ТЕХНИЧЕСКИЕ НАУКИ

УДК 621.397.01

Ю.А. Затолокин, магистрант, ФГБОУ ВО «Юго-Западный государственный университет» (Курск, Россия) (e-mail: yuzato@list.ru)

Э.И. Ватутин, канд. техн. наук, доцент, ФГБОУ ВО «Юго-Западный государственный университет» (Курск, Россия) (e-mail: evatutin@rambler.ru)

В.С. Титов, д-р техн. наук, профессор, ФГБОУ ВО «Юго-Западный государственный университет» (Курск, Россия) (e-mail: titov-kstu@rambler.ru)

АЛГОРИТМИЧЕСКАЯ ОПТИМИЗАЦИЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ АЛГОРИТМОВ УМНОЖЕНИЯ ПЛОТНЫХ ВЕЩЕСТВЕННЫХ МАТРИЦ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ С ПОДДЕРЖКОЙ ТЕХНОЛОГИИ OpenGL

Приведено описание подходов к выполнению операции умножения плотных вещественных матриц одинарной точности на видеокартах с поддержкой технологии OpenCL. Произведен обзор известных подходов к алгоритмической оптимизации процедуры умножения матриц и оценка возможности их использования с учетом особенностей организации и программирования для GPU. Сделан сравнительный анализ производительности выполняемых действий без характерных для GPU оптимизаций и с оптимизациями, который показал, что вычисления без оптимизации работы с глобальной памятью GPU имеют низкую производительность обработки данных. Оптимизация распределения данных в глобальной и локальной памяти GPU позволяет многократно сократить время вычисления и увеличить реальную производи-тельность. Для сравнения производительности разработанных программных реализаций для технологий OpenCL и CUDA выполнены идентичные расчёты на одинаковых GPU, которые показали более высокую реальную производительность при использовании CUDA-ядер. Значения производительности оценивались для всех реализаций процедуры умножения матриц. Сравнение полученных результатов показывает, что наиболее эффективным подходом среди реализованных является блочное умножение, при котором производится разделение исходной матрицы на подматрицы (блоки), размещаемые в локальной памяти GPU, что позволяет экономить обращения к глобальной памяти и максимально повторно использовать данные в локальной памяти. Результаты измерения реальной достигнутой производительности на GPU NVidia GeForce GTX 960M показали величину 275,3 GFLOP/s, что приблизительно на 10–20% меньше аналогичных результатов, получаемых при аналогичных условиях вычислительного эксперимента для той же GPU с использованием инструментария CUDA.

Ключевые слова: *умножение матриц, алгоритмическая оптимизация, OpenGL, CUDA.*

DOI: 10.21869/2223-1560-2017-21-5-06-15

Ссылка для цитирования: Затолокин Ю.А., Ватутин Э.И., Титов В.С. Алгоритмическая оптимизация программной реализации алгоритмов умножения плотных вещественных матриц на графических процессорах с поддержкой технологии OpenGL // Известия Юго-Западного государственного университета. 2017. Т. 21, № 5(74). С. 6-15.

***

Задача нахождения произведения плотных матриц встречается в ряде научно-технических направлений (геометрическое моделирование, современная ускорительная техника, проектирование роботизированных средств, классификация бинарных отношений [1], численное решение дифференциальных уравнений, обработка сигналов и др.). Операции над матрицами используются при математическом моделировании разнообразных процессов, систем и явлений.

При обработке больших объемов данных повышаются требования к времени нахождения произведения матриц. Зачастую время, затрачиваемое на выполнение расчетов с матрицами, является определяющим фактором, от которого зависит общее время решения поставленной прикладной задачи в целом. Оптимизация

выполняемых действий, включающая различные алгоритмические приемы и распараллеливание части выполняемых операций, позволяет существенно сократить время умножения матриц на современных вычислительных средствах.

Анализ эффективности различных подходов выполнен с помощью решения задачи общего вида (умножения квадратных матриц размера $N \times N$ из вещественных чисел одинарной точности) с использованием графических процессоров, поддерживающих технологию OpenGL [3] (в рамках концепции GPGPU).

Выполнение операций умножения матриц на GPU имеет свою специфику и подразделяется на три этапа [2]:

1. Исходные матрицы *A* и *B* передаются из оперативной памяти в глобальную память GPU.

