Исследование параллельных алгоритмов решения трехдиагональных систем линейных алгебраических уравнений на графическом вычислительном устройстве с применением различных видов памяти

К.С. Погорельских[1], **Л.В. Логанова**[1]

[1]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086

Аннотация. В данной работе произведено исследование и сравнение различных реализаций алгоритмов циклической редукции и прогонки на графическом процессорном устройстве (GPU) с применением различных видов памяти устройства. В результате выполнения работы было установлено, что для решения совокупности СЛАУ трехдиагонального вида следует использовать алгоритм прогонки. Однако при решении одной СЛАУ на GPU лучшие результаты показывает параллельная версия алгоритма циклической редукции с частичным использованием разделяемой памяти.

1. Введение

В данной статье исследуются параллельные алгоритмы решения систем линейных уравнений трехдиагонального вида. Применимость таких систем весьма обширна. Трехдиагональные матрицы играют крайне важную роль в разностных методах решения задач математической физики. Кроме того, многие задачи линейной алгебры, такие как решение уравнений и нахождение собственных значений, решаются через преобразования матриц общего вида к трехдиагональным. Также большую роль эти матрицы играют в теории ортогональных многочленов [1].

Для решения трехдиагональных СЛАУ существует множество алгоритмов и их параллельных версий. Необходимость распараллеливания естественно возникает, когда размер или количество решаемых систем весьма велики. Тот факт, что в основе большинства методов лежат многократно выполняемые простые арифметические операции, заставляет обратить внимание на реализацию параллельных версий алгоритмов на графическом процессоре.

Также в пользу реализаций на графическом процессоре говорит то, что GPU-кластеры потребляют меньшее количество электроэнергии, чем CPU. А развитие видеокарт позволяет проводить сложные вычисления даже на персональных компьютерах, что не может не привлекать исследователей.

При вычислениях на графическом процессоре существенное время тратится на работу с памятью [2]. Поэтому оптимально организовав работу с памятью, можно добиться значительного увеличения ускорения.

Для проведения вычислений принято использовать глобальную, разделяемую и константную память GPU. Константная память невелика и неизменяема, она мало подходит для решения

больших систем. Поэтому в данной работе исследуются глобальная и разделяемая память графического процессорного устройства.
Разделяемая память значительно быстрее глобальной, однако её размеры весьма ограничены. Эти особенности видов памяти необходимо учитывать при реализации алгоритмов на GPU.
Традиционно для решения СЛАУ трехдиагонального вида используют метод прогонки и метод циклической редукции [3].
Преимуществом метода циклической редукции является его высокая степень параллелизма [3]. Однако он также характеризуется большим объемом вычислений. При реализации на центральном процессорном устройстве с использованием технологии MPI метод требует множество пересылок, что существенно замедляет работу программы [4]. Однако, в технологии CUDA пересылки не требуются, так что метод привлекает внимание исследователей для реализации на видеокарте. Кроме того, он позволяет проводить распараллеливание для одной СЛАУ, что является несомненным его преимуществом [5].
В отличие от метода циклической редукции метод прогонки плохо распараллеливается при решении одной системы. Однако существует большое количество задач, требующих решения совокупности трехдиагональных СЛАУ. А для таких целей целесообразно использовать метод прогонки. Это может дать выигрыш в ускорении, потому что вычислительная сложность данного алгоритма сравнительно низкая.
В данной работе исследуются реализации алгоритма прогонки, а также последовательной и параллельной версий алгоритма циклической редукции с использованием различных комбинаций глобальной и разделяемой памяти GPU. Распараллеливание методов проводится с использованием программно-аппаратной архитектуры CUDA. Экспериментальные исследование проводятся на суперкомпьютере «Сергей Королев».

