МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

ЛАБОРАТОРНАЯ РАБОТА №3

Поиск суммы двух векторов с использованием технологий OpenMP и MPI

по курсу
Параллельное программирование

Студент гр. 6407 _____ В.С. Гринина
Преподаватель,
к.ф.-м.н. _____ Е.С. Козлова

САМАРА 2018

ЗАДАНИЕ

Произвести запуск программ для поиска суммы векторов, которые используют технологии MPI и OpenMP, на различном количестве нитей (процессов).

На основе технологии OpenMP реализовать три варианта работы с использованием различных настроек планировщика задач.

На основе технологии MPI реализовать два варианта программы:

1. Количество элементов массива кратно количеству процессов;
2. Количество элементов массива не кратно количеству процессов.

В программе с использованием технологией MPI использовать схему распределенного хранения данных: каждый поток хранит только ту часть исходных данных, которая необходима ему для расчетов. На 0-м процессе также хранятся полные копии обрабатываемых и результирующих векторов.

В ходе анализа работы программы оценить время ее выполнения на различном количестве исполняющих нитей (процессов). Оценить влияние различных функций и директив (опций) на скорость работы приложений. [1]

ВВЕДЕНИЕ

В большинстве случаев распараллеливание задачи на равные одинаковые куски нецелесообразно, так как нити/процессы имеют различные ресурсы. Для решения данной проблемы в OpenMP существует планировщик задач, который сам распределяет подзадачи между нитями в зависимости от настроек самого планировщика и загруженности нити.

Многие задачи также включают в себя коллективные операции над большим количеством данных. Такие задачи прекрасно подходят для того, чтобы их распараллеливать, особенно с технологией MPI, в которая полна коллективными операциями. Зачастую исходные данные в программе формируются на исходном нулевом процессе. Однако для обработки данные необходимо предоставить всем процессам. [1] В данной работе рассматривается несколько вариантов рассылки и сбора информации в MPI.

1 Ход выполнения работы

Для удаленного доступа к командной строке сервера будем использовать протокол SSH. Для простоты будем использовать командную строку. Для синхронизации файлов между компьютером и сервером используем протокол SSHFS, который позволяет примонтировать удаленную директорию. Редактировать код будем в SublimeText.

Компиляция и запуск файлов производится таким же образом, как и в предыдущих лабораторных работах.

1.1 Технология OpenMP

Для программ с технологией OpenMP используется опция регулировщика задач schedule в директиве for, где тип распределения элементов между нитями задается в параметрах. В данной лабораторной работе были использованы такие типы, как static (распределяет блоки фиксированной длины по всем нитям равномерно), dynamic (динамически распределяет блоки фиксированной длины между свободными нитями) и guided (динамически распределяет блоки, которые со временем уменьшаются в длине между свободными нитями).

Добавление данной опции так же, как и в предыдущей лабораторной работе, занимает всего одну строчку. Код программ, использующих технологию OpenMP, приведен в приложении.

1.2 Технология MPI

Для случая, когда количество элементов в массиве кратно количеству процессов в MPI, используется функция scatter, иначе scatterv. Первая функция производит рассылку блоками одинаковой длины, в случае вызова второй функции можно задать длину блока для каждого процесса. Сбор информации на 0-ой процесс происходит аналогичными функциями gather и gatherv.

Написание кода MPI так же, как и в предыдущей лабораторной работе, занимает несколько больше времени, чем в MPI. Однако, использование коллективных операций сильно удобнее, чем, например, пересылка точка-точка. Код программ, использующих технологию MPI, приведен в приложении.

1.3 Время выполнения программ

Для дополнительного анализа работы программ засечем время их выполнения и внесем эти данные в таблицу 1.

Из таблицы заметно, что с использованием регулировщика задач в технологии OpenMP быстрее всего работает guided распределение блоков.

На основе результатов выполнения программ с технологией MPI сложно сделать какой-либо вывод. Но можно заметить, что в среднем, когда количество элементов массива кратно количеству процессов, программа работает чуть быстрее, чем в ином случае.

Стоит также заметить, что время выполнения программ зависит нелинейно от количества нитей (процессов).

