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 №3

«Отладка на кластере простых программ с использованием MPI»

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 гибридном вычислительном кластере (ГВК) СФМЭИ простые тестовые программы на языке Си с использованием технологии MPI. В качестве тестовых программ использовать программы 1- 8 из лекции 4 по курсу «Вычислительные системы».

2. Программу 1 запустить на текущем узле с числом процессов, равным **номеру по журналу+2**.

3. Программу 2 запустить на текущем узле с числом процессов, равным **номеру по журналу+1**.

4. Программу 3 запустить с числом процессов, равным **номеру по журналу+3** на одном, двух и трех узлах. Запустить программу 3 на одном, двух и трех узлах с максимальным числом процессов.

5. Программу 4 запустить несколько раз с числом процессов, равным **номеру по журналу+4.** Убедиться, что результат меняется от запуска к запуску.

6. Программу 5 запустить на текущем узле с числом процессов, равным **номеру по журналу+5**.

7. Программу 6 запустить с числом процессов, равным **10.** Модифицируйте программу 6 таким образом, чтобы в результате выполнения обмена сообщениями процессы 2 и 3 содержали следующие значения: **a=номер по журналу, b=номер по журналу+1**.

8. Программу 7 запустить с числом процессов, равным **10.** Убедиться в том, что показания таймера могут быть различными в разных процессах.

9. Программу 8 запустить с числом процессов, равным **10.**

10. Результат выполнения каждого примера дополнить кратким описанием (анализом, объяснением).

11. Выполнение примеров показать преподавателю.

12. Все выполненные задания оформить в виде отчета.

13. Отчет должен содержать: тексты примеров, команды компиляции и запуска, результаты запуска, объяснение результатов.

Ход работы

1. Программу 1 запустить на текущем узле с числом процессов, равным **номеру по журналу+2**.

Код

```c
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    int i;
    MPI_Init(&argc, &argv);
    if (i == MPI_SUCCESS)
        printf("Successful initialization with code %d\n", i);
    else
        printf("Initialization failed with error code %d\n", i);
    MPI_Finalize();
    if (i == MPI_SUCCESS)
        printf("Successful MPI_Finalize() with code %d\n", i);
    else
        printf("MPI_Finalize() failed with error code %d\n", i);
    return (0);
}
```

```
[starostenkov_aa@mng1 3]$ mpirun -n 21 1
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful initialization with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
Successful MPI_Finalize() with code 0
[starostenkov_aa@mng1 3]$
```

Рисунок 1 – Запуск с 21 процессором

Код выполнился штатно, успешно выведено по 21 строки инициализации.

2. Программу 2 запустить на текущем узле с числом процессов, равным **номеру по журналу+1**.

Код

```c
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
    int version, sub_version;

    MPI_Init(&argc, &argv);

    // специальная функция для определения версии
    MPI_Get_version(&version, &sub_version);
    printf("MPI Version %d.%d\n", version, sub_version);

    MPI_Finalize();
    return 0;
}
```

```
[starostenkov_aa@mng1 3]$ mpirun -n 20 2
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
MPI Version 3.0
[starostenkov_aa@mng1 3]$
```

Рисунок 2 – вывод версии

Запускается 20 процессов, которые выводят версию MPI.

3. Программу 3 запустить с числом процессов, равным **номеру по журналу+3** на одном, двух и трех узлах. Запустить программу 3 на одном, двух и трех узлах с максимальным числом процессов.

