|  |  |
| --- | --- |
| ФИО Студента | Витомсков Алексей Вадимович |
| Группа | КЭ-220 |
| Курс | Технологии параллельного программирования |
| Задание | Задание 32. Программа «Число 𝜋» |
| Отчет по выполнению практического задания | |

***1. Реализуйте программу вычисления числа 𝜋 (см. задание 8) с использованием MPI+OpenMP.  
Входные данные: одно целое число N (точность вычисления).  
Выходные данные: одно вещественное число pi.***

*Bat файл для запуска программы:*

|  |
| --- |
| CMD /c  CLS &  "C:\Program Files\Microsoft MPI\Bin\mpiexec.exe" -np X Lab1.exe  Y Z &  PAUSE |

Здесь:

X - количество процессов MPI, передаваемое в mpiexec.exe;

Lab1.exe – наименование исполняемого файла программы;

Y – количество нитей OpenMP (передается в Lab1.exe);

Z – количество итераций цикла для вычисления числа pi.

*Код программы:*

|  |
| --- |
| #include <mpi.h>  #include <omp.h>  #include <stdio.h>  #include <iostream>  #include <windows.h>  int main(int argc, char\* argv[])  {  MPI\_Init(&argc, &argv);  double t1 = omp\_get\_wtime();  int num\_threads;  num\_threads = atoi(argv[1]);  omp\_set\_num\_threads(num\_threads);  int N;  N = atoi(argv[2]);    int rank\_mpi; //номер текущего процесса  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank\_mpi);  int size\_mpi; //Количество процессов в приложении  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size\_mpi);    int position = rank\_mpi \* (N / size\_mpi);  double sum = 0;  #pragma omp parallel reduction(+:sum)  {  int rank\_omp, size\_omp;  rank\_omp = omp\_get\_thread\_num();  size\_omp = omp\_get\_num\_threads();  printf("I am %d thread from %d> process. Number of hybrid threads = %d. position = %d\n", rank\_omp,  rank\_mpi, size\_omp \* size\_mpi, position);  #pragma omp for schedule(static)  for (int i = position; i < (position + (N / size\_mpi)); i++) {  double x = (i + 0.5) / N;  sum += 4 / (1 + pow(x, 2));  }  int num = omp\_get\_thread\_num();  printf("%d thread sum = %.10f\n", num, sum);  }  sum /= N; //результат суммы в каждом потоке  double buf = sum; //сообщение  printf("Summ of process %d = %.10f\n", rank\_mpi, sum);    MPI\_Status status; //атрибуты принятого сообщения    if (rank\_mpi == 0) { //схема мастер-рабочие  for (int i = 1; i < size\_mpi; i++) {  MPI\_Recv(&buf, 1, MPI\_DOUBLE, i, 1, MPI\_COMM\_WORLD, &status);  printf("receive message <%.10f> from <%d>\n", buf, i);  sum += buf;  }  double t2 = omp\_get\_wtime();  printf("\nPi = %.10f, time = %.10f\n\n", sum, t2-t1);  }  else {  MPI\_Send(&buf, 1, MPI\_DOUBLE, 0, 1, MPI\_COMM\_WORLD);  }  MPI\_Finalize();  return 0;  } |

*Описание работы программы*

В программу mpiexec.exe передается число процессов X, с которым запускается программа Lab1.exe. В свою очередь, в программу Lab1.exe передается число нитей в каждом процессе Y и число итераций цикла for Z.

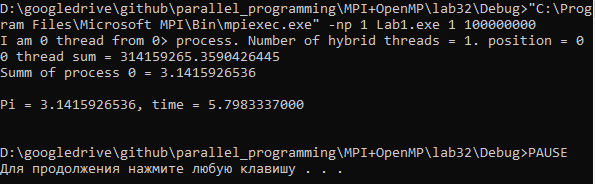
Функция MPI\_Init() инициализирует Х процессов MPI, каждому процессу передаются аргументы, переданные в командной строке. В t1 записывается время начала работы алгоритма. Задается число нитей в каждом процессе. В переменные rank\_mpi и size\_mpi считывается номер текущего процесса и количество процессов. В переменную N считывается число итераций цикла for Z.

Все итерации цикла for равномерно распределяются между процессами. Для этого для каждого процесса рассчитывается переменная position, начиная с которой будет вычислен цикл for для N / size\_mpi итераций. Создается глобальная переменная sum, в которой в параллельном openmp регионе выполняется суммирование результата, вычисляемого нитями. Далее идет параллельный регион с циклом for. Оператор schedule распределяет выполнение итераций между нитями последовательно (static).

