Филиал федерального государственного бюджетного образовательного учреждения высшего образования «Национальный исследовательский университет «МЭИ» в г. Смоленске

Кафедра вычислительной техники

Направление: 09.04.01. «Информатика и вычислительная техника»

Профиль: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

Практическая работа №2
«Отладка простых программ на вычислительном кластере с использованием OpenMP»

по курсу:

«Вычислительные системы»

Студент: Старостенков А.А.

Группа: ВМ-22(маг)

Вариант: 19

Преподаватель: Федулов А.С.

Смоленск, 2023

Задание

1. Отладить, скомпилировать и запустить на ГВК простые тестовые примеры на языке Си с использованием технологии OpenMP. В качестве базовых примеров использовать примеры из лекции № 3 по технологии OpenMP по курсу «Вычислительные системы». При выводе результатов для каждой программы предусмотреть вывод своей фамилии и инициалов (в латинице).

1.1. Определить версию OpenMP (пример 1).

1.2. Организовать задержку выполнения программы на заданный интервал времени и измерить этот интервал с помощью функций OpenMP (пример 2). Интервал задержки в секундах выбрать равным номеру по журналу. Отлаженную программу запустить несколько раз (2-3). Убедиться в том, что результат отличается от запуска к запуску. Объяснить причину этого.

1.3. Определить максимально возможное число нитей (потоков), заданное по умолчанию (пример 3). Изменить это число с помощью команды **export OMP_NUM_THREADS=n.** В качестве числа потоков n в параллельной области использовать значение «номер по журналу»+4. Вновь запустить пример 3. Оценить результаты.

1.4. В примере 4 проверить определение числа нитей в параллельной области тремя способами: с помощью переменной окружения (присвоить значение «номер по журналу»+2), с помощью функции OpenMP (присвоить значение «номер по журналу»+3), с помощью опции директивы pragma omp parallel (присвоить значение «номер по журналу»+4). Оценить результаты выполнения примера 4.

1.5. Пример 5 выполнить с числом потоков в параллельной области, равным номеру по журналу+8. Число потоков в параллельной области задать любым способом из рассмотренных в примере 4.

1.6. Пример 6 выполнить с числом потоков в параллельной области, равным номеру по журналу+9.

1.7. Программу из примера 7 запустить несколько раз. Убедиться в наличии гонок по данным.

1.8. Примеры 9 выполнить с числом потоков в параллельной области, равным номеру по журналу+7. Выполнить программу с опцией reduction и без нее. Объяснить полученные результаты.

1.9. Пример 9.1 выполнить без модификаций.

1.10. Пример 10 выполнить без модификаций. Определить число итераций цикла, реализованных каждой нитью.

Ход работы

1.1. Определить версию OpenMP (пример 1).

Код:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

int main(int argc, char *argv[])
{
    printf("Starostenkov A.A. VM-22 (mag.)\n");

    printf("Example 1.\n");
#ifdef _OPENMP
    printf("OpenMP Version = %d\n", _OPENMP);
#else
    printf("Sequential Version");
#endif

    return 0;
}
```

```
[starostenkov_aa@mng1 2]$ ./1
Starostenkov A.A. VM-22 (mag.)
Example 1.
OpenMP Version = 201511
[starostenkov_aa@mng1 2]$
```

Рисунок 1 – Запрос версии

Успешно, на сервере установлен OpenMP

1.2. Организовать задержку выполнения программы на заданный интервал времени и измерить этот интервал с помощью функций OpenMP (пример 2). Интервал задержки в секундах выбрать равным номеру по журналу. Отлаженную программу запустить несколько раз (2-3). Убедиться в том, что результат отличается от запуска к запуску. Объяснить причину этого.

Код:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

int main(int argc, char *argv[]){
    printf("\n\nExample 2.\n\n");

    double start_time, end_time, tick;
    start_time = omp_get_wtime(); // Начальная отсечка времени
    sleep(19);//Задержка на 19 секунд
    // sleep(1);                  // Задержка на 1 секунду
    end_time = omp_get_wtime(); // Конечная отсечка времени
    tick = omp_get_wtick();     // Точность таймера
    printf("Time range %e\n", end_time - start_time);
    printf("Accuracy of timer %e\n", tick);

    return 0;
}
```

```
[starostenkov_aa@mng1 2]$ ./2


 Example 2.

 Time range 1.900010e+01
 Accuracy of timer 1.000000e-09
[starostenkov_aa@mng1 2]$
```