Код

```c
#include "mpi.h"
#include <stdio.h>

int main(int argc, char *argv[])
{
    int rank, size, len;
    char name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Процедура возвращает в
    //строке name имя узла, на котором запущен вызвавший процесс.
    //В переменной len возвращается количество символов в имени,
    //не превышающее значения  константы MPI_MAX_PROCESSOR_NAME
    MPI_Get_processor_name(name, &len);
    printf("Hello.  I am %d of %d on %s\n", rank, size, name);
    MPI_Finalize();
    return 0;
}
```

```
[starostenkov_aa@mng1 3]$ mpicc 3.c -o 3
[starostenkov_aa@mng1 3]$ mpirun -n 22 3
Hello.  I am 13 of 22 on mng1.sbmpei
Hello.  I am 1 of 22 on mng1.sbmpei
Hello.  I am 2 of 22 on mng1.sbmpei
Hello.  I am 4 of 22 on mng1.sbmpei
Hello.  I am 0 of 22 on mng1.sbmpei
Hello.  I am 3 of 22 on mng1.sbmpei
Hello.  I am 5 of 22 on mng1.sbmpei
Hello.  I am 7 of 22 on mng1.sbmpei
Hello.  I am 6 of 22 on mng1.sbmpei
Hello.  I am 8 of 22 on mng1.sbmpei
Hello.  I am 9 of 22 on mng1.sbmpei
Hello.  I am 10 of 22 on mng1.sbmpei
Hello.  I am 11 of 22 on mng1.sbmpei
Hello.  I am 12 of 22 on mng1.sbmpei
Hello.  I am 14 of 22 on mng1.sbmpei
Hello.  I am 21 of 22 on mng1.sbmpei
Hello.  I am 15 of 22 on mng1.sbmpei
Hello.  I am 16 of 22 on mng1.sbmpei
Hello.  I am 17 of 22 on mng1.sbmpei
Hello.  I am 18 of 22 on mng1.sbmpei
Hello.  I am 19 of 22 on mng1.sbmpei
Hello.  I am 20 of 22 on mng1.sbmpei
[starostenkov_aa@mng1 3]$
```

