

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ

Кафедра ВТ

Лабораторная работа №2  
**«Оценка производительности процессора»**

Факультет: АВТФ

Группа: АПИМ-25

Выполнил: Бадагов А.В., Клименко К.В.

Преподаватель: Перышкова Е.Н.

Отметка о защите:

Новосибирск 2025

**Задание:** Реализовать программу для оценки производительности процессора (benchmark).

1. Написать программу(ы) (benchmark) на языке С/C++/C# для оценки производительности процессора. В качестве набора типовых задач использовать либо минимум 3 функции выполняющих математические вычисления, либо одну функцию по работе с матрицами и векторами данных с несколькими типами данных. Можно использовать готовые функции из математической библиотеки (math.h) [3], библиотеки BLAS [4] (англ. Basic Linear Algebra Subprograms — базовые подпрограммы линейной алгебры) и/или библиотеки LAPACK [5] (Linear Algebra PACKage). Обеспечить возможность в качестве аргумента при вызове программы указать общее число испытаний для каждой типовой задачи (минимум 10). Входные данные для типовой задачи сгенерировать случайным образом

2. С помощью системного таймера (библиотека time.h, функции clock() или gettimeofday()) или с помощью процессорного регистра счетчика TSC реализовать оценку в секундах среднего времени испытания каждой типовой задачи. Оценить точность и погрешность (абсолютную и относительную) измерения времени (рассчитать дисперсию и среднеквадратическое отклонение).

3. Результаты испытаний в самой программе (или с помощью скрипта) сохранить в файл в формате CSV со следующей структурой: [PModel;Task;OpType;Opt;InsCount;Timer;Time;LNum;AvTime;AbsErr;RelErr;TaskPerf], где:

- PModel – Processor Model, модель процессора, на котором проводятся испытания; Task – название выбранной типовой задачи (например, sin, log, saxpy, dgemv, sgemm и др.);
- OpType – Operand Type, тип операндов используемых при вычислениях типовой задачи; Opt – Optimisations, используемые ключи оптимизации (None, O1, O2 и др.);
- InsCount – Instruction Count, оценка числа инструкций при выполнении типовой задачи; Timer – название функции обращения к таймеру (для измерения времени);
- Time – время выполнения отдельного испытания;
- LNum – Launch Numer, номер испытания типовой задачи.
- AvTime – Average Time, среднее время выполнения типовой задачи из всех испытаний[секунды];
- AbsError – Absolute Error, абсолютная погрешность измерения времени в секундах;
- RelError – Relative Error, относительная погрешность измерения времени в %;

- TaskPerf – Task Performance, производительность (быстродействие) процессора при выполнении типовой задачи.

3. \* Оценить среднее время испытания каждой типовой задачи с разным типом входных данных (целочисленные, с одинарной и двойной точностью).

3. \*\* Оценить среднее время испытания каждой типовой задачи с оптимизирующими преобразования исходного кода компилятором (ключи –Debug, –Release).

4. Построить сводную диаграмму производительности в зависимости от задач и выбранных исходных параметров испытаний. Оценить среднее быстродействие (производительность) для равновероятного использования типовых задач.

### **Ход работы:**

Для оценки производительности процессора. В качестве набора типовых задач использовалось функция по работе с матрицами векторами данных с несколькими типами данных. Генерация значений организована случайным образом.

Листинг функции отвечающая за измерения времени на C#

```
// INT
static void BenchmarkMatrixInt(
    string cpuModel,
    string opt,
    string timerName,
    int runs,
    int n,
    StreamWriter writer)
{
    string taskName = $"MatMul_{n}x{n}";
    string opType = "Int32";

    // Оценка числа арифметических операций:
    double insCount = (double)n * n * (2 * n - 1);

    var rnd = new Random(135);
    int[,] A = GenerateMatrixInt(n, n, rnd);
    int[,] B = GenerateMatrixInt(n, n, rnd);
    int[,] C = new int[n, n];

    double[] times = new double[runs];
    var sw = new Stopwatch();

    for (int i = 0; i < runs; i++)
    {
        sw.Restart();
        MultiplyInt(A, B, C);
        sw.Stop();

        times[i] = sw.Elapsed.TotalSeconds;
    }

    CSV(
        cpuModel, taskName, opType, opt, timerName,
        insCount, times, runs, writer);
}
```

