# Исследование модели доступа к оперативной памяти в гетерогенной вычислительной системе

**А.А. Колпаков[1], Ю.А. Кропотов[1], А.А. Белов[1]**

[1]Владимирский государственный университет имени Столетовых, Орловская 23, Муром, Россия, 602264

**Аннотация.** Вопрос создания высокопроизводительных вычислительных комплексов на базе гетерогенных компьютерных систем является актуальным, так как объемы обрабатываемой информации, вычислений и исследований с большими массивами данных постоянно увеличиваются. Целью работы является экспериментальное исследование ранее разработанных моделей прогнозирования производительности гетерогенных компьютерных систем в телекоммуникациях. В результате Исследование показало, что разработанные модели позволяют получить адекватную оценку возможного времени работы алгоритма при различных параметрах работы GPU с некоторыми ограничениями.

## 1. Введение

Одним из наиболее динамично развивающихся направлений в параллельном программировании в настоящий момент является использование вычислительных систем с гетерогенной архитектурой, в которых присутствуют вычислительные устройства с разной архитектурой и, соответственно, с разными методами использования параллельных вычислений. Самым распространенным подходом в проектировании таких систем стало использование в качестве основного параллельного вычислителя графических видеокарт или устройств на их основе. Эта все возрастающая потребность в решении больших проблем стимулирует исследования и инновации в области параллельных вычислений в общем и в разработке методов для графических процессоров в частности.

## 2. Разработка и исследование моделей гетерогенных вычислительных систем на базе графических процессоров

Современные графические процессоры (graphic processor unit, GPU) – это параллельные процессоры. Точнее, они известны как потоковые процессоры, поскольку они способны выполнять различные функции в потоке входящих данных. Они представляют собой усовершенствованные архитектуры, которые предназначены для параллельной обработки данных (в первую очередь графических). На текущий момент они являются чрезвычайно мощными программируемыми процессорами, возможностями архитектуры MIMD с некоторыми ограничениями.

По мере развития технологий, языков и аппаратного обеспечения исследователи смогли использовать дополнительную гибкость графических процессоров при развертывании

неграфических приложений на GPU (GPGPU), особенно при обработке изображений. Более подробно история развития GPGPU представлена в работе [1].

Дальнейшим импульсом развития стало появление CUDA, среды разработки GPGPU на основе C от NVIDIA. CUDA позволяет разработчикам, незнакомым с графическим программированием, писать код, который может быть выполнен на графическом процессоре. CUDA предоставляет необходимые абстракции для разработчика для написания многопоточных программ с небольшим знанием или без знания графических API. С тех пор для графических процессоров разработано множество реализаций распараллеленных приложений, многие из которых предлагают значительное ускорение по сравнению с последовательными реализациями на процессоре.

В работе [2] приведена модель верхней оценки времени работы алгоритма на графическом процессоре в среде CPU-GPU, которая основана на абстрактной модели PRAM[3]. Верхняя оценка времени работы алгоритма на графическом процессоре в среде CPU-GPU согласно данной модели вычисляется согласно формуле, приведенной ниже

$$T_{GPU}(N) = \frac{N_{HD}(N)}{S_{HD}} + \sum_{i=1}^{B(N)} T_{iG}(N) + \frac{N_{DH}(N)}{S_{DH}}. \quad (1)$$

Данная модель не учитывает некоторых особенностей работы графического процессора, таких, например, как размер warp-а, времени доступа к памяти графического процессора и т.д., поэтому в работе [4] была приведена модель прогнозирования производительности GPGPU, которая представляет собой комбинацию известных моделей параллельных вычислений. Учитывая сложную архитектуру графического процессора, ни одна из этих моделей не является полной, и требуется комбинация из них наряду с несколькими расширениями. При разработке модели использовались:

1. Модель PRAM [3];
2. Модель BSP [5];
3. Модель QRQW [6].

Окончательное уравнение данной модели выглядит следующим образом:

$$T(K) = \frac{N_B(K) \cdot N_w(K) \cdot N_t(K) \cdot C_T(K)}{N_C \cdot D \cdot R} \quad (2)$$

Все параметры данной модели приведены в таблице 1.