Рисунок 3 – запуска на mng1

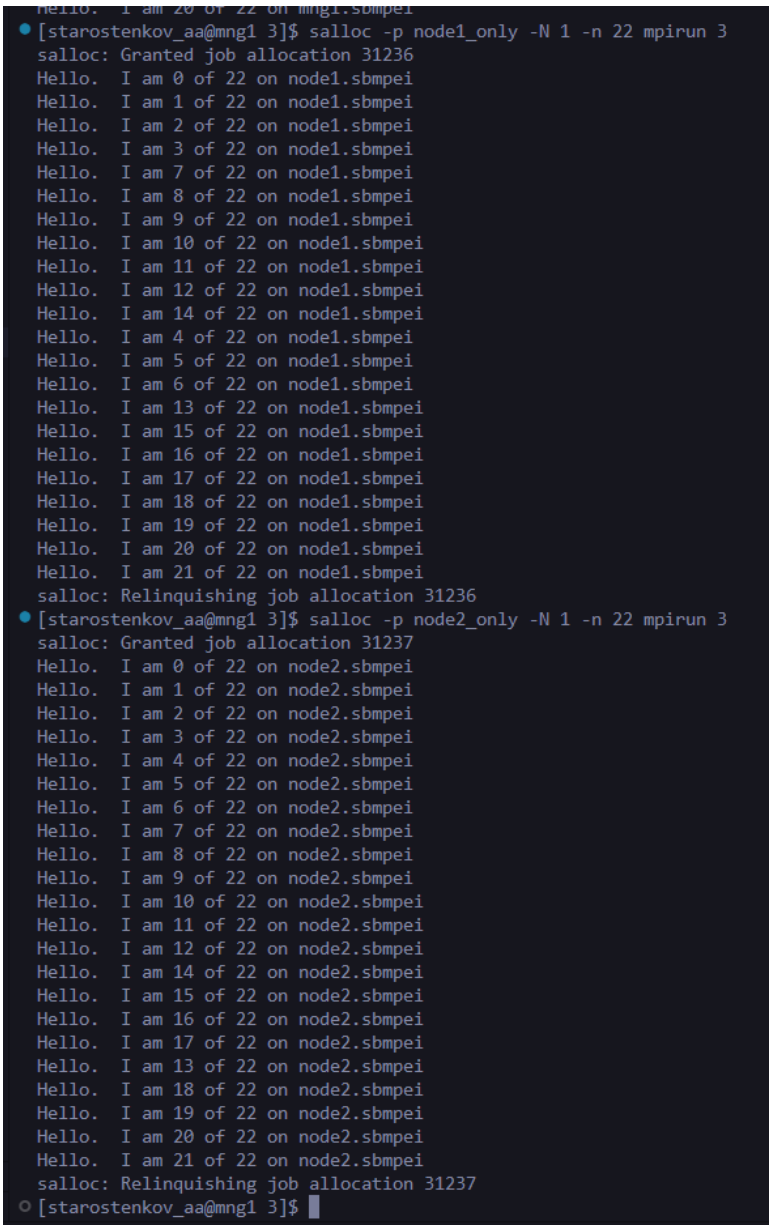

Рисунок 4 – запуск на node1 и node2 по отдельности

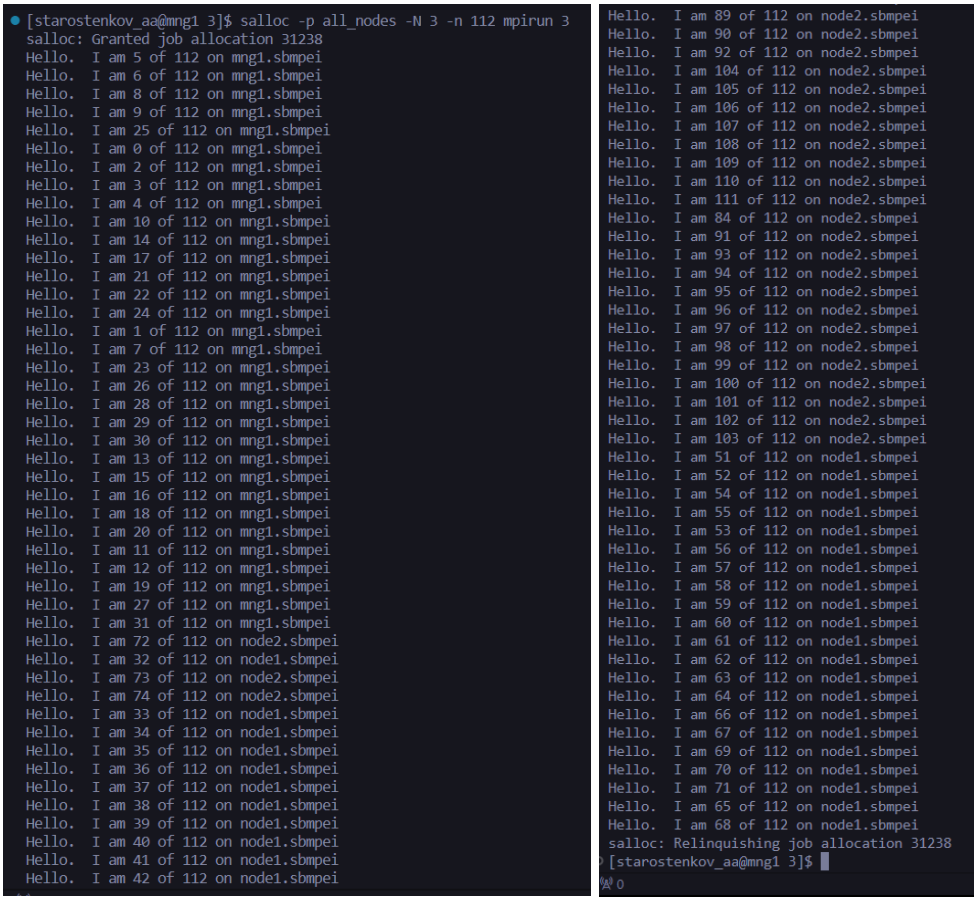

Рисунок 5 – запуск all_nodes

Запуск по узлам выполняется штатно.

4. Программу 4 запустить несколько раз с числом процессов, равным **номеру по журналу+4.** Убедиться, что результат меняется от запуска к запуску.

Код

```c
// Число процессов задается параметрами среды выполнения
// MPI при запуске, а не в программе!
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("process %d, size %d\n", rank, size);
    MPI_Finalize();
    return (0);
}
```

