, i, k)$, является аргументом для вычислений на итерациях $(k, i, j)$; $d^{a(i,j),a(i,k)} = (1, 0, j-k)$;

• $S_1(k-1, k, j) \to S_1(k, i, j)$: $a(k, j)$, вычисленное на итерации $(k-1, k, j)$, является аргументом для вычислений на итерациях $(k, i, j)$; $d^{a(i,j),a(k,j)} = (1, i-k, 0)$.

## Блочный алгоритм

Блочный алгоритм Флойда – Уоршелла с трехмерными (3D) блоками впервые предложен в работе [1]. Выделим $Q \times Q \times Q$ блоков размером $r \times r \times r$, где $Q = \left\lceil \frac{n}{r} \right\rceil$, $r$ – параметр, задающий размер. Отметим, что одинаковые размеры блока имеют существенное значение, а не выбраны для простоты.

Пусть $k^{gl}, i^{gl}, j^{gl}$ – номера частей, на которые при формировании блоков разбиваются области значений параметров $k, i, j$ циклов, $0 \le k^{gl}, i^{gl}, j^{gl} \le Q - 1$. Блоки вычислений называют также тайлами. Блок $\mathrm{Tile}\left(k^{gl}, i^{gl}, j^{gl}\right)$ имеет следующий вид:

do $k = 1 + k^{gl}r, \min\left(\left(k^{gl}+1\right)r, n\right)$
  do $i = 1 + i^{gl}r, \min\left(\left(i^{gl}+1\right)r, n\right)$
    do $j = 1 + j^{gl}r, \min\left(\left(j^{gl}+1\right)r, n\right)$
      $a(i, j) = \min\left(a(i, j), a(i, k) + a(k, j)\right)$
    enddo
  enddo
enddo

Установим корректный порядок выполнения блоков и обоснуем корректный порядок выполнения вычислений в блоке.

Рассмотрим блоки некоторой блочной итерации $k^{gl}$ (некоторого блочного слоя $k^{gl}$), т. е. блоки $\text{Tile}\left(k^{gl},\ i^{gl},\ j^{gl}\right)$ при фиксированном $k^{gl}$. Назовем:

• $\text{Tile}\left(k^{gl},\ k^{gl},\ k^{gl}\right)$ – ведущим блоком;

• $\text{Tile}\left(k^{gl},\ k^{gl},\ j^{gl}\right)$, $0 \le j^{gl} \le Q-1$, $j^{gl} \ne k^{gl}$, – блоком ведущей строки;

• $\text{Tile}\left(k^{gl},\ i^{gl},\ k^{gl}\right)$, $0 \le i^{gl} \le Q-1$, $i^{gl} \ne k^{gl}$, – блоком ведущего столбца.

Анализ зависимостей показывает следующее.

1. Вычисления любого ведущего блока $\text{Tile}\left(k^{gl},\ k^{gl},\ k^{gl}\right)$ не зависят от вычислений других блоков блочного слоя, для вычисления элементов ведущего блока нужны только его элементы. Ведущий блок на блочном слое следует вычислять первым. Ведущие блоки назовем I-блоками (independent blocks [4]).

2. Вычисления блоков ведущей строки и ведущего столбца зависят от вычислений ведущего блока блочного слоя. Для вычислений этих блоков необходимы их собственные элементы и уже подсчитанные элементы ведущего блока. Между собой эти блоки не конкурируют, поэтому последовательность их вычислений на блочном слое может быть произвольной. Назовем блоки ведущих строк и столбцов SD-блоками (singly dependent blocks).

3. Остальные блоки $\text{Tile}\left(k^{gl},\ i^{gl},\ j^{gl}\right)$, $0 \le i^{gl}, j^{gl} \le Q-1$, $i^{gl} \ne k^{gl}$, $j^{gl} \ne k^{gl}$, зависят от вычислений блоков ведущих строк и столбцов. Для вычислений этих блоков нужны их собственные элементы, а также элементы соответствующих блоков ведущей строки и ведущего столбца. Указанные блоки вычисляются на блочном слое в произвольном порядке после вычисления блоков ведущей строки и столбца. Блоки вне ведущих строк и столбцов назовем DD-блоками (doubly dependent blocks).

