Использование возможностей графических процессоров для математических вычислений

В.А. Егунов, В.А. Шабаловский

Волгоградский государственный технический университет

Аннотация: В представленной работе исследуется актуальная задача использования графических процессоров (GPU) в вычислительных процессах, которые традиционно выполняются на центральных процессорах (CPU). С развитием технологий и появлением специализированных архитектур и библиотек, GPU стали незаменимыми в областях, требующих интенсивных вычислений. Статья подробно рассматривает преимущества использования GPU по сравнению с традиционными CPU, обосновывая это их способностью к параллельной обработке и высокой пропускной способности, что делает их идеальным инструментом для работы с большими объемами данных.
Ключевые слова: графические процессоры, GPU, CUDA, OpenCL, cuBLAS, CLBlast, rocBLAS, параллельная обработка данных, математические вычисления, оптимизация кода, управление памятью, машинное обучение, научные исследования.

Введение. В современном мире научных и инженерных исследований наблюдается стремительный рост интереса к использованию графических процессоров (GPU) для выполнения математических вычислений. Это обусловлено не только их способностью к обработке огромных объемов данных за счет параллельной обработки, но и высокой эффективностью по сравнению с традиционными центральными процессорами (CPU) [1, 2].

Сегодня GPU играет ключевую роль в различных областях, где требуются интенсивные вычисления. Преимущество графических процессоров в этих областях заключается в их высокой пропускной способности и способности выполнять тысячи потоков вычислений параллельно. Однако, несмотря на значительные преимущества, использование GPU в математических вычислениях сопряжено с рядом технических и алгоритмических вызовов, таких, как необходимость оптимизации кода под архитектуру GPU, управление памятью и синхронизацию потоков.

Основные технологии и архитектуры GPU и их применение в математических задачах. Применение технологий GPU, таких, как CUDA,

OpenCL, а также специализированных библиотек cuBLAS, CLBlast и rocBLAS, представляет собой революцию в подходах к решению математических и инженерных задач [3].

cuBlas представляет собой высокопроизводительную библиотеку от компании NVIDIA, предназначенную для выполнения базовых функций линейной алгебры BLAS (Basic Linear Algebra Subprograms) на GPU [4]. Использование cuBlas для реализации ключевых операций BLAS, таких как умножение матриц, позволяет добиться значительного ускорения вычислений по сравнению с CPU.

CLBlast предлагает схожую функциональность для устройств с поддержкой OpenCL, что позволяет использовать GPU различных производителей для ускорения операций BLAS. Библиотека оптимизирует выполнение задач под конкретную архитектуру GPU, обеспечивая высокую производительность в разнообразных вычислительных средах [5].

rocBLAS входит в состав экосистемы AMD ROCm и представляет собой библиотеку, ориентированную на высокопроизводительные вычисления с использованием GPU AMD [6]. Она предлагает оптимизированные реализации BLAS, обеспечивает масштабируемость и оптимизацию процессов.

Исследование. Предметом исследования являются различные реализации операции умножения плотных матриц с последующим анализом и сравнением полученных показателей эффективности. На первом этапе рассмотрим базовую реализацию данной операции, заключающуюся в вычислении элементов результирующей матрицы C как скалярных произведений соответствующих строк и столбцов исходных матриц. Приведем фрагмента кода, демонстрирующий этот процесс.

```
for (int row = 0; row < N; ++row) {
    for (int col = 0; col < N; ++col) {
```

```
C[row][col] = 0;
for (int k = 0; k < N; ++k) {
   C[row][col] += A[row][k] * B[k][col];
}}}
```

В этом фрагменте внешние циклы перебирают строки и столбцы матрицы C, а внутренний цикл вычисляет необходимые скалярные произведения. Приведенная методика, являясь интуитивно понятной и простой в реализации, имеет ряд существенных недостатков.

1) Неэффективность использования кеша. Доступ к элементам матриц не оптимизирован в контексте использования кеш-памяти, что ведет к частым промахам кеша и, как следствие, к замедлению вычислений.

2) Отсутствие параллелизма: Вычисления выполняются последовательно и не используют возможностей параллельных вычислений на современных многоядерных процессорах и GPU.

3) Масштабируемость: При увеличении размера матриц наблюдается существенный рост времени выполнения, что делает данный метод непригодным для работы с большими размерами матриц.

Время, необходимое для расчета с использованием базового алгоритма (Standard_matrix), приведено в таблице 1.

Таблица 1

Время выполнения программы в секундах.