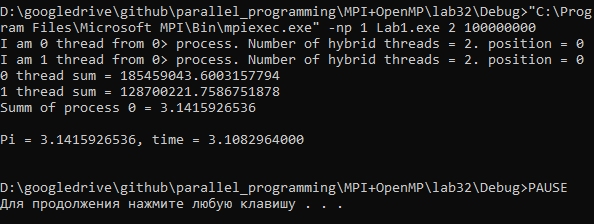
После выхода из параллельного openmp региона имеется X переменных sum. Далее используется MPI схема «Мастер-рабочие». Все процессы, кроме процесса 0 (рабочие), направляют процессу 0 (мастеру) свое значение sum. Используется функция передачи и приема с блокировкой. Мастер прибавляет к своему значению sum переданные значения. После завершения приема от всех процессов в t2 записывается время окончания работы алгоритма. Результат выводится на экран, он равен числу pi.

*Результат работы программы:*

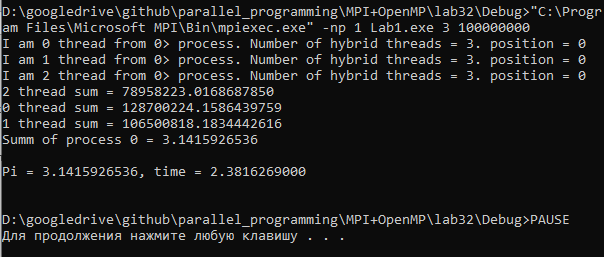
***число процессов = 1, число нитей = 1, N = 100000000:***



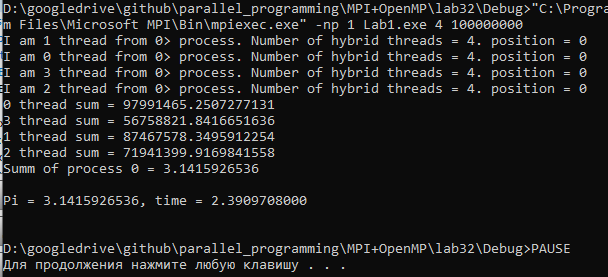
***число процессов = 1, число нитей = 2, N = 100000000:***



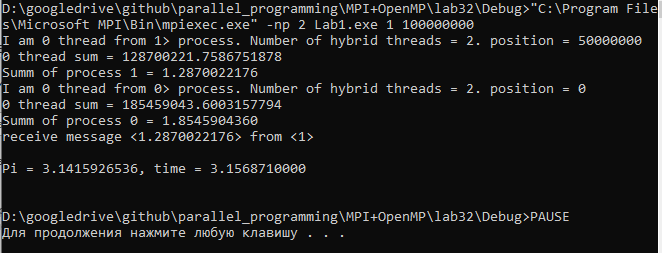
***число процессов = 1, число нитей = 3, N = 100000000:***



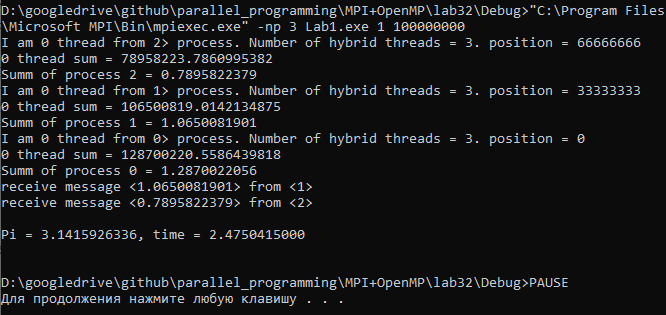
***число процессов = 1, число нитей = 4, N = 100000000:***



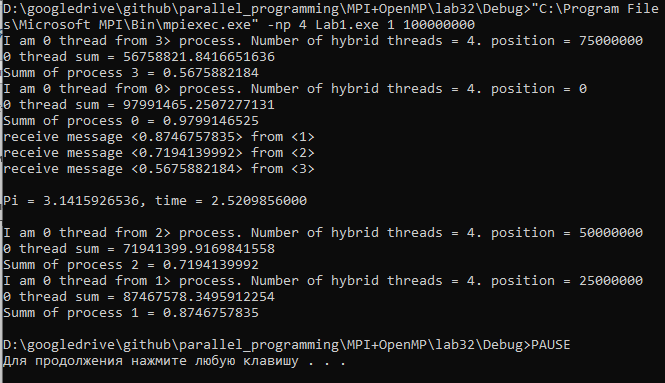
***число процессов = 2, число нитей = 1, N = 100000000:***