Опишем шаги блочной итерации $k^{gl}$:

1) производятся вычисления ведущего блока (I-блока). Фактически выполняется обычный точечный алгоритм Флойда – Уоршелла, в итоге сохраняется версия элементов подматрицы ведущего блока на итерации $\left(k^{gl}+1\right)r$ (или $\min\left(\left(k^{gl}+1\right)r,\ n\right)$ для последнего слоя);

2) производятся вычисления блоков ведущей строки и ведущего столбца (SD-блоков). При обращении к элементам ведущего блока происходит обращение к их последней версии. Блоки могут вычисляться независимо, в произвольном порядке;

3) производятся вычисления оставшихся блоков (DD-блоков). При обращении к элементам блоков ведущей строки и ведущего столбца происходит обращение к их последней версии. Блоки могут вычисляться независимо, в произвольном порядке.

*Замечание 1.* Результаты промежуточных вычислений точечного и блочного алгоритмов Флойда – Уоршелла могут не совпадать. Тем не менее блочный алгоритм Флойда – Уоршелла приводит к корректному результату [2].

Основная часть алгоритма Флойда – Уоршелла с выделенными 3D-блоками имеет следующий вид [4] (циклы, итерации которых заведомо можно выполнять независимо, запишем как dopar):

```
do k^gl = 0, Q – 1
   Tile(k^gl, k^gl, k^gl) // вычисления I-блока
   dopar j^gl = 0, Q – 1 (j^gl ≠ k^gl)
      Tile(k^gl, k^gl, j^gl) // вычисления SD-блоков ведущей строки
   enddopar
   dopar i^gl = 0, Q – 1 (i^gl ≠ k^gl)
      Tile(k^gl, i^gl, k^gl) // вычисления SD-блоков ведущего столбца
   enddopar
   dopar i^gl = 0, Q – 1 (i^gl ≠ k^gl)
      dopar j^gl = 0, Q – 1 (j^gl ≠ k^gl)
         Tile(k^gl, i^gl, j^gl) // вычисления DD-блоков
      enddopar
   enddopar
enddo (k^gl)
```

*Замечание 2.* В DD-блоках вычисления всех $a(i, j)$ происходят независимо друг от друга: $a(i, k)$ и $a(k, j)$ вычисляются вне DD-блока, между операциями блока остаются только зависимости, задаваемые вектором $d^{a(i,j),\,a(i,j)} = (1, 0, 0)$. На практике в DD-блоках производят перестановку циклов так, чтобы цикл с параметром $k$ стал самым внутренним. Обозначим такой блок через $\mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}, i^{gl}, j^{gl}\right)$ и запишем его явный вид:

dopar $i = 1 + i^{gl}r,\ \min\left(\left(i^{gl}+1\right)r,\ n\right)$
  dopar $j = 1 + j^{gl}r,\ \min\left(\left(j^{gl}+1\right)r,\ n\right)$
    do $k = 1 + k^{gl}r,\ \min\left(\left(k^{gl}+1\right)r,\ n\right)$
      $a(i, j) = \min\left(a(i, j),\ a(i, k) + a(k, j)\right)$
    enddo
  enddopar
enddopar

## Модифицированный блочный алгоритм

В рассмотренном блочном алгоритме последовательно выполняются блочные итерации $k^{gl}$. Для каждого фиксированного $k^{gl}$ сначала производятся вычисления I-блока $\mathrm{Tile}\left(k^{gl}, k^{gl}, k^{gl}\right)$, затем (в произвольном порядке) – вычисления $2(Q-1)$ SD-блоков $\mathrm{Tile}\left(k^{gl}, k^{gl}, j^{gl}\right)$ и $\mathrm{Tile}\left(k^{gl}, i^{gl}, k^{gl}\right)$, далее (в произвольном порядке) – вычисления $(Q-1)\times(Q-1)$ DD-блоков $\mathrm{Tile}\left(k^{gl}, i^{gl}, j^{gl}\right)$. Как уже отмечалось, размеры блока ($r \times r \times r$ итераций) могут быть только одинаковые.