Данная функция отвечает за измерения времени выполнения умножения матрицы. На вход функции подаются параметры, полученные в функции *Main*, потом они передаются в функцию *CSV* формирования csv файла. Переменная *insCount* содержит количество арифметических действий, при перемножении двух матриц формула (1), рассмотрим пример с матрицами  $A_{3 \times 3}$   $B_{3 \times 3}$ .

$$C_{01} = A_{00} * B_{00} + A_{01} * B_{10} + A_{02} * B_{20} \quad (1)$$

По формуле (1) можно понять, что для расчёта одного значения итоговой матрицы С необходимо свершить 3 перемножения и два сложения, т.е. n перемножений и n-1 действий сложения, матрица С имеет размер NxN, при  $A_{NxN}$   $B_{NxN}$ . Таким образом *insCount* =  $(NxN) * (n + n - 1) = n * n * (2n - 1)$ . Команда *Random()* нужен, чтобы генератор случайных чисел всегда стартовал из одного и того же состояния, это необходимо, чтобы для каждого запуска программы матрицы А и В получались одинаковыми. Функция *GenerateMatrixInt(n, n, rnd)* нужна для формирования матриц и будет рассмотрена далее. *Stopwatch()* является командой таймером, она осуществляет расчёт времени с момента запуск *sw.Restart()* и до остановки *sw.Stop()*, полученнное значение рассчитывается в секунды *sw.Elapsed.TotalSeconds*.

Листинг функции отвечающей за создание матриц на C#

```
static int[,] GenerateMatrixInt(int rows, int cols, Random rnd)
{
    int[,] m = new int[rows, cols];
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            m[i, j] = rnd.Next(-10, 11); // [-10; 10]
    return m;
```

На вход функции подается количество строк, столбцов и созданный генератор случайных чисел *Random rnd*. Команда *rnd.Next()* выдаёт целое число в диапазоне.

### Листинг функции отвечающей за перемножения матриц на C#

```
// Умножение матриц
static void MultiplyInt(int[,] A, int[,] B, int[,] C)
{
    int n = A.GetLength(0);
    int kDim = A.GetLength(1);
    int m = B.GetLength(1);

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            int sum = 0;
            for (int k = 0; k < kDim; k++)
            {
                sum += A[i, k] * B[k, j];
            }
            C[i, j] = sum;
        }
    }
}
```

На вход функции подаются 2 сформированные двумерные матрицы и матрица в которую будут вносится результаты перемножения. *A.GetLength(0)* используется для определения кол-ва строк в массиве, *B.GetLength(1)* определяет количество столбцов. Цикл осуществляют перебор значений по строкам матрица А и по столбцам матрицы В.

### Листинг функции отвечающей за перемножения матриц на C#

```
// статистика
static void CSV(
    string cpuModel,
    string taskName,
    string opType,
    string opt,
    string timerName,
    double insCount,
    double[] times,
    int runs,
    StreamWriter writer)
{
    double mean = times.Average();

    // Дисперсия
    double variance = 0.0;
    for (int i = 0; i < runs; i++)
    {
        double diff = times[i] - mean;
        variance += diff * diff;
    }

    variance /= runs;
    double stdDev = Math.Sqrt(variance);

    // Абсолютная погрешность оценки среднего
    double absError = stdDev / Math.Sqrt(runs);

    // Относительная погрешность в процентах
    double relError = absError / mean * 100;

    // Производительность
    double taskPerf = insCount / mean;
```

```

for (int i = 0; i < runs; i++)
{
    int launchNum = i + 1;
    double t = times[i];

    writer.WriteLine(
        string.Join(";", 
            cpuModel,
            taskName,
            opType,
            opt,
            insCount,
            timerName,
            t,
            launchNum,
            mean,
            absError,
            relError,
            taskPerf
        ));
}
}