Размерность	64	128	256	512	1024
Standard_matrix	0.00703	0.05657	0.42733	3.66166	34.71356

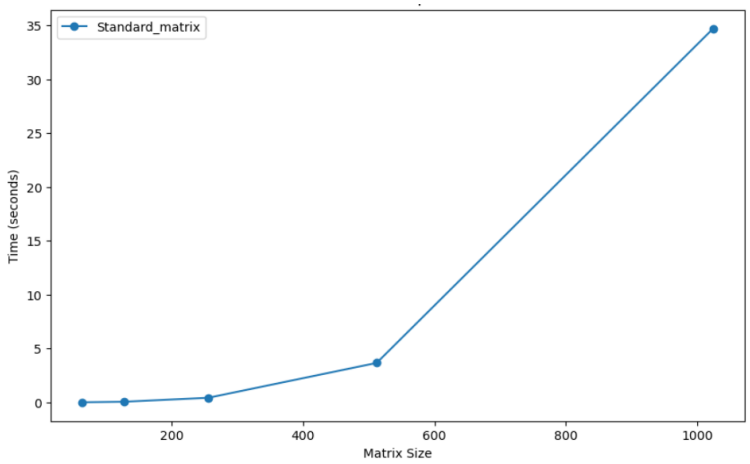

Рис 1 - Время выполнения базового алгоритма

Теперь рассмотрим алгоритм, который использует возможности CUDA [7] и библиотеку cuBLAS для ускорения вычислений на GPU. Данный подход значительно отличается по эффективности и скорости выполнения, благодаря параллельной обработке данных на графических процессорах. Важно отметить, что для управления данными на GPU используются специальные функции, такие как cudaMalloc, которые обеспечивают выделение памяти в глобальном адресном пространстве GPU.

cudaMalloc(&d_A, bytes);

Вычисления осуществляются с помощью библиотеки cuBLAS.

cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, n, n, n, &alpha, d_A, n, d_B, n, &beta, d_C, n);

Данный вызов запускает на GPU процесс умножения матриц A и B, результат записывается в матрицу C. Параметры alpha и beta позволяют масштабировать результат и накапливать его в результирующей матрице.

Преимущества данной реализации заключаются в значительном ускорении вычислений благодаря параллелизму, доступному на GPU. Такое ускорение становится особенно заметным при работе с большими матрицами, где разница во времени выполнения по сравнению с последовательными алгоритмами на CPU может достигать нескольких порядков.

Однако, несмотря на значительные преимущества, у данной реализации есть свои недостатки [8-10]. Во-первых, требуется наличие совместимой с CUDA видеокарты, что может ограничить доступность данного подхода. Во-вторых, программирование для CUDA требует понимания специфики архитектуры GPU, что увеличивает сложность разработки по сравнению с традиционными методами. В-третьих, управление памятью на GPU может стать дополнительным источником ошибок, особенно при работе с большими объемами данных. Результаты замеров времени выполнения данной программы (cuBlas_matrix) представлены в таблице 2.

Для преодоления перечисленных недостатков был использован другой подход к решению задачи, в рамках которого применяется оптимизированная реализация операции умножения матриц с использованием фреймворка OpenCL. Подобный подход может использоваться на различных аппаратных платформах, включая графические процессоры (GPU). Алгоритм использует локальное разбиение данных и технику тайлинга (разбиение на блоки) для эффективного использования кеш-памяти и уменьшения числа обращений к глобальной памяти, что крайне важно для ускорения вычислений на GPU.

Ключевые аспекты реализации:

1) Локальное разбиение данных. Каждый поток копирует элемент из глобальной памяти в локальную (shared) память, что уменьшает задержки, связанные с доступом к памяти. Размер блока (localSize) заранее определен как 16, что является оптимальным выбором для многих архитектур GPU из-за хорошей балансировки между размером локальной памяти и количеством потоков на блок.

2) Тайлинг. Алгоритм разбивает матрицы на подматрицы (тайлы) размером 16x16, что позволяет каждому блоку потоков работать с небольшим набором данных, полностью умещающимся в локальной памяти. Это существенно ускоряет доступ к данным и повышает общую

производительность за счет сокращения числа обращений к глобальной памяти.