**Таблица 1.** Список параметров разработанной модели.

| Параметр | Описание |
|---|---|
| $D$ | Глубина конвейера ялра |
| $N_c$ | Количество ядер на SM |
| $R$ | Тактовая частота GPU |
| $C_t(K)$ | Максимальное количество тактов, потребляемое любой нитью в ядре K |
| $N_t$ | Количество потоков в warp = 32 |
| $N_w$ | Количество warp-ов на блок |
| $N_B(K)$ | Количество блоков на ядро |
| $K_i$ | i-е ядро на графическом процессоре |
| $T(K)$ | Время, затраченное ядром K |
| $T(P)$ | Время, затраченное программой P |

Производительность ядра CUDA может сильно варьироваться с небольшими изменениями в зависимости от стратегий доступа к памяти. Использование общей памяти может обеспечить производительность в 20 раз лучше, чем использование глобальной памяти, а использование объединенных глобальных доступов к памяти может привести к увеличению производительности в 5 раз по сравнению с не выровненным доступом. Арифметические операции также требуют для выполнения различное количество тактов, например, операции, такие как целочисленное суммирование, требуют 4 такта, тогда как вычисление целочисленного модуля занимает 48 тактов [7,8]. Любая модель, которая не фиксирует эти изменения, вряд ли будет точной.

### 3. Использование шаблонов доступа к памяти GPU для повышения производительности гетерогенной вычислительной системы

Поскольку при чтении данных из глобальной памяти графического процессора задержка доступа до 200 раз выше, чем при чтении данных из регистров, обеспечение наиболее эффективного способа доступа к оперативной памяти GPU имеет решающее значение для повышения производительности графического процессора в частности и всей гетерогенной системы в целом. Следовательно, оптимизация доступа к глобальной памяти становится единственным наиболее важным фактором в программировании для архитектуры GPGPU.

Глобальная память GPU осуществляет чтение и запись данных потоками по половине warp-а (16 потоков), которые оптимизируются устройством всего в одну глобальную транзакцию памяти, если выполняются определенные требования доступа. Для видеокарты GTX 280 для определения количества транзакций, используемых полу-warp-ом, применяется следующий протокол:

1. Осуществляется поиск сегмента памяти, в котором содержится адресный запрос от активного потока с наименьшим номером. Размер сегмента составляет 32 байта для 8-битных данных, 64 байта для 16-битных данных и 128 байтов для 32-, 64- и 128-битных данных.
   - если размер транзакции составляет 128 байт и используется только верхняя или нижняя половина сегмента, размер транзакции уменьшается до 64 байт;
   - если размер транзакции составляет 64 байта и используется только нижняя или верхняя половина, размер транзакции уменьшается до 32 байт.
2. Выполняется транзакция, обслуживаемые потоки помечаются как неактивные.
3. Повторяется до тех пор, пока все нити в полу-warp-е не будут обслужены.

Когда более одного потока запрашивает данные с адресов, которые попадают в один и тот же сегмент, одна транзакция может удовлетворить все такие потоки. Такое обслуживание нескольких запросов одной транзакцией называется объединением (coalescing). Таким образом, очевидно, что определенные шаблоны доступа к памяти GPU обязательно будут давать положительный эффект, тогда как другие будут постепенно увеличивать задержку по мере того, как адреса памяти, запрашиваемые потоками полу-warp-а, будут расходиться.

Общая память GPU разделяется на модули памяти одинакового размера, называемые банками, к которым могут одновременно обращаться несколько нитей. Таким образом, любой запрос на чтение или запись в память, состоящей из $n$ адресов, попадающих в $n$ отдельных банков памяти, может осуществляться одновременно, что дает эффективную пропускную способность, в $n$ раз превышающую пропускную способность одного модуля. Однако, если два адреса запроса памяти попадают в один и тот же банк памяти, возникает конфликт банка, и доступ должен быть сериализован. Графический процессор разделяет запросы доступа к памяти с конфликтами банков на столько отдельных запросов без конфликтов, сколько необходимо, уменьшая эффективную полосу пропускания на коэффициент, равный количеству отдельных запросов к памяти. Для GTX280 размер warp-а равен 32, а количество банков – 16. Доступ к общей памяти для warp-а делится на один запрос для первой половины warp-а и один запрос для второй половины warp-а. Как следствие, не может быть никакого конфликта банка между нитью, принадлежащей первой половине warp-а, и нитью, принадлежащей второй половине того же warp-а.

Общая память также имеет механизм широковещательной передачи, который позволяет считать 32-разрядное слово и транслировать его нескольким нитям одновременно одной транзакцией на чтение из памяти. Это уменьшает количество конфликтов в банке, когда несколько нитей полу-warp-а читают с адреса в пределах одного и того же 32-разрядного слова. Точнее, запрос чтения из памяти, сделанный на несколько адресов, обслуживается в несколько этапов в течение времени – один шаг каждые два такта, обслуживая одно бесконфликтное подмножество этих адресов за шаг, пока все адреса не будут обслужены. На каждом шаге подмножество строится из оставшихся адресов, которые еще предстоит обслуживать, используя следующую процедуру:

1. Выбрать одно из слов, на которые указывают оставшиеся адреса, в качестве транслируемого слова.