Реорганизуем порядок выполнения блоков вычислений таким образом, чтобы атомарно, как одна макрооперация, выполнялось некоторое количество тайлов с идущими подряд номерами первой блочной координаты и фиксированными номерами второй и третьей координаты. Такие объединенные тайлы будем называть мультитайлами (вообще говоря, мультитайл может содержать только один тайл). В макрооперациях-мультитайлах выполнение большего числа идущих подряд итераций $k$ приводит к более редкому вытеснению из памяти с быстрым доступом (кеши, регистры) элементов массива на вхождениях $a(i, j)$, требующих как операций чтения, так и операций записи, в то время как вхождения $a(i, k)$ и $a(k, j)$ требуют только операций чтения. Можно ожидать, что при реализации алгоритма будет более эффективно использоваться память с быстрым доступом.

Пусть к – некоторое число в пределах от 1 до $Q$, $l$ – некоторое число в пределах от 0 до к – 1. Параметр к задает количество блочных итераций $k^{gl}$ (число блочных слоев), используемых для образования мультитайлов. От параметра $l$ зависит число блочных итераций $k^{gl}$ (число блочных слоев), которые объединяются для получения мультитайлов.

Введем в рассмотрение процедуры выполнения мультитайлов.

$\mathrm{CalcLeadBlock}\left(k^{gl}, l\right)$ – процедура вычисления (при фиксированных параметрах $k^{gl}$, $l$ процедуры) мультитайла

$$\mathrm{Tile}\left(k^{gl}, k^{gl}, k^{gl}\right), \text{ если } l = 0,$$

$$\bigcup_{m=0}^{l-1} \mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}+m, k^{gl}+l, k^{gl}+l\right) \bigcup \mathrm{Tile}\left(k^{gl}+l, k^{gl}+l, k^{gl}+l\right), \text{ если } l \neq 0,$$

объединяющего $l$ DD-блоков и один I-блок исходного блочного алгоритма. Мультитайл включает I-блок и невычисленные DD-блоки (если таковые имеются, т. е. если $l > 0$) с такими же, как у I-блока, номерами второй и третьей блочной координаты и с меньшими номерами первой блочной координаты.

$\mathrm{CalcLeadRowAndColumn}\left(k^{gl}, l\right)$ – процедура вычисления $Q - 1$ мультитайлов, каждый из которых включает один SD-блок $(k^{gl} + l)$-й блочной строки исходного блочного алгоритма, и $Q - 1$ мультитайлов, каждый из которых включает один SD-блок $\left(k^{gl} + l\right)$-го блочного столбца исходного блочного алгоритма. Кроме того, каждый мультитайл включает DD-блоки (если таковые имеются) с такими же, как у SD-блока, номерами второй и третьей блочной координаты и с меньшими номерами первой блочной координаты.

Мультитайл с SD-блоком $\left(k^{gl}+l\right)$-й блочной строки имеет вид

$$\mathrm{Tile}\left(k^{gl},\ k^{gl},\ j^{gl}\right),$$

$$\text{если } l=0,\ j^{gl}=0,\ 1,\ \ldots,\ Q-1\ \left(j^{gl}\neq k^{gl}\right),$$

$$\bigcup_{m=0}^{l-1}\mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}+m,\ k^{gl}+l,\ j^{gl}\right)\bigcup\mathrm{Tile}\left(k^{gl}+l,\ k^{gl}+l,\ j^{gl}\right),$$