```

Функция *CSV* в твоей программе выполняет роль модуля статистической обработки результатов измерений времени и их записи в файл *results.csv*. На вход она получает описание условий эксперимента (модель процессора, имя задачи, тип данных, режим оптимизации, название таймера, оценку числа операций), массив времён *times* и число запусков *runs*, *StreamWriter* необходим для записи в CSV. *times.Average()* рассчитывает среднее значение массива. В цикле рассчитывается отклонение от среднего значения, для определения абсолютной погрешности формула (2), относительной погрешности (3) и производительности формула (4). В цикле выполняются заполнение CSV файла с рассчитанными значениями метрик.

$$absError = \frac{\sqrt{\sum_{i=1}^{runs} (t_i - \bar{t})^2}}{runs} \quad (2)$$

$$relError = \frac{absError}{\bar{t}} \cdot 100\% \quad (3)$$

$$taskPerf = \frac{insCount}{\bar{t}} \quad (4)$$

где  $\bar{t}$  – среднее значение времени выполнения перемножения матрицы.

Листинг функции, отвечающей за определение статичных параметров на C#

```

static void Main(string[] args)
{
    // Количество запусков (испытаний) каждой задачи
    int runs = 10;
    // Размер квадратной матрицы N×N
    int n = 1000;

    if (args.Length >= 1 && int.TryParse(args[0], out var tmpRuns) && tmpRuns > 0)
        runs = tmpRuns;

    if (args.Length >= 2 && int.TryParse(args[1], out var tmpN) && tmpN > 0)
        n = tmpN;
}

```

```

string cpuModel = Environment.GetEnvironmentVariable("PROCESSOR_IDENTIFIER");

string opt = "Debug";      // "Debug" или "Release"
string timerName = "Stopwatch";
string outputFile = "results.csv";

Console.WriteLine("Matrix benchmark");
Console.WriteLine($"CPU: {cpuModel}");
Console.WriteLine($"Runs: {runs}, matrix size: {n}x{n}");
Console.WriteLine($"Results -> {outputFile}");

using (var writer = new StreamWriter(outputFile, false))
{
    // Заголовок CSV

writer.WriteLine("PModel;Task;OpType;Opt;InsCount;Timer;Time;LNum;AvTime;AbsErr;RelErr;
TaskPerf");

    // Три бенчмарка: int, float, double
    BenchmarkMatrixInt(cpuModel, opt, timerName, runs, n, writer);
    BenchmarkMatrixFloat(cpuModel, opt, timerName, runs, n, writer);
    BenchmarkMatrixDouble(cpuModel, opt, timerName, runs, n, writer);
}

}

```

В данной функции осуществляется ввод размера матрицы и количество испытаний, условия необходимо для проверки введенных данных с консоли. Команда *Environment.GetEnvironmentVariable("PROCESSOR\_IDENTIFIER")* позволяет получить описание процессора. Оператор *using()* открывает файл *result.csv* и создает заголовок в файле, затем запускаются функции отвечающая за измерения времени, в которые передаются статические параметры для записи со всеми рассчитанными значениями, а также *StreamWriter(outputFile, false)*, чтобы функция *CSV* записала строку, с рассчитанными метриками, в созданный уже файл.

Аналогичный функции написаны для типа данных *Float* и *Double* см. приложение А.

В таблице 1-4 приведены результаты выполнения программы, проведем анализ полученных значений рисунок 1.

3\* Среднее время испытания для разных типов входных данных

Для одной и той же задачи перемножения матрицы размером 1000x1000 среднее время испытания зависит и от типа данных, и от режима компиляции. В Debug все три типа дают близкие времена порядка 6 секунд. В Release времена уменьшаются до 3.8–4.3 с, при этом наименьшее время демонстрирует задача с типом *Float64* (*double*).

