Разработка модели прогнозирования производительности гетерогенной компьютерной системы в телекоммуникациях

А.А. Колпаков[1], Ю.А. Кропотов[1]

[1]Владимирский государственный университет имени Столетовых, ул. Орловская, 23, Муром, Владимирская область, 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 разработка модели прогнозирования производительности гетерогенных компьютерных систем в телекоммуникациях. В результате использование разработанной модели позволяет произвести адекватную оценку времени выполнения распараллеленной задачи с использованием гетерогенных компьютерных систем на базе графических процессоров.

1. Введение

Параллельное программирование стало темой исследований с самого развития современных компьютеров. Эта все возрастающая потребность в решении больших проблем стимулирует исследования и инновации в области параллельных вычислений.

Коммерчески параллельные вычисления появились в персональных компьютерах с появлением многоядерных процессоров – процессоров, которые содержат более одного процессора и могут запускать приложения параллельно. Эти системы обычно совместно используют одно и то же системную оперативную память и управляются операционной системой для параллельного запуска нескольких потоков.

2. Развитие гетерогенных компьютерных систем на основе графических процессоров

Общая память и многоядерные архитектуры сегодня являются основным направлением разработки для создания более быстрых и более эффективных приложений, которые могут использовать все большее число ядер на процессор. Существует ряд программных средств и сред, используемых для программирования этих новых процессоров, – это OpenMP, Intel Thread Building Blocks и т. д. Большое количество исследований ведется в сторону разработки более простых и более мощных парадигм программирования для программирования этих многоядерных чипов. Кроме того, основная цель разработки новых коммерческих операционных систем, таких как Microsoft Windows 10 и macOS 10.13 High Sierra, заключается в улучшении многопоточной производительности и предоставлении разработчикам возможности полного использования многоядерных чипов, включая графические процессоры.

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.

3. Разработка модели прогнозирования производительности GPGPU

Модель, которая приведена в данной работе, 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 [2];
2. Модель BSP [3];
3. Модель QRQW [4].

Программы CUDA записываются в единицах, называемых ядрами. Нити начинаются синхронно в начале каждого ядра и синхронизируются в конце каждого ядра. Таким образом, основной единицей синхронизации в программе CUDA является ядро. Это очень близко подходит к модели параллельных вычислений BSP с неявным вызовом для синхронизации в конце каждого ядра. Однако следует обратить внимание, что в то время как в модели BSP синхронизация выполняется через регулярные интервалы в L единиц времени, представленная модель устраняет это требование. Учитывая отсутствие какой-либо инфраструктуры маршрутизации в графическом процессоре, модель BSP используется только в том, что касается понятия супершагов [3].

Следует обратить внимание, что основным элементом вычислений в программе GPU является нить в ядре. Графический процессор не является унифицированной архитектурой. Время, затрачиваемое вычислительными операциями, может варьироваться от 4 тактов для простого добавления до 16 тактам для 32-разрядного целочисленного умножения и многих других для целочисленного модуля. Таким образом, чтобы получить лучшие результаты, нужно учитывать требование такта для вычислительных операций в потоке.

Следовательно, в разрабатываемой модели предлагается получить такты, необходимые для вычисления в потоке. Для этого можно использовать опубликованные архитектурные детали, чтобы увидеть такты, необходимые для каждой операции, и добавить их. Например, если поток имеет два целочисленных суммирования и два умножения, то для этого требуется 72 такта [5]. Число тактов также можно получить как функцию входного размера, как это сделано в типичном асимптотическом анализе. Таким образом, пусть N_{comp} – это количество тактов, необходимых для вычисления в потоке.

В графическом процессоре имеется глубокая иерархия памяти с большими вариациями времени доступа для каждого уровня иерархии памяти. Следовательно, для оценки времени, затрачиваемого потоком на чтение глобальной памяти, нужно быть более осторожным. Основными видами памяти GPU являются глобальная память и разделяемая память.

Чтение/запись из/в ячейку из глобальной памяти занимает 400-600 тактов [5]. В данной работе принимается среднее значение – 500 тактов за чтение.