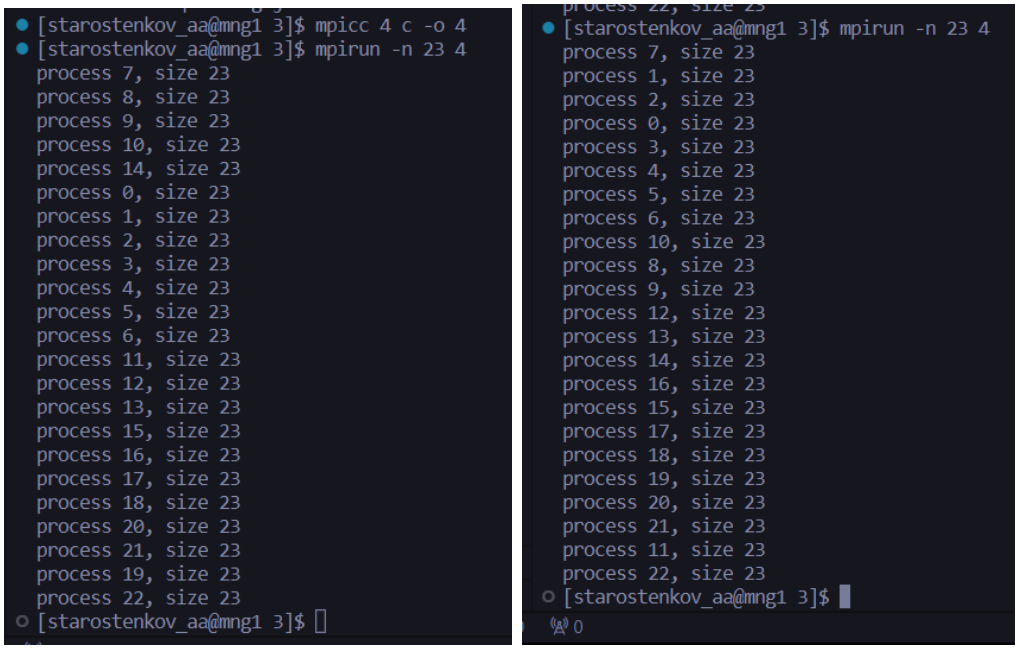

Рисунок 6 – процессы

Порядок выполнения процессов является недетерминированным, поэтому вывод программы меняется от запуска к запуску.

5. Программу 5 запустить на текущем узле с числом процессов, равным **номеру по журналу+5**.

Код

```c
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    printf("Before MPI_INIT\n");
    MPI_Init(&argc, &argv);
    printf("Parallel section\n");
    MPI_Finalize();
    printf("After MPI_FINALIZE\n");
    return (0);
}
```

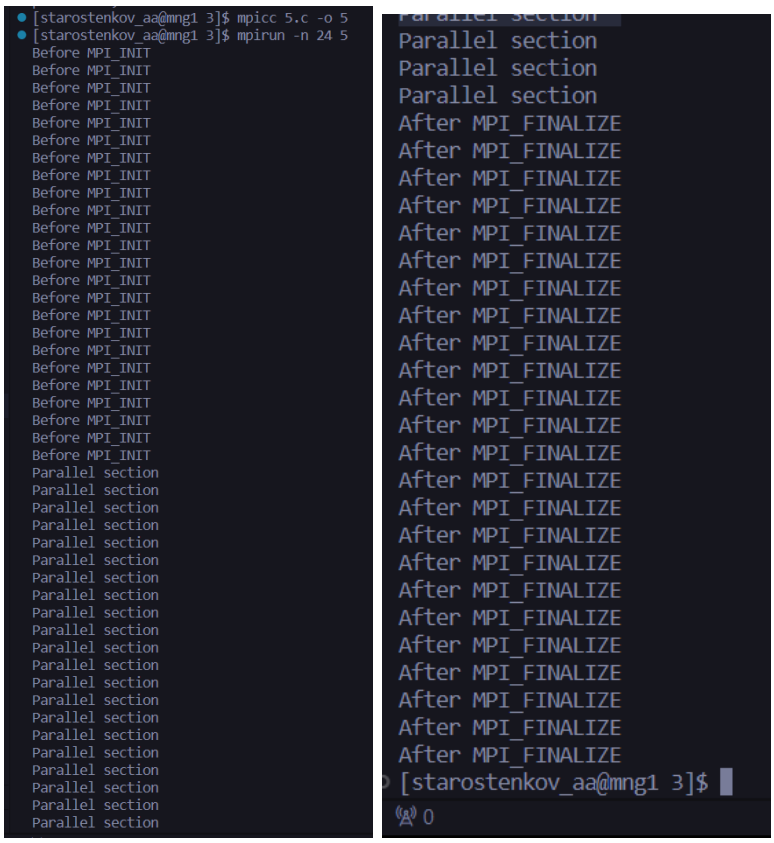

Рисунок 7 – Параллельное выполнение

Было выполнено по 3 блока, каждый одновременно имел 24 процесса, что наблюдается на рисунке.