2. Экспериментальные исследования

Данная работа началась с реализации и исследования алгоритма прогонки. Этот метод один из самых известных и популярных для решения трехдиагональных СЛАУ.
Для сравнения скорости работы последовательного метода прогонки с его параллельной версией он был реализован на CPU. Затем была написана реализация параллельного алгоритма прогонки на GPU с использованием только глобальной памяти. Следующей задачей было реализовать этот метод с использованием разделяемой памяти. Так как размер памяти невелик, необходимые коэффициенты порционно загружались в разделяемую память непосредственно перед использованием, а затем после всех необходимых операций с ними заменялись следующей порцией коэффициентов. Для хранения данных в разделяемой памяти были выделены массивы вспомогательных коэффициентов α и β и искомого вектора x размером $2 \cdot blocksize$ и массивы исходных коэффициентов a, b, c и k размером $blocksize$, где $blocksize$ – это размер блока. Все вычисления проводились в разделяемой памяти.
Эксперименты проводились для совокупности СЛАУ с матрицами размера $N = 4095$. Такой размер был выбран для сравнения полученных результатов с результатами экспериментов версий алгоритма циклической редукции, реализованных для размера матриц $N = 2^q - 1$, где $q \in \mathbb{N}$.
Количество СЛАУ варьировалось от 5000 до 20000. Количество систем менее 5000 не рассматривалось, так как при таком количестве работа с памятью на GPU занимает значительно больше времени, чем сами вычисления, и использование графического процессора не оправдано.
Полученные ускорения параллельных версий алгоритмов относительно последовательной версии на CPU приведены в таблице 1.
Среднее ускорение версии на GPU с использованием только глобальной памяти равно 1,26. Среднее ускорение с использованием разделяемой памяти – 1,86. Таким образом, использование разделяемой памяти дает увеличение ускорения на 47%.
Этот результат показывает, что время, затрачиваемое на копированные данных из глобальной памяти в разделяемую и обратно, оправдывается высокой скоростью вычислений, проводимых в разделяемой памяти.

Таблица 1. Зависимость ускорения параллельных версий алгоритма прогонки относительно последовательной при решении совокупности СЛАУ в зависимости от количества систем с матрицами размера $N = 4095$.

Количество СЛАУ	GPU с разделяемой памятью	GPU
5000	**2,19032**	1,680011
10000	**1,705628**	1,164633
15000	**1,903806**	1,274506
20000	**1,885846**	1,027733

Следующий шаг работы – реализация алгоритма последовательной циклической редукции SERICR. Этот алгоритм не дает возможности проводить параллельные вычисления для одной СЛАУ, однако аналогично методу прогонки распараллеливается для совокупности систем.
Циклическая редукция подробно рассмотрена в работе Р. Хокни [6]. В ней отмечается, что этот алгоритм подходит для любого количества уравнений. Однако для простоты автор приводит количество уравнений $N = 2^q - 1$, где q – целое число. Также для удобства обозначив, $N' = 2^q = N + 1$.
Приведем диаграмму маршрутизации этого алгоритма для $N' = 8$ на рисунке 1 (а). Ячейки, в которых хранится значащая информация имеют серый цвет.