$$\text{если } l\neq 0,\ j^{gl}=0,\ 1,\ \ldots,\ Q-1\ \left(j^{gl}\neq k^{gl},\ \ldots,\ k^{gl}+l\right),$$

$$\bigcup_{m=j^{gl}-k^{gl}+1}^{l-1}\mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}+m,\ k^{gl}+l,\ j^{gl}\right)\bigcup\mathrm{Tile}\left(k^{gl}+l,\ k^{gl}+l,\ j^{gl}\right),$$

$$\text{если } l\neq 0,\ j^{gl}=k^{gl},\ \ldots,\ k^{gl}+l-1.$$

Мультитайл с SD-блоком $\left(k^{gl}+l\right)$-го блочного столбца представляется в виде

$$\mathrm{Tile}\left(k^{gl},\ i^{gl},\ k^{gl}\right),$$

$$\text{если } l=0,\ i^{gl}=0,\ 1,\ \ldots,\ Q-1\ \left(i^{gl}\neq k^{gl}\right),$$

$$\bigcup_{m=0}^{l-1}\mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}+m,\ i^{gl},\ k^{gl}+l\right)\bigcup\mathrm{Tile}\left(k^{gl}+l,\ i^{gl},\ k^{gl}+l\right),$$

$$\text{если } l\neq 0,\ i^{gl}=0,\ 1,\ \ldots,\ Q-1\ \left(i^{gl}\neq k^{gl},\ \ldots,\ k^{gl}+l\right),$$

$$\bigcup_{m=i^{gl}-k^{gl}+1}^{l-1}\mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}+m,\ i^{gl},\ k^{gl}+l\right)\bigcup\mathrm{Tile}\left(k^{gl}+l,\ i^{gl},\ k^{gl}+l\right),$$

$$\text{если } l\neq 0,\ i^{gl}=k^{gl},\ \ldots,\ k^{gl}+l-1.$$

$\mathrm{CalcLeadRowAndColumnReverse}\left(k^{gl},\ \kappa,\ l\right)$ – процедура вычисления (при фиксированных параметрах $k^{gl}$, $\kappa$, $l$, здесь $l<\kappa-1$) мультитайлов $\left(k^{gl}+l\right)$-й блочной строки исходного блочного алгоритма и $\left(k^{gl}+l\right)$-го блочного столбца исходного блочного алгоритма. Каждый мультитайл объединяет $\kappa-l-1$ DD-блоков с фиксированными номерами второй и третьей блочной координаты и с большими, чем $k^{gl}+l$, номерами первой блочной координаты:

$$\bigcup_{m=l+1}^{\kappa-1}\mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}+m,\ k^{gl}+l,\ j^{gl}\right),$$

$$j^{gl}=0,\ 1,\ \ldots,\ k^{gl}+l,\ j^{gl}=k^{gl}+\kappa,\ \ldots,\ Q-1,$$

$$\bigcup_{m=l+1}^{\kappa-1}\mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}+m,\ i^{gl},\ k^{gl}+l\right),$$

$$i^{gl}=0,\ 1,\ \ldots,\ k^{gl}+l-1,\ i^{gl}=k^{gl}+\kappa,\ \ldots,\ Q-1.$$

$\mathrm{CalcRestBlocks}\left(k^{gl},\ \kappa\right)$, $\kappa\neq Q$, есть процедура вычисления $\left(Q-\kappa\right)\times\left(Q-\kappa\right)$ мультитайлов

$$\bigcup_{m=0}^{\kappa-1}\mathrm{Tile}_{\mathrm{DD}}\left(k^{gl}+m,\ i^{gl},\ j^{gl}\right),$$

$$i^{gl}=0,\ 1,\ \ldots,\ Q-1\ \left(i^{gl}\neq k^{gl},\ k^{gl}+1,\ \ldots,\ k^{gl}+\kappa-1\right),$$

$$j^{gl}=0,\ 1,\ \ldots,\ Q-1\ \left(j^{gl}\neq k^{gl},\ k^{gl}+1,\ \ldots,\ k^{gl}+\kappa-1\right),$$

каждый из которых объединяет $\kappa$ DD-блоков (вне ведущих блочных строк и столбцов) исходного блочного алгоритма.