6. Программу 6 запустить с числом процессов, равным **10.** Модифицируйте программу 6 таким образом, чтобы в результате выполнения обмена сообщениями процессы 2 и 3 содержали следующие значения: **a=номер по журналу, b=номер по журналу+1**.

Код

```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    int rank;
    float a, b;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    a = 0.0;
    b = 0.0;

    if (rank == 2)
    {
        b = 20.0;
        MPI_Send(&b, 1, MPI_FLOAT, 3, 5, MPI_COMM_WORLD);
        MPI_Recv(&a, 1, MPI_FLOAT, 3, 10, MPI_COMM_WORLD, &status);
    }
    else if (rank == 3)
    {
        a = 19.0;
        MPI_Recv(&b, 1, MPI_FLOAT, 2, 5, MPI_COMM_WORLD, &status);
        MPI_Send(&a, 1, MPI_FLOAT, 2, 10, MPI_COMM_WORLD);
    }

    printf("process %d a = %f, b = %f\n", rank, a, b);
    MPI_Finalize();
    return 0;
}
```

```
[starostenkov_aa@mng1 3]$ mpicc 6.c -o 6
[starostenkov_aa@mng1 3]$ mpirun -n 10 6
process 0 a = 0.000000, b = 0.000000
process 1 a = 0.000000, b = 0.000000
process 4 a = 0.000000, b = 0.000000
process 5 a = 0.000000, b = 0.000000
process 7 a = 0.000000, b = 0.000000
process 2 a = 19.000000, b = 20.000000
process 3 a = 19.000000, b = 20.000000
process 6 a = 0.000000, b = 0.000000
process 8 a = 0.000000, b = 0.000000
process 9 a = 0.000000, b = 0.000000
[starostenkov_aa@mng1 3]$
```

Рисунок 8 – модифицированная программа

Процесс 2 задаёт значение b=20 и отправляет его процессу 3, затем получает значение a от процесса 3. Процесс 3 получает значение b, затем задаёт значение a=19 и отправляет его процессу 2. В результате, процессы 2 и 3 содержат следующие значения: a=19, b=20.

7. Программу 7 запустить с числом процессов, равным **10.** Убедиться в том, что показания таймера могут быть различными в разных процессах.

Код

```c
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    double start_time, end_time, tick;
    MPI_Init(&argc, &argv);
    start_time = MPI_Wtime();
    sleep(1);
    end_time = MPI_Wtime();
    tick = MPI_Wtick();
    printf("Time to measure %E\n", end_time - start_time);
    printf("Accuracy of timer %E\n", tick);
    MPI_Finalize();
    return 0;
}
```

