Московский авиационный институт
(национальный исследовательский университет)
Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект по курсу «Численные методы»

на тему

«Распараллеливание алгоритмов решения задач линейной алгебры»

Работу выполнил:
Лопатин А. О.
Группа: М80-307Б
Преподаватель:
Ревизников Д. Л.

Москва
2020

Содержание

Постановка задачи **3**

1. Синхронный алгоритм Якоби **3**

2. Параллельный алгоритм Якоби **4**

3. Тест производительности **6**

4. Техническая реализация **7**

Заключение **9**

Перечень использованных источников **10**

Постановка задачи

Необходимо распараллелить алгоритм Якоби для решения полной проблемы собственных значений и собственных векторов симметрических матриц и сравнить его производительность с синхронным алгоритмом.

1. Синхронный алгоритм Якоби

Пусть дана симметрическая матрица А. Требуется для нее вычислить с точностью ϵ все собственные значения и соответствующие им собственные векторы. Алгоритм метода вращения следующий:
Пусть известна матрица $A^{(k)}$ на k–й итерации, при этом для k=0 $A = A^{(0)}$.

1. Выбирается максимальный по модулю внедиагональный элемент $a_{ij}^{(k)}$ матрицы $A^{(k)}$.

2. Ставится задача найти такую ортогональную матрицу $U^{(k)}$, чтобы в результате преобразования подобия $A^{(k+1)} = U^{(k)T} A^{(k)} U^{(k)}$ произошло обнуление элемента $a_{ij}^{(k+1)}$ матрицы $A^{(k+1)}$. В качестве ортогональной матрицы выбирается матрица вращения, в которой на пересечении i—й строки и j—го столбца находится элемент $u_{ij}^{(k)} = -sin\ \phi^{(k)}$ где $\phi^{(k)}$ - угол вращения, подлежащий определению. Симметрично относительно главной диагонали (j-я строка, i-й столбец) расположен элемент $u_{ji}^{(k)} = sin\ \phi^{(k)}$; Диагональные элементы $u_{ii}^{(k)}$ и $u_{jj}^{(k)}$ равны $cos\ \phi^{(k)}$; другие диагональные элементы равны единице, а все остальные элементы равны нулю.
   Угол вращения $\phi^{(k)}$ определяется из условия $a_{ij}^{(k+1)} = 0$:

$$\phi^{(k)} = 0.5 arctg\ \frac{2a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}}$$

   причем если $a_{ii}^{(k)} = a_{jj}^{(k)}$, то $\phi = \frac{\pi}{4}$.

3. Строится матрица $A^{(k+1)} = U^{(k)T} A^{(k)} U^{(k)}$, в которой элемент $a_{ij}^{(k+1)} \approx 0$. В качестве критерия окончания итерационного процесса используется условие малости суммы квадратов внедиагональных элементов:

$$t(A^{(k+1)}) = \sqrt{\sum a_{lm}^{(k+1)\ 2}}$$

   Если $t(A^{(k+1)}) > \epsilon$, то итерационный процесс продолжается. Если $t(A^{(k+1)}) \leq \epsilon$, то итерационный процесс останавливается, и в качестве искомых собственных значений принимаются $\lambda_1 \approx a_{11}^{(k+1)}, \lambda_2 \approx a_{22}^{(k+1)}, ..., \lambda_n \approx a_{nn}^{(k+1)}$.
   Координатными столбцами собственных векторов матрицы А в единичном базисе будут столбцы матрицы $U = U^{(0)} U^{(1)} ... U^{(k)}$. [2]

Замечание: не нужно умножать матрицы $U^{(k)}$ и $A^{(k)}$ напрямую: при умножении в матрице А изменятся только две строки/столбца, можно просто посчитать только эти две строчки или два столбца, тем самым улучшив асимптоматику с $O(n^3)$ до $O(n^2)$. Данное замечание лежит в основе параллельного алгоритма Якоби.

2. Параллельный алгоритм Якоби

После умножения транспонированной матрицы поворота слева $U^T * A = A'$ меняются только две строки, после умножения $A' * U$ справа меняются только два столбца. Следовательно, если будет выбрано $k \leq n/2$ элементов M из матрицы A таких, что любой элемент не лежит на диагонали и любая пара элементов не лежит на одной строчке или столбце, $U^{(i)}$ - матрица вращения для i-того элемента M, $A' = U^{(0)T}U^{(1)T}...U^{(k-1)T}A *$ $* U^{(k-1)}U^{(k-2)}...U^{(0)}$, то:

- Пользуясь ассоциативностью умножения матриц, можно сначала вычислить матрицу $A^R = U^{(0)T}U^{(1)T}...U^{(k-1)T}A$, потом вычислить $A' = A^C = A^R U^{(k-1)}U^{(k-2)}...U^{(0)}$;

- Следуя из построения множества M, все матрицы $U^{(i)T}$ и $U^{(i)}$ изменяют только строки/столбцы p и q, причем никакая другая матрица поворота не будет их изменять. Это свойство дает нам возможность распараллелить вычисление матриц A^R и A^C.