2. Выполняются операции с матрицами на GPU.

3. Для получения результирующей матрицы C данные передаются из глобальной памяти GPU в оперативную память.

Реальная производительность обработки [6] оценивается по формуле $P = \frac{V}{t}$, где $V = 2N^3 - N^2 \approx 2N^3$ – объем выполняемых вычислений; t – суммарное время, включающее выполнение операций с матрицами и обмен данными.

Расчёт произведения матриц будем выполнять на платформе OpenGL [4]. Данная платформа предназначена для написания программ, связанных с параллельными вычислениями, и задумывалась для создания приложений, которые работали бы в гетерогенной среде. Поэтому стандарт OpenGL предоставляет унифицированный API, позволяющий работать с вычислительными устройствами независимо от их архитектуры.

Для сравнения производительности платформы OpenGL в поставленной задаче так же выполнены замеры производительности вычислений аналогичных алгоритмов для матриц соответствующей размерности на платформе CUDA [2].

Вычисление произведения матриц выполнено в соответствии с архитектурой OpenGL [7]. Управляющее устройство (host), в роли которого выступает центральный процессор (CPU), содержит специально оформленную текстовую строку кода (ядро или kernel), которая после компиляции средствами OpenGL будет выполнена в процессе работы программы (runtime) на выбранном в данный момент вычислительном устройстве (device). В нашем случае вычислительным устройством будет GPU. Производительность вычисления произведения матриц для CPU была получена на базе методики [6].

Вычисление произведения матриц без оптимизаций выполняется широко известным параллельным алгоритмом умножения. Код OpenGL ядра, использующего этот алгоритм, приведён ниже.

```
__kernel void matrixMultiplica-
tion(const __global float* A, const
__global float* B, __global float* C,
const int n)
{
const int i = get_global_id(0);
const int j = get_global_id(1);

float s = 0.0f;
for (int k = 0; k<n; k++)
     s += A[i*n + k] * B[k*n + j];

     C[i*n + j] = s;
}
```

Выполнение этого OpenGL ядра запускается с размерностью work-group, равной 32 для тестовых GPU, в качестве которых выбраны GeForce GTX 960M и

16 для AMD Radeon R7 200. Выбрано максимальное количество work-item (поток в терминологии CUDA) в work-group (блок в терминологии CUDA). Разница в значениях обусловлена аппаратными ограничениями используемых видеокарт.

Таблица 1

Результаты сопоставления производительности обработки на CPU и GPU для реализации без оптимизации, CPU Intel Core i7-4750HQ + GPU GeForce GTX 960M

N×N, объем данных	t_{CPU}, P	$t_{GPU\ CUDA}$, P	$t_{GPU\ OPENGL}$, P	Выигрыш в производительности (раз) CPU:GPU-CUDA / CPU:GPU-OpenGL / GPU-CUDA:GPU-OpenGL
256×256 (2×256 КБ)	0,06 с 0,56 GFLOP/s	0,005 с 6,2 GFLOP/s	0,008 с 4,4 GFLOP/s	11 / 7,9 / 1,4
512×512 (2×1 МБ)	0,63 с 0,43 GFLOP/s	0,03 с 9 GFLOP/s	0,03 с 7.9 GFLOP/s	21 / 18,4 / 1,1
1024×1024 (2×4 МБ)	9,06 с 0,24 GFLOP/s	0,2 с 10,5 GFLOP/s	0,2 с 10,4 GFLOP/s	43,8 / 43,3 / 1
2048×2048 (2×16 МБ)	87,15 с 0,2 GFLOP/s	1,53 с 11,2 GFLOP/s	1,55 с 11,1 GFLOP/s	56 / 55,5 / 1

Полученные результаты позволяют сделать вывод о том, что производительность современных видеокарт в исследуемой задаче превышает производительность одного ядра CPU. Сравнение результатов относительно выполняемых расчетов на данном GPU с помощью платформ OpenGL и CUDA (табл. 1) позволяет сделать вывод, что без оптимизации алгоритма расчета эти две технологии показывают приблизительно сопоставимый результат (в пределах погрешности измерений).

