Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

09.03.01 Информатика и вычислительная техника
код и наименование направления подготовки

ОТЧЕТ
по учебной практике

по направлению 09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) – «Электронно-вычислительные машины, комплексы, системы и сети», квалификация – бакалавр,
программа академического бакалавриата,
форма обучения – очная, год начала подготовки (по учебному плану) – 2016

Выполнил:
студент гр. ИВ-621
«06» июля 2018 г. _____ /Дьяченко Д.В./

Оценка «_____»

Руководитель практики
от университета
д.т.н., доцент Кафедры ВС
«07» июля 2018 г. _____ /Родионов А.С./

Новосибирск 2018

ПЛАН-ГРАФИК ПРОВЕДЕНИЯ ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ

Тип практики: учебная практика по получению первичных профессиональных
Способ проведения практики: стационарная
Форма проведения практики: дискретно по периодам проведения практики

Тема: Разработка алгоритма ленточного перемножения матриц с оптимизацией кэша.

Содержание практики

Наименование видов деятельности	Дата (начало – окончание)
Общее ознакомление со структурным подразделением СибГУТИ, вводный инструктаж по технике безопасности	05.02.18-17.02.18
Выдача задания на практику, определение конкретной индивидуальной темы, формирование плана работ	19.02.18-24.02.18
Работа с библиотечными фондами структурного подразделения или предприятия, сбор и анализ материалов по теме практики	26.02.18-24.03.18
Выполнение работ в соответствии с составленным планом	26.03.18-02.06.18
Анализ полученных результатов и произведенной работы, составление отчета по практике	02.07.18-07.07.18

Согласовано:

Руководитель практики
от университета
д.т.н., доцент Кафедры ВС _____ /Родионов А.С./

ЗАДАНИЕ НА ПРАКТИКУ

Разработать алгоритм ленточного перемножения матриц с оптимизацией кэша с использованием библиотеки MPI.

ВВЕДЕНИЕ

1. Описание аппаратного и программного обеспечения.

Исследование было произведено на вычислительном кластере Jet, укомплектованного 18 узлами.

Таблица 1 Конфигурация вычислительного узла

Системная плата	Intel S5000VSA (Серверная платформа Intel SR2520SAF)
Процессор	2 x Intel Xeon E5420 (2,5 GHz; Intel-64)
Оперативная память	8 GB (4 x 2GB PC-5300)
Жесткий диск	SATAII 500GB (Seagate 500Gb Barracuda)
Сетевая карта	2 x Intel Gigabit Ethernet (Integrated Intel PRO/1000 EB, 80003ES2LAN Gigabit Ethernet Controller) 1 x Intel PRO/1000 MT Server Adapter (PWLA8490MT, 82572EI Gigabit Ethernet Controller)
Корпус	Rack mount 2U

Схема исследования:

N – количество элементов массива (каждый массив состоит из NxN элементов).

NumNodes – количество задействованных узлов.

NumCores – количество задействованных ядер процессора.

Таблица 2 Схема исследования

NumNodes	NumCores	N	NumNodes	NumCores	N	NumNodes	NumCores	N
1	1	1024	1	1	2048	1	1	4096
	2	1024		2	2048		2	4096
	4	1024		4	2048		4	4096
	6	1024		6	2048		6	4096
	8	1024		8	2048		8	4096
2	1	1024	2	1	2048	2	1	4096
	2	1024		2	2048		2	4096
	4	1024		4	2048		4	4096
	6	1024		6	2048		6	4096
	8	1024		8	2048		8	4096
4	1	1024	4	1	2048	4	1	4096
	2	1024		2	2048		2	4096
	4	1024		4	2048		4	4096
	6	1024		6	2048		6	4096
	8	1024		8	2048		8	4096
8	1	1024	8	1	2048	8	1	4096
	2	1024		2	2048		2	4096
	4	1024		4	2048		4	4096
	6	1024		6	2048		6	4096
	8	1024		8	2048		8	4096

