**Практическая работа № 2 (10 баллов)**

**Задание 1 (5).**

*Язык программирования: C#, класс Parallel*

Необходимо реализовать и сравнить алгоритмы параллельного умножения квадратных матриц с распараллеливанием по строкам и столбцам. Матрицы генерировать случайными целочисленными значениями. Для перебора использовать методы класса Parallel (C#).

Запустите и определите время работы для разных размерностей и для разного количества потоков.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Размерность | Потоки | | | | | | |
| 1 | 2 | 4 | 8 | 12 | 16 | 20 |
| 1000x1000 |  |  |  |  |  |  |  |
| 2000x2000 |  |  |  |  |  |  |  |
| 5000x5000 |  |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |  |

По результатам составить Excel-таблицы, в которых сравнить время выполнения алгоритмов для матриц разных размерностей и разных способов декомпозиции.

Помимо таблицы со временем, добавьте расчет и таблицу/график ускорения (Speedup) и эффективности (Efficiency) для каждого количества потоков.

* Ускорение: S(n) = T(1) / T(n), где T(1) - время на 1 потоке, T(n) - время на n потоках.
* Эффективность: E(n) = S(n) / n \* 100%. Показывает, насколько эффективно используются потоки.

Ответьте на вопросы:

* Какой метод декомпозиции (по строкам, столбцам) оказался наиболее эффективным для каждого размера матрицы и почему?
* Наблюдается ли ожидаемое ускорение? Для каких размерностей матриц ускорение ближе к идеальному (линейному), а для каких — сильно отстает?
* Существует ли "точка насыщения", после которой добавление новых потоков не уменьшает, а может даже увеличивает время выполнения?

**Задание 2 (5).**

*Язык программирования: C#, класс Thread*

1. Выберите функцию, для которой нужно вычислить значение определенного интеграла (определите a и b – границы интегрирования). Желательно выбирать функцию, вычисление которой является затратным (например, sin(x) / x, e^(-x^2), log(1 + x^2)).
2. Для определения значения интеграла выберите один из методов численного интегрирования (левых, правых или средних прямоугольников).
3. Реализуйте метод последовательно и параллельно. Разделите участок интегрирования на равные части (в зависимости от метода).
4. Исследуйте зависимость времени работы алгоритма от числа потоков (1,2,4,…,20) и желаемой точности   
    (ε=0.001, 0.0001, 0.00001, …, 0.000000001, …). Для достижения необходимой точности на каждом шаге увеличивать количество разбиений в 2 раза, пока уточнение результата не станет изменяться меньше, чем на ε.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Точность | Потоки | | | | | | |
| 1 | 2 | 4 | 8 | 12 | 16 | 20 |
| 0.001 |  |  |  |  |  |  |  |
| 0.0001 |  |  |  |  |  |  |  |
| 0.00001 |  |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |  |
| 0.000000001 |  |  |  |  |  |  |  |

Ответьте на вопросы:

* Во сколько раз параллельная версия ускорила вычисление для разных значений точности ε?
* Как увеличение требуемой точности (уменьшение ε) влияет на оптимальное количество потоков? Есть ли точность, после которой использование большого числа потоков становится невыгодным?

**Дополнительные материалы:**

1. Использование класса Parallel в C#

Класс Parallel (пространство имен System.Threading.Tasks) предоставляет простой способ организации параллельных циклов, что идеально подходит для задач с большим объемом независимых вычислений (например, перебор элементов матрицы или отрезков интегрирования).

Основные методы:

* Parallel.For: Параллельный аналог цикла for.
* Parallel.ForEach: Параллельный аналог цикла foreach.

Базовый синтаксис Parallel.For:

*// Последовательный цикл*

for (int i = 0; i < n; i++)

{

*// Работа, которая будет выполнена параллельно*

DoWork(i);

}

*// Его параллельный эквивалент*

Parallel.For(0, n, i =>

{

DoWork(i);

});

Важные моменты

Потокобезопасность: Код внутри параллельного цикла должен быть потокобезопасным. При заполнении матрицы в разных потоках важно, чтобы никакие два потока не заполняли одну и ту же ячейку.

Настройка параллелизма: Вы можете ограничить максимальное количество потоков, используемых циклом, с помощью класса ParallelOptions.

var options = new ParallelOptions { MaxDegreeOfParallelism = 4 }; *// Не более 4 потоков*

Parallel.For(0, n, options, i =>

{

DoWork(i);

});

2. Ограничение времени выполнения (Timeout)

Часто необходимо ограничить время выполнения длительной операции, чтобы программа не "зависала". Для отмены выполнения потока можно использовать CancellationToken.

public static double CalculateWithTimeout(int maxSeconds)

{

// Создаем источник токена отмены

var cts = new CancellationTokenSource();

// Устанавливаем таймаут на maxSeconds секунд

cts.CancelAfter(TimeSpan.FromSeconds(maxSeconds));

CancellationToken token = cts.Token;

double result = 0;

try

{

while (true)

{

// На каждой итерации проверяем, не потребовали ли отмену

token.ThrowIfCancellationRequested();

// Здесь код вычислений

}

}

catch (OperationCanceledException)

{

// Токен сработал по таймауту

Console.WriteLine($"Вычисление прервано по таймауту ({maxSeconds} сек).");

return result; // Возвращаем последний вычисленный результат

}

}