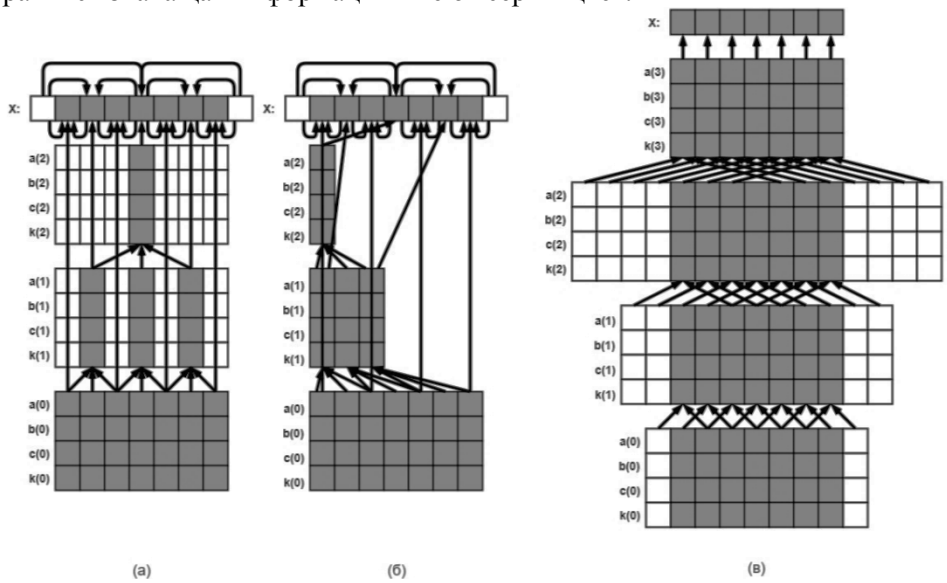

Рисунок 1. Схемы маршрутизации алгоритмов для $N' = 8$: последовательной циклической редукции SERICR (а), последовательной циклической редукции SERICR с измененной индексацией (б), параллельной циклической редукции PARACR (в).

Стоит отметить, что в описываемом алгоритме необходимо хранить в памяти коэффициенты всех уровней редукции. Это требует большого количества памяти. Кроме того, как это видно из схемы множество ячеек (белых) не хранят полезную информацию. Поэтому была изменена индексация алгоритма для исключения использования лишней памяти. Полученная схема маршрутизации приведена на рисунке 1 (б).
Приведем алгоритм с измененной индексацией. На первом этапе считаются новые коэффициенты для левых и правых частей для уровней редукции $l = \overline{1, q-1}$ с шагом $h = 1$ от $i = 0$ до $i = N^{(l)}$. Коэффициенты уровня редукции $l = 0$ принимаются равными $a_i^{(0)} = a_i$, $b_i^{(0)} = b_i$, $c_i^{(0)} = c_i$ и $k_i^{(0)} = k_i$. Сначала находим вспомогательные коэффициенты:

$$\alpha_i = -\frac{a_{2i+1}^{(l-1)}}{b_{2i}^{(l-1)}};$$
$$\gamma_i = -\frac{c_{2i+1}^{(l-1)}}{b_{2i+2}^{(l-1)}};$$

Затем на основе полученных α_i и γ_i и значений коэффициентов левых и правых частей предыдущего уровня редукции получаем:

$$a_i^{(l)} = \alpha_i a_{i-2^{l-1}}^{(l-1)};$$
$$c_i^{(l)} = \gamma_i c_{i+2^{l-1}}^{(l-1)};$$
$$b_i^{(l)} = b_i^{(l-1)} + \alpha_i c_{i-2^{l-1}}^{(l-1)} + \gamma_i a_{i+2^{l-1}}^{(l-1)};$$
$$k_i^{(l)} = k_i^{(l-1)} + \alpha_i k_{i-2^{l-1}}^{(l-1)} + \gamma_i k_{i+2^{l-1}}^{(l-1)}.$$

На втором этапе на основе полученных коэффициентов левых и правых частей для уровней редукции $l = \overline{0, q-1}$ находим вектор решений x. Принимается, что $x_0 = x_{N'} = 0$. Для $l = \overline{q, 1}$, для i, меняющегося с шагом $h_2 = 2^{(l-1)}$ от $i = 2^{(l-1)}$ до $i = N' - 2^{(l-1)}$, считаем:

$$x_i = \frac{k_i^{(l-1)} - a_i^{(l-1)} x_{i-2^{(l-1)}} - c_i^{(l-1)} x_{i+2^{(l-1)}}}{b_i^{(l-1)}}.$$

Однако полученной экономии в использовании памяти недостаточно для использования этого алгоритма на больших данных. Выходом в данной ситуации является хранение только двух уровней редукции и пересчет коэффициентов, когда они нужны.

Алгоритм был реализован на центральном и графическом процессорных устройствах. При реализации алгоритма на CPU пересчет коэффициентов не проводился, так как на центральном процессоре достаточно памяти для хранения коэффициентов всех уровней редукции. При реализации на GPU пересчет был необходим. На видеокарте алгоритм был реализован и с использованием только глобальной памяти, и с использованием разделяемой памяти.