Программа была написана на языке программирования Си, с использованием компилятора mpicc. Время замерялось при помощи функции wtime из библиотеки sys/time.h.

```
double wtime()
{
   struct timeval t;
   gettimeofday(&t, NULL);
   return (double)t.tv_sec + (double)t.tv_usec * 1E-6;
}
```

2. Краткое описание алгоритма

1) Инициализация данных для обработки функцией init_matrix (массивов A[N*N], B[N*N], C[N*N]). Для эксперимента использовались статичные данные.

```
1.  void init_matrix(double *a, double *b, double *c, int n)
2.  {
3.      int i, j, k;
4.      for (i = 0; i < n; i++) {
5.          for (j = 0; j < n; j++) {
6.              for (k = 0; k < n; k++) {
7.                  *(a + i * n + j) = 1.0;
8.                  *(b + i * n + j) = 2.0;
9.                  *(c + i * n + j) = 0.0;
10.             }
11.         }
12.     }
13. }
```

2) Нахождение номера начала и конца строки, необходимого для обработки, основываясь на количестве процессов и номере процесса, выполняемого код в данный момент.

```
1.  int rank;
2.  int size;
3.  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
4.  MPI_Comm_size(MPI_COMM_WORLD, &size);
5.  int st = (N / size) * rank;
6.  int fn = ((rank + 1) == size) ? N : (st + (N / size));
```

3) Выполнение умножения матриц функцией dgemm_parallel (цикл ikj).

```
1.  void dgemm_parallel(double *a, double *b, double *c, int n, int st, int fn)
2.  {
3.      int i, j, k;
4.
5.      for (i = st; i < fn; i++) {
6.          for (k = 0; k < n; k++) {
7.              for (j = 0; j < n; j++) {
8.                  *(c + i * n + j) += *(a + i * n + k) * *(b + k * n + j);
9.              }
10.         }
11.     }
12. }
```

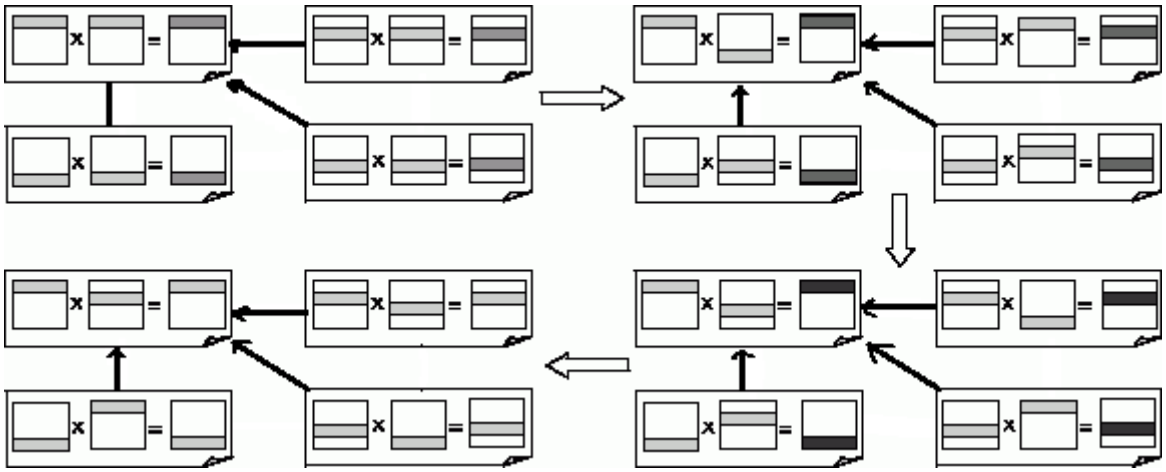
4) Отправка подсчитанных строк массива в главный узел для объединения данных остальными процессами и принятие этим процессом данных.

```
if (rank != 0) {
        MPI_Send(&t, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }
    if (rank == 0) {
        for (int i = 1; i < size; i++) {
            MPI_Recv(&t_all[i], 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD,
&status);
        }
}
```

5) Завершение, очистка памяти

```
MPI_Finalize();
free(A);
free(B);
free(C);
```

Рисунок 1 Общая схема алгоритма перемножения матриц



Результаты исследования

N – количество элементов массива (каждый массив состоит из NxN элементов).