На графическом процессоре нити выполняются как пакет, называемый warp. GPU обращается к глобальной памяти в смежных кусках по 128 байтов, называемых сегментами. Нити в halfwarp, т. е. половина числа нитей в warp-е, выполняются одновременно без влияния извне. В этом случае одна транзакция считывания сегмента из глобальной памяти выполняется для обслуживания всех нитей half-warp-а. Если транзакция обслуживает t потоков в warp-е, тогда среднее отношение стоимости доступа для этих t потоков в этой ситуации может быть принята равной $(500 + t) / t$.

GPU обеспечивает общую память для нитей, которая идеально подходит для часто используемых переменных, которые необходимы нитям. Это низкоприоритетная память в иерархии, на нее отводится около 4 тактов на доступ, но имеется несколько ограничений. Общая память имеет очень маленький размер (16 КБ на потоковый мультипроцессор (streaming multiprocessor, SM)) и должна использоваться всеми потоками, запланированными на SM. Кроме того, если несколько потоков одновременно обращаются к одному и тому же банку в общей памяти, это приводит к конфликту памяти, что может увеличить стоимость доступа.

В разрабатываемой модели подразумевается, что если обращения, производимые нитями, не детерминированы, а рандомизированы, то с высокой вероятностью будет присутствовать определенное количество конфликтов, что необходимо учесть для оценки стоимости доступа к общей памяти.

Итак, пусть N_{memory} – это количество тактов, необходимых для всех обращений к памяти нитью, причем это значение включает стоимость как доступа к глобальной памяти, так и к общей памяти нити в ядре.

Следовательно, применяется следующий подход. Пусть $C(T)$ – количество тактов, необходимых нити. Лучшим решением для планирования является полное скрытие задержек. Таким образом, количество тактов, требуемых нити, равно $C(T) = \max\{N_{comp}; N_{memory}\}$. Этот случай в рамках представленной работы обозначается как модель MAX. Если планирование не применяется, то количество тактов, требуемых нити, равно $C(T) = N_{comp} + N_{memory}$. Этот случай обозначается как модель SUM. В любом случае наличие 4-этапного конвейера в каждом ядре графического процессора оказывает отдельное влияние, которое анализируется ниже.

Таким образом, время, затраченное программой P на выполнение на GPU, обозначается как T(P). Модель BSP позволяет нам рассматривать это время как сумму времени в разных ядрах. Таким образом, с учетом программы CUDA P с r ядрами K_1; K_2; …; K_r, формула вычисления времени выполнения, затраченного программой P на выполнение на GPU, выглядит следующим образом:

$$T(P) = \sum_{i-1}^{r} T(K_i) \text{ секунд} \quad (1)$$

где $T(K_i)$ обозначает время, затраченное ядром K_i.

Для ядра K мы теперь должны рассмотреть модель исполнения GPU. Напомним, что блоки назначаются SM и каждый блок состоит из N_w warp-ов. Каждый warp состоит из нитей N_t и нити в каждом warp-е выполняются параллельно. Нужно также учитывать тот факт, что каждое из N_c ядер в SM на графическом процессоре имеет D-этапный конвейер, который влияет на выполнение D нитей параллельно.

Кроме того, также необходимо оценить время, требуемое для выполнения такта для одного потока. Это можно сделать, оценивая время вычислений и доступа к памяти. Предполагается, что количество тактов, используемых ядром, равно максимуму, используемому некоторой нитью в этом ядре. Пусть максимальное количество тактов, используемых любой нитью, выполняющейся ядром K, является $C_T(K)$. Таким образом, $C_T(K)$ можно выразить как максимум по всем $C(T)$ для нити T, выполняющейся ядром K. Следовательно,

$$C_T(K) = \max_T C(T) \quad (2)$$

Обратите внимание, что если используется модель MAX (SUM), то параметр CT(K) в приведенном выше случае должен быть получен с использованием модели MAX (соответственно SUM).

Наконец, время, затрачиваемое на выполнение ядра K, оценивается следующим образом. Пусть $N_B(K)$ – количество блоков, назначенных каждому SM последовательно в ядре K, $N_w(K)$ – количество warp-ов в каждом блоке в ядре K, $N_t(K)$ – количество нитей в warp-е в ядре K. Тогда число тактов, необходимых для ядра K, обозначаемое $C(K)$, равно:

$$C(K) = N_B(K) \cdot N_w(K) \cdot N_t(K) \cdot C_T(K) \cdot \frac{1}{N_C \cdot D} \quad (3)$$

Чтобы преобразовать такты в секундах, необходимо умножить формулу (3) на тактовую частоту GPU, как в приведенном ниже уравнении, где R – тактовая частота ядра GPU.

$$T(K) = \frac{C(K)}{R} \quad (4)$$

Это приводит к окончательному уравнению

$$T(K) = \frac{N_B(K) \cdot N_w(K) \cdot N_t(K) \cdot C_T(K)}{N_C \cdot D \cdot R} \quad (5)$$

Поскольку каждое ядро может иметь различную структуру по числу блоков, warp-ов на каждый блок и т. д., эти величины определяются в соответствии с ядром.

