ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ УМНОЖЕНИЯ ТРАНСПОНИРОВАННОЙ МАТРИЦЫ НА ВЕКТОР

ВВЕДЕНИЕ

Умножение матрицы на вектор — одна из ключевых операций в линейной алгебре, которая встречается в различных областях, таких как численные методы, машинное обучение, компьютерная графика и т.д. В условиях больших матриц и векторов эта операция становится вычислительно затратной, особенно когда требуется трансформация матрицы, например, её транспонирование.Для оптимизации этой задачи применяются параллельные вычисления, которые позволяют разделить работу между несколькими процессами и тем самым ускорить выполнение алгоритма.

1.ПОСТАНОВКА ЗАДАЧИ

- Задача состоит в вычислении произведения транспонированной матрицы на A на вектор x То есть:

$$b = A^T \cdot x$$

- A — исходная матрица размером $N \times M$,
- x — вектор размером M,
- b — результирующий вектор размером N.

3. ТЕОРЕТИЧЕСКАЯ ОСНОВА ПАРАЛЛЕЛЬНОГО УМНОЖЕНИЯ ТРАНСПОНИРОВАННОЙ МАТРИЦЫ НА ВЕКТОР

Разделение матрицы и вектора

Для распределения данных между процессами:

1.Транспонированная матрица делится на блоки строк. Каждый процесс получит одну или несколько строк матрицы для обработки.

2.Вектор x рассылается каждому процессу, так как каждая часть строки транспонированной матрицы должна быть умножена на весь вектор x.

Каждый процесс:
1.Получает свою часть матрицы A^T и вектор x.
2.Выполняет частичное умножение: для каждого элемента результирующего вектора b, процесс вычисляет скалярное произведение своей части строки матрицы на вектор x.

После вычисления частичных результатов, они отправляются на корневой процесс (или остаются на каждом процессе), где они суммируются с помощью функции MPI Reduce, которая суммирует результаты и собирает их в один конечный вектор bbb

Так как массив A уже разбит на блоки, разобъём вектор на блоки, согласованно с разбиением матрицы. Дальше на каждом процессе будет происходить умножение частей матриц на части вектора. Полученные вектора с разных процессов нужно будет сложить. Схему этой операции можно увидеть на следующем рисунке

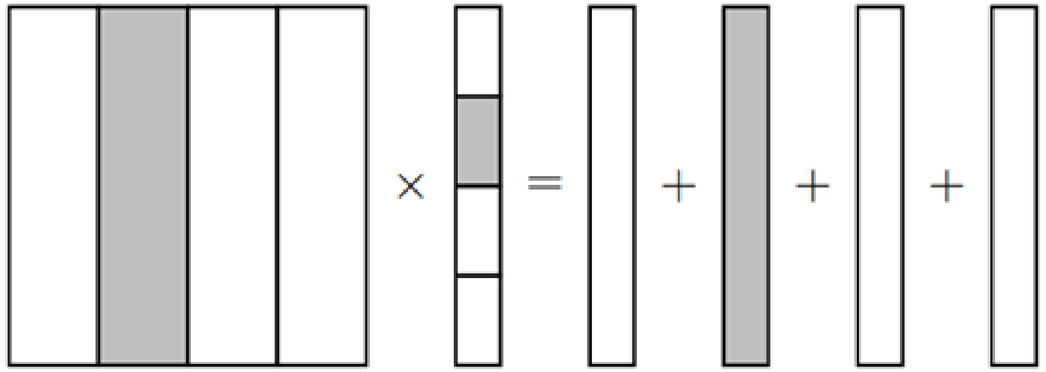

Рис. 2.2: Схема умножения транспонированной матрицы на вектор

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ УМНОЖЕНИЯ ТРАНСПОНИРОВАННОЙ МАТРИЦЫ НА ВЕКТОР

```
from mpi4py import MPI
from numpy import empty, array, int32, float64, dot

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
numprocs = comm.Get_size()
```