Рисунок 2 – Задержки

Вывод: измеренный временной интервал слегка отличается от запуска к запуску. Это может происходить по целому ряду причин:

1. Linux является многопользовательской системой с вытесняющей многозадачностью – часть процессорного времени уходит на другие задачи.
2. Различное время доступа к памяти, в зависимости от загрузки шин и состояния кэшей процессора
3. Состояние конвейера процессора – время выполнения одних и тех же машинных команд может быть различным.

1.3. Определить максимально возможное число нитей (потоков), заданное по умолчанию (пример 3). Изменить это число с помощью команды export OMP_NUM_THREADS=n. В качестве числа потоков n в параллельной области использовать значение «номер по журналу»+4. Вновь запустить пример 3. Оценить результаты.

Код:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

int main(int argc, char *argv[])
{
    printf("\n\nExample 3.\n\n");

    omp_set_num_threads(23);

    printf("Sequential region 1\n");
#pragma omp parallel
    { // Начало пар
        int thread_num = omp_get_thread_num();
        printf("Parallel region: Thread %d\n", thread_num);
    } // Конец пар

    printf("Sequential region 2\n");

    return 0;
}
```

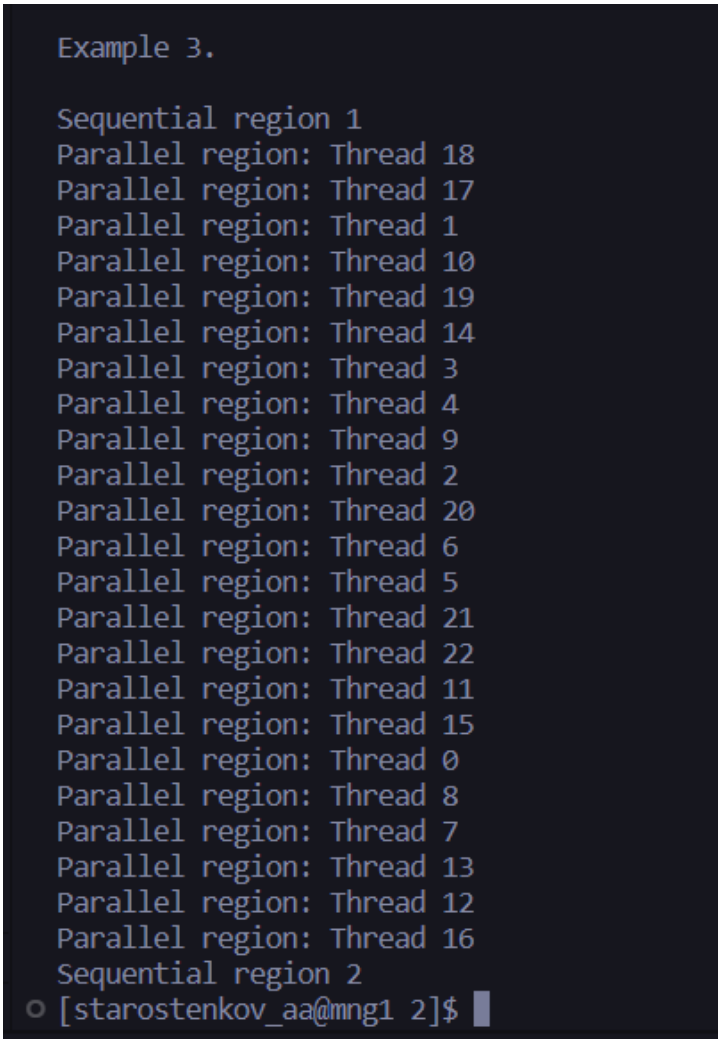

Рисунок 3 – Измененная программа

Ограничили количество потоков 23, что и видно на рисунке, вывелось 23 строки.

1.4. В примере 4 проверить определение числа нитей в параллельной области тремя способами: с помощью переменной окружения (присвоить значение «номер по журналу»+2), с помощью функции OpenMP (присвоить значение «номер по журналу»+3), с помощью опции директивы pragma omp parallel (присвоить значение «номер по журналу»+4). Оценить результаты выполнения примера 4.

Код

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

void printThreadsProc()
{
    int num_threads, num_proc;

    num_threads = omp_get_max_threads(); // максимально допустимое число нитей
    num_proc = omp_get_num_procs();      // число доступных ядер с учетом гипертрейдинга