Все параметры разработанной 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 неклассифицированным 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 [5]. Любая модель, которая не фиксирует эти изменения, вряд ли будет точной.

4. Экспериментальная проверка моделирования доступа к памяти

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, Nthread;

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 halfwarp-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 halfwarp-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 к чисто памяти, т. е. с небольшим количеством вычислений, отличается от модели, из-за ограниченного знания аппаратного обеспечения доступа к памяти.

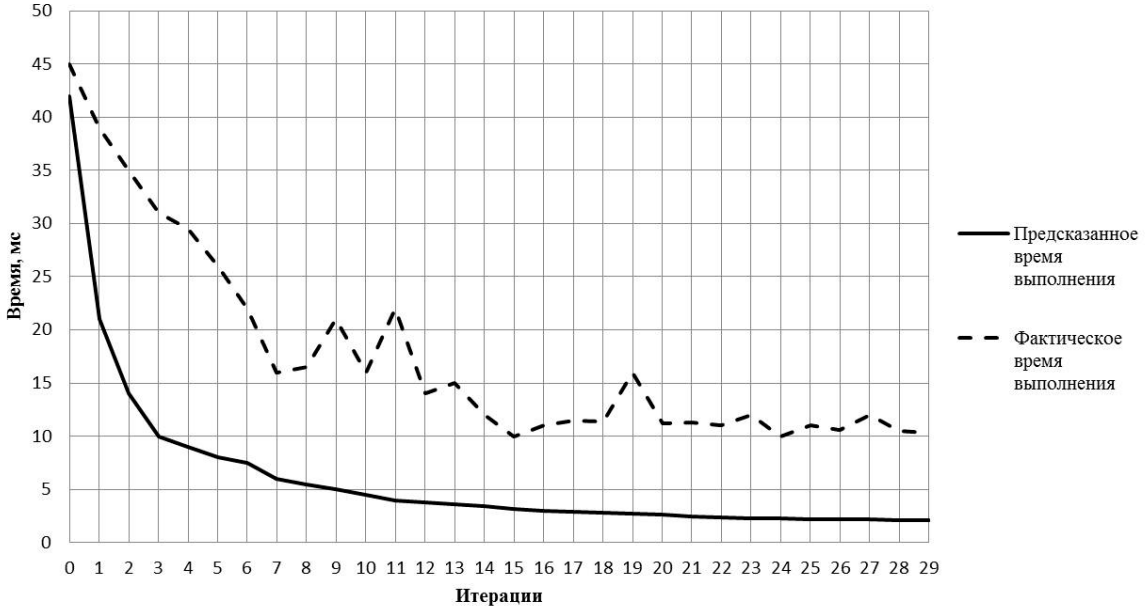

Рисунок 1. Результаты экспериментальных исследований моделирования доступа к глобальной памяти с применением модели MAX.

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, Nthread;
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[IDthread × 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. Фактическое время выполнения и время выполнения, предсказанное разработанной моделью, показаны на рисунке 2. Как видно, существует линейная зависимость количества конфликтов от времени выполнения программы.

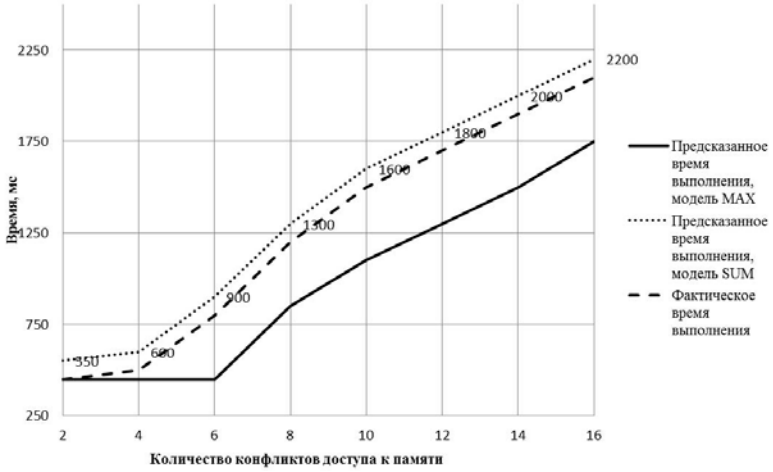

Рисунок 2. Результаты экспериментальных исследований влияния конфликтов доступа при использовании общей памяти.