Для оптимизации вычислений при обработке на CPU используется буферизация обрабатываемого *j*-го столбца матрицы *B* [2]. Для OpenGL платформы кеширование *j*-го столбца матрицы B будем производить в быстрой локальной памяти work-group. Выполняемое ядро имеет следующий вид:

```
 __kernel void matrixMultiplica-
tion(const __global float* A, const
 __global float* B, __global float* C,
const int n)

 {
 int j = get_group_id(0);
 int i = get_local_id(0);
 __local float t[BS];
 t[i] = B[i*n + j];  // None coalesced
доступ к памяти
 barrier(CLK_LOCAL_MEM_FENCE);
 float s = 0.0f;
 for (int k = 0; k < n; k++)
    s += A[i*n + k] * t[k];
 C[i*n + j] = s;
 }
```

Результаты изменения времени обработки и достигнутой при этом реальной производительности приведены в таблице 2.

Таблица 2

Результаты буферизованного умножения с кэшированием столбца на GPU, CPU Intel Core i7-4750HQ + GPU GeForce GTX 960M

N	$t_{GPU\ CUDA}$, *P*	$t_{GPU\ OPENGL}$, *P*	Разница, раз
256×256	0,005 с 6,3 GFLOP/s	0,01 с 3 GFLOP/s	2,1
512×512	0,03 с 9,1 GFLOP/s	0,36 с 7,5 GFLOP/s	1,2
1024×1024	0,21 с 10,4 GFLOP/s	0,21 с 10,3 GFLOP/s	1

Анализ полученных результатов показывает, что данная оптимизация вычислений не даёт видимого результата. Виной этому является неоптимизированный доступ к памяти (англ. none coalesced). Вычисление на платформе CUDA обеспечивает б**о**льшую производительность по сравнению с OpenGL.

Для оптимизации обращения к локальной памяти также был реализован алгоритм умножения с кешированием i-ой строки матрицы A, соответствующее ядро которого приведено ниже.

```
__kernel void matrixMultiplica-
tion(const __global float* A, const
__global float* B, __global float*
C, const int n){
   int i = get_group_id(0);
   int j = get_local_id(0);
   __local float t[BS];
   t[j] = A[i*n + j];  // Coalesced
доступ к памяти
   barrier(CLK_LOCAL_MEM_FENCE);
   float s = 0.0f;
   for (int k = 0; k < n; k++)
         s += t[k] * B[k*n + j];
   C[i*n + j] = s;
}
```

Результаты использования данного OpenGL-ядра приведены в таблице 3.

Таблица 3

Результаты буферизованного умножения с кэшированием строки на GPU, CPU Intel Core i7-4750HQ + GPU GeForce GTX 960M

N	$t_{GPU\ CUDA}$, P	$t_{GPU\ OPENGL}$, P	Разница
256×256	0,003 с 10,5 GFLOP/s	0,008 с 4,4 GFLOP/s	2,4
512×512	0,006 с 45,2 GFLOP/s	0,01 с 25 GFLOP/s	1,8
1024×1024	0,025 с 85,9 GFLOP/s	0,037 с 57 GFLOP/s	1,5

Оптимизированный доступ к глобальной памяти для work-item показывает выигрыш в производительности по сравнению с двумя предыдущими алгоритмами.

Повысить реальную производительность обработки можно, применив алгоритм блочного умножения [2]. Код соответствующего OpenGL-ядра приведён ниже.

```
__kernel void matrixMultiplica-
tion(const __global float* A, const
```

```
__global float* B, __global float* C,
const int n){
int bx = get_group_id(0);
int by = get_group_id(1);
int tx = get_local_id(1);
int ty = get_local_id(0);
int x0 = bx*BS;
int y0 = by*BS;
float sum = 0.0f;
for (int z = 0; z < n / BS; z++)
{
__local float ta[BS][BS];
__local float tb[BS][BS];

int zb = z*BS;
ta[tx][ty] = A[(x0 + tx)*n + (zb + ty)];
tb[tx][ty] = B[(zb + tx)*n + (y0 + ty)];
// Синхронизация загрузки данных всеми
work-item в work-group
barrier(CLK_LOCAL_MEM_FENCE);
// Умножение
for (int k = 0; k < BS; k++)
sum += ta[tx][k] * tb[k][ty];
barrier(CLK_LOCAL_MEM_FENCE);
}
C[(x0 + tx)*n + (y0 + ty)] = sum;
}
```

Выполнение умножения с максимальным размером блока дает лучшую производительность вычислений [2] вследствие оптимизации работы подсистемы быстрой локальной памяти GPU и кэш-памяти CPU. Для платформы OpenGL был организован схожий эксперимент со следующими результатами в сравнении с CUDA (табл. 4).