Эксперименты проводились для совокупности СЛАУ с матрицами размера $N = 4095$. Количество СЛАУ варьировалось от 5000 до 25000. Полученные ускорения приведены в таблице 2.

Таблица 2. Зависимость ускорения алгоритма последовательной циклической редукции на GPU относительно реализации на CPU.

Количество СЛАУ	SERICR на GPU с разделяемой памятью	SERICR на GPU
5000	0,316570	0,193076
10000	0,259339	0,191475
15000	0,258175	0,186748
20000	0,264068	0,190208
25000	0,253464	0,185693

Реализации на GPU показали замедление работы. Это объясняется необходимостью проводить многократные пересчеты коэффициентов.

Заключительный этап работы – реализация алгоритма параллельной циклической редукции. Этот алгоритм предложен в работе Р. Хокни [6]. Его преимущество в том, что он поддается распараллеливанию для одной СЛАУ. Кроме того, следующие уровни редукции замещаются предыдущими, что дает заметный выигрыш в памяти. Схема маршрутизации алгоритма представлена на рисунке 1 (в).

Алгоритм был реализован на GPU с использованием только глобальной памяти, а также с загрузкой в разделяемую память всех коэффициентов и с загрузкой только части коэффициентов – исходных данных.

Эксперименты проводились для СЛАУ с матрицами размера $N = 4095$. Количество систем варьировалось от 5000 до 25000. Полученные ускорения представлены в таблице 3.

Таблица 3. Зависимость ускорения реализаций алгоритма параллельной редукции на GPU относительно алгоритма последовательной циклической редукции на CPU.

Количество СЛАУ	PARACR на GPU с частичным использованием разделяемой памяти	PARACR на GPU с разделяемой памятью	PARACR на GPU
5000	**2,006864**	1,198271	1,970711
10000	**2,02118**	1,206249	1,972713
15000	**2,008165**	1,201463	1,973943
20000	**2,007109**	1,202857	1,962254
25000	**2,017338**	1,203156	1,987762

Из таблицы видно, что лучшие результаты показывает реализация с частичной загрузкой в разделяемую память. Она на 2% быстрее реализации, не использующей разделяемую память, и на 41% быстрее реализации с полной загрузкой в разделяемую память.

Результаты объясняются тем, что некоторые коэффициенты используются в вычислениях лишь единожды. И время их копирования в разделяемую память практически не компенсируется выигрышем во времени выполнения вычислений. Поэтому в разделяемую память имеет смысл загружать только те данные, которые используются многократно. В этом случае данные, загружаемые в разделяемую память, используются дважды, поэтому существенного выигрыша использование разделяемой памяти не даёт.

Сравним лучшие реализации алгоритмов прогонки и циклической редукции. Время работы алгоритмов при решении совокупности СЛАУ с матрицами размера $N = 4095$ представлено в таблице 4.

Таблица 4. Время работы реализаций алгоритма прогонки на GPU с полной загрузкой в разделяемую память и алгоритма параллельной циклической редукции на GPU с частичной загрузкой в разделяемую память на совокупности СЛАУ с матрицами размера $N = 4095$.

Количество СЛАУ	Алгоритм прогонки, с	PARACR, с
5000	**0,09**	0,59
10000	**0,19**	1,19
15000	**0,28**	1,78
20000	**0,38**	2,38
25000	**0,47**	2,96

График времени работы алгоритмов при решении одной СЛАУ представлено на рисунке 2.

