

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школы электроники и компьютерных наук
Кафедра системного программирования

ОТЧЕТ
о лабораторной работе №3.1
по дисциплине «Технологии параллельного программирования»

Выполнил:
студент группы КЭ-220
_____/Голенищев А. Б.
_____ 2024 г.

Отчет принял:
_____/Жулев А. Э.
_____ 2024 г.

Челябинск
2024

Задание 6. Распараллеливание циклов в OpenMP: программа «Сумма чисел»

Приведен код программы, листинг

1.

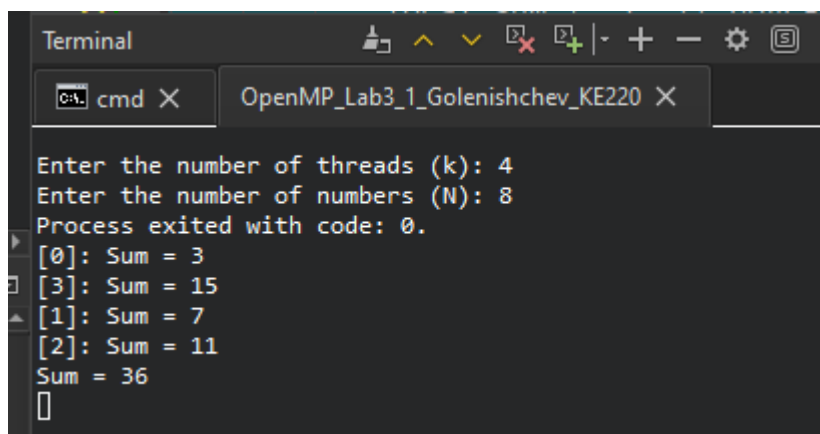
Продемонстрирована ее работа, рисунок 1.

```
#include <stdio>
#include <omp.h>
// Golenishchev Artem, KE-220 Task 6
int main() {
    int k, N;
    // Ввод: количество потоков (k) и верхняя граница суммы (N)
    printf("Enter the number of threads (k): ");
    scanf("%d", &k);
    printf("Enter the number of numbers (N): ");
    scanf("%d", &N);
    int total_sum = 0; // Общая сумма
    // Установка количества потоков
    omp_set_num_threads(k);
#pragma omp parallel
    {
        int thread_num = omp_get_thread_num(); // Номер текущего потока
        int local_sum = 0; // Локальная сумма для текущего потока
        // Параллельное распределение работы между потоками
#pragma omp for reduction(+ : total_sum)
        for (int i = 1; i <= N; ++i) {
            local_sum += i; // Подсчет локальной асуммы
            total_sum += i; // Добавление к общей сумме
        }

        // Вывод частичной суммы для каждого потока
#pragma omp critical
        {
            printf("[%d]: Sum = %d\n", thread_num, local_sum);
        }
    }

    // Вывод общей суммы
    printf("Sum = %d\n", total_sum);
    return 0;
}
```

Листинг 1. Код программы для подсчета суммы чисел



```
Terminal
C:\> cmd X
OpenMP_Lab3_1_Golenishchev_KE220 X

Enter the number of threads (k): 4
Enter the number of numbers (N): 8
Process exited with code: 0.
[0]: Sum = 3
[3]: Sum = 15
[1]: Sum = 7
[2]: Sum = 11
Sum = 36
█
```

Рисунок 1. Пример выполнения программы для подсчета суммы чисел

Программа вычисляет сумму чисел от 1 до NNN с использованием kkk потоков через OpenMP. Она распределяет итерации цикла между потоками с помощью директивы `#pragma omp for`, где каждый поток вычисляет свою частичную сумму. Эти частичные суммы автоматически суммируются в общую с использованием механизма `reduction(+ : total_sum)`, который обеспечивает корректное параллельное сложение. Каждый поток выводит свою частичную сумму, а затем один раз выводится итоговая сумма. Параллельное выполнение позволяет ускорить процесс при больших значениях NNN за счет разделения работы между потоками.