NumCores – количество задействованных ядер процессора.

Time – время выполнения

Speedup – коэффициент ускорения, рассчитываемый по формуле $S_p(n) = \frac{T_1(n)}{T_p(n)}$, где p – количество процессов.

1. При одном узле

Таблица 3 Таблица времени и ускорения на одном узле для N=1024

N	1024				
NumCores	1	2	4	6	8
Time	10.738666	5.371339	2.814382	1.848182	1.477680
Speedup	1	1.999253073	3.815639099	5.810394214	7.267247307

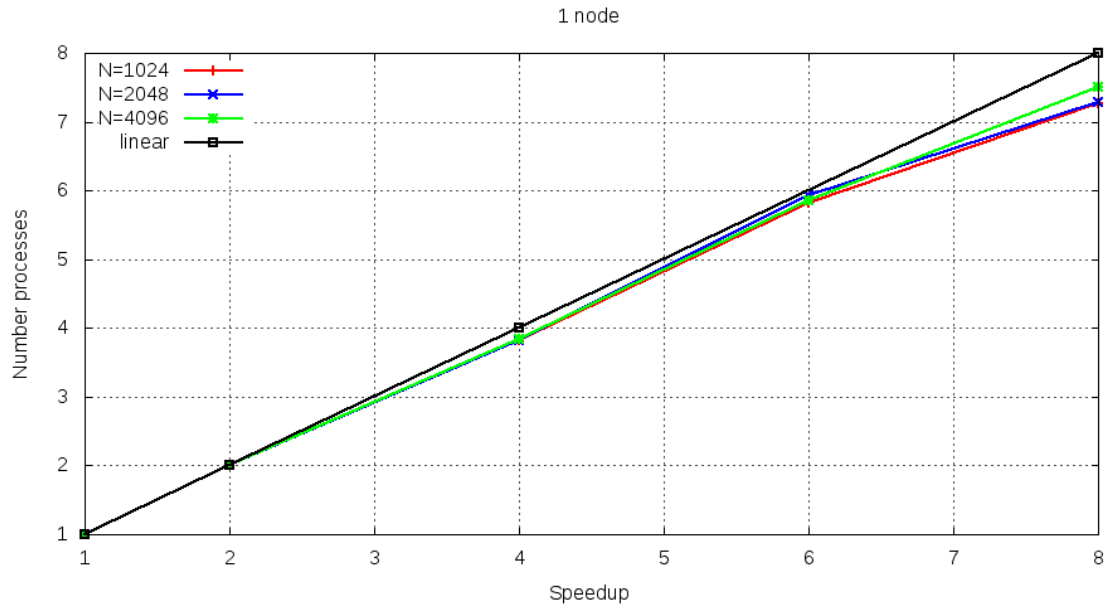
Таблица 4 Таблица времени и ускорения на одном узле для N=2048

N	2048				
NumCores	1	2	4	6	8
Time	86.507901	43.158811	22.622286	14.599645	11.875154
Speedup	1	2.00440881	3.824012348	5.925342774	7.284781402

Таблица 5 Таблица времени и ускорения на одном узле для N=4096

N	4096				
NumCores	1	2	4	6	8
Time	696.882661	347.676537	181.444636	118.917351	92.918897
Speedup	1	2.004399454	3.840745455	5.860226915	7.499902426