    printf("\n----------------\n");
    printf("number of treads %d\n", num_threads);  // равно значению OMP_NUM_THREADS
    printf("number of proceccors %d\n", num_proc); // равно значению OMP_NUM_THREADS
    printf("---------------\n");
}

int main(int argc, char *argv[])
{
    printf("\n\nExample 4.\n\n");

    // Переменная окружения export OMP_NUM_THREADS=21 (19 + 2)
    // из файла .bashrc
    printf("Переменная окружения export OMP_NUM_THREADS=21 (19 + 2) из файла .bashrc\n");
    // Получение значения переменной окружения OMP_NUM_THREADS
    char *env_num_threads = getenv("OMP_NUM_THREADS");
    if (env_num_threads != NULL)
    {
        int num_threads_env = atoi(env_num_threads);
        omp_set_num_threads(num_threads_env);
        printf("\n");
        printThreadsProc();
        printf("\n");
    }
    else
    {
        // Обработка случая, когда переменная окружения не была задана или не существует
        // По умолчанию устанавливаем желаемое количество нитей
        printf("\n\n!!!!!!!!!!!!!\nError! Can't get OMP_NUM_THREADS!\n\n");
        omp_set_num_threads(21);
    }

#pragma omp parallel
    {
        int thread_num = omp_get_thread_num();
        printf("Parallel region 1: Thread #%d\n", thread_num);
    }

    printf("\n$$$$$$$$$$$$$$$$$$$$$$\n");

    // задаем 22 потока в параллельной области с помощью ф-ии
    printf("\n\nЗадаем 22 потока в параллельной области с помощью ф-ии omp_set_num_threads\n\n");
    omp_set_num_threads(22);
    printThreadsProc();
#pragma omp parallel
    {
        int thread_num = omp_get_thread_num();
        printf("Parallel region 2: Thread #%d\n", thread_num);
    }

    printf("\n$$$$$$$$$$$$$$$$$$$$$$\n");

    // задаем 23 потока с помощью опции директивы

    printf("\n\nЗадаем 23 потока с помощью опции директивы num_threads(23)\n\n");

#pragma omp parallel num_threads(23)
    {
        int thread_num = omp_get_thread_num();
        printf("Parallel region 3: Thread #%d\n", thread_num);
    }

    return 0;
}
```