Отметим, что во всех процедурах вычисление мультитайлов при фиксированных $k^{gl}$, $\kappa$, $l$ можно выполнять независимо (кроме процедуры $\mathrm{CalcLeadBlock}\left(k^{gl},\ l\right)$, вычисляющей только один мультитайл).

Основную часть обобщенного блочного алгоритма Флойда – Уоршелла можно представить следующим образом:

do $k^{gl} = 0, Q - 1, \kappa$ // вычисления с шагом $\kappa$
do $l = 0, \kappa - 1$
CalcLeadBlock$\left(k^{gl}, l\right)$
CalcLeadRowAndColumn$\left(k^{gl}, l\right)$
enddo
do $l = \kappa - 2, 0, -1$ // вычисления с шагом –1
CalcLeadRowAndColumnReverse$\left(k^{gl}, \kappa, l\right)$
enddo
CalcRestBlocks$\left(k^{gl}, \kappa\right)$
enddo $\left(k^{gl}\right)$

Если $\kappa = 1$, то получим известный блочный алгоритм Флойда – Уоршелла (цикл do $l = \kappa - 2, 0, -1$ отсутствует).

Если $\kappa = 2$, то имеем случай, рассмотренный в магистерской диссертации О. И. Сычевой[1].

Если $\kappa = Q$, то внешний цикл do $k^{gl} = 0, Q - 1, \kappa$ вырождается в одну итерацию $k^{gl} = 0$, циклы с параметром $l$ охватывают вычисление всех блоков, функция CalcRestBlocks$\left(k^{gl}, \kappa\right)$ отсутствует.

## Реализация на графическом процессоре

Графический процессор осуществляет множество параллельных потоков вычислений. Потоки объединяются в блоки вычислений, каждый блок потоков выполняется атомарно на одном из мультипроцессоров графического процессора. При этом должны быть указаны блоки, которые могут выполняться мультипроцессорами одновременно и независимо друг от друга.

Точечный алгоритм Флойда – Уоршелла обладает естественным параллелизмом в пределах одной итерации. Поэтому на каждой итерации $k$ ($k = 1, 2, \ldots, n$) можно выделить двумерные (2D) блоки вычислений, которые могут выполняться независимо друг от друга. GPU-реализация алгоритма Флойда – Уоршелла, основанная на 2D-блоках вычислений, предложена в работе [10]. Матрица $A$ хранится в глобальной памяти GPU, поэтому на каждой итерации $k$ необходима запись всех обновленных элементов матрицы в глобальную память, из которой считываются все нужные данные, подсчитанные на предыдущей итерации.

При вычислениях на GPU быстрым является процесс обращения к разделяемой памяти мультипроцессора и к кешам, но не обращение к глобальной памяти GPU. В работе [11] реализован на GPU алгоритм Флойда – Уоршелла с 3D-блоками. Использование блочного алгоритма с 3D-блоками позволило существенно уменьшить время выполнения алгоритма. Оно сократилось главным образом за счет того, что запись обновленных элементов матрицы в глобальную память производится не на каждой итерации $k$, а на каждой $r$-й итерации $k$. На каждой блочной итерации $k^{gl}$ требуются три так называемых запуска ядра (т. е. три процедуры выполнения блоков вычислений):

• запускаются вычисления ведущего блока (I-блока). Используется $r \times r$ потоков – один поток вычисляет один элемент матрицы. Все потоки в одном блоке можно запустить, если $r \le 32$ (в одном блоке может быть до 1024 потоков). Для каждого потока нужно один элемент скопировать из глобальной памяти в разделяемую, обновить его на $r$ слоях и вернуть новое значение в глобальную память;

