Абдрашева Асия, группа БПИ194. Вариант №1.

Задание: Вычислить векторное произведение квадратных матриц A и B. Входные данные: произвольные квадратные матрицы A и B 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.

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

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

```
//вывод матриц в консоль (вывод матриц A и B 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. Но эту проблему просто решает метод `omp_set_num_threads(threadsNum)`, который устанавливает количество потоков.

```
int main()
{
	//вводимые пользователем значения размера матрицы и количества потоков
	int matrixSize;
	int threadsNum;

	std::cout << "input matrix size: ";
	std::cin >> matrixSize;
	std::cout << "input threads number: ";
	std::cin >> threadsNum;

	//обработка данных
	if (threadsNum <= 0 || matrixSize <= 0)
	{
		std::cout << "Wrong! Threads number or matrix size can not be less than
0." << std::endl;
	}
	else if (threadsNum > matrixSize)
	{
		std::cout << "Wrong! Threads number can not be more than matrix size."
<< std::endl;
	}
```

```
	else {
		int i, j, k;

		int** A = new int* [matrixSize];
		int** B = new int* [matrixSize];
		int** resultMatrix = new int* [matrixSize];

		for (int i = 0; i < matrixSize; i++)
		{
			A[i] = new int[matrixSize];
			B[i] = new int[matrixSize];
			resultMatrix[i] = new int[matrixSize];
		}


		// заполнение массивов A и B
		for (i = 0; i < matrixSize; i++) {
			for (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;
			}
		}

		cout << "A matrix" << endl;
		//вывод первой матрицы
		matrixPrint(A, matrixSize);
		cout << "B matrix" << endl;

		//вывод второй матрицы
		matrixPrint(B, matrixSize);

		omp_set_num_threads(threadsNum);

#pragma omp parallel shared(a,b,c) private(i,j,k)
		{
#pragma omp for  schedule(static)
			for (i = 0; i < matrixSize; i++)
				for (j = 0; j < matrixSize; j++)
					for (k = 0; k < matrixSize; k++)
						resultMatrix[i][j] += (A[i][k] * B[k][j]);
		}

		std::cout << "\n";
		std::cout << "Result matrix = A * B" << std::endl;
		std::cout << "Result matrix:" << std::endl;
		matrixPrint(resultMatrix, matrixSize);
	}
	return EXIT_SUCCESS;
}
```