3\*\* Влияние оптимизирующих преобразований (Debug / Release)

Опираясь на таблицы 1 – 6 можно прийти к заключению, что включение оптимизирующих преобразований (переход от Debug к Release) уменьшает среднее время

испытания каждой типовой задачи примерно в 1.4–1.55 раза. Эффект оптимизации наиболее заметен для задач с *Float32* и *Double*.

4. Сводную диаграмму производительности процессора по итогу перемножения матрицы 1000x1000 с 10 запусками. Оценить среднее быстродействие (производительность) для равновероятного использования типовых задач.

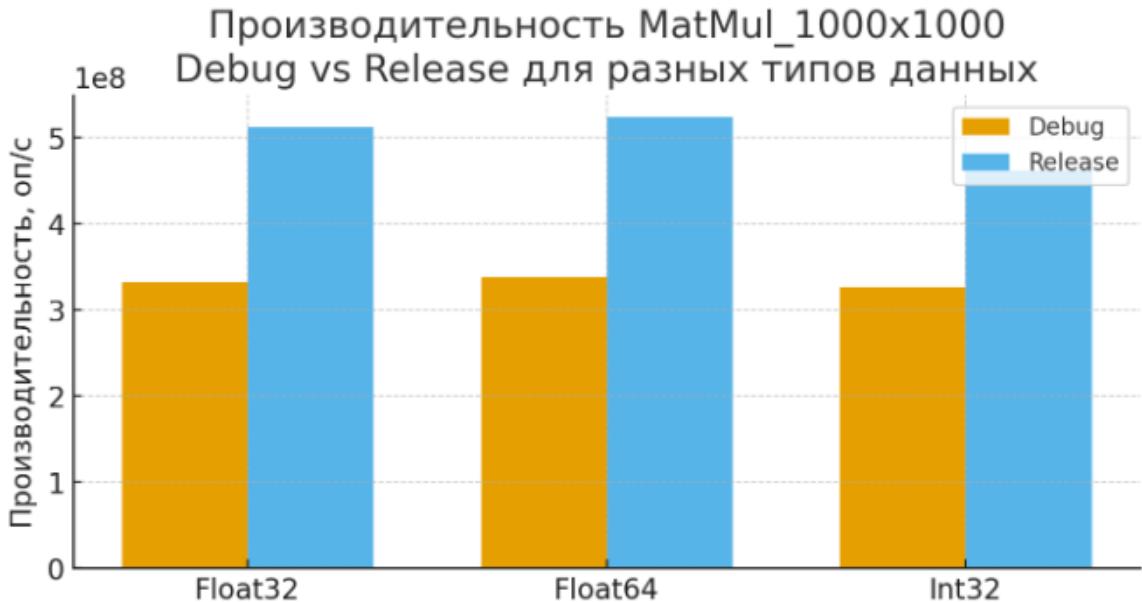


Рисунок 1 – Диаграмму производительности

По формуле (5) рассчитаем быстродействие при равновероятном значении использования типовой задачи: *Int*, *Float*, *Double*.

$$\omega = \left( \sum_{i=1}^L \frac{\pi_i}{\omega_i} \right)^{-1} \quad (5)$$

$$\omega_{Debug} = \frac{3}{\frac{1}{TaskPerf_{Int}} + \frac{1}{TaskPerf_{Float}} + \frac{1}{TaskPerf_{Double}}} = \frac{3}{\frac{1}{3,27*10^8} + \frac{1}{3,33*10^8} + \frac{1}{3,37*10^8}} = 3,322*10^8$$

$$\omega_{Release} = \frac{3}{\frac{1}{TaskPerf_{Int}} + \frac{1}{TaskPerf_{Float}} + \frac{1}{TaskPerf_{Double}}} = \frac{3}{\frac{1}{4,62*10^8} + \frac{1}{5,12*10^8} + \frac{1}{5,24*10^8}} = 4,98*10^8$$

При равновероятном использовании трёх типовых задач переход от *Debug* к *Release* увеличивает среднее быстродействие процессора примерно в 1.5 раза.