Таблица 1 – Время выполнения программ при различных параметрах

Количество нитей (процессов)	OpenMP			MPI	
	Static	Dynamic	Guided	Кратно	Не кратно
1	0.011358	0.011343	0.011066	0.023205	0.023549
8	0.010863	0.015432	**0.005514**	0.016103	0.021322
16	0.036912	0.079350	0.095770	0.015081 (2n/8ppn) 0.015680 (4n/4ppn)	0.014758 (2n/8ppn) 0.026161 (4n/4ppn)

ЗАКЛЮЧЕНИЕ

В ходе проведения данной лабораторной работы в технологии OpenMP были сравнены различные распределения информации между нитями с использованием регулировщика задач. Практическим путем было установлено, что guided распределение является более предпочтительным.

С использованием технологии MPI были написаны программы для двух случаев: когда количество элементов массива кратно количеству процессов и иначе. Сделать какие-либо точные оценки по результатам выполнения программ MPI сложно.

Было выявлено, что распараллеливание программы не всегда приходит к более быстрому ее выполнению.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Козлова Е.С. Методические указания к лабораторной работе №1 по курсу «Параллельное программирование» Методические указания к лабораторным работам [Текст]/ Сост. Козлова Е.С. – Самара, 2017. 12 с.

[2] Параллельные вычисления [Электронный ресурс] // Википедия: свободная энцикл. – Электрон. дан. – [Б. м.], 2017. – URL: https://en.wikipedia.org/wiki/Concurrent_computing (дата обращения: 10.09.2018).

[3] Справочное руководство по OpenMP [Электронный ресурс] // OpenMP: – Электрон. дан. – [Б. м.], 2015. – URL: https://www.openmp.org/resources/refguides/ (дата обращения: 08.10.2018).

[4] Справочное руководство в примерах по MPI [Электронный ресурс] // MPI Tutorials: – Электрон. дан. – [Б. м.], 2018. – URL: http://mpitutorial.com/tutorials/ (дата обращения: 10.10.2018).

[5] Страуструп, Б. Язык программирования С++ [Текст]/ / Б. Страуструп. – М:Бином, 2011 –1136 с

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ OPENMP DYNAMIC

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define CHUNK 100
#define NMAX 1500000
#include<omp.h>

int main(int argc, char* argv[]) {
  int i;
  double *a, *b, *sum, s = 0;
  int chunk, n;
  chunk = CHUNK;
  n = NMAX;
  omp_set_num_threads(16);

  a = malloc(sizeof(double) * n);
  b = malloc(sizeof(double) * n);
  sum = malloc(sizeof(double) * n);
  // инициализация данных
  for (i = 0; i < n; i++) {
    a[i] = (double)rand();
    b[i] = (double)rand();
  }
  double st_time, end_time;
  st_time = omp_get_wtime();
  #pragma omp parallel for schedule(dynamic, chunk) shared(a,
b, sum, n) private(i)
  for (i = 0; i < n; i++) {
    // суммирование векторов
    sum[i] = a[i] + b[i];
  }
  end_time = omp_get_wtime();
  s = end_time - st_time;

  long long hash_sum = 0;
  for (i = 0; i < n; i++) {
    hash_sum ^= *(long long *)(sum+i);
  }
  printf("\nHASH SUM OF RESULT IS %lld ", hash_sum);
  printf("\nTIME OF WORK IS %f ", s);
  free(a);
  free(b);
  free(sum);
  return 0;
}
```

ПРИЛОЖЕНИЕ Б
КОД ПРОГРАММЫ OPENMP GUIDED

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define CHUNK 100
#define NMAX 1600000
#include<omp.h>

int main(int argc, char* argv[]) {
  int i;
  double *a, *b, *sum, s = 0;
  int chunk, n;
  chunk = CHUNK;
  n = NMAX;
  omp_set_num_threads(16);

  a = malloc(sizeof(double) * n);
  b = malloc(sizeof(double) * n);
  sum = malloc(sizeof(double) * n);
  // инициализация данных
  for (i = 0; i < n; i++) {
    a[i] = (double)rand();
    b[i] = (double)rand();
  }

  double st_time, end_time;
  st_time = omp_get_wtime();
  #pragma omp parallel for schedule(guided, chunk) shared(a,
b, sum, n) private(i)
  for (i = 0; i < n; i++) {
    // суммирование векторов
    sum[i] = a[i] + b[i];
  }
  end_time = omp_get_wtime();
  s = end_time - st_time;
  long long hash_sum = 0;
  for (i = 0; i < n; i++) {
    hash_sum ^= *(long long *)(sum+i);
  }
  printf("\nHASH SUM OF RESULT IS %lld ", hash_sum);
  printf("\nTIME OF WORK IS %f ", s);
  free(a);
  free(b);
  free(sum);
  return 0;
}
```

