

РАЗРАБОТКА ПАРАЛЛЕЛЬНОГО АЛГОРИТМА МЕТОДА ЦИКЛИЧЕСКОЙ РЕДУКЦИИ, ПРЕДНАЗНАЧЕННОГО ДЛЯ РЕАЛИЗАЦИИ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

Л.В. Шимченко¹

¹Самарский университет, г. Самара
e-mail: lev.shimchlev@gmail.com

Аннотация

В работе представлена CUDA-реализация метода циклической редукции для решения трёхдиагональных систем линейных алгебраических уравнений (СЛАУ). Проведён сравнительный анализ производительности с последовательным алгоритмом. Продемонстрировано ускорение до 6,52 раз на системах с размерностью 2^{23} – 1.

Ключевые слова: CUDA, графический процессор, параллельный алгоритм, метод циклической редукции, метод конечных разностей, уравнение теплопроводности.

Введение

Метод циклической редукции (PCR) является эффективным подходом для решения задач с трёхдиагональными матрицами. Благодаря высокой степени параллелизма, PCR находит широкое применение в вычислительной гидродинамике [1], акустике [2], финансовой математике [3] и других областях, что обуславливает целесообразность его реализации на GPU. Однако в рассмотренной литературе отсутствует описание адаптации данного алгоритма для CUDA, несмотря на то что его реализация является нетривиальной задачей ввиду специфики хранения редуцированных матриц в памяти. Данная работа восполняет этот пробел. Тестирование алгоритма проводилось на неявной конечно-разностной схеме модели теплопроводности в однородном теплоизолированном с боков шаре.

1. Программная реализация параллельного алгоритма

Для применения на GPU реализовано две функции ядра – для прямого и обратного ходов редукции. На рисунке 1 представлен программный код этих функций.

```

__global__ void forwardStepKernel(double* c, double* a, double* b, double* d, int a_size,
    int a_size_next, int c_size, int a_start_id, int c_start_id, int k)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < a_size_next)
    {
        int id_prev = a_start_id + idx * 2 + 1; // предыдущие коэффициенты
        int id_prev_c = c_start_id + idx * 2 + 1; //
        int id_cur = a_start_id + a_size + idx; // текущие коэффициенты
        int id_cur_c = c_start_id + c_size + idx; //
        double repeat_coef_1 = c[id_prev_c - 1] / a[id_prev - 1];
        double repeat_coef_2 = b[id_prev_c] / a[id_prev + 1];
        if (k == 0)
        {
            if (idx > 0) c[id_cur_c - 1] = -repeat_coef_1 * c[id_prev_c - 2];
            a[id_cur] = a[id_prev] - repeat_coef_1 * b[id_prev_c - 1] - repeat_coef_2 * c[id_prev_c];
            if (idx < a_size_next - 1) b[id_cur_c] = -repeat_coef_2 * b[id_prev_c + 1];
        }
        d[id_cur] = d[id_prev] - repeat_coef_1 * d[id_prev - 1] - repeat_coef_2 * d[id_prev + 1];
    }
}

__global__ void backStepKernel(double* c, double* a, double* b, double* d, int a_size,
    int c_size, int a_start_id, int c_start_id, int a_sum, int i, int q_input, int eq_use_count, int i)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < q_input - 1 && idx < eq_use_count)
    {
        int shift = 1 << i;
        int id_cur = a_sum + shift + idx * (shift << 1) - 1;
        int j_2 = idx * 2;
        if (idx > 0 && idx < eq_use_count - 1)
        {
            d[id_cur] = (d[a_start_id + j_2] - c[c_start_id + j_2 - 1] * d[id_cur - shift] -
                b[c_start_id + j_2] * d[id_cur + shift]) / a[a_start_id + j_2];
        }
        else if (idx == 0)
        {
            d[id_cur] = (d[a_start_id] - b[c_start_id] * d[id_cur + shift]) / a[a_start_id];
        }
        else
        {
            d[id_cur] = (d[a_start_id + a_size - 1] - c[c_start_id + c_size - 1] * d[id_cur - shift])
                / a[a_start_id + a_size - 1];
        }
    }
    else if (idx == i / 2) d[a_sum + i / 2] = d[a_sum - 1] / a[a_sum - 1];
}