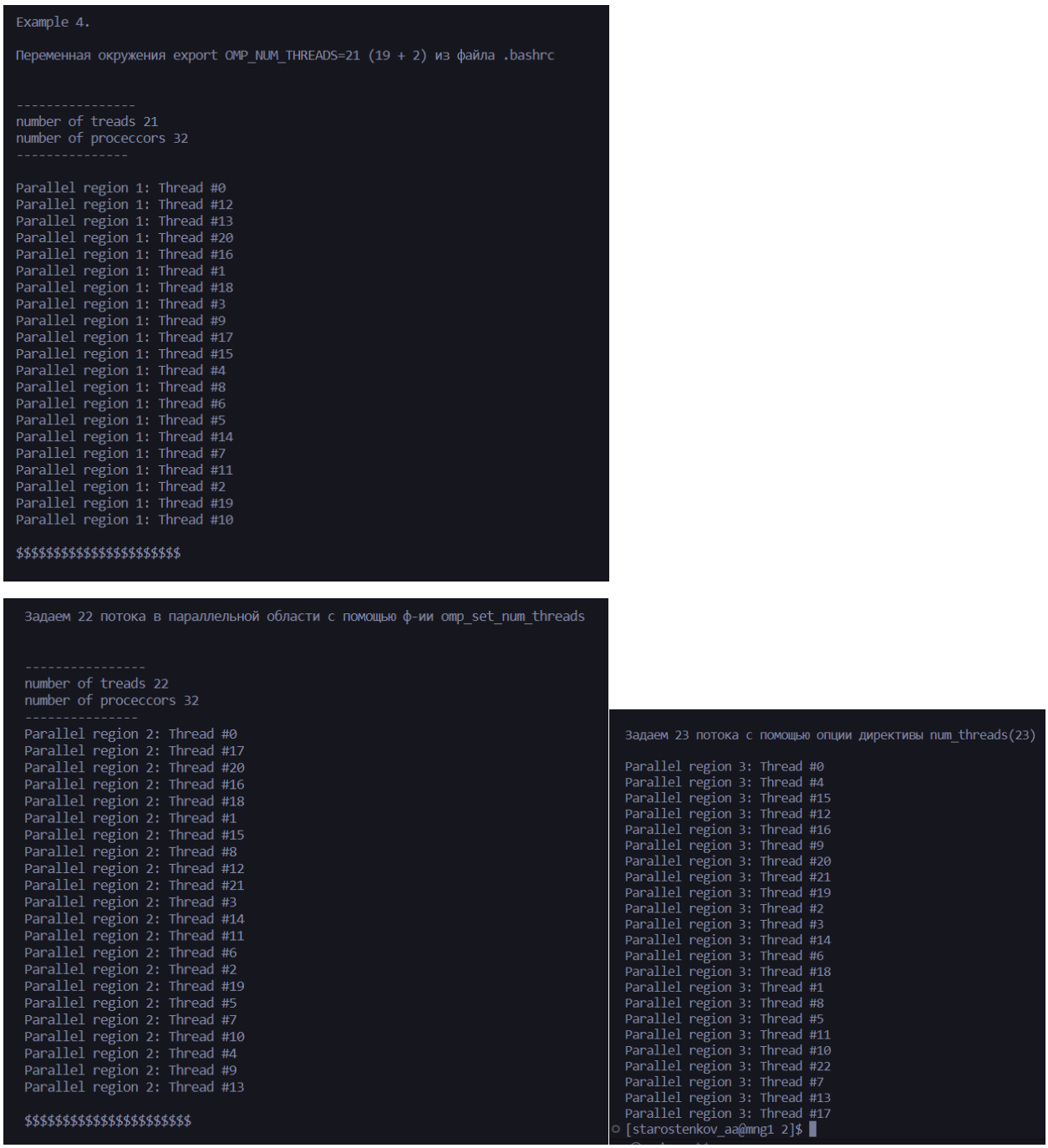

Рисунок 5 - Изменение потоков

Программа работает, число потоков изменяется тремя разными способами. Видно, что Опция num_threads() имеет приоритет над функцией omp_set_num_threads(), которая, в свою очередь, приоритетна над переменной окружения OMP_NUM_THREADS.

1.5. Пример 5 выполнить с числом потоков в параллельной области, равным номеру по журналу+8. Число потоков в параллельной области задать любым способом из рассмотренных в примере 4.

Код

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>


int main(int argc, char *argv[])
{
    printf("\n\nExample 5.\n\n");
    int count;
    omp_set_num_threads(27);
#pragma omp parallel
    {
        int num;
        count = omp_get_num_threads(); // число нитей всего
        num = omp_get_thread_num();    // номер нити, у каждой свой
        if (num == 0)
        {
            printf("Quantity of treads: %d\n", count);
        }
        else
        {
            printf("Number of tread %d\n", num);
        }
    }

    return 0;
}
```

```
Example 5.

Number of tread 17
Number of tread 2
Number of tread 23
Number of tread 8
Number of tread 20
Number of tread 9
Number of tread 24
Quantity of treads: 27
Number of tread 5
Number of tread 7
Number of tread 19
Number of tread 13
Number of tread 4
Number of tread 22
Number of tread 25
Number of tread 16
Number of tread 10
Number of tread 1
Number of tread 3
Number of tread 15
Number of tread 21
Number of tread 26
Number of tread 6
Number of tread 11
Number of tread 12
Number of tread 18
Number of tread 14
[starostenkov_aa@mng1 2]$
```

Рисунок 6 – Изменение потоков

Результаты различаются от запуска к запуску. Это происходит от того, что потоки не синхронизированы между собой и в качестве текущего берётся первый попавшийся свободный поток.

1.6. Пример 6 выполнить с числом потоков в параллельной области, равным номеру по журналу+9.

Код:

