



МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.Ломоносова



Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу
«Суперкомпьютеры и параллельная обработка данных»

ЗАДАНИЕ №10

Разработка параллельной версии программы для задачи
Red-Black 3D

ОТЧЕТ

о выполненном задании

студента 320 учебной группы факультета ВМК МГУ

Кириллова Алексея Константиновича

Москва, 2022

Содержание

1	Постановка задачи	2
2	Использованные библиотеки	2
2.1	OpenMP	2
2.2	MPI	3
3	Компиляция и запуск	3
4	Тестирование	4
4.1	Скорость работы OpenMP	4
4.2	Скорость работы MPI	4
5	Выводы	5
6	Приложения	6
6.1	Оригинальный код	6
6.2	OpenMP-версия программы	7
6.3	MPI-версия программы	9
6.4	Скрипт для тестирования	12

1 Постановка задачи

Целью задания была разработка параллельной версии игры Red-Black3D. Код для игры уже был предоставлен и необходимо было написать его параллельную версию. Подзадачи:

- Использовать OpenMP для реализации параллельности программы
- Использовать функционал MPI для распараллеливания
- Провести сравнительный анализ, построить графики зависимости
- Написать отчет

Код находится в приложении к документу, а также доступен на гитхабе по адресу <https://github.com/Alexkkir/red-black-3d-parallel>

2 Используемые библиотеки

Далее будет рассмотрено, с помощью каких методов выполнялось распараллеливание программы

2.1 OpenMP

Фреймворк OpenMP дает возможность выполнять распараллеливания программы с помощью очень удобного интерфейса. Места кода, которые могут быть ускорены, помечаются с помощью директивы компилятора

```
#pragma omp директива [клаузы]
```

Это позволяет распараллелить код, не внося в него существенных изменений. Рассмотрим, какие функции использовались:

- `#pragma omp parallel` - самая главная директива. В месте кода, где она используется, создается n процессов
- `#pragma omp for` - директива, позволяющая распараллелить цикл `for`. Есть несколько способов распараллелить цикл:
 - `schedule(static)` - цикл бьется на n равных диапазонов
 - `schedule(dynamic, n)` - цикл бьется на небольшие диапазоны (размера n), они обрабатываются в порядке "живой очереди"
 - `schedule(guided, n)` - похоже на `static`, но немного другой принцип разбиения
 - `schedule(runtime, n)` - способ разбиения определяется в аргументах командной строки
 - `schedule(auto)` - на этапе компиляции выбирается оптимальный способ разбиения; используется по умолчанию
- `#pragma omp barrier` - барьер; нити останавливаются у барьера, пока не соберутся все
- `shared` - позволяет разделить переменную между потоками; по умолчанию все переменные являются `shared`

- `private` - делают копию переменной в каждом потоке
- `collapse(n)` - если есть вложенный цикл глубины n , то этот цикл выпрямляется в один длинный
- `reduction(op: var)` - создает в каждом потоке `private` копию переменной `var` и когда параллельный блок заканчивает редуцирует все переменные к одной, используя операцию `op`

2.2 MPI

Перечислим, какие использовались функции из библиотеки MPI

- `MPI_Init` - создает процессы
- `MPI_Comm_rank` - определяет число созданных процессов
- `MPI_Comm_size` - определяет номер данного процесса
- `MPI_Barrier` - барьер
- `MPI_Send` - стандартная блокирующая передача. Ждет, пока принимающий процесс прочитает
- `MPI_Isend` - стандартная неблокирующая передача. Не ждет завершения обмена, лишь инициализируя ее. Работа программы ускоряется за счет того, что копирование сообщение в буфер и вычисления могут происходить одновременно.
- `MPI_Recv` - стандартные блокирующий прием
- `MPI_Irecv` - стандартные неблокирующий прием
- `MPI_BSend` - передача с буферизацией. Источник копирует сообщение в буфер и передает его в неблокирующем режиме
- `MPI_Rsend` - передача по готовности. Выкидывает сообщение в коммуникация, не дожидаясь установления соединения
- `MPI_Ssend` - синхронная передача. Завершение передачи происходит сразу после того, как прием сообщения инициализирован другим процессом
- `MPI_Reduce` - передает переменную заданному процессу и выполняет редукцию
- `MPI_Allreduce` - как предыдущее, но результат записывается в каждый процесс