В этом коде мы импортируем библиотеки mpi4py для параллельных вычислений с использованием MPI и numpy для работы с матрицами и векторами, создаем объект comm, представляющий группу процессов, участвующих в вычислениях, где каждый процесс имеет уникальный идентификатор rank, а numprocs указывает общее количество процессов, запущенных для выполнения программы.

```
# Размеры матрицы
N = 3  # Число строк матрицы A
M = 4  # Число столбцов матрицы A (равно размеру вектора x)

# Матрица и вектор на процессе с рангом 0
if rank == 0:
    A = array([[1.0, 2.0, 3.0, 4.0],
               [5.0, 6.0, 7.0, 8.0],
               [9.0, 10.0, 11.0, 12.0]], dtype=float64)
    x = array([1.0, 2.0, 3.0, 4.0], dtype=float64)
else:
    A = None
    x = None
```

Здесь мы определяем размеры матрицы A и вектора x.
N — количество строк матрицы A (3).
M — количество столбцов матрицы A (4), и также это размер вектора x.
Только процесс с рангом 0 (корневой процесс) инициализирует полную матрицу A и вектор x.
Остальные процессы пока не имеют данных (A = None, x = None).

РАЗДЕЛЕНИЕ МАТРИЦЫ И ВЕКТОРА МЕЖДУ ПРОЦЕССАМИ

- Здесь происходит подготовка к распределению данныхмежду процессами.
- rcounts хранит количество строк, которые будут переданы каждому процессу.
- displs хранит смещения (индексы начала строк), откуда каждый процесс должен начинать чтение.
- В корневом процессе (rank == 0) мы определяем, сколько строк должно быть передано каждому процессу.
- ave — это среднее количество строк, которые получает каждый процесс.
- res — остаток, который мы добавляем к распределению строк для равномерного разделения данных.
- Затем цикл распределяет строки между процессами с рангами от 1 до numprocs−1. Корневой процесс с rank=0 не участвует в вычислениях.

```
# Распределение данных между процессами
rcounts = empty(numprocs, dtype=int32)  # Количество строк для каждого процесса
displs = empty(numprocs, dtype=int32)   # Индексы смещений для каждого процесса

if rank == 0:
    ave, res = divmod(M, numprocs - 1)
    rcounts[0] = 0
    displs[0] = 0
    for k in range(1, numprocs):
        rcounts[k] = ave + 1 if k <= res else ave
        displs[k] = displs[k - 1] + rcounts[k - 1]
```

РАССЫЛКА ВЕКТОРА И ДАННЫХ МАТРИЦЫ

```
# Рассылка вектора x всем процессам
x_part = empty(M, dtype=float64)
comm.Bcast([x, MPI.DOUBLE], root=0)

# Каждый процесс получает часть матрицы A^T
A_part = empty((M, N), dtype=float64)
comm.Scatterv([A, rcounts, displs, MPI.DOUBLE], A_part,
root=0)
```

В этой части происходит распределение данныхмежду процессами.
Вектор x передается каждому процессу с использованием MPI-функции Bcast (broadcast). Это важно, так как каждый процесс должен иметь полный вектор для выполнения умножения.
Для матрицы используется функция Scatterv, которая раздает строки матрицы транспонированного вида (A^T) каждому процессу. Каждый процесс получает только свою часть строк матрицы A^T.

Вычисление частичного результата

```
# Часть вычисления произведения A^T * x
b_temp = dot(A_part.T, x_part)
```

После того как каждый процесс получил свою часть транспонированной матрицы и вектор, происходит вычисление частичного произведения.
Функция dot() выполняет операцию умножения каждой строки матрицы A^T на вектор x для получения частичного результата.
Важно: каждый процесс вычисляет только свою часть итогового вектора b.

СБОР РЕЗУЛЬТАТОВ И ВЫВОД ИТОГОВ

```
# Сбор всех частичных результатов на корневом процессе
b = None
if rank == 0:
    b = empty(N, dtype=float64)
comm.Reduce(b_temp, b, op=MPI.SUM, root=0)

# Вывод результата на процессе 0
if rank == 0:
    print("Результат умножения A^T на x:", b)
```

- *Каждый процесс вычисляет свою часть результата в переменной b_temp.*
- *С помощью функции Reduce все частичные результаты собираются и суммируются на корневом процессе (rank=0).*
- *Аргумент op=MPI.SUM означает, что результаты будут суммироваться.*
- *Корневой процесс выводит на экран окончательный результат умножения транспонированной матрицы на вектор.*