```c
int main(int argc, char *argv[]){
    printf("\n\nExample 6.\n");
    omp_set_num_threads(28); // 10 потоков
    printThreadsProc();
    int n = 1;
    printf("n in sequential region (begin): %d\n", n);
#pragma omp parallel private(n)
    {
        printf("Value of n in tread (input): %d\n", n);
        /* Value of n equal number of particular tread */
        n = omp_get_thread_num();
        printf("Value of n in tread (output): %d\n", n);
    }
    printf("n in sequential region (end): %d\n", n);

    printf("\n\n---------------\nEND.\n---------------\n");
    return 0;
}
```

```
Example 6.

----------------
number of treads 28
number of proceccors 32
---------------
n in sequential region (begin): 1
Value of n in tread (input): 0
Value of n in tread (output): 15
Value of n in tread (input): 0
Value of n in tread (output): 0
Value of n in tread (input): 0
Value of n in tread (output): 1
Value of n in tread (input): 0
Value of n in tread (output): 26
Value of n in tread (input): 0
Value of n in tread (output): 5
Value of n in tread (input): 0
Value of n in tread (output): 2
Value of n in tread (input): 0
Value of n in tread (output): 27
Value of n in tread (input): 0
Value of n in tread (output): 20
Value of n in tread (input): 0
Value of n in tread (output): 7
Value of n in tread (input): 0
Value of n in tread (output): 22
Value of n in tread (input): 0
Value of n in tread (output): 23
Value of n in tread (input): 0
Value of n in tread (output): 11
Value of n in tread (input): 0
Value of n in tread (output): 14
Value of n in tread (input): 0
Value of n in tread (output): 19
Value of n in tread (input): 0
Value of n in tread (output): 25
Value of n in tread (input): 0
Value of n in tread (output): 13
Value of n in tread (input): 0
Value of n in tread (output): 9
Value of n in tread (input): 0
Value of n in tread (output): 18
Value of n in tread (input): 0
Value of n in tread (output): 24
Value of n in tread (input): 0
```

Рисунок 7 - Потоки

1.7. Программу из примера 7 запустить несколько раз. Убедиться в наличии гонок по данным.

Код:

```c
int main(int argc, char *argv[])
{
    printf("\n\nExample 7.\n");
    int count = 0;
#pragma omp parallel shared(count)
    {
        count = count + 1;
    }
    printf("Number of treads: %d\n", count);

    printf("\n\n---------------\nEND.\n---------------\n");
    return 0;
}
```

```
[starostenkov_aa@mng1 2]$ ./7


Example 7.
Number of treads: 4


---------------
END.
---------------
[starostenkov_aa@mng1 2]$ ./7


Example 7.
Number of treads: 3


---------------
END.
---------------
[starostenkov_aa@mng1 2]$ ./7


Example 7.
Number of treads: 6


---------------
END.
---------------
[starostenkov_aa@mng1 2]$ ./7


Example 7.
Number of treads: 4


---------------
END.
---------------
[starostenkov_aa@mng1 2]$
```

Рисунок 8 – Гонки данных

Наблюдаются гонки данных, выводится в программе то 4, то 5 поток.

1.8. Примеры 9 выполнить с числом потоков в параллельной области, равным номеру по журналу+7. Выполнить программу с опцией reduction и без нее. Объяснить полученные результаты.

Код

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void printThreadsProc()
{
    int num_threads, num_proc;

    num_threads = omp_get_max_threads(); // максимально допустимое число нитей
    num_proc = omp_get_num_procs();      // число доступных ядер с учетом гипертрейдинга

    printf("\n---------------\n");
    printf("number of threads %d\n", num_threads); // равно значению OMP_NUM_THREADS
    printf("number of processors %d\n", num_proc); // равно значению OMP_NUM_THREADS
    printf("---------------\n");
}

int main(int argc, char *argv[])
{
    printf("\n\nExample 9.\n");

    omp_set_num_threads(26);
    printThreadsProc();

    int count = 0;
#pragma omp parallel reduction(+ : count)
    {
        count++;
        printf("Value of count: %d\n", count);
    }
    printf("Number of treads: %d\n", count);

    printf("\n\n---------------\nEND.\n---------------\n");
    return 0;
}
```

```
Example 9.

---------------
number of threads 26
number of processors 32
---------------
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Value of count: 1
Number of treads: 26


---------------
END.
---------------
[starostenkov_aa@mng1 2]$
```