Таким образом, если размер матрицы n x n, на одном шаге можно провести до $n/2$ параллельных вращений исходной матрицы. [1]
Для достаточного условия сходимости наложим дополнительное условие на множество M: из всех возможных вариантов выбора элементов возьмем тот, у которого сумма модулей выбранных элементов максимальна (можно сказать, что это условие "обобщает" условие выбора элемента в синхронном алгоритме).
Покажем, как можно построить множество M жадным алгоритмом. Хоть жадный алгоритм и не выполнит достаточного условия, он прост в реализации и относительно быстр.

1. Инициализируем массив булевых значений $Locked[n]$ значением False, создаем пустой массив $maxElems[n/2]$ для хранения найденных элементов;

2. Ищем максимальный по модулю внедиагональный элемент $A[i, j]$ такой, что $!Locked[i]$ и $!Locked[j]$ верны;

3. Такой элемент из предыдущего пункта найден, ставим значения $Locked[i] = true$ и $Locked[j] = true$, записываем найденный элемент в массив $maxElems$, если количество элементов в нем меньше n/2, то переходим к пункту 2, иначе выходим из алгоритма;

Оценим асимптотическую сложность этого алгоритма. Если просматривать элементы на каждом шаге последовательно, то получается $O(n^2)$ на шаг, учитывая, что шагов $n/2$, получаем асимптотическую сложность $O(n^3)$.
Теперь можно описать параллельный алгоритм Якоби:

1. Строим множество M по предложенному выше алгоритму, для каждого элемента вычисляем ϕ;

2. Запускаем несколько параллельных процессов для подсчета A^R (изменения сразу записываются в матрицу A) и ждем, пока они все закончат свою работу;

3. Запускаем несколько параллельных процессов для подсчета A^C (изменения сразу записываются в матрицу A) и для подсчета матрицы собственных векторов и ждем, пока они все закончат свою работу;

4. Проверяем условие малости внедиагональных элементов, если их норма больше ϵ, то переходим к пункту 1, иначе выходим из алгоритма;

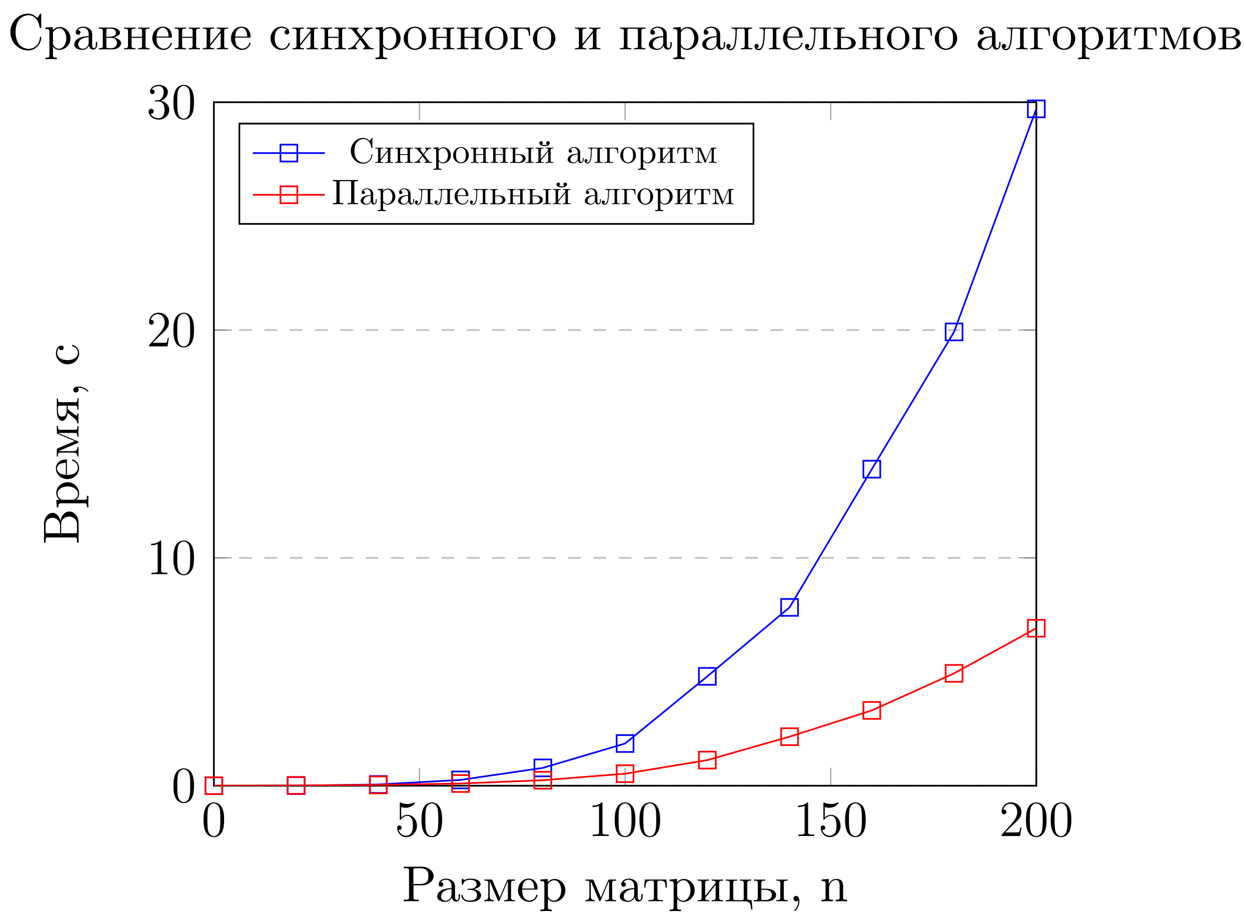
Заметим, что на последнем шаге алгоритм может запустить больше процессов вычисления, чем необходимо для заданной точности. Чтобы это предотвратить, попробуем оценить нижнюю границу модуля внедиагонального элемента:

