МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по учебному курсу

"Суперкомпьютеры и параллельная обработка данных"

Задание №1:

Разработка параллельной версии программы Jacobi-1D Stencil с использованием технологии OpenMP.

Отчет

студента 328 группы

факультета ВМК МГУ

Медведева Ивана Владимировича

2019 год

Постановка задачи

Имеется одномерная краевая задача для уравнения теплопроводности, которая решается методом Якоби - для каждой ячейки её значение вычисляется как среднее арифметическое ее значения и значения соседей. Имеется последовательная реализация Jacobi-1D Stencil.

Требуется:

1) Разработать параллельную версию программы для задачи Jacobi-1D Stencil с использованием технологии OpenMP,
2) Исследовать масштабируемость полученной программы, построить графики зависимости времени её выполнения от числа используемых нитей и объёма входных данных.

Код программы

```
#include <omp.h>

double bench_t_start, bench_t_end;

static
double rtclock()
{
    struct timeval Tp;
    int stat;
    stat = gettimeofday (&Tp, NULL);
    if (stat != 0)
      printf ("Error return from gettimeofday: %d", stat);
    return (Tp.tv_sec + Tp.tv_usec * 1.0e-6);
}

void bench_timer_start()
{
  bench_t_start = rtclock ();
}

void bench_timer_stop()
{
  bench_t_end = rtclock ();
}

void bench_timer_print()
{
  printf ("Time in seconds = %0.6lf\n", bench_t_end - bench_t_start);
}

static
void init_array (int n,
   float A[n],
   float B[n])
{
  int i;

  for (i = 0; i < n; i++)
      {
 A[i] = ((float) i+ 2) / n;
 B[i] = ((float) i+ 3) / n;
```

```
      }
}

static
void print_array(int n,
        float A[n])
{
  int i;

  fprintf(stderr, "==BEGIN DUMP_ARRAYS==\n");
  fprintf(stderr, "begin dump: %s", "A");
  for (i = 0; i < n; i++)
    {
      if (i % 20 == 0) fprintf(stderr, "\n");
      fprintf(stderr, "%0.2f ", A[i]);
    }
  fprintf(stderr, "\nend   dump: %s\n", "A");
  fprintf(stderr, "==END   DUMP_ARRAYS==\n");
}

static
void kernel_jacobi_1d(int tsteps,
       int n,
       float A[n],
       float B[n])
{
  int t, i;

  for (t = 0; t < tsteps; t++) {
      #pragma omp parallel shared(B) private(i)
      {
          #pragma omp for
          for (i = 1; i < n - 1; i++) {
              B[i] = 0.33333 * (A[i-1] + A[i] + A[i + 1]);
          }
      }
      #pragma omp barrier
      #pragma omp parallel shared(A) private(i)
      {
          #pragma omp for
          for (i = 1; i < n - 1; i++) {
              A[i] = 0.33333 * (B[i-1] + B[i] + B[i + 1]);
          }
      }
      #pragma omp barrier
  }
}

int main(int argc, char** argv)
{
    int procs = omp_get_num_threads();
    printf("%d\n", procs);
    int nes[] =      {30, 120, 400, 2000, 4000, 16000, 32000, 64000, 128000};
    int tstepses[] = {20, 40, 100, 500, 1000, 4000, 8000, 16000, 32000};
    int n, tsteps;
    int i;
    for (i = 0; i < 9; i++) {
        n = nes[i];
        tsteps = tstepses[i];
```

```
        float (*A)[n];
        A = (float(*)[n]) malloc((n) * sizeof(float));
        float (*B)[n];
        B = (float(*)[n]) malloc ((n) * sizeof(float));

        init_array(n, *A, *B);

        printf("%d, %d\n", n, tsteps);
        bench_timer_start();
        kernel_jacobi_1d(tsteps, n, *A, *B);
        bench_timer_stop();
        bench_timer_print();
        print_array(n, *A);

        free((void*)A);
        free((void*)B);
    }
    return 0;
}
```

Распараллелена программа классическими приёмами, рассмотренными на лекции и в учебном пособии Антонова А. С. «ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ OpenMP»

Результаты замеров времени выполнения

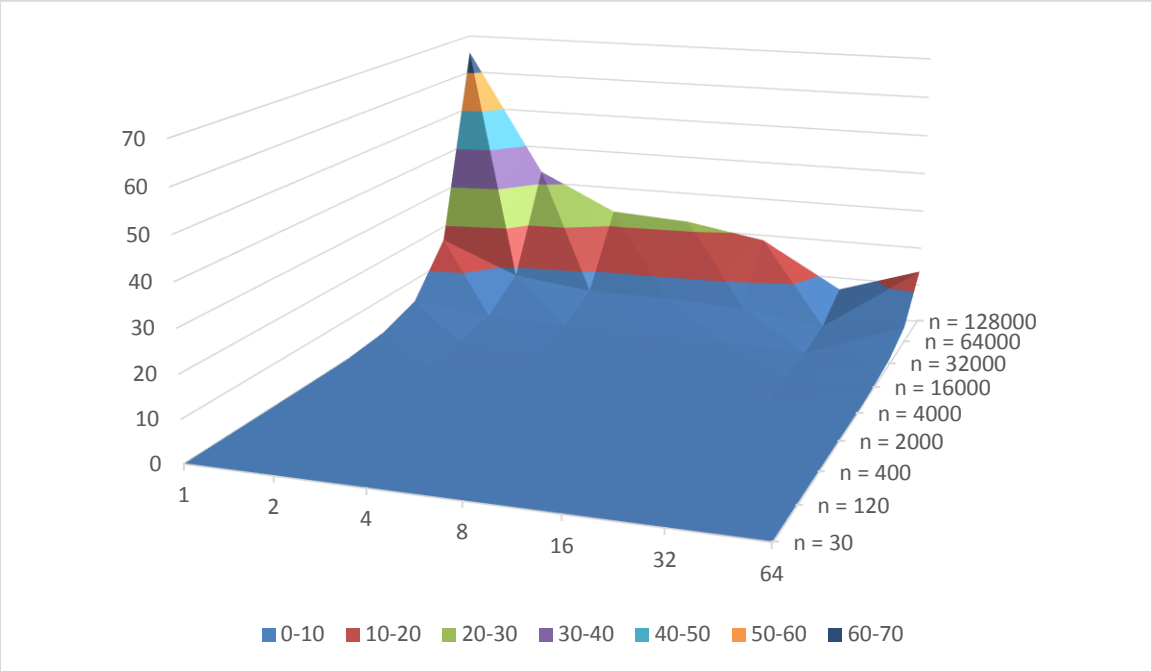
Работа задачи рассмотрена на суперкомпьютере Polus с различным числом нитей (1 - 64) и различными размерами одномерной матрицы (30 — 128000). Каждое измерение проводилось 3 раза. В таблице и на графиках записаны усредненные результаты времени выполнения.