6. Экспериментальное исследование фундаментальных параллельных алгоритмов

В качестве экспериментальных задач рассматриваются следующие задачи: умножение матриц, сортировка списка и генерация гистограммы. Эти задачи охватывают все особенности разработанной модели. Ядро, реализующее умножение матриц, вычислительно интенсивно, ядро, реализующее ранжирования списков, интенсивно использует глобальную память и является популярным примером для нерегулярных алгоритмов. Ядро, реализующее генерацию гистограммы, использует общую память, что приводит к конфликтам доступа. Следовательно, выбор экспериментальных задач обоснован. Поскольку в каждом из экспериментов анализируется только одно ядро, для простоты при вычислениях отбрасывается параметр K в таких переменных, как $N_B(K)$, и просто используется параметр NB.

Умножение матриц является очень популярной задачей в параллельных вычислениях. В экспериментальной задаче применяется алгоритм, рассмотренный в работе [37, глава 6].

Для умножения матриц размера $N \times N$ общее число блоков должно быть равно $N^2/256$, каждый блок должен состоять из $N_w = 8$ warp-ов, каждый warp состоять из $N_t = 32$ нитей. Таким образом, каждый SM имеет $N_B = N^2/(256 \cdot 30)$.

В соответствии с вышесказанным [5], работа, выполняемая каждой нитью, масштабируется в зависимости от количества строк и столбцов. Размеры блока нитей составляют 16 × 16. Каждая нить загружает значение из матриц A и B в общую память, итеративно вычисляет

каждый элемент C_{sub} и записывает его обратно в память. Для этого требуется $N/16$ итераций для каждой нити. Количество тактов вычислений, необходимых для каждой нити для матрицы $N \times N$, можно подсчитать как $N_{comp} = 760N/16$. Поскольку каждая нить выполняет три доступа к глобальной и два к общей памяти на каждый вычисленный элемент, такты, потраченные на операции с памятью в этой нити можно считать равным $N_{memory} = 240N/16$.

Таким образом, при использовании модели MAX время вычислений превышает время доступа к памяти. Пусть $C_T = \max\{N_{comp}; N_{memory}\}$. Используя формулу 4, общее время, необходимое для умножения матриц в модели MAX, получается следующее:

$$\left\lceil \frac{N^2}{256 \cdot 30} \right\rceil \cdot 8 \cdot 32 \cdot \frac{760N}{16} \cdot \frac{1}{32 \times 1.3 \times 10^9} \text{ секунд.}$$

При $N = 128$ расчетное время с использованием модели MAX составляет около 0,11 мс, что выгодно отличается от фактического времени 0,16 мс. Прогнозируемое время выполнения этого алгоритма для моделей SUM и MAX для различных значений N приведено на рисунке 3. Умножение матриц требует блочной синхронизации, которая может быть предсказана, и, следовательно, существует некоторое отклонение от фактического времени выполнения.

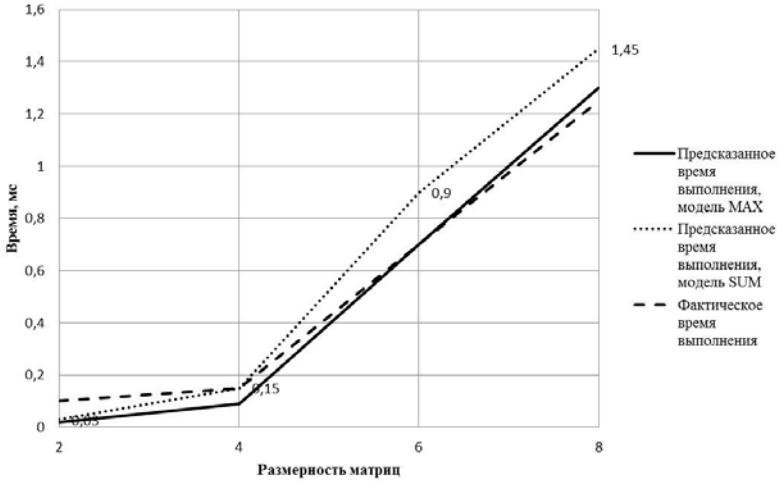

Рисунок 3. Результаты экспериментальных исследований алгоритма умножения матриц размером 2^n.