Задание 7. Распараллеливание циклов в OpenMP: параметр *schedule*

Представлен код программы для вывода номеров нитей в циклах, листнинг 2.

```
#include <stdio>
#include <string>
#include <omp.h>
// Golenishchev Artem, KE-220 Task 7
int main() {
    int k, N, chunk_start, chunk_end;
    char schedule_type[20];

    // Ввод параметров
    printf("Threads (k): ");
    scanf("%d", &k);
    printf("Numbers (N): ");
    scanf("%d", &N);
    printf("Schedule (static, dynamic, guided): ");
    scanf("%s", schedule_type);
    printf("Chunk size range (from-to): ");
    scanf("%d %d", &chunk_start, &chunk_end);

    omp_set_num_threads(k); // Установка потоков
    omp_sched_t schedule = (strcmp(schedule_type, "static") == 0) ? omp_sched_static :
                           (strcmp(schedule_type, "dynamic") == 0) ? omp_sched_dynamic :
                           omp_sched_guided;

    for (int chunk = chunk_start; chunk <= chunk_end; ++chunk) {
        omp_set_schedule(schedule, chunk); // Установка расписания
        printf("\nSchedule(%s, %d):\n", schedule_type, chunk);

#pragma omp parallel
        {
#pragma omp for schedule(runtime)
            for (int i = 1; i <= N; i++)
                printf("[Thread %d]: Iteration %d\n", omp_get_thread_num(), i);
        }

        return 0;
    }
}
```

Листнинг 2. Код программы для заполнения таблицы распределения итераций цикла

Запустим программу. Рассмотрим результат её выполнения для $k = 4$, $N = 10$ и различных параметров *schedule*, рисунки 2-4. Результаты работы программы представлены в таблице 1.

```
Terminal
OpenMP_Lab3_1_Golenishchev_KE220 X

Threads (k): 4
Numbers (N): 10
Schedule (static, dynamic, guided): static
Chunk size range (from-to): 0 2

Schedule(static, 0):
[Thread 0]: Iteration 1
[Thread 3]: Iteration 9
[Thread 3]: Iteration 10
[Thread 2]: Iteration 7
[Thread 2]: Iteration 8
[Thread 1]: Iteration 4
[Thread 1]: Iteration 5
[Thread 1]: Iteration 6
[Thread 0]: Iteration 2
[Thread 0]: Iteration 3

Schedule(static, 1):
[Thread 3]: Iteration 4
[Thread 3]: Iteration 8
[Thread 2]: Iteration 3
[Thread 2]: Iteration 7
[Thread 0]: Iteration 1
[Thread 0]: Iteration 5
[Thread 0]: Iteration 9
[Thread 1]: Iteration 2
[Thread 1]: Iteration 6
[Thread 1]: Iteration 10

Schedule(static, 2):
[Thread 1]: Iteration 3
[Thread 1]: Iteration 4
[Thread 2]: Iteration 5
[Thread 2]: Iteration 6
[Thread 3]: Iteration 7
[Thread 3]: Iteratio
Process exited with code: 0.n 8
[Thread 0]: Iteration 1
[Thread 0]: Iteration 2
[Thread 0]: Iteration 9
[Thread 0]: Iteration 10
```

Рисунок 2. Результат выполнения программы для schedule static

```
Terminal
OpenMP_Lab3_1_Golenishchev_KE220 X

Threads (k): 4
Numbers (N): 10
Schedule (static, dynamic, guided): dynamic
Chunk size range (from-to): 0 2

Schedule(dynamic, 0):
[Thread 0]: Iteration 1
[Thread 0]: Iteration 5
[Thread 0]: Iteration 6
[Thread 0]: Iteration 7
[Thread 0]: Iteration 8
[Thread 0]: Iteration 9
[Thread 0]: Iteration 10
[Thread 1]: Iteration 2
[Thread 2]: Iteration 3
[Thread 3]: Iteration 4

Schedule(dynamic, 1):
[Thread 3]: Iteration 1
[Thread 3]: Iteration 5
[Thread 3]: Iteration 6
[Thread 3]: Iteration 7
[Thread 3]: Iteration 8
[Thread 3]: Iteration 9
[Thread 3]: Iteration
Process exited with code: 0. 10
[Thread 0]: Iteration 4
[Thread 1]: Iteration 2
[Thread 2]: Iteration 3
Schedule(dynamic, 2):
[Thread 2]: Iteration 1
[Thread 2]: Iteration 2
[Thread 2]: Iteration 9
[Thread 2]: Iteration 10
[Thread 3]: Iteration 3
[Thread 3]: Iteration 4
[Thread 0]: Iteration 7
[Thread 0]: Iteration 8
[Thread 1]: Iteration 5
[Thread 1]: Iteration 6
```