• запускаются вычисления блоков ведущей строки и ведущего столбца (SD-блоков), в каждом блоке используется $r \times r$ ($r \le 32$) потоков. Напомним, что для вычислений этих блоков необходимы их собственные элементы и уже подсчитанные элементы ведущего блока. Для каждого из запускаемых блоков в разделяемой памяти хранятся две матрицы размером $r \times r$: помимо своих элементов потокам суммарно требуются $r \times r$ элементов I-блока;

• запускаются вычисления DD-блоков, в каждом блоке используется $r \times r$ потоков ($r \le 32$). Для каждого из запускаемых блоков в разделяемой памяти хранятся три матрицы размером $r \times r$: помимо своих элементов потокам суммарно требуются $r \times r$ элементов блока ведущей строки и $r \times r$ элементов блока ведущего столбца.

[1] *Сычева О. И.* Разработка и программная реализация новых параллельных версий алгоритма Флойда – Уоршелла : магист. дис. Минск : БГУ, 2016. 58 с.

*Замечание 3.* При вычислении DD-блоков в разделяемой памяти достаточно хранить только матрицы, связанные с вхождениями $a(i, k)$ и $a(k, j)$: потокам не понадобится обращаться к элементам $a(i, j)$, которые пересчитывают другие потоки, – нужен будет только свой элемент и элементы двух SD-блоков. Каждый поток может хранить элемент, который он пересчитывает, в своей регистровой памяти.

Дальнейшая оптимизация алгоритма путем уменьшения объема занимаемой разделяемой памяти в блочном алгоритме Флойда – Уоршелла предложена в работе [5]. Главная идея подхода – многостадийное чтение SD-блоков при вычислении DD-блоков. Сокращение размеров разделяемой памяти, необходимой блоку в каждый момент времени, обусловливает заметный выигрыш в производительности, так как уменьшение объема разделяемой памяти, используемой в блоке, позволяет мультипроцессору выполнять большее количество блоков одновременно.

С помощью построенного обобщенного алгоритма Флойда – Уоршелла можно реализовать на графическом процессоре 3D-блоки размером $(l+1)r \times r \times r$, где, напомним, $l$ изменяется от 0 до к – 1. Запись обновленных элементов матрицы в глобальную память производится на $(l+1)r$-й итерации $k$, а не $r$-й итерации, как в случаях блоков размером $r \times r \times r$.

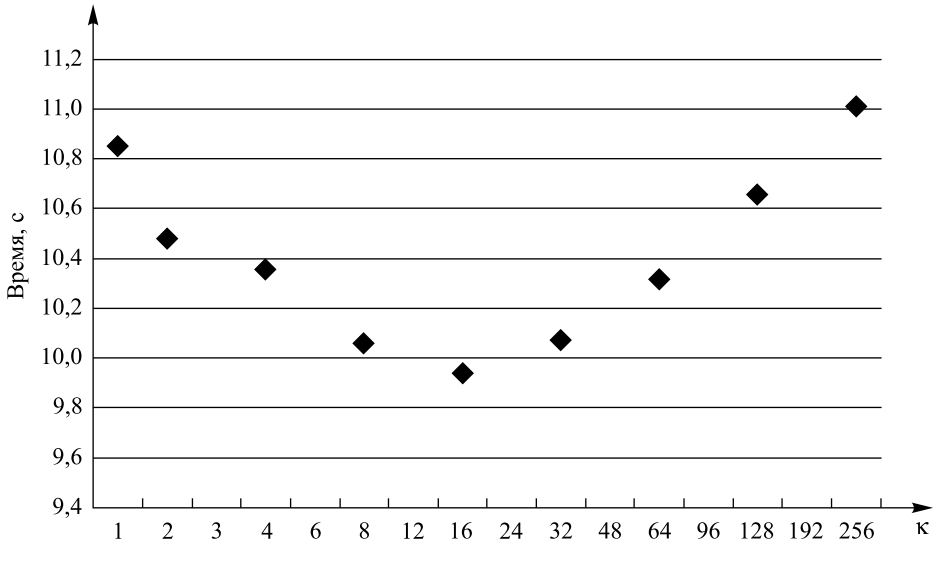
Опишем вычислительные эксперименты, в которых использовалось многостадийное чтение SD-блоков, число потоков в одном блоке 1024 ($r \times r = 32 \times 32$).