В параллельных вычислениях сортировка списков является одной из фундаментальных операций. Для симметричных мультипроцессоров в работе [6] предложен алгоритм для сортировки списков, который имеет сложность O(log$N$) для списка из N элементов и обладает высокой вероятностью удачной сортировки, когда число процессоров невелико по сравнению с размером входного списка.

При списке в N элементов и количеством разделителей $p = N/\log N$ требуется $N/\log N$ нитей. Эти нити сгруппированы в $N/512\log N$ блоков по 512 нитей каждый. Из них не более $N_B = [N/512 \times 30 \times \log N]$ блоков назначаются любому одиночному SM на GPU. Каждый из этих блоков состоит из $N_w = 16$ warp-ов на $N_t = 32$ нити каждый.

Используя случайный список в качестве входного, вполне вероятно, что некоторые нити обрабатывают больше элементов по сравнению с другими [7,8]. Как правило, наиболее вероятным размером подсписка является $4\log N$ элемента. Такты доступа к памяти, используемые нитью, могут быть вычислены следующим образом: каждая нить включает три чтения/записи в глобальную память для каждого элемента. Все эти обращения, как правило, не объединены. Таким образом, $N_{\text{memory}} = 4\log N \times 3 \times 500$. Количество вычислений в каждой нити очень минимально. Поэтому возможно полностью игнорировать N_{comp} и установить $C_T(K) = N_{memory}$ как в модели SUM, так и в MAX.

Общее время, затрачиваемое ядром на вычисление локальных рангов для каждого подсписка, может быть вычислено с использованием формулы 3.4 как:

$$\left\lceil \frac{N}{512 \cdot 30 \cdot \log N} \right\rceil \cdot 16 \cdot \frac{32}{8 \times 4} \cdot 4 \log N \cdot 3 \cdot 500 \cdot \frac{1}{1.3 \times 10^9} \text{ секунд.}$$

Для $N = 2^{22}$ мы получаем время на SM равным приблизительно 21 миллисекунде. Это выгодно отличается от фактического времени 24 миллисекунды. На рисунке 4 показано сравнение оцениваемого и фактического времен по различным размерам списка, от 256K до 4M элементов.

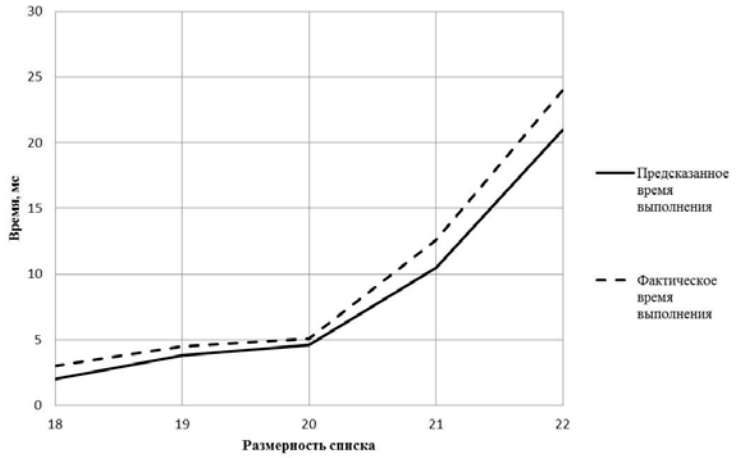

Рисунок 4. Результаты экспериментальных исследований алгоритма сортировки списка размером от 2^{18} до 2^{22}.

Стоит отметить, что поскольку вычисления в каждом потоке очень минимальны по сравнению с временем доступа к памяти, модели MAX и SUM демонстрируют одинаковое поведение. Поэтому показаны только оценки модели MAX [9,10].