3) Синхронизация потоков. Внутри ядра используются барьеры синхронизации (barrier(CLK_LOCAL_MEM_FENCE)), чтобы гарантировать, что все потоки блока завершили копирование данных в локальную память перед началом вычисления и перед копированием следующего тайла. Это обеспечивает корректность результатов при параллельной работе. Ниже представлен фрагмент кода ядра умножения матриц:

```
for (int tileIdx = 0; tileIdx < N / localSize; ++tileIdx) {
   Asub [localRow][localCol] = A [globalRow * N + (tileIdx * localSize + localCol)];
   Bsub [localRow][localCol] = B[(tileIdx * localSize + localRow) * K + globalCol];
   barrier (CLK_LOCAL_MEM_FENCE);
   for (int k = 0; k < localSize; ++k) {
      sum += Asub[localRow][k] * Bsub[k][localCol];
   }
   barrier (CLK_LOCAL_MEM_FENCE);
}
```

Хотя оптимизированная реализация умножения матриц с использованием OpenCL значительно ускоряет вычисления, особенно на GPU, она также предъявляет ряд требований и вводит определенные ограничения. Размеры матриц должны быть кратны размеру блока, что может потребовать дополнительной предварительной обработки. Для достижения максимальной производительности на конкретной аппаратной платформе необходима тщательная оптимизация памяти и выбор параметров, учет разнообразие архитектур GPU и их оптимизационные характеристики. Эффективное использование такой реализации также зависит от качества

поддержки OpenCL на целевой платформе, поскольку производительность и стабильность различных реализаций могут значительно различаться. Кроме того, необходимы глубокие знания в области параллельного программирования и оптимизации кода для GPU, включая эффективное использование локальной памяти, синхронизацию потоков и минимизацию обращений к глобальной памяти, что может существенно увеличить сложность кода. Результат замеров времени работы метода (OpenCL_matrix) представлен в таблице 2.

Таблица 2

Результат замеров времени выполнения программ в секундах.

Размерность	512	1024	2048	4096	8192	16384
cuBlas_matrix	0.00021	0.00089	0.00465	0.03409	0.24540	1.43031
OpenCL_matrix	0.00065	0.00117	0.00353	0.01344	0.05336	0.19334

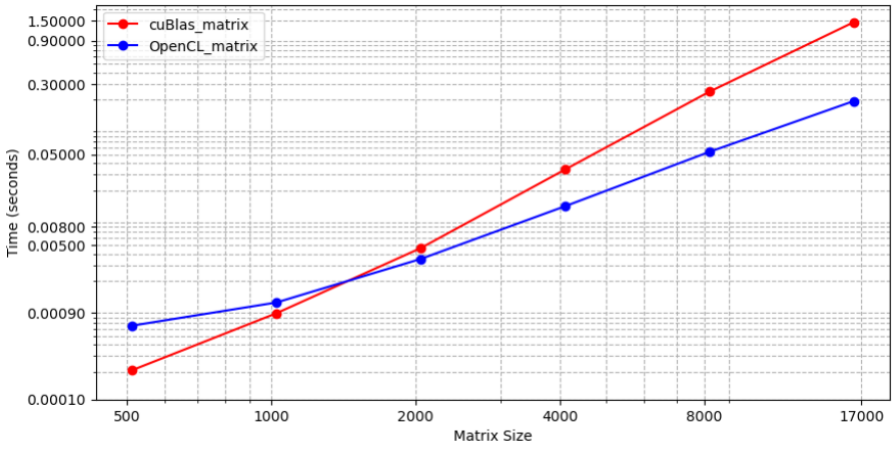

Рис 2 - Время реализации исследуемых алгоритмов с использованием GPU

Из графика видно, что оба алгоритма масштабируются с увеличением размера матриц, но OpenCL демонстрирует лучшую производительность по сравнению с cuBLAS. Это может быть связано с различной степенью оптимизации библиотек под конкретные вычислительные задачи и архитектуру используемых GPU.

Заключение. Использование графических процессоров (GPU) для математических вычислений, особенно в задачах умножения матриц, значительно расширяет возможности научных и инженерных исследований. С помощью различных технологий и библиотек достигается значительное ускорение обработки данных и выполнения вычислений по сравнению с традиционными методами на CPU. Это не только улучшает эффективность исследовательских работ, но и позволяет преодолевать сложные технические и алгоритмические проблемы.

Однако, для успешного применения GPU в вычислительных задачах, необходимы глубокие знания в области параллельного программирования и оптимизации кода. Важно учитывать особенности работы с памятью на GPU, синхронизацию потоков и другие ключевые аспекты, чтобы максимизировать производительность и стабильность приложений. В перспективе, развитие технологий GPU обещает еще большие ускорения в вычислениях, что будет способствовать быстрому прогрессу в науке и технике.