Рисунок 9 - C reduction

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void printThreadsProc()
{
    int num_threads, num_proc;

    num_threads = omp_get_max_threads(); // максимально допустимое число нитей
    num_proc = omp_get_num_procs();      // число доступных ядер с учетом гипертрейдинга

    printf("\n----------------\n");
    printf("number of threads %d\n", num_threads); // равно значению OMP_NUM_THREADS
    printf("number of processors %d\n", num_proc); // равно значению OMP_NUM_THREADS
    printf("---------------\n");
}

int main(int argc, char *argv[])
{
    printf("\n\nExample 9.\n");

    omp_set_num_threads(26);
    printThreadsProc();

    int count = 0;
#pragma omp parallel
//reduction(+ : count)
    {
        count++;
        printf("Value of count: %d\n", count);
    }
    printf("Number of treads: %d\n", count);

    printf("\n\n---------------\nEND.\n---------------\n");
    return 0;
}
```

```
Example 9.

----------------
number of threads 26
number of processors 32
---------------
Value of count: 1
Value of count: 3
Value of count: 14
Value of count: 19
Value of count: 24
Value of count: 6
Value of count: 8
Value of count: 9
Value of count: 10
Value of count: 4
Value of count: 11
Value of count: 12
Value of count: 13
Value of count: 15
Value of count: 16
Value of count: 17
Value of count: 2
Value of count: 18
Value of count: 21
Value of count: 22
Value of count: 20
Value of count: 5
Value of count: 23
Value of count: 25
Value of count: 7
Value of count: 26
Number of treads: 26


---------------
END.
---------------
[starostenkov_aa@mng1 2]$
```

Рисунок 10 – Без reduction

При использовании опции reduction произошло корректное суммирование значений переменной count от каждого потока. Без опции reduction программа превратилась в ошибочную и непредсказуемую: переменная count стала глобальной, потоки стали беспорядочно её перезаписывать, в том числе, «вклиниваясь» во время записи этой же переменной другими потоками. Извлечь полезных результатов из такой программы нельзя.

1.9. Пример 9.1 выполнить без модификаций.

Код

```
int main(int argc, char *argv[])
{
    printf("\n\nExample 9.\n");

    omp_set_num_threads(26);
    printThreadsProc();

    int count = 0;
#pragma omp parallel reduction(+ : count)
    {
        count++;
        printf("Value of count: %d\n", count);
    }
    printf("Number of treads: %d\n", count);