ПРИЛОЖЕНИЕ В
КОД ПРОГРАММЫ OPENMP STATIC

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define CHUNK 100
#define NMAX 1600000
#include<omp.h>

int main(int argc, char* argv[]) {
  int i;
  double *a, *b, *sum, s = 0;
  int chunk, n;
  chunk = CHUNK;
  n = NMAX;
  omp_set_num_threads(8);

  a = malloc(sizeof(double) * n);
  b = malloc(sizeof(double) * n);
  sum = malloc(sizeof(double) * n);
  // инициализация данных
  for (i = 0; i < n; i++) {
    a[i] = (double)rand();
    b[i] = (double)rand();
  }

  double st_time, end_time;
  st_time = omp_get_wtime();
  #pragma omp parallel for schedule(static, chunk) shared(a,
b, sum, n) private(i)
  for (i = 0; i < n; i++) {
    // суммирование векторов
    sum[i] = a[i] + b[i];
  }
  end_time = omp_get_wtime();
  s = end_time - st_time;
  long long hash_sum = 0;
  for (i = 0; i < n; i++) {
    hash_sum ^= *(long long *)(sum+i);
  }
  printf("\nHASH SUM OF RESULT IS %lld ", hash_sum);
  printf("\nTIME OF WORK IS %f ", s);
  free(a);
  free(b);
  free(sum);
  return 0;
}
```

ПРИЛОЖЕНИЕ Г
КОД ПРОГРАММЫ MPI (КРАТНЫЙ СЛУЧАЙ)

```
#include <stdlib.h>
#include "mpi.h"
#include <math.h>
#include <stdio.h>
#define NMAX 1600000

int main(int argc, char* argv[]) {
  double *a, *b, *sum = NULL, s = 0;
  int i, j;
  double *a_loc, *b_loc, *sum_loc;
  int ProcRank, ProcNum, N = NMAX;
  MPI_Status Status;
  double st_time, end_time;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
  int count = N / ProcNum;
  // инициализация 0-ым процессом исходных данных
  srand(time(NULL));
  if (ProcRank == 0) {
    a = malloc(sizeof(double) * N);
    b = malloc(sizeof(double) * N);
    sum = malloc(sizeof(double) * N);
    for (i = 0; i < N; i++) {
      a[i] = (double)rand();
      b[i] = (double)rand();
    }
  }

  a_loc = (double*)malloc(count * sizeof(double));
  b_loc = (double*)malloc(count * sizeof(double));
  sum_loc = (double*)malloc(count * sizeof(double));

  st_time = MPI_Wtime();
  // рассылка 0-ым процессом по всем остальным
  MPI_Scatter(a, count, MPI_DOUBLE, a_loc, count, MPI_DOUBLE,
0, MPI_COMM_WORLD);
  MPI_Scatter(b, count, MPI_DOUBLE, b_loc, count, MPI_DOUBLE,
0, MPI_COMM_WORLD);
  // получение локальной суммы векторов
  for (i = 0; i < count; i++) {
    sum_loc[i] = a_loc[i] + b_loc[i];
  }
  // сборка результата 0-ым процессом
```

```
  MPI_Gather(sum_loc, count, MPI_DOUBLE, sum, count,
MPI_DOUBLE, 0, MPI_COMM_WORLD);

  end_time = MPI_Wtime();

  s = end_time - st_time;

  if (ProcRank == 0) {
    // printf("a ");
    // for (i = 0; i < 3; i++) {
    //   printf("%.0f ", a[i]);
    // }
    // printf("...\n");
    // printf("b ");
    // for (i = 0; i < 3; i++) {
    //   printf("%.0f ", b[i]);
    // }
    // printf("...\n");
    // printf("c ");
    // for (i = 0; i < 3; i++) {
    //   printf("%.0f ", sum[i]);
    // }
    // printf("...\n");
    printf("\nTIME OF WORK IS %f ", s);
    free(a);
    free(b);
    free(sum);
  }
  free(a_loc);
  free(b_loc);
  free(sum_loc);
  MPI_Finalize();
  return 0;
```

ПРИЛОЖЕНИЕ Д
КОД ПРОГРАММЫ MPI (НЕКРАТНЫЙ СЛУЧАЙ)