3 Компиляция и запуск

Для копирования исходника на сервер и его компиляции были написаны две длинные команды. Для OpenMP:

```
scp redb_3d_omp.c skipod:. && ssh skipod
gcc ./redb_3d_omp.c -o redb_3d_omp -Ofast -fopenmp -Wall -Werror \
    && time OMP_NUM_THREADS=16 ./redb_3d_omp
```

Для MPI:

```

scp redb_3d_mpi.c skipod:. && ssh skipod
module load SpectrumMPI \
    && mpicc redb_3d_mpi.c -o redb_3d_mpi -Wall -Werror \
    && mpirun -np 2 redb_3d_mpi

```

4 Тестирование

Для тестирования программа запускалась с разными количеством потоков и разными размерами матрицы. В результате были построены трехмерные графики, иллюстрирующие скорость работы программы. Тестирование проводилось на сервере Polus. Для замера скорости был написан скрипт (см. приложение), которые запускает скомпилированную программу 3 раза с разным числом потоков и замеряет среднее время выполнения. Также скрипт проверяет, что программа выводит каждый раз одно и то же.

4.1 Скорость работы OpenMP

На графике представлена зависимость скорости работы алгоритма, написанного с использованием библиотеки OpenMP



Можно сделать наблюдение, что с ростом числа потоков увеличивается скорость работы. Однако когда процессов слишком много, накладные расходы существенно замедляют время работы

4.2 Скорость работы MPI

На графике представлена зависимость скорости работы алгоритма, написанного с использованием MPI



Выводы аналогичны предыдущим. Из отличий, время работы получилось в 10 раз хуже, чем для OpenMP. Также при увеличении числа потоков накладные расходы гораздо сильнее сказываются на производительности, чем в случае OpenMP.

5 Выводы

Оба фреймворка OpenMP и MPI позволяют ускорить программу. Однако первый из них позволяет сделать это с меньшими изменениями кода и при этом большим ускорением. С увеличением числа потоков возрастает производительность, однако при слишком большом числе потоков замедление из-за накладных расходов компенсирует полученное ускорение.

6 Приложения

6.1 Оригинальный код

Ниже представлен оригинальный код программы:

```
1 #include <math.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #define Max(a, b) ((a) > (b) ? (a) : (b))
5
6 #define N (2 * 2 * 2 * 2 * 2 * 2 * 2 + 2)
7 double maxeps = 0.1e-7;
8 int itmax = 100;
9 int i, j, k;
10 double w = 0.5;
11 double eps;
12
13 double A[N][N][N];
14
15 void relax();
16 void init();
17 void verify();
18
19 int main(int an, char **as)
20 {
21     int it;
22
23     init();
24
25     for (it = 1; it <= itmax; it++)
26     {
27         eps = 0.;
28         relax();
29         printf("it=%4i    eps=%f\n", it, eps);
30         if (eps < maxeps)
31             break;
32     }
33
34     verify();
35
36     return 0;
37 }
38
39 void init()
40 {
41     for (k = 0; k <= N - 1; k++)
42         for (j = 0; j <= N - 1; j++)
43             for (i = 0; i <= N - 1; i++)
44             {
45                 if (i == 0 || i == N - 1 || j == 0 || j == N - 1 || k == 0 || k == N
46                     - 1)
47                     A[i][j][k] = 0.;
48                 else
49                     A[i][j][k] = (4. + i + j + k);
50             }
51 }
52
53 void relax()
54 {
```

```

55 for (k = 1; k <= N - 2; k++)
56     for (j = 1; j <= N - 2; j++)
57         for (i = 1 + (k + j) % 2; i <= N - 2; i += 2)
58             {
59                 double b;
60                 b = w * ((A[i - 1][j][k] + A[i + 1][j][k] + A[i][j - 1][k] + A[i][j
+ 1][k] + A[i][j][k - 1] + A[i][j][k + 1]) / 6. - A[i][j][k]);
61                 eps = Max(fabs(b), eps);
62                 A[i][j][k] = A[i][j][k] + b;
63             }
64
65 for (k = 1; k <= N - 2; k++)
66     for (j = 1; j <= N - 2; j++)
67         for (i = 1 + (k + j + 1) % 2; i <= N - 2; i += 2)
68             {
69                 double b;
70                 b = w * ((A[i - 1][j][k] + A[i + 1][j][k] + A[i][j - 1][k] + A[i][j
+ 1][k] + A[i][j][k - 1] + A[i][j][k + 1]) / 6. - A[i][j][k]);
71                 A[i][j][k] = A[i][j][k] + b;
72             }
73 }
74
75 void verify()
76 {
77     double s;
78
79     s = 0.;
80     for (k = 0; k <= N - 1; k++)
81         for (j = 0; j <= N - 1; j++)
82             for (i = 0; i <= N - 1; i++)
83                 {
84                     s = s + A[i][j][k] * (i + 1) * (j + 1) * (k + 1) / (N * N * N);
85                 }
86     printf("    S = %f\n", s);
87 }