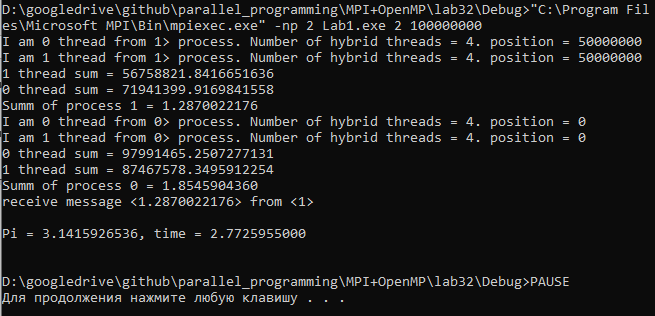
***число процессов = 3, число нитей = 1, N = 100000000:***



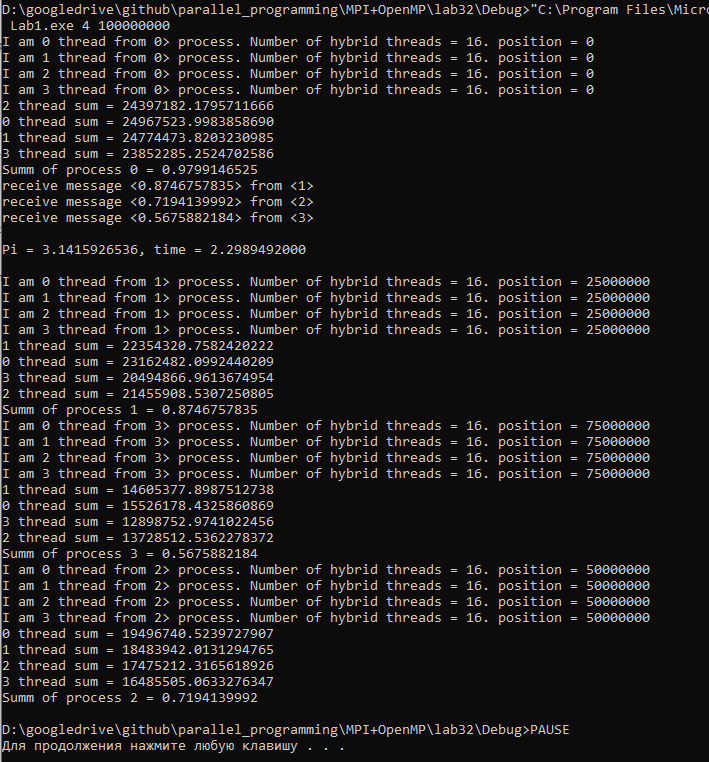
***число процессов = 4, число нитей = 1, N = 100000000:***



***число процессов = 2, число нитей = 2, N = 100000000:***



***число процессов = 4, число нитей = 4, N = 100000000:***



***число процессов = 1, число нитей = 100, N = 100000000:***

***число процессов = 100, число нитей = 1, N = 100000000:***



***число процессов = 100, число нитей = 100, N = 100000000:***



***Время работы программы для N=100 000 000 представлено в таблице 1.***

***Таблица 1***

|  |  |  |
| --- | --- | --- |
| ***число процессов*** | ***число нитей*** | ***время, с  на intel i5-2450M*** |
| ***1*** | ***1*** | ***5,8*** |
| ***1*** | ***2*** | ***3,1*** |
| ***1*** | ***3*** | ***2,4*** |
| ***1*** | ***4*** | ***2,4*** |
| ***2*** | ***1*** | ***3,2*** |
| ***3*** | ***1*** | ***2,5*** |
| ***4*** | ***1*** | ***2,5*** |
| ***2*** | ***2*** | ***2,8*** |
| ***4*** | ***4*** | ***2,3*** |
| ***1*** | ***100*** | ***2,2*** |
| ***1*** | ***200*** | ***2,3*** |
| ***100*** | ***1*** | ***1,5*** |
| ***200*** | ***1*** | ***0,6*** |
| ***100*** | ***100*** | ***3,6*** |

Таким образом, наилучшего результата удалось добиться при 200 процессах MPI, время выполнения 0,6 с. Наихудший результат при 1 процессе и 1 нити, как и следовало ожидать.

***5. Ответить на вопросы***

1) Поясните, из каких соображений выбраны способы распределения итераций по процессам и нитям.

При MPI подходе накладными расходами будет являться пересылка сообщений между процессами. В данной программе пересылаются всего 99 сообщений, поэтому этим можно пренебречь. Также каждому процессу необходимо хранить в памяти свои переменные.

Нити в рамках процесса работают в общей памяти, поэтому в целях экономии ресурсов на архитектуре с общей памятью выгоднее использовать OpenMP. Таблица показывает, что по времени выполнения MPI и OpenMP подходы практически равнозначны для данной задачи.