2. Включить в подмножество:
   - все адреса, которые находятся в пределах транслируемого слова;
   - один адрес для каждого банка, указанных в оставшихся адресах.

Пространство константной памяти кэшируется, поэтому чтение из константной памяти осуществляется с задержкой как при операции чтения из глобальной памяти только при отсутствии кэша, в противном случае производится транзакция из константной кэш-памяти. Для всех нитей полу-warp-а чтение из константного кэша происходит так же быстро, как чтение из регистров, если все нити читают один и тот же адрес. Время доступа масштабируется линейно в зависимости от количества разных адресов, читаемых всеми нитями.

Текстурная память позволяет кэшировать данные, присутствующие в глобальной памяти. Если запрашивается кэшированный элемент данных, то он обслуживается в одном запросе. Отсутствие кэша приводит к операции чтения глобальной памяти, что занимает гораздо больше времени.

Для сравнения времени доступа к различным типам памяти было произведено 1000000 операций чтения из каждого типа памяти. Доступ выполнялся как последовательно, так и случайным образом. Тесты выполнялись с использованием графического процессора NVIDIA GTX280. Результаты представлены в таблице 2. Как можно заметить, общая память обеспечивает наилучшую производительность, за ней следует кэш-память констант, а затем кэш-память текстур. Глобальная память показывает наибольшую задержку.

**Таблица 2.** Список параметров разработанной модели.

| Тип памяти | Последовательный доступ, мс | Случайный доступ, мс |
|---|---|---|
| Глобальная | 969 | 21777 |
| Общая | 51 | 86.6 |
| Константная | 35 | 192 |
| Текстурная | 140 | 247 |

## 4. Экспериментальное моделирование шаблонов доступа к памяти GPU

Для большинства параллельных вычислительных платформ моделирование шаблонов доступа к памяти и связанных с ними затрат является самой сложной и наиболее важной частью. Для экспериментальной проверки утверждений о процессе доступа к памяти на графическом процессоре используется следующий алгоритм:

Алгоритм 1 Контрольный показатель доступа к глобальной памяти

Входные данные: количество элементов N, шаг stride, смещение offset, массив A в глобальной памяти.

1: Рассчитать количество элементов в потоке, $N_{thread}$;
2: Рассчитать диапазон данных этого потока, используя stride и offset;
3: while index находится в диапазоне do
4: Чтение A[index] в переменную R;
5: Инкрементирование R и сохранение обратно в A[index];
6: index = index + stride;
7: end while

Используя приведенный алгоритм, был смоделирован эксперимент, который показывает, какой выигрыш от совместного доступа зависит от количества нитей в warp-е. Это контролируется переменной stride. Stride обозначает промежуток между элементами, к которым обращаются последовательно одной нитью. Следовательно, нити в полу-warp-е могут получить преимущество от объединенного доступа, если значение stride велико. Например, когда stride = 32, каждая нить warp-а получает последовательные элементы, что обеспечивает полное объединение. Когда stride равен 1, каждая нить считывает по одному элементу, которые смещены на 32, поэтому они полностью не объединены и требуют 16 транзакций памяти, которые будут обслуживаться для полу-warp-а. Чтобы обеспечить справедливое сравнение, в приведенном коде количество обращений по потоку не зависит от stride.

В коде, приведенном в алгоритме 1, количество вычислений на итерацию очень мало по сравнению с задержкой доступа к памяти для stride = 1. Однако, по мере увеличения значения stride, доступ к памяти и вычисления занимают приблизительно одинаковое количество тактов. Используя модель MAX, можно предположить время выполнения этого ядра и сравнить его с фактическим на рисунке 1. График выполнения программы приведен для различных значений stride. Следует отметить, что базовый код доступа только к памяти, т. е. с небольшим количеством вычислений, отличается от модели, из-за ограниченной информации об аппаратном обеспечении [9,10].