**Вывод:** ходе работы была разработана и реализована на языке C# benchmark-программа для оценки производительности процессора при выполнении типовой задачи, умножения квадратных матриц размера 1000x1000 с различными типами данных: *Int*, *Float* и *double*. Для генерации входных данных использовался детерминированный генератор случайных чисел. Измерение времени выполнялось с применением *Stopwatch*, результаты каждого запуска сохранялись в CSV-файл в формате. При расчёте среднего быстродействия процессора при равновероятном использовании трёх типовых задач (*Int*, *Float*, *Double*) по

формуле гармонического среднего были получены значения порядка  $\omega_{Debug} = 3,322 \cdot 10^8$  и  $\omega_{Release} = 4,98 \cdot 10^8$ . Таким образом, при равновероятном использовании задач переход от Debug к Release увеличивает среднее быстродействие процессора примерно в 1.5 раза.

Таблица 1 – Часть результата программы на C# с типом данных integer без оптимизации

PModel	Task	OpType	Opt	InsCount	Timer	Time	LNum	AvTime	AbsErr	RelErr	TaskPerf
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	6,181345	1	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	5,872984	2	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	6,348024	3	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	6,3801	4	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	6,124527	5	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	6,395593	6	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	6,149086	7	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	6,219542	8	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	5,671385	9	6,118061	0,074289	1,214255	3,27E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Debug	2E+09	Stopwatch	5,83802	10	6,118061	0,074289	1,214255	3,27E+08

Таблица 2 - Часть результата программы на C# с типом данных float без оптимизации

PModel	Task	OpType	Opt	InsCount	Timer	Time	LNum	AvTime	AbsErr	RelErr	TaskPerf
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	6,126497	1	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	5,864765	2	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	5,92622	3	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	5,914844	4	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	6,119102	5	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	6,054105	6	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	5,988394	7	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	6,148257	8	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	5,922365	9	6,001279	0,030806	0,513327	3,33E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Debug	2E+09	Stopwatch	5,948241	10	6,001279	0,030806	0,513327	3,33E+08

Таблица 3 – Часть результата программы на C# с типом данных double без оптимизации

PModel	Task	OpType	Opt	InsCount	Timer	Time	LNum	AvTime	AbsErr	RelErr	TaskPerf
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	5,783409	1	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	6,267597	2	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	5,956892	3	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	6,11108	4	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	6,018104	5	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	5,824202	6	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	5,773168	7	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	5,779535	8	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	5,955106	9	5,92806	0,04998	0,843107	3,37E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Debug	2E+09	Stopwatch	5,811506	10	5,92806	0,04998	0,843107	3,37E+08

Таблица 4 - Часть результата программы на C# с типом данных integer и ключом оптимизации Release

PModel	Task	OpType	Opt	InsCount	Timer	Time	LNum	AvTime	AbsErr	RelErr	TaskPerf
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,344818	1	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,372222	2	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,322663	3	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,332778	4	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,232916	5	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,194063	6	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,410934	7	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,367249	8	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,326698	9	4,327547	0,019887	0,45954	4,62E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Int32	Release	2E+09	Stopwatch	4,371133	10	4,327547	0,019887	0,45954	4,62E+08

Таблица 5 - Часть результата программы на C# с типом данных float и ключом оптимизации Release

PModel	Task	OpType	Opt	InsCount	Timer	Time	LNum	AvTime	AbsErr	RelErr	TaskPerf
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,932163	1	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,936906	2	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,920521	3	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,887612	4	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,89315	5	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,892859	6	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,880601	7	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,884267	8	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,894023	9	3,901358	0,00615	0,15765	5,12E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float32	Release	2E+09	Stopwatch	3,891476	10	3,901358	0,00615	0,15765	5,12E+08

Таблица 6 - Часть результата программы на C# с типом данных double и ключом оптимизации Release