```
#include <stdlib.h>
#include "mpi.h"
#include <math.h>
#include <stdio.h>
#define NMAX 1500000

int main(int argc, char* argv[]) {
  double *a, *b, *sum = NULL, s = 0;
  int i, j;
  double *a_loc, *b_loc, *sum_loc;
  int ProcRank, ProcNum, N = NMAX;
  MPI_Status Status;
  double st_time, end_time;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

  int *counts;
  counts = malloc(sizeof(int) * ProcNum);
  int *displs;
  displs = malloc(sizeof(int) * ProcNum);

  int prevCounts = 0;
  for (i = 0; i < ProcNum; i++) {
    counts[i] = (N - prevCounts) / (ProcNum - i);
    displs[i] = prevCounts;
    prevCounts += counts[i];
  }

  // инициализация 0-ым процессом исходных данных
  srand(time(NULL));
  if (ProcRank == 0) {
    a = malloc(sizeof(double) * N);
    b = malloc(sizeof(double) * N);
    sum = malloc(sizeof(double) * N);
    for (i = 0; i < N; i++) {
      a[i] = (double)rand();
      b[i] = (double)rand();
    }
  }

  a_loc = (double*)malloc(counts[ProcRank] * sizeof(double));
  b_loc = (double*)malloc(counts[ProcRank] * sizeof(double));
```

```
  sum_loc = (double*)malloc(counts[ProcRank] *
sizeof(double));

  st_time = MPI_Wtime();
  // рассылка 0-ым процессом по всем остальным
  MPI_Scatterv(a, counts, displs, MPI_DOUBLE, a_loc,
counts[ProcRank], MPI_DOUBLE, 0, MPI_COMM_WORLD);
  MPI_Scatterv(b, counts, displs, MPI_DOUBLE, b_loc,
counts[ProcRank], MPI_DOUBLE, 0, MPI_COMM_WORLD);
  // получение локальной суммы векторов
  for (i = 0; i < counts[ProcRank]; i++) {
    sum_loc[i] = a_loc[i] + b_loc[i];
  }
  // сборка результата 0-ым процессом
  MPI_Gatherv(sum_loc, counts[ProcRank], MPI_DOUBLE, sum,
counts, displs, MPI_DOUBLE, 0, MPI_COMM_WORLD);

  end_time = MPI_Wtime();
  s = end_time - st_time;

  if (ProcRank == 0) {
    // printf("a ");
    // for (i = 0; i < 3; i++) {
    //   printf("%.0f ", a[i]);
    // }
    // printf("...\n");
    // printf("b ");
    // for (i = 0; i < 3; i++) {
    //   printf("%.0f ", b[i]);
    // }
    // printf("...\n");
    // printf("c ");
    // for (i = 0; i < 3; i++) {
    //   printf("%.0f ", sum[i]);
    // }
    // printf("...\n");
    printf("\nTIME OF WORK IS %f ", s);
    free(a);
    free(b);
    free(sum);
  }
  free(counts);
  free(displs);
  free(a_loc);
  free(b_loc);
  free(sum_loc);
  MPI_Finalize();
  return 0;
}
```

ПРИЛОЖЕНИЕ Е
КОД СКРИПТА ЗАПУСКА MPI

```
#!/bin/bash
#PBS -N MPI_mapo_8
#PBS -l walltime=00:01:10
#PBS -l nodes=1:ppn=8
#PBS -j oe
#PBS -A tk

cd $PBS_O_WORKDIR

module load impi/4
export I_MPI_DEVICE=rdma
export I_MPI_DEBUG=0
export I_MPI_FALLBACK_DEVICE=disable
mpirun -r ssh -machinefile $PBS_NODEFILE -np $PBS_NP ./MPI
```