Рисунок 3. Результат выполнения программы для schedule dynamic

```
Terminal
OpenMP_Lab3_1_Golenishchev_KE220 X

Threads (k): 4
Numbers (N): 10
Schedule (static, dynamic, guided): guided
Chunk size range (from-to): 0 2

Schedule(guided, 0):
[Thread 0]: Iteration 1
[Thread 0]: Iteration 2
[Thread 0]: Iteration 3
[Thread 0]: Iteration 9
[Thread 0]: Iteration 10
[Thread 3]: Iteration 6
[Thread 3]: Iteration 7
[Thread 1]: Iteration 4
[Thread 1]: Iteration 5
[Thread 2]: Iteration 8

Schedule(guided, 1):
Process exited with code: 0.
[Thread 2]: Iteration 1
[Thread 2]: Iteration 2
[Thread 2]: Iteration 3
[Thread 2]: Iteration 9
[Thread 2]: Iteration 10
[Thread 0]: Iteration 8
[Thread 3]: Iteration 6
[Thread 3]: Iteration 7
[Thread 1]: Iteration 4
[Thread 1]: Iteration 5
Schedule(guided, 2):
[Thread 1]: Iteration 1
[Thread 1]: Iteration 2
[Thread 1]: Iteration 3
[Thread 1]: Iteration 10
[Thread 2]: Iteration 4
[Thread 2]: Iteration 5
[Thread 0]: Iteration 8
[Thread 0]: Iteration 9
[Thread 3]: Iteration 6
[Thread 3]: Iteration 7
```

Рисунок 4. Результат выполнения программы для schedule guided

Таблица 1 – Распределение итераций цикла по нитям от параметра schedule

Номер итерации	Значение параметра schedule						
	static	static, 1	static, 2	dynamic	dynamic, 2	guided	guided, 2
1	0	0	0	0	2	0	1
2	0	1	0	1	2	0	1
3	0	2	1	2	3	0	1
4	1	3	1	3	3	1	2
5	1	0	2	0	1	1	2
6	1	1	2	0	1	3	3
7	2	2	3	0	0	3	3
8	2	3	3	0	0	2	0
9	3	0	0	0	2	0	0
10	3	1	0	0	2	0	1

Таким образом, из таблицы видно, что static использует фиксированное распределение (0 – равномерное, 1 – по одной итерации между потоками, 2 – блоками по 2), dynamic распределяет итерации в порядке освобождения потоков (0 – начиная с первой итерации последовательно, 2 – блоками по 2), guided распределяет по размеру блоков от крупных до chunk (0 – потоки 0 и 1 нагружаются сперва потоки 0 и 1, 2 – указание минимального размера блока 2). Для равномерных задач лучше подходит static, для неравномерных dynamic или guided.