PModel	Task	OpType	Opt	InsCount	Timer	Time	LNum	AvTime	AbsErr	RelErr	TaskPerf
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,823966	1	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,82101	2	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,785605	3	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,815044	4	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,822062	5	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,78097	6	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,862826	7	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,846361	8	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,826754	9	3,818459	0,007552	0,197775	5,24E+08
AMD64 Family 25 Model 80	MatMul_1000x1000	Float64	Release	2E+09	Stopwatch	3,799987	10	3,818459	0,007552	0,197775	5,24E+08

## Приложение А.

Листинг всей программы на C#

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;

class Program
{
    static void Main(string[] args)
    {
        // Количество запусков (испытаний) каждой задачи
        int runs = 10;
        // Размер квадратной матрицы N×N
        int n = 1000;

        if (args.Length >= 1 && int.TryParse(args[0], out var tmpRuns) && tmpRuns > 0)
            runs = tmpRuns;

        if (args.Length >= 2 && int.TryParse(args[1], out var tmpN) && tmpN > 0)
            n = tmpN;

        string cpuModel =
Environment.GetEnvironmentVariable("PROCESSOR_IDENTIFIER");

        string opt = "Debug";           // "Debug" или "Release"
string timerName = "Stopwatch";
string outputFile = "results.csv";

Console.WriteLine("Matrix benchmark");
Console.WriteLine($"CPU: {cpuModel}");
Console.WriteLine($"Runs: {runs}, matrix size: {n}x{n}");
Console.WriteLine($"Results -> {outputFile}");

using (var writer = new StreamWriter(outputFile, false))
{
    // Заголовок CSV

writer.WriteLine("PModel;Task;OpType;Opt;InsCount;Timer;Time;LNum;AvTime;AbsErr;RelErr;
TaskPerf");

    // Три бенчмарка: int, float, double
BenchmarkMatrixInt(cpuModel, opt, timerName, runs, n, writer);
BenchmarkMatrixFloat(cpuModel, opt, timerName, runs, n, writer);
BenchmarkMatrixDouble(cpuModel, opt, timerName, runs, n, writer);
}

// INT
static void BenchmarkMatrixInt(
    string cpuModel,
    string opt,
    string timerName,
    int runs,
    int n,
    StreamWriter writer)
{
    string taskName = $"MatMul_{n}x{n}";
```

```

        string opType = "Int32";

        // Оценка числа арифметических операций:
        double insCount = (double)n * n * (2 * n - 1);

        var rnd = new Random(135);
        int[,] A = GenerateMatrixInt(n, n, rnd);
        int[,] B = GenerateMatrixInt(n, n, rnd);
        int[,] C = new int[n, n];

        double[] times = new double[runs];
        var sw = new Stopwatch();

        for (int i = 0; i < runs; i++)
        {
            sw.Restart();
            MultiplyInt(A, B, C);
            sw.Stop();

            times[i] = sw.Elapsed.TotalSeconds;
        }

        CSV(
            cpuModel, taskName, opType, opt, timerName,
            insCount, times, runs, writer);
    }

    // FLOAT
    static void BenchmarkMatrixFloat(
        string cpuModel,
        string opt,
        string timerName,
        int runs,
        int n,
        StreamWriter writer)
    {
        string taskName = $"MatMul_{n}x{n}";
        string opType = "Float32";

        double insCount = (double)n * n * (2 * n - 1);

        var rnd = new Random(456);
        float[,] A = GenerateMatrixFloat(n, n, rnd);
        float[,] B = GenerateMatrixFloat(n, n, rnd);
        float[,] C = new float[n, n];

        double[] times = new double[runs];
        var sw = new Stopwatch();

        for (int i = 0; i < runs; i++)
        {
            sw.Restart();
            MultiplyFloat(A, B, C);
            sw.Stop();

            times[i] = sw.Elapsed.TotalSeconds;
        }

        CSV(
            cpuModel, taskName, opType, opt, timerName,
            insCount, times, runs, writer);
    }