$$||A|| = \sqrt{\sum A[i,j]^2} \geq \sqrt{\sum \tfrac{4\epsilon^2}{n^2}} = \epsilon$$

Нижняя граница $\frac{2\epsilon}{n}$. Таким образом, если в алгоритме построения множества M на шаге 3 был найден элемент такой, что он меньше $\frac{2\epsilon}{n}$, значит, что все остальные рассмотренные элементы как минимум не больше найденного, а, следовательно, необходимо прервать поиск и выйти из алгоритма. Причем данная модификация не "зациклит" алгоритм, ведь в таком случае получится, что норма действительно меньше заданной точности. На практике алгоритм с этой модификацией работает в среднем на 5-7% быстрее.
Самой дорогой операцией в этом алгоритме является построение множества M. Вполне возможно, что существует способ улучшить асимптоматику этой операции, но он пока что не был найден.

3. Тест производительности

Скорость работы параллельной и синхронной вариаций алгоритма замерялись на одних и тех же данных на ПК с процессором Intel Core i5-4590 с 4 ядрами и частотой 3.3 GHz. Программа автоматически тестируется с помощью скрипта, написанного на языке оболочки PowerShell.



В среднем, параллельный алгоритм работает быстрее синхронного в 3-4 раза.

4. Техническая реализация

Для реализации алгоритма Якоби мною был выбран язык программирования C#, поскольку с помощью средств библиотеки TPL можно очень просто распараллелить задачи. Также я использую самописный класс матрицы, его исходный код (и весь остальной код программы) можно посмотреть на Github'е, здесь я приведу листинг метода нахождения множества M и, собственно, самого параллельного алгоритма Якоби:

```
private (int, int) FindMax()
    {
        (int, int) result = (0, 0);
        double max = 0;
        for(int i = 0; i < A.Rows; i++)
            for (int j = 0; j < i; j++)
            {
                if (isLocked[i])
                    break;
                if (Math.Abs(A[i, j]) > max && !isLocked[j])
                    (result, max) = ((i, j), Math.Abs(A[i, j]));
            }
        return result;
    }
```

Листинг 1: построение множества M

```
private void Rotate()
    {
        List<(int, int, double)> maxElems = new List<(int, int,
        ↪  double)>();
        isLocked = new bool[A.Columns];
        int count = (A.Columns % 2 == 1) ? (A.Columns / 2) : Math.Max(1,
        ↪  A.Columns / 2 - 1);
        for (int k = 0; k < count; k++)
        {
            int i = 0;
            int j = 0;
            double fi = 0;
            (i, j) = FindMax();
            if (Math.Abs(A[i, j]) < eps / A.Rows)
                break;
            isLocked[i] = true;
            isLocked[j] = true;
            fi = 0.5 * Math.Atan(2 * A[i, j] / (A[i, i] - A[j, j]));
            maxElems.Add((i, j, fi));
        }
        Task[] taskArray = new Task[maxElems.Count];
        for (int k = 0; k < maxElems.Count; k++)
        {
            int i, j = 0;
            double fi = 0;
            (i, j, fi) = maxElems[k];
            taskArray[k] = Task.Run(() => SumRows(A, i, j, fi));
        }
        Task.WaitAll(taskArray);
        taskArray = new Task[maxElems.Count * 2];
        for (int k = 0; k < maxElems.Count; k++)
        {
            int i, j = 0;
            double fi = 0;
            (i, j, fi) = maxElems[k];
            taskArray[k] = Task.Run(() => SumColumns(A, i, j, fi));
            taskArray[maxElems.Count + k] = Task.Run(() =>
            ↪  SumColumns(eigenVectorsRaw, i, j, fi));
        }
        Task.WaitAll(taskArray);
    }
```

Листинг 2: параллельный алгоритм Якоби

Заключение

В ходе работы над данным курсовым проектом я поближе познакомился со средствами разработки параллельных приложений на языке программирования $C\#$, разобрался в работе метода вращений Якоби для нахождения собственных значений и собственных векторов симметричной матрицы, понял, как можно автоматизировать проверку производительности используя расширяемое средство автоматизации PowerShell.

Перечень использованных источников

1. Jim Lambers, CME 335, Lecture 7 Notes. — URL: https://web.stanford.edu/class/cme335/lecture7.pdf.
2. Численные методы, Формалев В.Ф., Ревизников Д.Л., 2006.