Таблица 4

Результаты блочного умножения с размером блока 32 на GPU (CPU Intel Core i7-4750HQ + GPU GeForce GTX 960M)

N	$t_{GPU\ CUDA}$, P	$t_{GPU\ OPENGL}$, P	Разница
256×256	0,0026 с 12,7 GFLOP/s	0,006 с 5,3 GFLOP/s	2,4
512×512	0,005 с 53,6 GFLOP/s	0,011 с 23,7 GFLOP/s	2,3
1024×1024	0,017 с 126 GFLOP/s	0,047 с 45 GFLOP/s	2,8
2048×2048	0,1 с 164 GFLOP/s	0,2 66 GFLOP/s	2,5

Полученные результаты для алгоритма блочного умножения на двух вычислительных платформах аналогичны рассмотренным выше: производительность платформы CUDA в 2,3–2,8 раз больше, чем OpenGL.

В используемых ранее алгоритмах тело цикла состоит из зависящих друг от друга действий (RAW-зависимости). Это не позволяет увеличить параллелизм на уровне команд (англ. Instruction Level Parallelism, ILP), что приводит к низкой загрузке исполнительных устройств процессора. Повысить степень загрузки исполнительных устройств можно путем раскрутки внутреннего цикла программы.

Часть OpenGL ядра с алгоритмом раскрутки цикла на несколько итераций приведена ниже.

```
const int row = get_local_id(0); //
Local row
const int col = get_local_id(1); //
Local col
const int globalRow =
TS*get_group_id(0) + row; // Row ID of
C const int globalCol =
TS*get_group_id(1) + col; // Col ID of
C
```

```
// Локальная память для хранения бло-
ков размерностью TS*TS
__local float Asub[TS][TS];
__local float Bsub[TS][TS];
// Инициализация регистров хранения
float acc[WPT]; // WPT – количество
циклов (4)
for (int w = 0; w<WPT; w++) {
     acc[w] = 0.0f;
}

// Цикл по всем блокам
const int numTiles = K / TS;
for (int t = 0; t<numTiles; t++) {
// загрузка блоков из A и B в локаль-
ную память
for (int w = 0; w<WPT; w++) {
const int tiledRow = TS*t + row;
const int tiledCol = TS*t + col;
Asub[col + w*RTS][row] = A[(tiledCol +
w*RTS)*M +
     globalRow];
     Bsub[col + w*RTS][row] =
B[(globalCol + w*RTS)*K +
     tiledRow];
     }
// Ожидание загрузки всех данных
barrier(CLK_LOCAL_MEM_FENCE);

for (int k = 0; k<TS; k++) {
for (int w = 0; w<WPT; w++) {
acc[w] += Asub[k][row] * Bsub[col +
w*RTS][k];
}
}
// Синхронизация перед загрузкой сле-
дующего блока
barrier(CLK_LOCAL_MEM_FENCE);
}
// Запись результата в  C
for (int w = 0; w<WPT; w++) {
C[(globalCol + w*RTS)*M + globalRow] =
acc[w];
     }
```

Результаты измерения производительности данного ядра приведены в таблице 5.

Полученный результат вычислений при использовании алгоритма блочного умножения с раскруткой цикла на 4 итерации для платформы OpenGL в 21 раз превосходит полученные результаты этой платформы при вычислениях без оптимизации.

На рис. 1 и 2 приведены графики зависимости производительности от выбранного алгоритма вычисления и размерности матрицы для платформ OpenGL и CUDA.