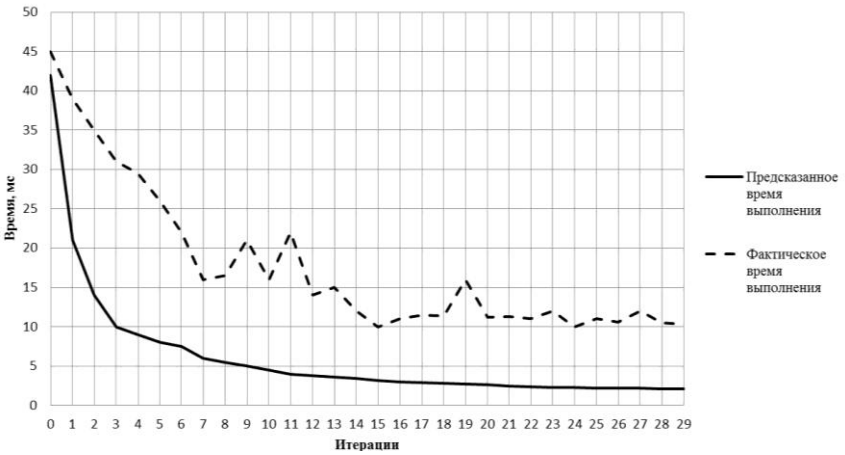


**Рисунок 1.** Результаты экспериментальных исследований моделирования доступа к глобальной памяти с применением модели MAX.

Как видно из рисунка 1, величина stride существенно влияет на время выполнения алгоритма. Т.к. объем операций собственно вычислений в алгоритме очень мал, то по приведенному графику можно видеть, что для увеличения производительности параллельного алгоритма требуется считывать элементы массива, находящиеся на расстоянии друг от друга не менее, чем на величину warp-а для того, чтобы получить преимущество от объединенного доступа к памяти. Также из рисунка 1 видно, что хотя график предсказанного времени выполнения и не совпадает с графиком фактического времени, но при это предсказанное время выполнения позволяет получить достаточно информации о снижении производительности при уменьшении расстояния между считываемыми элементами. Это позволяет утверждать, что представленная выше модель адекватно описывает производительность GPGPU.

## 5. Экспериментальная проверка влияния конфликтов доступа при использовании общей памяти

В этом эксперименте, сохраняя общую структуру глобальных доступов к памяти, как в предыдущем эксперименте, каждый поток записывает элемент в общую память. Шаблон доступа к общей памяти управляется переменной bank, которой может быть задано значение от 0 до 16. При большем значении bank мы можем таким образом увеличить количество конфликтов доступа.

Алгоритм 2 Контрольный показатель доступа к общей памяти

Входные данные: количество элементов N, шаг stride, смещение offset, управляющая переменная bank, массив A в глобальной памяти, массив B в общей памяти

1: Рассчитать количество элементов в потоке, $N_{thread}$;
2: Рассчитать диапазон данных этого потока, используя stride и offset;
3: while index находится в диапазоне do
4: for i = 0 to 10000 do
5: Чтение A[index] и его сохранение;
6: $B[ID_{thread} \times bank \ (mod \ sizeblock)]$;
7: end for

8: end while

Ядро в данном алгоритме имеет около 16 тактов вычисления на итерацию, и есть 64000 итераций. Количество тактов, необходимых для доступа к памяти, составляет около bank×4 за итерацию [11,12,13]. Фактическое время выполнения и время выполнения, предсказанное разработанной моделью, показаны на рисунке 2.

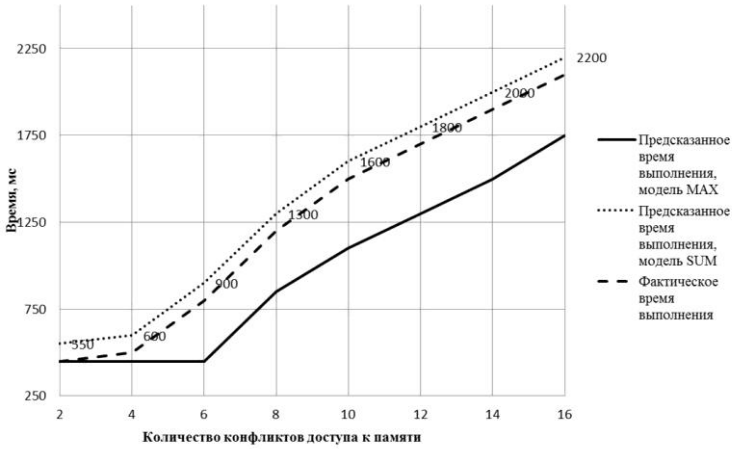


**Рисунок 2.** Результаты экспериментальных исследований влияния конфликтов доступа при использовании общей памяти.

Как видно из рисунка 2, существует линейная зависимость количества конфликтов от времени выполнения программы. Таким образом, для увеличения производительности параллельных алгоритмов требуется исключить пересечение считываемых данных для разных нитей. Также из рисунка 2 можно сделать вывод, что представленная модель позволяет адекватно оценить время выполнения алгоритма при наличии конфликтов памяти, хотя и не дает точной информации из-за закрытой аппаратной платформы.