Подсчет элементов той же категории является распространенной проблемой, распространяющейся на широкий спектр задач [11]. Это один из основных примитивов в статистике, обработке изображений и данных. Пусть N наблюдений выбираются независимо и равномерно случайным образом между 1 и B, включительно. Предположим, что эти N наблюдений должны быть помещены в ячейки B.

В используемой реализации каждая нить строит локальную гистограмму элементов N = (1920 × 256) на 256 ячеек. Эта операция включает в себя считывание каждого элемента из глобальной памяти объединенным способом и обновление счетчика в соответствующей переменной в общей памяти. Таким образом, объемы вычислений и доступа к памяти для каждого элемента очень малы.

Используя формулу 5, получается оценка времени выполнения согласно вариантам MAX и SUM. Фактическое и расчетное время приведены на рисунке 5.

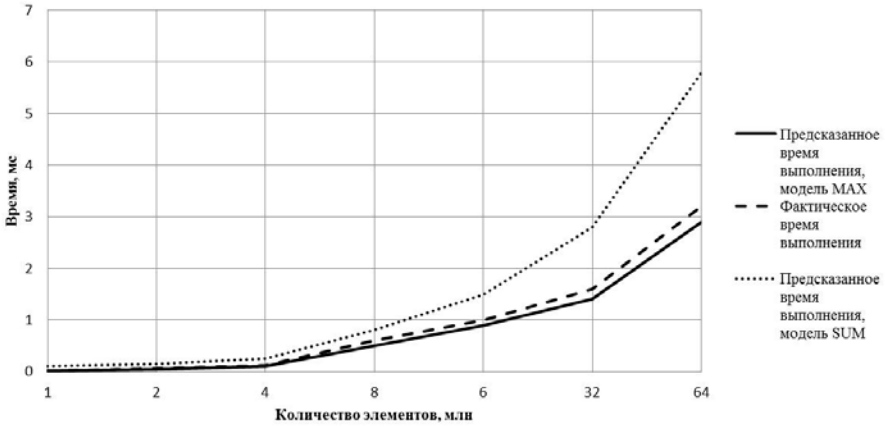

Рисунок 5. Результаты экспериментальных исследований построения гистограмм.

7. Выводы

В данной работе представлена модель прогнозирования производительности гетерогенной компьютерной системы в телекоммуникациях. Главное ее достоинство – это адекватная оценка возможного времени работы алгоритма при различных параметрах работы GPU, что позволяет оценить время выполнения всей задачи в целом без необходимости проведения экспериментальных исследований. Стоит заметить, что оценка, полученная с использованием разработанной модели, могла бы быть более точной, однако для этого требуется информация об аппаратной реализации механизмов работы графического процессора, которая, к сожалению, производителем не предоставляется.

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

[3] Valiant, L.G. A Bridging Model for Parallel Computation / L.G. Valiant // Communications of the ACM. – 1990. – Vol. 33(8). – P. 103-111.

[4] 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.

[5] CUDA C Programming Guide [Электронный ресурс]. – Режим доступа: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (20.11.2017).

[6] Helman, D.R. Designing Practical Efficient Algorithms for Symmetric Multiprocessors / D.R. Helman, J. JaJa // Lecture Notes in Computer Science 1619, International Workshop ALENEX’99. – 1999. – P. 37-56.

[7] 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.

[8] Колпаков, А.А. Теоретическая оценка роста производительности вычислительной системы при использовании нескольких вычислительных устройств / А.А. Колпаков // В мире научных открытий. – 2012. – №1. – С. 206-209.

[9] 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.

[10] Кропотов, Ю.А. Алгоритм определения параметров экспоненциальной аппроксимации закона распределения вероятности амплитуд речевого сигнала / Ю.А. Кропотов // Радиотехника. – 2007. – №6. – С. 44-47.

[11] 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.

Development of a model for predicting the performance of a heterogeneous computer system in telecommunications s

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

[1]Vladimir State University named after Alexander and Nicholay Stoletovs, Orlovskaya street, 23, Murom, Vladimir Region, 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 to develop a model for predicting the performance of heterogeneous computer systems in telecommunications. As a result, the use of the developed model allows an adequate estimation of the execution time of a parallel task using heterogeneous computer systems based on graphics processors.

Keywords: graphic processor units, heterogeneous computing systems, parallel calculations.