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.

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

Приведен код программы, листнинг 1.   
Продемонстрирована ее работа, рисунок 1.

#include <cstdio>

#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 суммы

}

// Вывод частичной суммы для каждого потока

printf("[%d]: Sum = %d\n", thread\_num, local\_sum);

}

// Вывод общей суммы

printf("Sum = %d\n", total\_sum);

return 0;

}

Листнинг 1. Код программы для подсчета суммы чисел

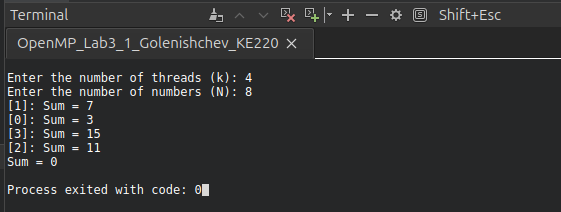


Рисунок 1. Пример выполнения программы для подсчета суммы чисел

Программа предназначена для вычисления суммы чисел от 1 до N с использованием многопоточности. Пользователь вводит количество потоков k и верхнюю границу суммы N. Работа распределяется между потоками с использованием OpenMP, где каждому потоку назначается диапазон чисел. Локальные суммы подсчитываются в каждом потоке и автоматически складываются в общую сумму\_total\_sum благодаря механизму reduction. Частичные суммы для каждого потока и итоговая сумма выводятся на экран.

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

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

Листнинг 2. Код программы для заполнения таблицы распределения   
итераций цикла

#include <cstdio>

#include <cstring>

#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

for (int chunk = chunk\_start; chunk <= chunk\_end; ++chunk) {

// Установка расписания

if (strcmp(schedule\_type, "static") == 0) {

printf("\nSchedule(static, %d):\n", chunk);

} else if (strcmp(schedule\_type, "dynamic") == 0) {

printf("\nSchedule(dynamic, %d):\n", chunk);

} else if (strcmp(schedule\_type, "guided") == 0) {

printf("\nSchedule(guided, %d):\n", chunk);

} else {

printf("Invalid schedule type.\n");

return 1; }

#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;

}

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.