## 6. Выводы

В данной работе представлено экспериментальное исследование ранее разработанных моделей прогнозирования производительности гетерогенной компьютерной системы в телекоммуникациях. Исследование показало, что разработанные модели позволяют получить адекватную оценку возможного времени работы алгоритма при различных параметрах работы GPU. Однако стоит заметить, что оценка, полученная с использованием разработанных моделей, не является точной, т.к. используется среднее время доступа для всех уровней иерархии памяти. В дальнейшем планируется доработка моделей с учетом использования величины времени доступа к памяти, подчиняющейся ранговому распределению, например, Парето или Зипфа.

## 7. Литература

[1] Owens, J.D. A survey of general-purpose computation on graphics hardware / J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, T.J. Purcell // Computer Graphics Forum. – 2007. – Vol. 26(1). – P. 80-113.

[2] Колпаков, А.А. Разработка модели прогнозирования производительности гетерогенной компьютерной системы в телекоммуникациях / А.А. Колпаков, Ю.А. Кропотов // Сборник трудов международной конференции и молодежной школы «Информационные технологии и нанотехнологии» (ИТНТ) – Самара: Новая техника, 2018. – С. 2265-2274.

[3] Fortune, S. Parallelism in Random Access Machines / S. Fortune, J. Wyllie // Proceedings of 10th Annual ACM Symposium on Theory of Computing (STOC) – ACM New York, USA, 1978. – P. 114-118.

[4] Kropotov, Y.A. Experimental study of the model for predicting the performance of a heterogeneous computer system in telecommunications / Y.A. Kropotov, A.A. Kolpakov // 12th

International Scientific and Technical Conference "Dynamics of Systems, Mechanisms and Machines", 2019. DOI: 10.1109/Dynamics.2018.8601478.
[5] Valiant, L.G. A Bridging Model for Parallel Computation // Communications of the ACM. – 1990. – Vol. 33(8). – P. 103-111.
[6] Gibbons, P.B. The Queue-Read Queue-Write PRAM Model: Accounting for Contention in Parallel Algorithms / P.B. Gibbons, Y. Matias, V. Ramachandran // SIAM Journal of Computation. – 1999. – Vol. 28(2). – P. 733-769.
[7] CUDA C Programming Guide [Электронный ресурс]. – Режим доступа: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (20.11.2017).
[8] Helman, D.R. Designing Practical Efficient Algorithms for Symmetric Multiprocessors / D.R. Helman, J. JaJa // Lecture Notes in Computer Science, International Workshop ALENEX. – 1999. – Vol. 1619. – P. 37-56.
[9] Kolpakov, A.A. Advanced mixing audio streams for heterogeneous computer systems in telecommunications / A.A. Kolpakov, Y.A. Kropotov // CEUR Workshop Proceedings. – 2017. – Vol. 1902. – P. 32-36.
[10] Колпаков, А.А. Теоретическая оценка роста производительности вычислительной системы при использовании нескольких вычислительных устройств // В мире научных открытий. – 2012. – № 1. – С. 206-209.
[11] Кропотов, Ю.А. Вопросы обработки экспериментальных временных рядов в электронной системе автоматизированного контроля / Ю.А. Кропотов, А.А. Белов, А.Ю. Проскуряков // Вопросы радиоэлектроники. – 2010. – Т. 1, № 1. – С. 95-101.
[12] Кропотов, Ю.А. Алгоритм определения параметров экспоненциальной аппроксимации закона распределения вероятности амплитуд речевого сигнала / Ю.А. Кропотов // Радиотехника. – 2007. –№ 6 – С. 44-47.
[13] Кропотов, Ю.А. Модель закона распределения вероятности амплитуд сигналов в базисе экспоненциальных функций системы / Ю.А. Кропотов, А.А. Быков // Проектирование и технология электронных средств. – 2007. – № 2. – С. 30-34.

# Investigation of the RAM access model in a heterogeneous computing system

**A.A. Kolpakov[1], Yu.A. Kropotov[1], A.A. Belov[1]**

[1]State University named after Alexander and Nicholay Stoletovs, Orlovskaya street 23, Murom, Russia, 602264

**Abstract.** The issue of creating high-performance computing systems based on heterogeneous computer systems is topical, since the volumes of processed information, calculations and studies with large data sets are constantly increasing. The aim of the work is an experimental study of previously developed models for predicting the performance of heterogeneous computer systems in telecommunications. As a result, the study showed that the developed models allow us to obtain an adequate estimate of the possible time of the algorithm for various parameters of the GPU with some limitations.