Таблица с результатами

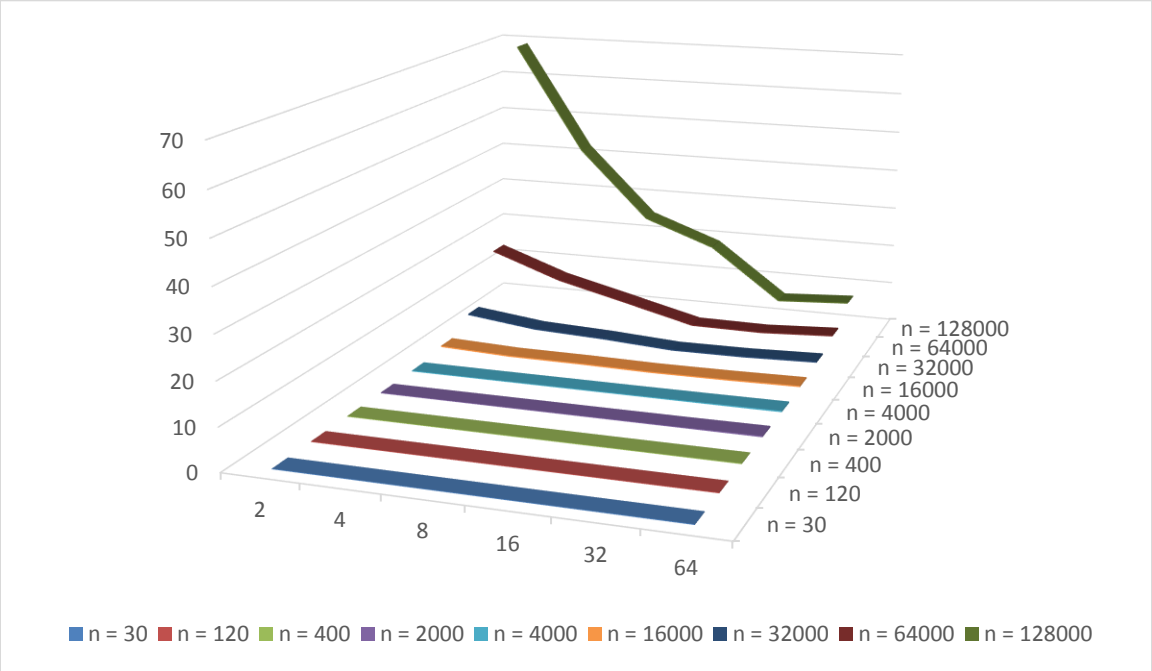
Нити и количество итераций / размер матрицы	n = 30 t = 20	n = 120 t = 40	n = 400 t = 100	n = 2000 t = 500	n = 4000 t = 1000	n = 16000 t = 4000	n = 32000 t = 8000	n = 64000 t = 16000	n = 128000 t = 32000
1	0.000052	0.000124	0.000748	0.016474	0.064761	1.023893	4.086248	16.357408	65.417132
2	0.000125	0.000146	0.000579	0.009855	0.034440	0.520177	2.060155	8.203113	33.054454
4	0.000436	0.000562	0.002838	0.011489	0.033665	0.399453	1.498244	5.795863	23.317382
8	0.000455	0.000568	0.001531	0.009989	0.028635	0.328693	1.466929	5.688870	22.117210
16	0.000699	0.000726	0.003182	0.017688	0.043695	0.386435	1.290843	4.812450	18.628048
32	0.001523	0.001961	0.005095	0.026586	0.053706	0.270972	0.701824	2.062651	6.840621
64	0.005974	0.003737	0.009634	0.049674	0.101793	0.498348	1.302753	3.931723	13.653818

Графики: время выполнения программы в зависимости от размера матрицы и количества ядер Polus

В виде поверхности:



В виде линий в трехмерном пространстве:



Вывод:

Задача хорошо поддается распараллеливанию при помощи технологии OpenMP. Было получено 10-кратное ускорение программы. Как показали эксперименты, для рассмотренных массивов маленького размера (30-2000) выгодно использовать 2-4 нити, а для больших (>32000) – 16-32. При увеличении числа нитей наблюдается снижение производительности, что происходит из-за больших накладных расходов.