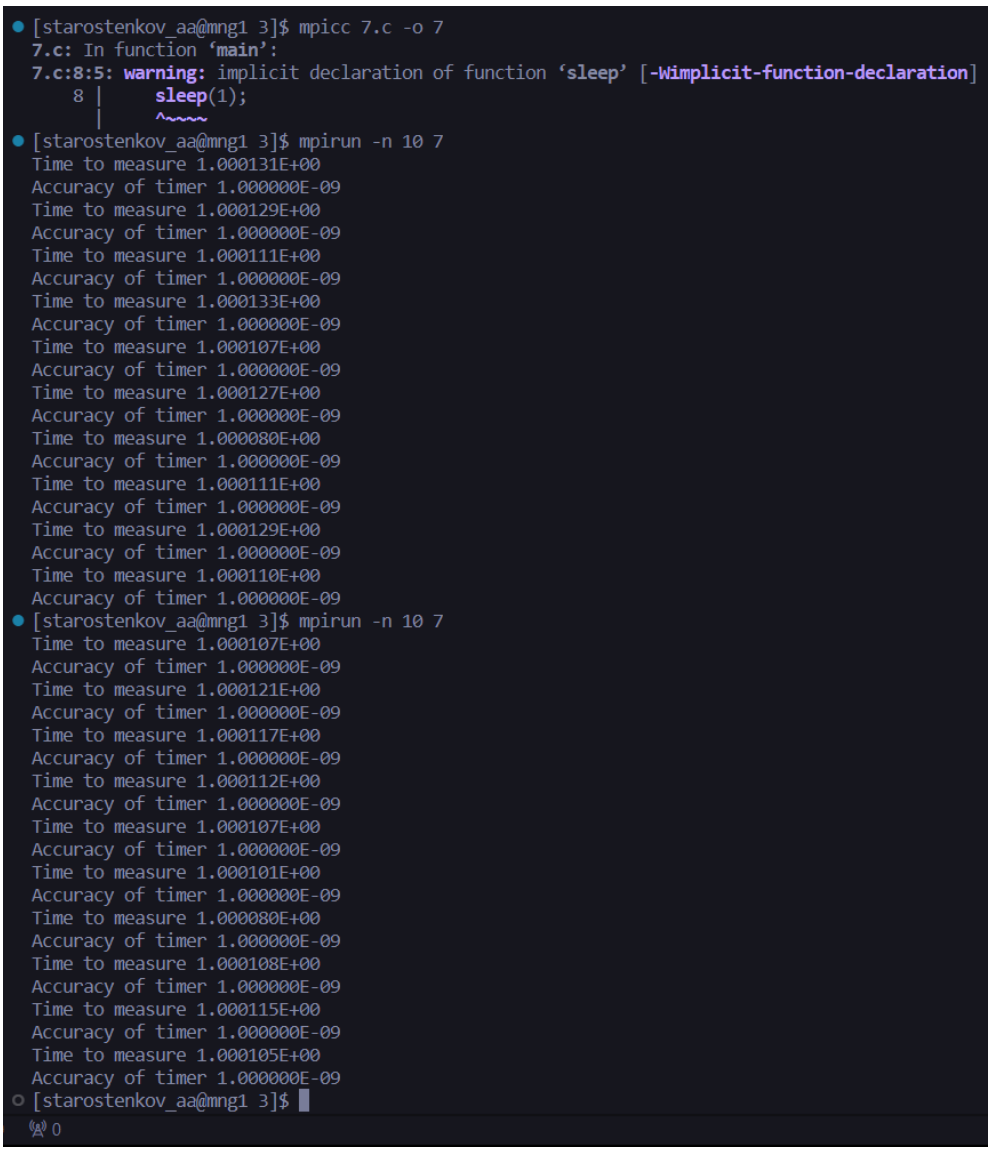

Рисунок 9 - Показания таймера

Вывод: программа работает корректно. Измеренное время 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.

Точность таймера MPI составляет $1 \cdot 10^{-6}$ с, в то время как точность таймера OpenMP составляла $1 \cdot 10^{-9}$ с. Таким образом, таймер MPI на три порядка менее точен.

8. Программу 8 запустить с числом процессов, равным **10.**

Код

```c
#include <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
    int size, rank, a, b;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    // Действия, выполняемые только процессом с рангом 0
    {
        a = 1;
    }
    else // Действия, выполняемые остальными процессами
    {
        a = 0;
    }
    // Широковещательная рассылка //«a» от нулевого процесса всем остальным
    MPI_Bcast(&a, 1, MPI_INT, 0, MPI_COMM_WORLD);

    // Редукция в //виде суммы «a» от всех процессов на нулевом процессе. Результат суммы «a» //
записывается в «b»
    MPI_Reduce(&a, &b, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Все процессы печатают свой номер и «b». //Для всех процессов, кроме нулевого, «b» не определено.
Для нулевого: b=size
    printf("process %d, b = %d\n", rank, b);
    MPI_Finalize();
    return 0;
}
```

```
[starostenkov_aa@mng1 3]$ mpicc 8 c -o 8
[starostenkov_aa@mng1 3]$ mpirun -n 10 8
process 5, b = 2031034048
process 8, b = -1252538160
process 9, b = -1121298864
process 1, b = -744477792
process 2, b = 648424464
process 3, b = -133608400
process 6, b = 577271792
process 7, b = -563184496
process 0, b = 10
process 4, b = 1918642256
[starostenkov_aa@mng1 3]$
```

Рисунок 10 – Выполнение кода

Программа работает корректно. Каждый процесс определяет свой ранг, после чего действия в программе разделяются. Все процессы, кроме процесса

с рангом 0, передают значение своего ранга нулевому процессу. Процесс с рангом 0 сначала печатает значение своего ранга, а далее последовательно принимает сообщения с рангами процессов и также печатает их значения. Порядок приема сообщений заранее не определен и зависит от условий выполнения параллельной программы (более того, этот порядок может изменяться от запуска к запуску).