Литература

1. Быков Д.В., Неретин А.Д. Прогнозирование производительности при реализации алгоритмов генерации случайных последовательностей больших размерностей на реконфигурируемых архитектурах с сопроцессорами // Инженерный вестник Дона, 2014. №2. URL: ivdon.ru/ru/magazine/archive/n2y2014/2414.

2. Егунов В.А., Королева И.Ю., Типаев Д.В. Алгоритмы движения мобильного робота с построением карты местности в реальном времени // Инженерный вестник Дона. 2022. № 4. URL: ivdon.ru/ru/magazine/archive/n4y2022/7570.

3. Якушев В. Л., Филимонов А. В., Солдатов П. Ю. Использование технологии CUDA для оптимизации прямого решателя СЛАУ // Вестник кибернетики. 2014. №1 (13). URL: cyberleninka.ru/article/n/ispolzovanie-tehnologii-cuda-dlya-optimizatsii-pryamogo-reshatelya-slau.

4. cuBLAS Basic Linear Algebra on NVIDIA GPUs // Nvidia. URL: docs.nvidia.com/cuda/cublas/ (дата обращения: 29.03.2024).

5. Nugteren C. CLBlast: A Tuned OpenCL BLAS Library // In Proceedings of the International Workshop on OpenCL (IWOCL '18). New York, NY, USA, Article 5, pp. 1–10. URL: doi.org/10.1145/3204919.3204924.

6. rocBLAS Documentation // AMD URL: rocm.docs.amd.com/projects/rocBLAS/en/latest/ (дата обращения: 29.03.2024).

7. Исупов К.С., Князьков В.С. Матрично-векторное умножение многократной точности на графическом процессоре // Программные системы: теория и приложения. 2020. Т. 11. № 3 (46). С. 33-59.

8. Безрученко А.Ю., Егунов В.А. Исследование эффективности методов оптимизации программ для параллельных вычислительных систем с GPU // Вестник Дагестанского государственного технического университета. Технические науки. 2023. Т. 50, № 4. С. 59-74. URL: vestnik.dgtu.ru/jour/article/view/1392/842.

9. CUDA C++ programming guide // docs.nvidia.com. URL: docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (дата обращения: 29.03.2024)

10. CUDA Pro Tip: Write Flexible Kernels with Grid-Stride Loops // Nvidia technical blog: URL: developer.nvidia.com/blog/cuda-pro-tip-write-flexible-kernels-grid-stride-loops/ (дата обращения: 29.03.2023).

References

1. Bykov D.V., Neretin A.D. Inzhenernyj vestnik Dona, 2014, №2. URL: ivdon.ru/ru/magazine/archive/n2y2014/2414.

2. Egunov V.A., Koroleva I.YU., Tipaev D.V. Inzhenernyj vestnik Dona, 2022, №4. URL: ivdon.ru/ru/magazine/archive/n4y2022/7570.

3. YAkushev V. L., Filimonov A. V., Soldatov P. YU. Vestnik kibernetiki. 2014. №1 (13). URL: cyberleninka.ru/article/n/ispolzovanie-tehnologii-cuda-dlya-optimizatsii-pryamogo-reshatelya-slau

4. cuBLAS Basic Linear Algebra on NVIDIA GPUs. Nvidia URL: docs.nvidia.com/cuda/cublas/ (date assessed: 29/03/2024).

5. Nugteren C. Proceedings of the International Workshop on OpenCL (IWOCL '18). New York, NY, USA, Article 5, pp. 1–10. URL: doi.org/10.1145/3204919.3204924.

6. rocBLAS Documentation. URL: rocm.docs.amd.com/projects/rocBLAS/en/latest/ (date assessed: 29/03/2024).

7. Isupov K.C., Knyaz'kov V.S. Programmnye sistemy: teoriya i prilozheniya. 2020. T. 11. № 3 (46). Pp. 33-59.

8. Bezruchenko A.Yu., Egunov V.A. Vestnik Dagestanskogo gosudarstvennogo tekhnicheskogo universiteta. Tekhnicheskie nauki. 2023. V. 50, № 4. Pp. 59-74. URL: vestnik.dgtu.ru/jour/article/view/1392/842.

9. CUDA C++ programming guide. URL: docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (date assessed: 29/03/2024)

10. CUDA Pro Tip: Write Flexible Kernels with Grid-Stride Loops. Nvidia technical blog: URL: developer.nvidia.com/blog/cuda-pro-tip-write-flexible-kernels-grid-stride-loops/ (date assessed: 29/03/2024)

Дата поступления: 29.02.2024

Дата публикации: 9.04.2024