Рисунок 2 График зависимости коэффициента ускорения от числа процессоров



2. При двух узлах

Таблица 6 Таблица времени и ускорения на двух узлах для N=1024

N	1024				
NumCores	1	2	4	6	8
Time	10.738666	5.371339	2.814382	1.848182	1.477680
Speedup	1	1.999253073	3.815639099	5.810394214	7.267247307

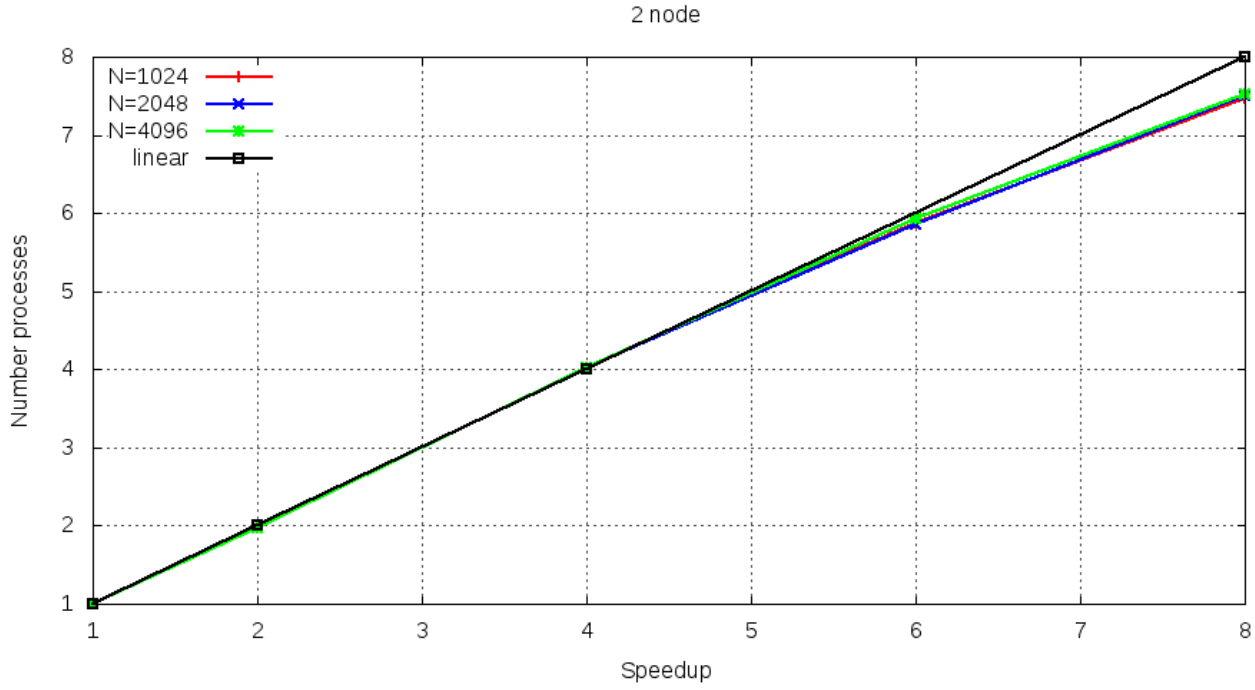
Таблица 7 Таблица времени и ускорения на двух узлах для N=2048

N	2048				
NumCores	1	2	4	6	8
Time	86.507901	43.158811	22.622286	14.599645	11.875154
Speedup	1	2.00440881	3.824012348	5.925342774	7.284781402

Таблица 8 Таблица времени и ускорения на двух узлах для N=4096

N	4096				
NumCores	1	2	4	6	8
Time	696.882661	347.676537	181.444636	118.917351	92.918897
Speedup	1	2.004399454	3.840745455	5.860226915	7.499902426

Рисунок 3 График зависимости коэффициента ускорения от числа процессоров



3. При четырех узлах

Таблица 9 Таблица времени и ускорения на двух узлах для N=1024

N	1024				
NumCores	1	2	4	6	8
Time	2.841816	1.442992	0.670140	0.607875	0.425006
Speedup	1	1.969391376	4.240630316	4.675000617	6.686531484