```

6.2 OpenMP-версия программы

Ниже представлена реализация кода программы с использованием OpenMP

```

1 #include <math.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <omp.h>
5 #define Max(a, b) ((a) > (b) ? (a) : (b))
6
7 #define N (2 * 2 * 2 * 2 * 2 * 2 * 2 + 2)
8 double maxeps = 0.1e-7;
9 int itmax = 100;
10 int i, j, k;
11 double w = 0.5;
12 double eps;
13
14 double A[N][N][N];
15
16 void relax();
17 void init();
18 void verify();
19
20 int main(int an, char **as)

```



```

21 {
22     int it;
23     printf("num_threads: %d, max: %d\n", omp_get_num_threads(),
24           omp_get_max_threads());
25     #pragma omp parallel shared(A) // default is shared
26     init();
27     for (it = 1; it <= itmax; it++)
28     {
29         eps = 0.;
30         #pragma omp parallel private(i, j, k) shared(A, eps)
31         relax();
32         printf("it=%4i    eps=%f\n", it, eps);
33         if (eps < maxeps)
34             break;
35     }
36
37     verify();
38
39     return 0;
40 }
41
42 void init()
43 {
44     #pragma omp for private(i, j, k) schedule(static) collapse(3)
45     for (i = 0; i <= N - 1; i++)
46         for (j = 0; j <= N - 1; j++)
47             for (k = 0; k <= N - 1; k++)
48             {
49                 if (i == 0 || i == N - 1 || j == 0 || j == N - 1 || k == 0
50                     || k == N - 1)
51                     A[i][j][k] = 0.;
52                 else
53                     A[i][j][k] = (4. + i + j + k);
54             }
55 }
56
57 void relax()
58 {
59     #pragma omp for schedule(static) collapse(2) reduction(max \
60                                     : eps)
61     for (i = 1; i <= N - 2; i++)
62         for (j = 1; j <= N - 2; j++)
63             for (k = 1 + (i + j) % 2; k <= N - 2; k += 2)
64             {
65                 double b;
66                 b = w * ((A[i - 1][j][k] + A[i + 1][j][k] + A[i][j - 1][k] +
67                     A[i][j + 1][k] + A[i][j][k - 1] + A[i][j][k + 1]) / 6. - A[i][j][k]);
68                 eps = Max(fabs(b), eps);
69                 A[i][j][k] = A[i][j][k] + b;
70             }
71
72     #pragma omp barrier
73
74     #pragma omp for schedule(static) collapse(2)
75     for (i = 1; i <= N - 2; i++)
76         for (j = 1; j <= N - 2; j++)
77             for (k = 1 + (i + j + 1) % 2; k <= N - 2; k += 2)
78             {
79                 double b;

```

```

78         b = w * ((A[i - 1][j][k] + A[i + 1][j][k] + A[i][j - 1][k] +
79         A[i][j + 1][k] + A[i][j][k - 1] + A[i][j][k + 1]) / 6. - A[i][j][k]);
80         A[i][j][k] = A[i][j][k] + b;
81     }
82 }
83 void verify()
84 {
85     double s;
86
87     s = 0.;
88 #pragma omp parallel for private(i, j, k) shared(A) schedule(static)
89     collapse(3) reduction(+ \
90         : s)
91     for (i = 0; i <= N - 1; i++)
92         for (j = 0; j <= N - 1; j++)
93             for (k = 0; k <= N - 1; k++)
94             {
95                 s = s + A[i][j][k] * (i + 1) * (j + 1) * (k + 1) / (N * N *
96                 N);
97             }
98     printf("    S = %f\n", s);
99 }