```

Рисунок 1 – Программный код функций ядра прямого и обратного ходов редукции

Ядро *forwardStepKernel* вычисляет коэффициенты редуцированных матриц. Индексы *id_prev* и *id_prev_c* указывают на элементы текущей матрицы, *id_cur* и *id_cur_c* — на соответствующие элементы редуцированной. Множители *repeat_coef_1* и *repeat_coef_2*, повторяющиеся в формулах, используются для расчёта новых диагональных элементов *a* и правой части *d*. Алгоритм адаптирован для решения нестационарных задач: на первом временном слое дополнительно вычисляются коэффициенты редуцированной матрицы *c*, *a* и *b*; на остальных слоях — только *d*. Расчёт продолжается до сведения системы к одному уравнению. Коэффициенты СЛАУ: верхняя диагональ *c*, главная диагональ *a* и нижняя диагональ *b*, а также искомый вектор *x* и правая часть *d* хранятся в одномерных массивах.

На этапе обратного хода ядро *backStepKernel* восстанавливает неизвестные *x* от самой маленькой системы к исходной. Параметр *i* задаёт уровень редукции, а *shift*, равный двойке в степени *i* — расстояние между уже найденными компонентами в массиве *d*. Нити параллельно вычисляют искомые *x*: индекс *id_cur* указывает позицию в *d*, а формула выбирается по положению уравнения (первое, внутреннее или последнее) с подстановкой соседних значений *x* со сдвигом *shift*. Для системы из одного уравнения выполняется прямое деление правой части на *a*. Так последовательно находятся все *x* исходной системы.

2. Исследование эффективности метода и вычислительный эксперимент

Последовательная версия алгоритма реализована на C++. Параллельная версия разработана с использованием CUDA, каждая нить обрабатывает одно уравнение на каждом этапе редукции. Вычислительные эксперименты проводились на конфигурации, которая представлена на таблице 1.

Таблица 1 – Конфигурация вычислительной машины

Устройство	Описание
Центральный процессор	Intel Core i5-12450H 8 ядер, 2.0 GHz
Оперативная память	16 GB DDR4, 3200 MHz
Графический процессор	Nvidia GeForce RTX 4050 Laptop, 2560 CUDA ядер, 1605-2370 MHz, 6 GB, GDDR6

Ускорение считалось, как отношение среднего времени выполнения последовательного алгоритма к среднему параллельному:

$$S = \frac{T_s}{T_p}.$$

В таблице 2 представлены последовательное и параллельное время, а также ускорение алгоритма циклической редукции.

Таблица 2 – Последовательное, параллельное время
и ускорение алгоритма циклической редукции

N	T_s	T_p	S	N	T_s	T_p	S
$2^6 - 1$	0,00000256	0,00077871	0,00329388	$2^{16} - 1$	0,00119075	0,00101144	1,17728416
$2^7 - 1$	0,00000366	0,00101809	0,00359495	$2^{17} - 1$	0,00216847	0,00116529	1,86087401
$2^8 - 1$	0,00000757	0,00089855	0,00843024	$2^{18} - 1$	0,00444256	0,00158144	2,80917146
$2^9 - 1$	0,0000146	0,00094547	0,01544197	$2^{19} - 1$	0,00976660	0,00202350	4,82657814
$2^{10} - 1$	0,00002248	0,00095000	0,02366838	$2^{20} - 1$	0,01752385	0,00346072	5,06363768
$2^{11} - 1$	0,00005082	0,00103317	0,04918806	$2^{21} - 1$	0,03483096	0,00629764	5,53079083
$2^{12} - 1$	0,00010080	0,00096082	0,10491498	$2^{22} - 1$	0,07557847	0,01217662	6,20685173
$2^{13} - 1$	0,00016769	0,00098593	0,17008282	$2^{23} - 1$	0,15467423	0,02372273	6,52008421
$2^{14} - 1$	0,00028236	0,00094843	0,29771252	$2^{24} - 1$	0,30049644	0,04778012	6,28915150
$2^{15} - 1$	0,00056125	0,00093839	0,59809848	$2^{25} - 1$	0,55452103	0,09116737	6,08245032

На рисунке 2 представлены графики времени работы последовательного, параллельного алгоритмов и ускорение.