Таблица 5

Результаты блочного умножения с размером блока 32 и раскруткой цикла на 4 итерации для GPU (CPU Intel Core i7-4750HQ + GPU GeForce GTX 960M)

N	$t_{GPU\ CUDA}$, P	$t_{GPU\ OPENGL}$, P	Разница
256×256	0,0029 с 11,2 GFLOP/s	0,006 с 5,7 GFLOP/s	2
512×512	0,004 с 66,3 GFLOP/s	0,009 с 30,4 GFLOP/s	2,2
1024×1024	0,011 с 186 GFLOP/s	0,017 с 120,6 GFLOP/s	1,5
2048×2048	0,062 с 275,3 GFLOP/s	0,074 с 229 GFLOP/s	1,2

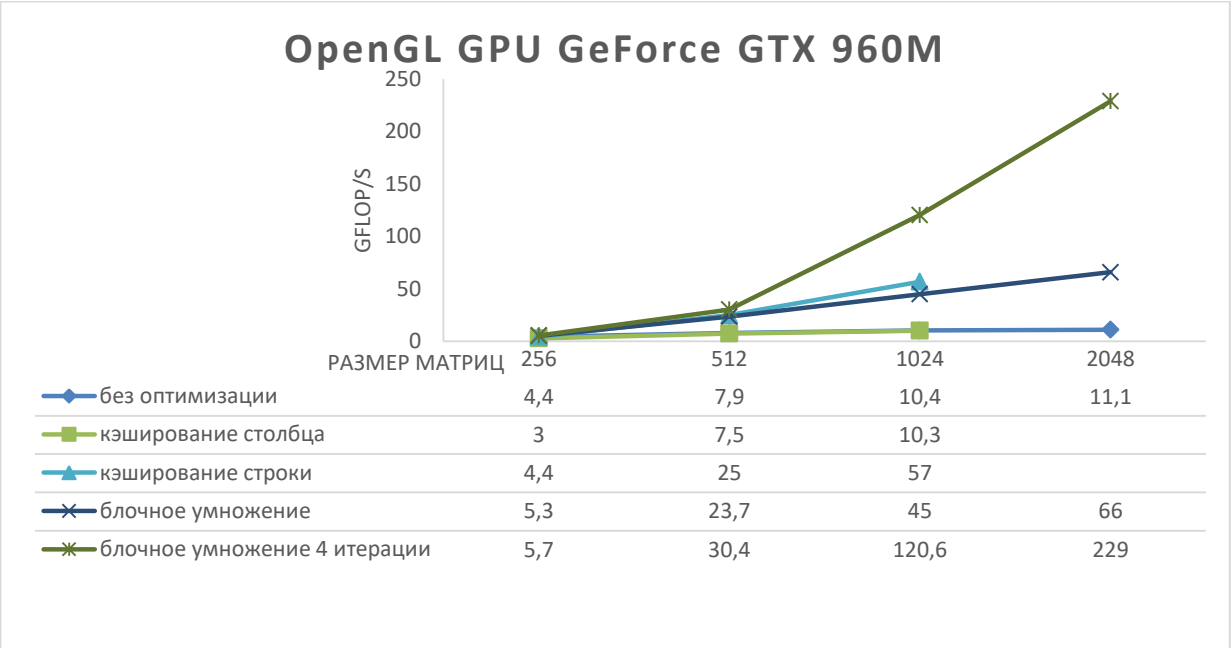


Рис. 1. Зависимость производительности OpenGL ядра от размера умножаемых матриц

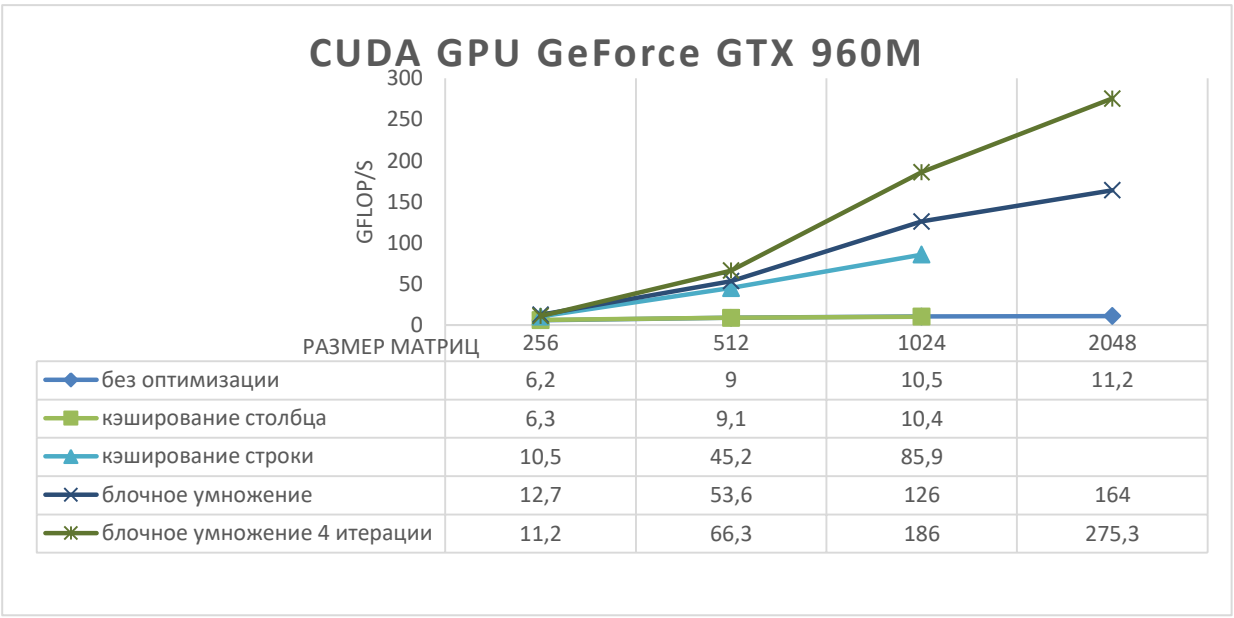
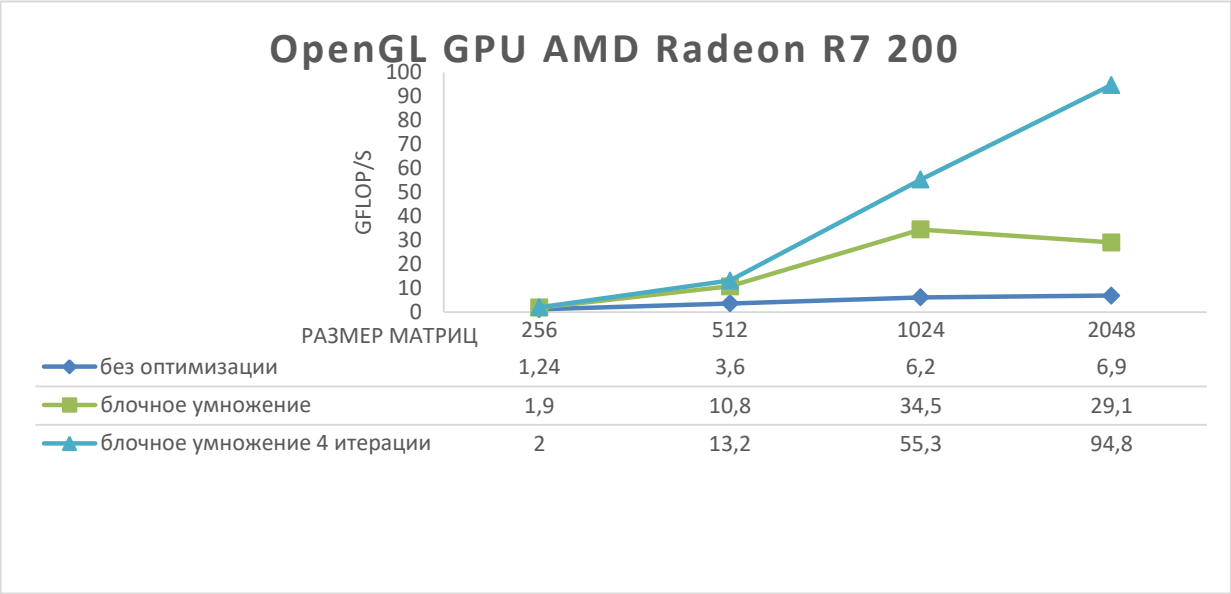


Рис. 2. Зависимость производительности CUDA ядра от размера умножаемых матриц

Платформа OpenGL может работать на любом вычислительном устройстве, если производитель предоставляет соответствующий драйвер, поддерживающий OpenGL. Это открывает возможность для тестирования на других аппаратных платформах, не поддерживающих CUDA. В качестве примера была выбрана следующая аппаратная платформа: CPU Intel Core 2 Duo E6550 + GPU AMD Radeon R7 200. Результаты измерения производительности в графическом виде представлены на рис. 3. Уменьшение производительности блочного умножения на GPU AMD Radeon R7 200 при размерности матрицы 2048 объясняется тем, что в данном случае исчерпан объем локальной памяти GPU и загрузка данных производится неоднократно.