```

6.3 MPI-версия программы

Ниже представлена параллельная версия кода, написанная с использованием MPI

```

1 #include <math.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <mpi.h>
5 #define Max(a, b) ((a) > (b) ? (a) : (b))
6
7 #define N (2 * 2 * 2 * 2 * 2 * 2 * 2 + 2)
8 #define REAL_INDEX(i_aux, startrow) (i_aux + startrow)
9 double maxeps = 0.1e-7;
10 int itmax = 100;
11 int i, j, k;
12 double w = 0.5;
13 double eps;
14
15 // double A[N][N][N];
16 typedef double type_array_2d[N][N];
17 type_array_2d *A; // we divide tensor A on n_procs parts and store each part
18 // in corresponding process
19
20 void relax();
21 void init();
22 void verify();
23
24 MPI_Request req[4];
25 int myrank, ranksize;
26 int startrow, lastrow, nrows;
27 MPI_Status status[4];
28
29 int ll, shift;
30
31 int main(int argc, char **argv)

```

```

31 {
32     MPI_Init(&argc, &argv);           /* initialize MPI system */
33     MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* my place in MPI system */
34     MPI_Comm_size(MPI_COMM_WORLD, &ranksz); /* size of MPI system */
35     MPI_Barrier(MPI_COMM_WORLD);       /* wait until all processes will be
        created */
36
37     /* rows of matrix A to be processed */
38     startrow = (myrank * (N - 2)) / ranksz; // we divide rows of matrix A
        between processes. Indexation starts from real row of matrix A
39     lastrow = (((myrank + 1) * (N - 2)) / ranksz) - 1;
40     nrow = lastrow - startrow + 1;
41
42     A = malloc((nrow + 2) * sizeof(type_array_2d)); // store matrix + upper
        and lower aux borders + left and right borders
43
44     int it;
45     init();
46
47     for (it = 1; it <= itmax; it++) /* main loop */
48     {
49         eps = 0.;
50         relax();
51         if (myrank == 0)
52             printf("it=%4i    eps=%f\n", it, eps);
53         if (eps < maxeps)
54             break;
55     }
56
57     verify();
58
59     MPI_Finalize();
60     return 0;
61 }
62
63 void init()
64 {
65     for (int i_aux = 0; i_aux < nrow + 2; i_aux++)
66         for (j = 0; j <= N - 1; j++)
67             for (k = 0; k <= N - 1; k++)
68             {
69                 i = REAL_INDEX(i_aux, startrow); // real index
70                 if (i == 0 || i == N - 1 || j == 0 || j == N - 1 || k == 0 || k == N
                    - 1)
71                     A[i_aux][j][k] = 0.;
72                 else
73                     A[i_aux][j][k] = (4. + i + j + k);
74             }
75 }
76
77 void relax()
78 {
79     double local_eps = eps;
80     for (i = 1; i <= nrow; i++) /* red cells */
81         for (j = 1; j <= N - 2; j++)
82             for (k = 1 + (REAL_INDEX(i, startrow) + j) % 2; k <= N - 2; k += 2)
83             {
84                 double b;
85                 b = w * ((A[i - 1][j][k] + A[i + 1][j][k] + A[i][j - 1][k] + A[i][j
                    + 1][k] + A[i][j][k - 1] + A[i][j][k + 1]) / 6. - A[i][j][k]);