Задание 8. Распараллеливание циклов в OpenMP: программа «Матрица»

```
#include <stdio>
#include <omp.h>
// Golenishchev Artem, KE-220 Task 8
int main() {
    int n;
    // Ввод размера матриц
    printf("Enter the size of the matrices (n x n): ");
    scanf("%d", &n);
    if (n < 1 || n > 10) {
        printf("Invalid size. n must be between 1 and 10.\n");
        return 1;
    }
    double A[10][10], B[10][10], C[10][10] = {{0}};
    // Ввод элементов матрицы A
    printf("Enter the elements of matrix A:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%lf", &A[i][j]);
    // Ввод элементов матрицы B
    printf("Enter the elements of matrix B:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%lf", &B[i][j]);
    // Параллельное вычисление произведения матриц
    #pragma omp parallel for collapse(2)
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    // Вывод результата
    printf("Resultant matrix C:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%.2lf ", C[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Листнинг 3. Код программы для умножения квадратных матриц

Представлен код программы, листинг 3. Найдем пример расчета умножения матриц традиционным способом, рисунок 5. Сравним с результатами работы программы, рисунок 6.

Пример 1. Найти матрицу C равную произведению матриц $A = \begin{pmatrix} 4 & 2 \\ 9 & 0 \end{pmatrix}$ и $B = \begin{pmatrix} 3 & 1 \\ -3 & 4 \end{pmatrix}$

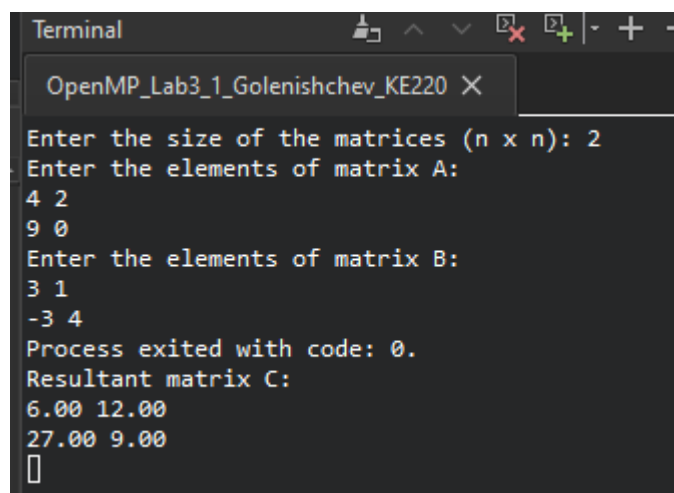
Решение:

$$C = A \cdot B = \begin{pmatrix} 4 & 2 \\ 9 & 0 \end{pmatrix} \cdot \begin{pmatrix} 3 & 1 \\ -3 & 4 \end{pmatrix} = \begin{pmatrix} 6 & 12 \\ 27 & 9 \end{pmatrix}$$

Элементы матрицы C вычисляются следующим образом:

$$c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21} = 4 \cdot 3 + 2 \cdot (-3) = 12 - 6 = 6$$
$$c_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22} = 4 \cdot 1 + 2 \cdot 4 = 4 + 8 = 12$$
$$c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21} = 9 \cdot 3 + 0 \cdot (-3) = 27 + 0 = 27$$
$$c_{22} = a_{21} \cdot b_{12} + a_{22} \cdot b_{22} = 9 \cdot 1 + 0 \cdot 4 = 9 + 0 = 9$$

Рисунок 5. Пример умножения матрицы на бумаге