РАЗМЕР МАТРИЦ	256	512	1024	2048
без оптимизации	1,24	3,6	6,2	6,9
блочное умножение	1,9	10,8	34,5	29,1
блочное умножение 4 итерации	2	13,2	55,3	94,8

Рис. 3. Результаты изменения производительности

Полученные оценки реальной производительности позволяют сделать вывод о том, что для оптимального использования вычислительных мощностей GPU необходимо соблюдать баланс между пропускной способностью глобальной памяти GPU, выбором типа памяти GPU для хранения данных, числом операций, выполняемых одной вычислительной единицей GPU и числом SIMD\PE-вычислителей.

Результат обработки на машине с CPU Intel Core i7-4750HQ + GPU GeForce GTX 960M при умножении матриц одинарной точности размером 2048×2048 без оптимизации показал производительность GPU 11,1 GFLOP/s для платформы OpenGL и 11,2 GFLOP/s для CUDA. Оптимизация с помощью блочного умножения позволяет увеличить производительность GPU до 66 GFLOP/s для OpenGL. Максимальная производительность в выполняемых вычислениях была достигнута при оптимизации блочным умножением и раскруткой внутреннего цикла с целью повышения параллелизма на уровне инструкций (ILP). При этом реальная производительность GPU с использованием OpenGL составляет 229 GFLOP/s и соответственно для CUDA 275,3 GFLOP/s.

Меньшая максимальная производительность при использовании платформы OpenGL по-видимому объясняется реализацией дополнительной трансляции кода OpenGL под архитектуру CUDA, т.к. OpenGL драйвер для видеокарт NVIDIA использует вызовы CUDA Toolkit. В отличие от этого, платформа CUDA использует специализированный компилятор, который преобразует код в более оптимальные для платформы от NVIDIA инструкции.

Исходя из выполненных тестовых расчетов и замеров производительности можно заметить, что расчеты OpenGL могут выполняться с меньшей скоростью в сравнении с CUDA, но платформа OpenGL обладает важным свойством переносимости и позволяет задействовать множество вычислительных устройств, в отличие от CUDA платформы, которая может использовать только NVIDIA устрой-

ства. Ввиду этого платформа OpenGL имеет хорошие перспективы развития и более широкую сферу применения, хотя и несколько меньшую производительность.

Работа была частично поддержана РФФИ (грант № 17-07-00317-а) и советом по грантам Президента РФ (грант МК-9445.2016.8).

Список литературы

1. Ватутин Э.И., Зотов И.В. Построение матрицы отношений в задаче оптимального разбиения параллельных управляющих алгоритмов // Известия Курского государственного технического университета. 2004. № 2. С. 85–89.

2. Ватутин Э.И., Мартынов И.А., Титов В.С. Оценка реальной производительности современных видеокарт с поддержкой технологии CUDA в задаче умножения матриц // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2014. № 2. С. 8–17.

3. OpenGL. URL: https://ru.wikipedia.org/ wiki/ OpenGL (дата обращения: 01.02.2017).

4. APP SDK – A Complete Development Platform // AMD: website. URL: http: // developer.amd.com/tools-and-sdks/ OpenGL-zone/amd-accelerated-parallel-pro-cessing-app-sdk/ (дата обращения: 01.02.2017).

5. CUDA АЛЬМАНАХ / Май 2015 г. 5 мая 2015. NVIDIA: сайт. URL: http: //www.nvidia.ru/docs/IO/141194/ CUDA-альманах-may-2015.pdf (дата обращения 01.02.2017).

6. Ватутин Э.И., Мартынов И.А., Титов В.С. Оценка реальной производительности современных процессоров в задаче умножения матриц для однопоточной программной реализации // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2013. № 4. С. 11–20.

7. Казеннов А.М. Основы технологии CUDA и OpenGL // НОЦ СКТ МФТИ: сайт. URL: http://hpc.mipt.ru/ wp-content/ uploads/ 2013/11/ CUDA+ OpenGL.pdf (дата обращения 01.02.2017).