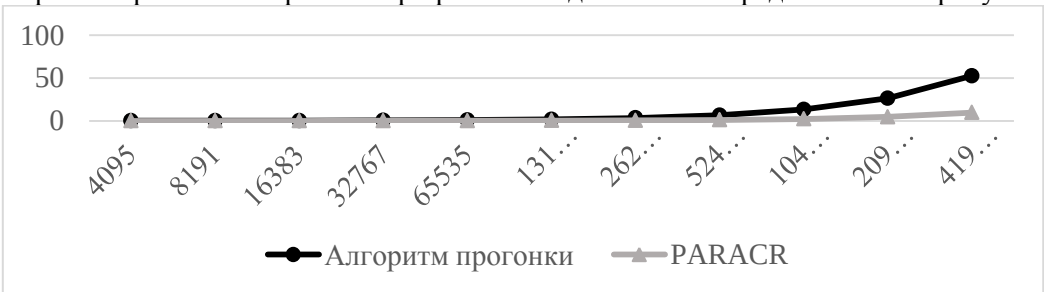

Рисунок 2. График времени работы реализаций алгоритма прогонки на GPU с полной загрузкой в разделяемую память и алгоритма параллельной циклической редукции на GPU с частичной загрузкой в разделяемую память при решении одной СЛАУ (на вертикальной оси показано время в секундах, на горизонтальной – размер матриц).

Таким образом при решении совокупности СЛАУ лучший результат показывает алгоритм прогонки на GPU с полной загрузкой в разделяемую память, время его работы в среднем в 6 раз меньше времени работы алгоритма циклической редукции. Однако при решении одной СЛАУ с матрицей большого размера целесообразнее использовать алгоритм параллельной циклической редукции с частичной загрузкой в разделяемую память, его реализация требует в 7 раз меньше времени на вычисления, чем реализации алгоритма прогонки.

3. Заключение

В результате данной работы было получено, что использование разделяемой памяти для алгоритма прогонки даёт увеличение ускорения на 47%. Это объясняется тем, что данные, загружаемые в разделяемую память, используются многократно.

Для алгоритма циклической редукции целесообразна загрузка в разделяемую память только тех коэффициентов, которые при вычислениях используются дважды. При этом ускорение увеличивается на 2%.

При решении совокупности СЛАУ лучшая реализация алгоритма прогонки в 6 раз быстрее лучшей реализации алгоритма циклической редукции. Однако при решении одной СЛАУ с матрицей большого размера результат обратный: алгоритм циклической редукции быстрее в 7 раз.

4. Литература

[1] Ильин, В.П. Трехдиагональные матрицы и их приложения / В.П. Ильин, Ю.И. Кузнецов. – М.: Наука, 1985. – 208 с.

[2] Технология CUDA в примерах: введение в программирование графических процессоров. – М.: ДМК Пресс, 2011. – 232 с.

[3] Шинкарук, Д.Н. Анализ эффективности применения технологии CUDA для решения систем линейных уравнений с трехдиагональными матрицами в задачах расчета цен опционов / Д.Н. Шинкарук, Ю.А. Шполянский, М.С. Косяков // Изв. вузов. Приборостроение. – 2012. – № 10. – С. 6.

[4] Малявко, А.А. Параллельное программирование на основе технологий OpenMP, MPI, CUDA / А.А. Малявко. – Новосибирск: Изд-во НГТУ, 2015. – 116 с.

[5] Ярмушкин, С.В. Исследование параллельных алгоритмов решения трехдиагональных систем линейных алгебраический уравнений / С.В. Ярмушкин, Д.Л. Головашкин // Вестник Самарского Государственного Технического Университета. Сер. Физико-математические науки. – 2004. – № 26. – С. 5.

[6] Хокни, Р. Параллельные ЭВМ. Архитектура, программирование и алгоритмы / Р. Хокни, К. Джессхоуп. – М.: Радио и связь, 1986. – 392 с.

Research of parallel algorithms for solving three-diagonal systems of linear algebraic equations on a graphical computing device using various types of memory

X.S. Pogorelskih[1], L.V. Loganova[1]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086

Abstract. In this paper, we research and compare different implementations of cyclic reduction and sweep algorithms on a graphics processing unit (GPU) using different types of device memory. As a result of the work, it was found that the algorithm of the run should be used to solve the set of the tridiagonal linear system. However, the best results are shown by a parallel version of the cyclic reduction algorithm with partial use of shared memory, when solving a single linear system on the GPU.