```

```

86         local_eps = Max(fabs(b), local_eps);
87         A[i][j][k] = A[i][j][k] + b;
88     }
89
90     if (myrank != 0) /* exchanging borders */
91         MPI_Irecv(&A[0], N * N, MPI_DOUBLE, myrank - 1, 1215, MPI_COMM_WORLD, &
92         req[0]);
93     if (myrank != ranksize - 1)
94         MPI_Isend(&A[nrows], N * N, MPI_DOUBLE, myrank + 1, 1215, MPI_COMM_WORLD
95         , &req[2]);
96     if (myrank != ranksize - 1)
97         MPI_Irecv(&A[nrows + 1], N * N, MPI_DOUBLE, myrank + 1, 1216,
98         MPI_COMM_WORLD, &req[3]);
99     if (myrank != 0)
100         MPI_Isend(&A[1], N * N, MPI_DOUBLE, myrank - 1, 1216, MPI_COMM_WORLD, &
101         req[1]);
102
103     ll = 4, shift = 0;
104     if (myrank == 0)
105     {
106         ll -= 2;
107         shift = 2;
108     }
109     if (myrank == ranksize - 1)
110     {
111         ll -= 2;
112     }
113
114     MPI_Waitall(ll, &req[shift], &status[0]); /* waiting until all swaps will
115     be done */
116
117     for (int i = 1; i <= nrows; i++) /* black cells */
118         for (j = 1; j <= N - 2; j++)
119             for (k = 1 + (REAL_INDEX(i, startrow) + j + 1) % 2; k <= N - 2; k +=
120             2)
121             {
122                 double b;
123                 b = w * ((A[i - 1][j][k] + A[i + 1][j][k] + A[i][j - 1][k] + A[i][j
124                 + 1][k] + A[i][j][k - 1] + A[i][j][k + 1]) / 6. - A[i][j][k]);
125                 A[i][j][k] = A[i][j][k] + b;
126             }
127
128     if (myrank != 0) /* exchanging borders */
129         MPI_Irecv(&A[0], N * N, MPI_DOUBLE, myrank - 1, 1215, MPI_COMM_WORLD, &
130         req[0]);
131     if (myrank != ranksize - 1)
132         MPI_Isend(&A[nrows], N * N, MPI_DOUBLE, myrank + 1, 1215, MPI_COMM_WORLD
133         , &req[2]);
134     if (myrank != ranksize - 1)
135         MPI_Irecv(&A[nrows + 1], N * N, MPI_DOUBLE, myrank + 1, 1216,
136         MPI_COMM_WORLD, &req[3]);
137     if (myrank != 0)
138         MPI_Isend(&A[1], N * N, MPI_DOUBLE, myrank - 1, 1216, MPI_COMM_WORLD, &
139         req[1]);
140
141     ll = 4, shift = 0;
142     if (myrank == 0)
143     {
144         ll -= 2;
145         shift = 2;
146     }

```

```

135 }
136 if (myrank == ranksize - 1)
137 {
138     ll -= 2;
139 }
140
141 MPI_Waitall(ll, &req[shift], &status[0]); /* waiting until all swaps will
    be done */
142
143 MPI_Allreduce(&local_eps, &eps, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
    /* updating global eps */
144 }
145
146 void verify()
147 {
148     double s = 0, local_s = 0;
149
150     for (i = 1; i <= nrows; i++)
151         for (k = 0; k <= N - 1; k++)
152             for (j = 0; j <= N - 1; j++)
153                 {
154                     local_s += A[i][j][k] * (REAL_INDEX(i, startrow) + 1) * (j + 1) * (k
    + 1) / (N * N * N);
155                 }
156
157 MPI_Allreduce(&local_s, &s, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD); /*
    updating global eps */
158 if (myrank == 0)
159     printf("    S = %f\n", s);
160 }

```

6.4 Скрипт для тестирования

Скрипт, с помощью которого запускался код:

```

1 import time
2 import os, sys
3
4 file = sys.argv[1]
5 framework = sys.argv[2]
6 assert framework in ['openmp', 'mpi']
7 threads_space = [1, 2, 4, 8, 16, 32, 64, 128]
8 results = []
9 out_vals = []
10 tmp_file = 'tmp.txt'
11
12 n_runs = 3
13
14
15
16 for n_threads in threads_space:
17     out_vals.append([])
18
19     start = time.time()
20     for _ in range(n_runs):
21         if framework == 'openmp':
22             os.system(f"OMP_NUM_THREADS={n_threads} ./ {file} > {tmp_file}")
23         else:
24             os.system(f"mpirun -np {n_threads} ./ {file} > {tmp_file}")
25     with open(tmp_file) as f:

```

```
26         out = f.readlines()[-1].split()[-1]
27         out_vals[-1].append(out)
28     end = time.time()
29
30     t = (end - start) / n_runs
31     results.append(t)
32     print(n_threads, t)
33
34 print(threads_space)
35 print(results)
36 print(out_vals)
```