Поступила в редакцию 12.09.17

UDC 621.397.01

Y.A. Zatolokin, Undergraduate, Southwest State University (Kursk, Russia) (e-mail: yuzato@list.ru)

E.I. Vatutin, Candidate of Engineering Sciences, Associate Professor**,** Southwest State University (Kursk, Russia) (e-mail: evatutin@rambler.ru)

V.S. Titov, Doctor of Engineering Sciences, Professor, Southwest State University (Kursk, Russia) (e-mail: titov-kstu@rambler.ru)

ALGORITHMIC OPTIMIZATION OF SOFTWARE IMPLEMENTATION OF ALGORITHMS FOR MULTIPLYING DENSE REAL MATRICES ON GRAPHICS PROCESSORS WITH OpenGL TECHNOLOGY SUPPORT

In the article was given statement of a problem of matrix multiplication. Is is show that desired problem can be simpl formulated but for its solving may be required both heuristic methods and set of algorithmic modifications relating to algorithmic and high-level software optimization taking into account the particular problem and allow to

increase the multiplication performance. These include: a comparative analysis of the performance of the actions performed without GPU-specific optimizations and with optimizations, which showed that computations without optimizing the work with global GPU memory have low processing performance. Optimizing data distribution in global and local memory The GPU allows you to reuse the calculation time and increase real performance. To compare the performance of the developed software implementations for OpenGL and CUDA technologies, identical calculations on identical GPUs were performed, which showed higher real performance when using CUDA cores. Specific values of generation performance measured for multi-threaded software implementation on GPU are given for all of described optimizations. It is shown that the most effective approach is based on the method we can get much more performance by technique of caching sub-blocks of the matrices (tiles) in the GPU's on-chip local memory, that with specialized software implementation is provide the performance of 275,3 GFLOP/s for GPU GeForce GTX 960M.

Key words: *Multiplication of matrices, algorithmic optimization, OpenGL, CUDA, algorithmic and high-level software optimization.*

DOI: 10.21869/2223-1560-2017-21-5-06-15

For citation: Zatolokin Y.A., Vatutin E.I., Titov V.S. Algorithmic optimization of software implementation of algorithms for multiplying dense real matrices on graphics processors with OpenGL technology support. Proceedings of the Southwest State University, 2017, vol. 21, no. 5(74), pp. 6-15 (in Russ.).

***

Reference

1. Vatutin Je.I., Zotov I.V. Postroe-nie matricy otnoshenij v zadache optimal'nogo razbienija parallel'nyh upravljajushhih algoritmov. Izvestija Kurskogo gosudarstvennogo tehnicheskogo universiteta, 2004. no. 2, pp. 85–89.

2. Vatutin Je.I., Martynov I.A., Titov V.S. Ocenka real'noj proizvoditel'nosti sovremennyh videokart s podderzhkoj tehnologii CUDA v zadache umnozhenija matric. Izvestija Jugo-Zapadnogo gosudarstvennogo universiteta. Serija: Upravlenie, vychislitel'naja tehnika, informatika. Medicinskoe priborostroenie, 2014, no. 2, pp. 8–17.

3. OpenGL. URL: https://ru.wikipedia.org/ wiki/ OpenGL (accessed 01.02.2017).

4. APP SDK – A Complete Development Platform // AMD: website. URL: http: // developer.amd.com/tools-and-sdks/ OpenGL -zone/amd-accelerated-parallel-pro-cessing-app-sdk/ (accessed 01.02.2017).

5. CUDA AL"MANAH / Maj 2015 g. 5 maja 2015. NVIDIA. URL: http: //www. nvidia.ru/docs/IO/141194/ CUDA-al'manah-may-2015.pdf (accessrd 01.02.2017).

6. Vatutin Je.I., Martynov I.A., Titov V.S. Ocenka real'noj proizvoditel'nosti sovremennyh processorov v zadache umnozhenija matric dlja odnopotochnoj programmnoj realizacii. Izvestija Jugo-Zapadnogo gosudarstvennogo universiteta. Serija: Upravlenie, vychislitel'naja tehnika, informatika. Medicinskoe priborostroenie, 2013, no. 4, pp. 11–20.

7. Kazennov A.M. Osnovy tehnologii CUDA i OpenGL // NOC SKT MFTI: http:// hpc.mipt.ru/ wp-content/uploads/ 2013/11/ CUDA+ OpenGL.pdf (data obrashhenija 01.02.2017).