    printf("\n\n---------------\nEND.\n---------------\n");
    return 0;
}
```

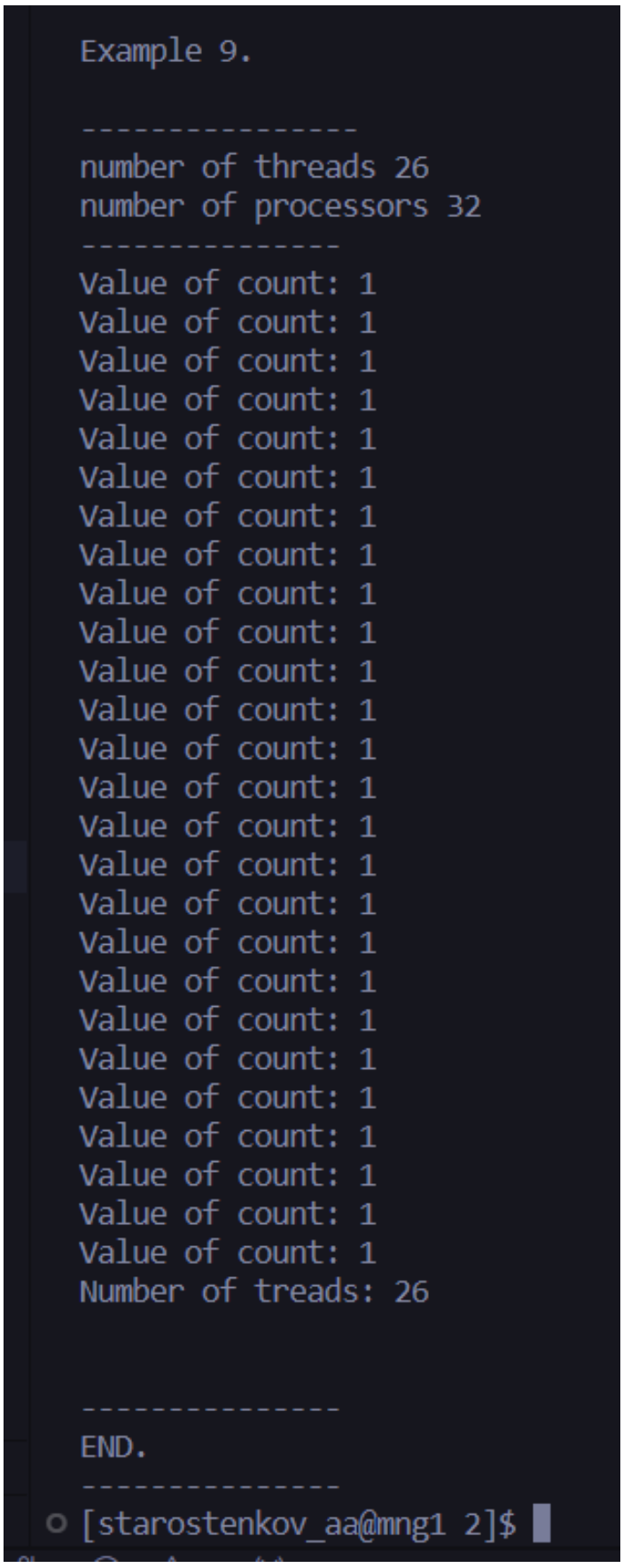

Рисунок 11 - Потоки

1.10. Пример 10 выполнить без модификаций. Определить число итераций цикла, реализованных каждой нитью.

Код

```c
int main(int argc, char *argv[])
{
    printf("\n\nExample 10.\n");

    int A[20], B[20], C[20], i, n;
    /* Заполнение массивов */
    for (i = 0; i < 20; i++)
    {
        A[i] = i;
        B[i] = i + 1;
        C[i] = 0;
    }
    omp_set_num_threads(5); // 5 потоков
#pragma omp parallel shared(A, B, C) private(i, n)
    { /* Номер текущей нити */
        n = omp_get_thread_num();
#pragma omp for
        for (i = 0; i < 20; i++)
        {
            C[i] = A[i] + B[i];
            printf("Tread %d number of array element %d\n ", n, i);
        }
    }

    printf("\n\n---------------\nEND.\n---------------\n");
    return 0;
}
```

```
[starostenkov_aa@mng1 2]$ ./10


Example 10.
Tread 0 number of array element 0
 Tread 0 number of array element 1
 Tread 0 number of array element 2
 Tread 0 number of array element 3
 Tread 3 number of array element 12
 Tread 3 number of array element 13
 Tread 3 number of array element 14
 Tread 3 number of array element 15
 Tread 2 number of array element 8
 Tread 2 number of array element 9
 Tread 2 number of array element 10
 Tread 2 number of array element 11
 Tread 1 number of array element 4
 Tread 1 number of array element 5
 Tread 1 number of array element 6
 Tread 1 number of array element 7
 Tread 4 number of array element 16
 Tread 4 number of array element 17
 Tread 4 number of array element 18
 Tread 4 number of array element 19


---------------
END.
---------------
[starostenkov_aa@mng1 2]$
```

Рисунок 12 – работы директивы for

Эта программа демонстрирует работу директивы for. 5 потоков суммируют два вектора по 20 элементов. OpenMP позаботился о том, чтобы равномерно распределить выполнение цикла между потоками. Число 20 делится на 5, так что каждому потоку выпало равное количество выполнений.