Эксперименты проводились на графическом процессоре NVIDIA GeForce GTX 670. Некоторые характеристики этого GPU:

| | |
|---|---|
| Число мультипроцессоров | 7 |
| Количество ядер в устройстве | 1344 |
| Объем глобальной памяти | 2 Гб |
| Объем разделяемой памяти | 48 Кб на каждый мультипроцессор |
| Количество 32-битных регистров в мультипроцессоре | 65 536 |
| Используемая архитектура | Kepler |

На рисунке представлен график зависимости времени вычислений реализации алгоритма от параметра к, влияющего на число записей в глобальную память.

Рисунок показывает, что при нескольких значениях параметра к обобщенного блочного алгоритма достигается меньшее время вычислений по сравнению со случаем к = 1 (классический блочный алгоритм). В этом примере число вершин графа равно 8192. Аналогичная картина наблюдается и в примерах с другим числом вершин.




Зависимость времени вычислений реализации обобщенного блочного алгоритма ($n = 8192$, $r = 32$) от параметра к

Dependency of computation time of generalized blocked algorithm implementation ($n = 8192$, $r = 32$) on the parameter к

*Замечание 4.* При обращении к функции CalcLeadBlock выполняется только один мультитайл, поэтому активен только один (из нескольких) мультипроцессор GPU. Можно организовать вычисление функции CalcLeadBlock одновременно на всех мультипроцессорах. Эксперименты показали, что общее время реализации алгоритма при этом уменьшается очень незначительно, так как почти все временные затраты приходятся на вычисление других функций.

Таким образом, в работе построено параметрическое семейство блочных алгоритмов Флойда – Уоршелла, которое включает в себя и классический блочный алгоритм, рассмотрена реализация предложенного алгоритма на графическом процессоре.

## Библиографические ссылки

1. Venkataraman G, Sahni S, Mukhopadhyaya S. A blocked all-pairs shortest-paths algorithm. *Journal of Experimental Algorithmics*. 2003;8:857–874. DOI: 10.1145/996546.996553.

2. Park J, Penner M, Prasanna VK. Optimizing graph algorithms for improved cache performance. *IEEE Transactions on Parallel and Distributed Systems*. 2004;15(9):769–782. DOI: 10.1109/TPDS.2004.44.

3. Srinivasan T, Balakrishnan R, Gangadharan SA, Hayawardh V. A scalable parallelization of all-pairs shortest path algorithm for a high performance cluster environment. In: *Proceedings of the 13th International Conference on Parallel and Distributed Systems; 2007 December 5–7; Hsinchu, Taiwan.* Washington: IEEE Computer Society; 2007. p. 1–8. DOI: 10.1109/ICPADS.2007.4447721.

4. Lund BD, Smith JW. A multi-stage CUDA kernel for Floyd – Warshall. arXiv:1001.4108. 2010 [Preprint]. 2010 [cited 2019 June 3]. Available from: https://arxiv.org/abs/1001.4108.

5. Mullapudi RT, Bondhugula U. Tiling for dynamic scheduling. In: *IMPACT 2014. Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques; 2014 January 20–22; Vienna, Austria.* [S. l.]: [s. n.]; 2014.

6. Прихожий АА, Карасик ОН. Разнородный блочный алгоритм поиска кратчайших путей между всеми парами вершин графа. *Системный анализ и прикладная информатика*. 2017;3:68–75. DOI: 10.21122/2309-4923-2017-3-68-75.

7. Воеводин ВлВ, Воеводин ВадВ. Спасительная локальность суперкомпьютеров. *Открытые системы. СУБД*. 2013; 9:12–15.

