**Абдрашева Асия**, группа БПИ194. Вариант №1.  
  
**Задание**: Вычислить векторное произведение квадратных матриц А и B. Входные данные: произвольные квадратные матрицы А и В одинаковой размерности. Размер матриц задается входным параметром. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

**Описание**

Для решения задачи был выбран итеративный параллелизм. Используется для реализации нескольких потоков (часто идентичных), каждый из которых содержит циклы. Потоки программы, описываются итеративными функциями и работают совместно над решением одной задачи.

Передача управления к другому потоку, когда операция записи не закончена, могут привести, например, к потере записи нескольких результатов или разрушению связей между элементами. Поэтому воспользуемся двоичным семафором, с помощью которого будет защищена операция записи.

Источники информации: Лекция №7 и семинар по Архитектуре вычислительных систем.

Метод fillTheMatrix() – функция для дочерних потоков, в которой вычисляются элементы матрицы, стоящие в строках, отвечающих за определенный поток.

//стартовая функция для дочерних потоков

void fillTheMatrix(int iTread, int iTN, int\*\* A, int\*\*B, int size, int\*\*C) {

for (int k = iTread; k < size; k += iTN) {

//вычисление элементов матрицы, стоящих в строке k

for (int i = 0; i < size; i++) {

C[k][i] = 0;

for (int j = 0; j < size; j++) {

C[k][i] += A[k][j] \* B[j][i];

}

//приведенный ниже код в комментариях не нужен был при выводе в условии,

//но если его раскомментировать, то можно отследить за какие строки отвечает определенный поток

////протокол входа в КС: закрыть двоичный семафор

////начало критической секции

//mtx.lock();

//std::cout « "Thread " « iTread « ": element [" « k « "][" « i « "] = " « C[k][i] « std::endl;

////конец критической секции

////протокол выхода из КС:

//mtx.unlock();

////открыть двоичный семафор

}

}

}

Метод matrixPrint() – метод, который выводит в консоль матрицы

//вывод матриц в консоль (вывод матриц А и В не прописан в условии,

//но для понимамания, что именно мы должны умножать, удобно будет их все же вывести)

void matrixPrint(int \*\*matrix, int matrixSize)

{

for (size\_t i = 0; i < matrixSize; i++)

{

for (size\_t j = 0; j < matrixSize; j++)

{

std::cout << matrix[i][j] << "\t";

}

std::cout << "\n";

}

}

Метод main() – точка входа в программу, метод, в котором создаются потоки, вводятся с консоли значения размерности матрицы и количества потоков. При этом количество потоков может быть не кратно размерности матрицы. В таком случае каждый поток отвечает за целое число от деления размерности матрицы на количество потоков, а последний поток берет на себя еще и остаток от деления размерности матрицы на количество потоков. Если же количество потоков кратно размерности матрицы, то в таком случае каждый поток просто отвечает за деление.

{

A[i] = new int[matrixSize];

B[i] = new int[matrixSize];

resultMatrix[i] = new int[matrixSize];

for (size\_t j = 0; j < matrixSize; j++)

{

A[i][j] = rand() % 20; //числа от рандома получаются слишком большими, просто для удобства возьмем остатки от деления на 20

B[i][j] = rand() % 20;

resultMatrix[i][j] = 0;

}

}

std::cout << "A matrix:" << std::endl;

//вывод первой матрицы

matrixPrint(A, matrixSize);

std::cout << "\n";

std::cout << "B matrix:" << std::endl;

//вывод второй матрицы

matrixPrint(B, matrixSize);

std::cout << "\n";

std::thread\* thr = new std::thread[threadsNum];

// Создание потоков

if (matrixSize % threadsNum != 0) {

for (size\_t i = 0; i < threadsNum - 1; i++)

{

//в один поток пытаемся положить [matrixSize//threadsNum] строк из matrixSize строк

thr[i] = std::thread{ fillTheMatrix, i, threadsNum, A, B, matrixSize, std::ref(resultMatrix) };

}

//если размерность матрицы не кратна количеству потоков => в последний поток кладем [matrixSize//threadsNum] + остаток от деления [matrixSize%threadsNum] строк

thr[threadsNum - 1] = std::thread{ fillTheMatrix, threadsNum - 1, 1, A, B, matrixSize, std::ref(resultMatrix) };

}

else {

for (size\_t i = 0; i < threadsNum; i++)

{

//в один поток кладем [matrixSize//threadsNum] строк из matrixSize строк

thr[i] = std::thread{ fillTheMatrix, i, threadsNum, A, B, matrixSize, std::ref(resultMatrix) };

}

}

for (int i = 0; i < threadsNum; i++) {

thr[i].join();

}

delete[] thr;

std::cout << "\n";

std::cout << "Result matrix = A \* B" << std::endl;

std::cout << "Result matrix:" << std::endl;

//вывод итоговой матрицы resultMatrix = A\*B

matrixPrint(resultMatrix, matrixSize);

//освобождение памяти для каждого элемнта A[i]

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

delete[] A[i];

//освобождение памяти B

delete[] A;

//освобождение памяти для каждого элемнта B[i]

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

delete[] B[i];

//освобождение памяти A

delete[] B;

//освобождение памяти для каждого элемнта resultMatrix[i]

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

delete[] resultMatrix[i];

//освобождение памяти resultMatrix

delete[] resultMatrix;

}

system("pause");

return 0;

}