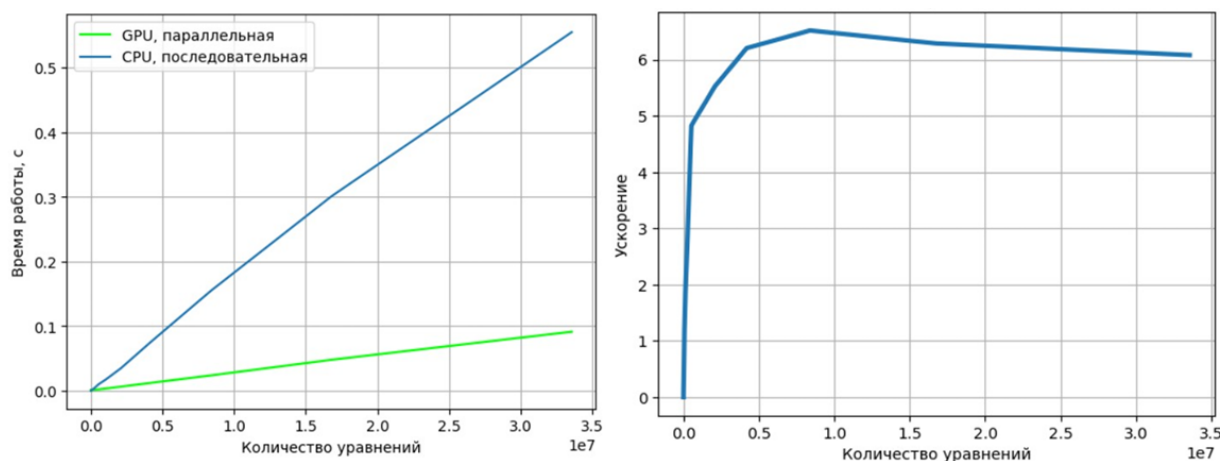


Рисунок 2 – Графики времени работы последовательного, параллельного алгоритмов
и ускорение

Эксперименты показали, что для малых систем (менее $2^{16} - 1$ уравнений) последовательная версия предпочтительнее из-за накладных расходов на организацию параллелизма. Для систем размером $2^{23} - 1$ уравнений достигнуто максимальное ускорение параллельной реализации – 6,52 раза. При дальнейшем росте размерности системы ускорение снижается ввиду насыщения вычислительных ресурсов GPU.

Заключение

В ходе работы разработан параллельный алгоритм метода циклической редукции на GPU с использованием CUDA, восполняющий отсутствие в литературе описания адаптации метода для архитектуры NVIDIA. Тестирование на модели теплопроводности показало, что максимальное ускорение достигает 6,52 для системы размером $2^{23} - 1$.

Список использованных источников

1. Zhang, Y. Parallel Implicit Solvers for 2D Numerical Models on Structured Meshes / Y. Zhang, M. Z. Al-Hamdan, X. Chao // *Mathematics*. – 2024. – V. 12, № 14. – P. 2184–2221. – DOI: 10.3390/math12142184.
2. Altybay, A. Numerical Simulation and Parallel Computing of Acoustic Wave Equation in Isotropic-Heterogeneous Media / A. Altybay, N. Tokmagambetov // *Computer Modeling in Engineering & Sciences*. – 2024. – V. 141, № 2. – P. 1867–1881.
3. Ghosh, A. Highly efficient parallel algorithms for solving the Bates PIDE for pricing options on a GPU / A. Ghosh, C. Mishra // *Applied Mathematics and Computation*. – 2021. – V. 409. – P. 126411.

PARALLEL IMPLEMENTATION OF THE CYCLIC REDUCTION METHOD ON A GRAPHICS PROCESSING UNIT

L.V. Shimchenko¹

¹*Samara University, Samara
e-mail: lev.shimchlev@gmail.com*

Abstract

This paper presents a CUDA implementation of the cyclic reduction method for solving tridiagonal systems of linear algebraic equations (SLAEs). A comparative performance analysis with a sequential algorithm is conducted. A speedup of up to 6.52 is demonstrated for a $2^{23} - 1$ size.

Keywords: CUDA, graphics processing unit, parallel algorithm, cyclic reduction method, finite difference method, heat equation.