```
Terminal
OpenMP_Lab3_1_Golenishchev_KE220 X
Enter the size of the matrices (n x n): 2
Enter the elements of matrix A:
4 2
9 0
Enter the elements of matrix B:
3 1
-3 4
Process exited with code: 0.
Resultant matrix C:
6.00 12.00
27.00 9.00
█
```

Рисунок 6. Результат выполнения программы в Qt Creator

Код реализует параллельное вычисление произведения двух квадратных матриц A и B размера $n \times n$, результат сохраняется в матрицу C . Пользователь вводит размер матриц (n), а затем значения элементов матриц A и B . Основной алгоритм для вычисления произведения матриц представлен в виде тройного цикла:

1. Внешний цикл проходит по строкам матрицы A.
2. Средний цикл проходит по столбцам матрицы B.
3. Внутренний цикл вычисляет сумму произведений элементов текущей строки из AAA и текущего столбца из BBB, формируя один элемент результата в CCC.

Внутренние два цикла, отвечающие за перебор строк и столбцов, распараллелены с помощью OpenMP, что позволяет одновременно обрабатывать разные элементы матрицы CCC. Это достигается с помощью директивы `#pragma omp parallel for collapse(2)`.

Ключевое слово `collapse` указывает, сколько циклов следует объединить (развернуть) в одну параллельную итерацию. В данном случае `collapse(2)` объединяет два вложенных цикла (`for (int i = 0; i < n; i++)` и `for (int j = 0; j < n; j++)`) в одну плоскую итерационную область, которая затем распараллеливается. Это особенно полезно, когда количество итераций во внутреннем цикле невелико, так как позволяет лучше распределить нагрузку между потоками.

Ответы на вопросы к лабораторной работе:

1. Почему в OpenMP реализован только цикл `for`? Любой ли корректный последовательный цикл `for` можно распараллелить?

Его итерации можно удобно разбивать на параллельные блоки (известно количество итераций) в тех случаях, когда результат следующей итерации не зависит от результата предыдущей.

2. На каком этапе выполняется распределение итераций по нитям, опишите алгоритм этого распределения.

Распределение итераций по нитям выполняется на этапе выполнения программы (runtime). Конкретный способ распределения задается параметром `schedule`, либо через программный код, либо через переменную окружения `OMP_SCHEDULE`. Например, команда `setenv OMP_SCHEDULE "dynamic,3"` устанавливает динамическое распределение с размером блока 3. Это позволяет гибко изменять стратегию распределения итераций без изменения исходного кода программы и повторной компиляции, что особенно полезно для тестирования и оптимизации работы программы под различные условия.

3. Опишите назначение параметра `schedule` и смысл его двух параметров.

При разном объеме вычислений в разных итерациях цикла желательно иметь возможность управлять распределением итераций цикла между потоками - в OpenMP это обеспечивается при помощи параметра `schedule` директивы `for`. Параметры - тип распределения итераций, минимальный размер блока.

static - статический способ распределения итераций до начала выполнения цикла. Если поле `chunk` не указано, то итерации делятся поровну между потоками. При заданном значении `chunk` итерации цикла делятся на блоки размера `chunk` и эти блоки распределяются между потоками до начала выполнения цикла.

dynamic - динамический способ распределения итераций. До начала выполнения цикла потокам выделяются блоки итераций размера `chunk` (если поле `chunk` не указано, то полагается значение `chunk=1`). Дальнейшее выделение итераций

(также блоками размера `chunk`) осуществляется в момент завершения потоками своих ранее назначенных итераций.

guided - управляемый способ распределения итераций. Данный способ близок к предшествующему варианту, отличие состоит только в том, что начальный размер блоков итераций определяется в соответствии с некоторым параметром среды реализации OpenMP, а затем уменьшается экспоненциально (следующее значение `chunk` есть некоторая доля предшествующего значения) при каждом новом выделении блока итераций. При этом получаемый размер блока итераций не должен быть меньше значения `chunk` (если поле `chunk` не указано, то полагается значение `chunk=1`).

4. Для чего введены динамические распределения итераций?

Динамические распределения (`dynamic`, `guided`) используются для балансировки нагрузки при нерегулярных вычислениях. Если некоторые итерации требуют больше времени, динамическое распределение позволяет равномерно загружать потоки, передавая им новые блоки по мере завершения работы.

5. В каких случаях и почему распределение типа `guided` будет работать эффективнее, чем `dynamic`, и наоборот?

guided эффективнее, если итерации имеют существенно разную сложность. Крупные начальные блоки минимизируют накладные расходы на распределение, а уменьшение размера блоков к концу помогает оптимально загрузить потоки.

dynamic предпочтительнее, если итерации примерно одинаковы по времени, но их порядок выполнения непредсказуем. Здесь равномерный размер блоков упрощает управление.

Выводы:

Изучили распараллеливание вычислений на примере умножения квадратных матриц с использованием OpenMP. Мы научились задавать параллельные области и эффективно управлять переменными: общими (shared) для всех потоков и частными (private) для каждого потока. С помощью директивы `#pragma omp parallel for` с модификатором `collapse` мы оптимизировали распределение нагрузки между потоками, объединив вложенные циклы в один. Это позволило значительно ускорить выполнение тройного вложенного цикла, отвечающего за вычисление произведения матриц. Работа продемонстрировала основы распараллеливания, особенности управления переменными и преимущества использования OpenMP в задачах линейной алгебры.