8. Buluc A, Gilberta JR, Budak C. Solving path problems on the GPU. *Parallel Computing*. 2010;36(5–6):241–253. DOI: 10.1016/j.parco.2009.12.002.

9. Лиходед НА, Полещук МА. Условия приватизации элементов массива потоками вычислений. *Журнал Белорусского государственного университета. Математика. Информатика.* 2018;3:59–67.

10. Harish P, Narayanan P. Accelerating large graph algorithms on the GPU using CUDA. In: *High Performance Computing – HiPC 2007. Proceedings of the 14th International Conference; 2007 December 18–21; Goa, India.* Berlin: Springer-Verlag; 2007. p. 197–208. DOI: 10.1007/978-3-540-77220-0_21.

11. Katz GJ, Kider JT. All-pairs shortest-paths for large graphs on the GPU. In: *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware; 2008 June 20–21; Sarajevo, Bosnia and Herzegovina.* Aire-la-Ville: Eurographics Association; 2008. p. 47–55.

## References

1. Venkataraman G, Sahni S, Mukhopadhyaya S. A blocked all-pairs shortest-paths algorithm. *Journal of Experimental Algorithmics*. 2003;8:857–874. DOI: 10.1145/996546.996553.

2. Park J, Penner M, Prasanna VK. Optimizing graph algorithms for improved cache performance. *IEEE Transactions on Parallel and Distributed Systems*. 2004;15(9):769–782. DOI: 10.1109/TPDS.2004.44.

3. Srinivasan T, Balakrishnan R, Gangadharan SA, Hayawardh V. A scalable parallelization of all-pairs shortest path algorithm for a high performance cluster environment. In: *Proceedings of the 13th International Conference on Parallel and Distributed Systems; 2007 December 5–7; Hsinchu, Taiwan.* Washington: IEEE Computer Society; 2007. p. 1–8. DOI: 10.1109/ICPADS.2007.4447721.

4. Lund BD, Smith JW. A multi-stage CUDA kernel for Floyd – Warshall. arXiv:1001.4108. 2010 [Preprint]. 2010 [cited 2019 June 3]. Available from: https://arxiv.org/abs/1001.4108.

5. Mullapudi RT, Bondhugula U. Tiling for dynamic scheduling. In: *IMPACT 2014. Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques; 2014 January 20–22; Vienna, Austria.* [S. l.]: [s. n.]; 2014.

6. Prihozhy AA, Karasik ON. Heterogenious blocked all-pairs shortest paths algorithm. *Sistemnyi analiz i prikladnaya informatika.* 2017;3:68–75. Russian. DOI: 10.21122/2309-4923-2017-3-68-75.

7. Voevodin VlV, Voevodin VadV. [The fortunate locality of supercomputers]. *Otkrytye sistemy. SUBD*. 2013;9:12–15. Russian.

8. Buluc A, Gilberta JR, Budak C. Solving path problems on the GPU. *Parallel Computing*. 2010;36(5–6):241–253. DOI: 10.1016/j.parco.2009.12.002.

9. Likhoded NA, Paliashchuk MA. Conditions for privatizing the elements of arrays by computing threads. *Journal of the Belarusian State University. Mathematics and Informatics.* 2018;3:59–67. Russian.

10. Harish P, Narayanan P. Accelerating large graph algorithms on the GPU using CUDA. In: *High Performance Computing – HiPC 2007. Proceedings of the 14th International Conference; 2007 December 18–21; Goa, India.* Berlin: Springer-Verlag; 2007. p. 197–208. DOI: 10.1007/978-3-540-77220-0_21.

11. Katz GJ, Kider JT. All-pairs shortest-paths for large graphs on the GPU. In: *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware; 2008 June 20–21; Sarajevo, Bosnia and Herzegovina.* Aire-la-Ville: Eurographics Association; 2008. p. 47–55.

*Статья поступила в редколлегию 22.09.2019.*
*Received by editorial board 22.09.2019.*