    // DOUBLE

```

```

static void BenchmarkMatrixDouble(
    string cpuModel,
    string opt,
    string timerName,
    int runs,
    int n,
    StreamWriter writer)
{
    string taskName = $"MatMul_{n}x{n}";
    string opType = "Float64";

    double insCount = (double)n * n * (2 * n - 1);

    var rnd = new Random(789);
    double[,] A = GenerateMatrixDouble(n, n, rnd);
    double[,] B = GenerateMatrixDouble(n, n, rnd);
    double[,] C = new double[n, n];

    double[] times = new double[runs];
    var sw = new Stopwatch();

    for (int i = 0; i < runs; i++)
    {
        sw.Restart();
        MultiplyDouble(A, B, C);
        sw.Stop();

        times[i] = sw.Elapsed.TotalSeconds;
    }

    CSV(
        cpuModel, taskName, opType, opt, timerName,
        insCount, times, runs, writer);
}

// статистика
static void CSV(
    string cpuModel,
    string taskName,
    string opType,
    string opt,
    string timerName,
    double insCount,
    double[] times,
    int runs,
    StreamWriter writer)
{
    double mean = times.Average();

    // Дисперсия
    double variance = 0.0;
    for (int i = 0; i < runs; i++)
    {
        double diff = times[i] - mean;
        variance += diff * diff;
    }

    variance /= (runs);
    double stdDev = Math.Sqrt(variance);

    // Абсолютная погрешность оценки среднего
    double absError = stdDev / Math.Sqrt(runs);

    // Относительная погрешность в процентах
}

```

```

        double relError = absError / mean * 100;

        // Производительность
        double taskPerf = insCount / mean;

        for (int i = 0; i < runs; i++)
        {
            int launchNum = i + 1;
            double t = times[i];

            writer.WriteLine(
                string.Join(";", 
                    cpuModel,
                    taskName,
                    opType,
                    opt,
                    insCount,
                    timerName,
                    t,
                    launchNum,
                    mean,
                    absError,
                    relError,
                    taskPerf
                ));
        }
    }

    // Генерация матриц
    static int[,] GenerateMatrixInt(int rows, int cols, Random rnd)
    {
        int[,] m = new int[rows, cols];
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                m[i, j] = rnd.Next(-10, 11); // [-10; 10]
        return m;
    }

    static float[,] GenerateMatrixFloat(int rows, int cols, Random rnd)
    {
        float[,] m = new float[rows, cols];
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                m[i, j] = (float)(rnd.NextDouble() * 2.0 - 1.0); // [-1; 1]
        return m;
    }

    static double[,] GenerateMatrixDouble(int rows, int cols, Random rnd)
    {
        double[,] m = new double[rows, cols];
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                m[i, j] = rnd.NextDouble() * 2.0 - 1.0; // [-1; 1]
        return m;
    }

    // Умножение матриц
    static void MultiplyInt(int[,] A, int[,] B, int[,] C)
    {
        int n = A.GetLength(0);
        int kDim = A.GetLength(1);
        int m = B.GetLength(1);

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

```

```

    {
        for (int j = 0; j < m; j++)
        {
            int sum = 0;
            for (int k = 0; k < kDim; k++)
            {
                sum += A[i, k] * B[k, j];
            }
            C[i, j] = sum;
        }
    }

    static void MultiplyFloat(float[,] A, float[,] B, float[,] C)
{
    int n = A.GetLength(0);
    int kDim = A.GetLength(1);
    int m = B.GetLength(1);

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            float sum = 0.0f;
            for (int k = 0; k < kDim; k++)
            {
                sum += A[i, k] * B[k, j];
                // здесь одна операция умножения и одна сложения
            }
            C[i, j] = sum;
        }
    }
}

static void MultiplyDouble(double[,] A, double[,] B, double[,] C)
{
    int n = A.GetLength(0);
    int kDim = A.GetLength(1);
    int m = B.GetLength(1);

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            double sum = 0.0;
            for (int k = 0; k < kDim; k++)
            {
                sum += A[i, k] * B[k, j];
            }
            C[i, j] = sum;
        }
    }
}

```