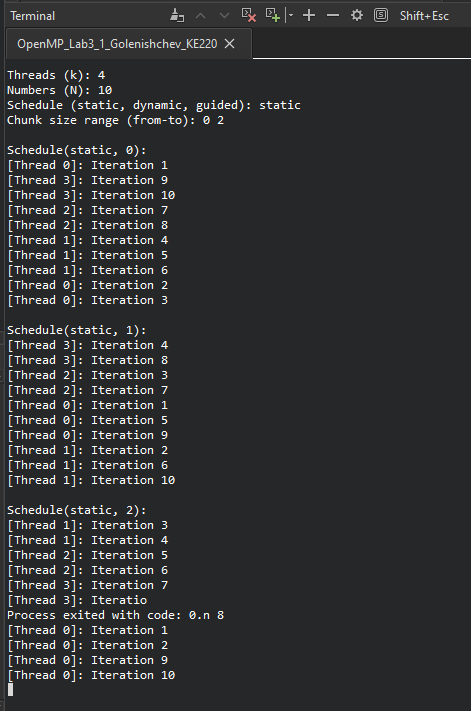


Рисунок 2. Результат выполнения программы для schedule static

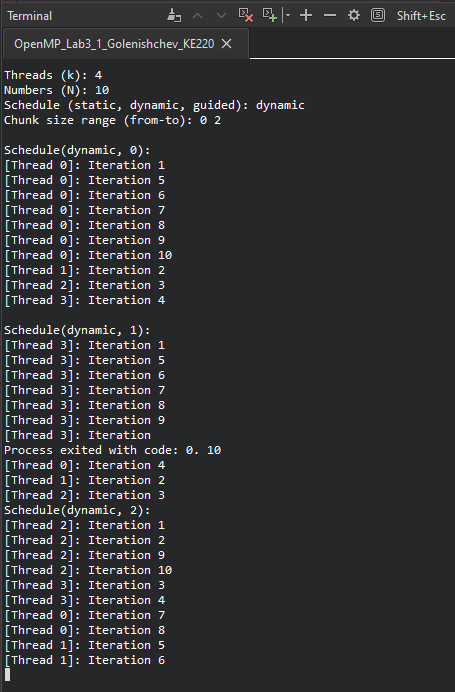


Рисунок 3. Результат выполнения программы для schedule dynamic

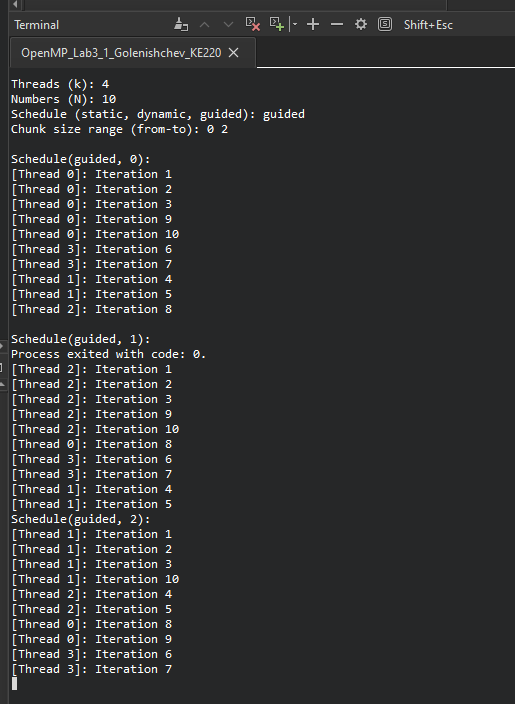


Рисунок 4. Результат выполнения программы для schedule guided

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

Таблица 1 – Распределение итераций цикла по нитям от параметра schedule

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 <cstdio>

#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, листнинг 3. Найдем пример расчета умножения матриц традиционным способом, рисунок 5. Сравнив с результатами работы программы, рисунок 6.

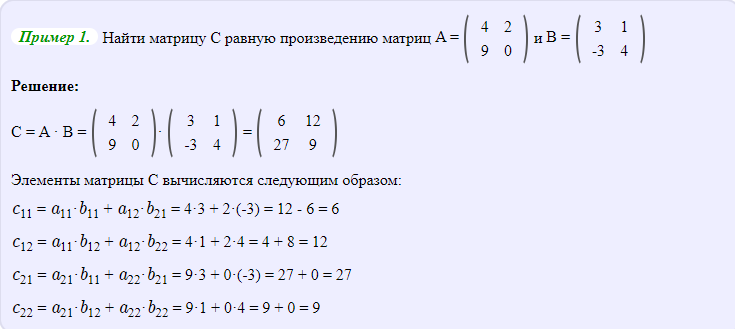


Рисунок 5. Пример умножения матрицы на бумаге

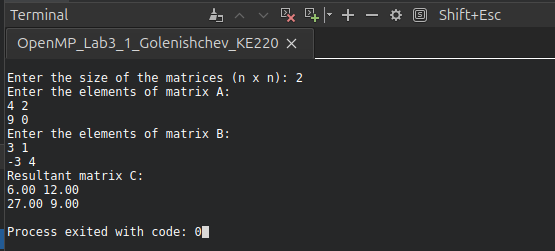


Рисунок 6. Результат выполнения программы в Qt Creator

Код реализует параллельное вычисление произведения двух квадратных матриц A и B размера n\*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:

Внешний цикл проходит по строкам матрицы A.

Средний цикл проходит по столбцам матрицы B.

Внутренний цикл вычисляет сумму произведений элементов текущей строки из A и текущего столбца из BB, формируя один элемент результата в C.

Внутренние два цикла, отвечающие за перебор строк и столбцов, распараллелены с помощью OpenMP, что позволяет одновременно обрабатывать разные элементы матрицы C. 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.

1. 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.

1. 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).

1. 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.

1. 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.