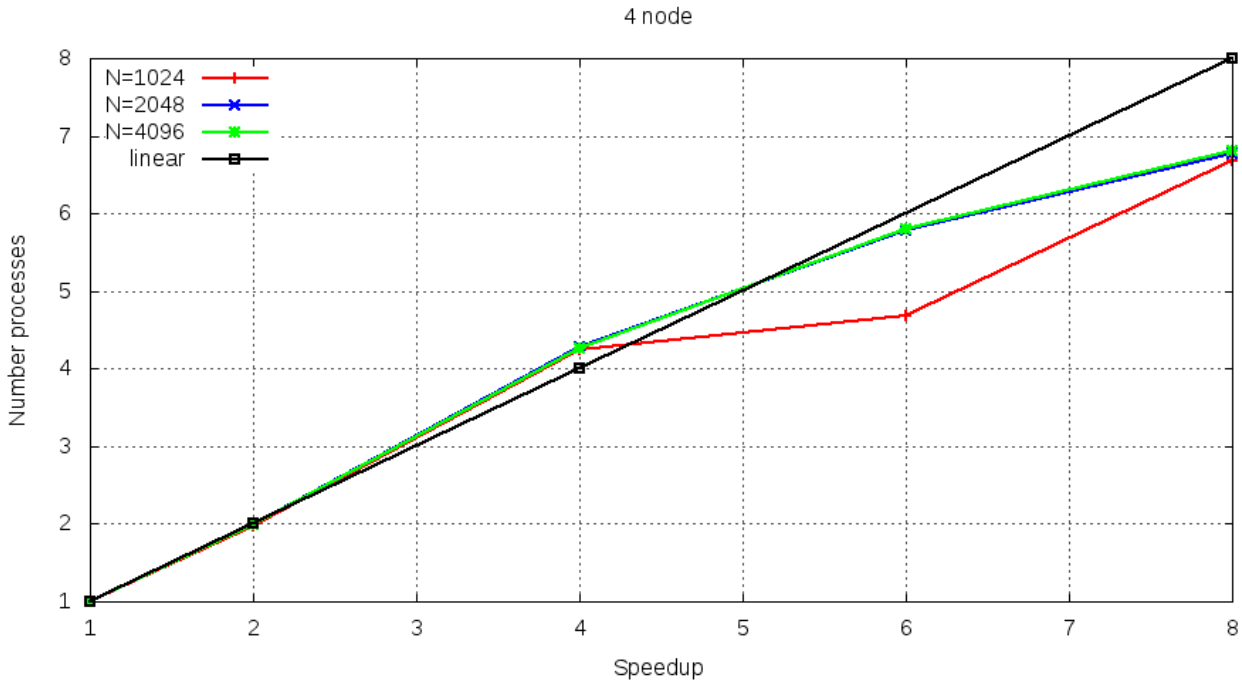
Таблица 10 Таблица времени и ускорения на двух узлах для N=2048

N	2048				
NumCores	1	2	4	6	8
Time	22.981416	11.552278	5.374940	3.976240	3.389849
Speedup	1	1.98934063	4.275660007	5.779685331	6.779578736

Таблица 11 Таблица времени и ускорения на двух узлах для N=4096

N	4096				
NumCores	1	2	4	6	8
Time	184.620054	92.616670	43.326093	31.847790	27.112465
Speedup	1	1.993378233	4.261174761	5.796950244	6.809416038

Рисунок 4 График зависимости коэффициента ускорения от числа процессоров



4. При восьми узлах

Таблица 12 Таблица времени и ускорения на восьми узлах для N=1024

N	1024				
NumCores	1	2	4	6	8
Time	10.738666	5.371339	2.814382	1.848182	1.477680
Speedup	1	1.999253073	3.815639099	5.810394214	7.267247307

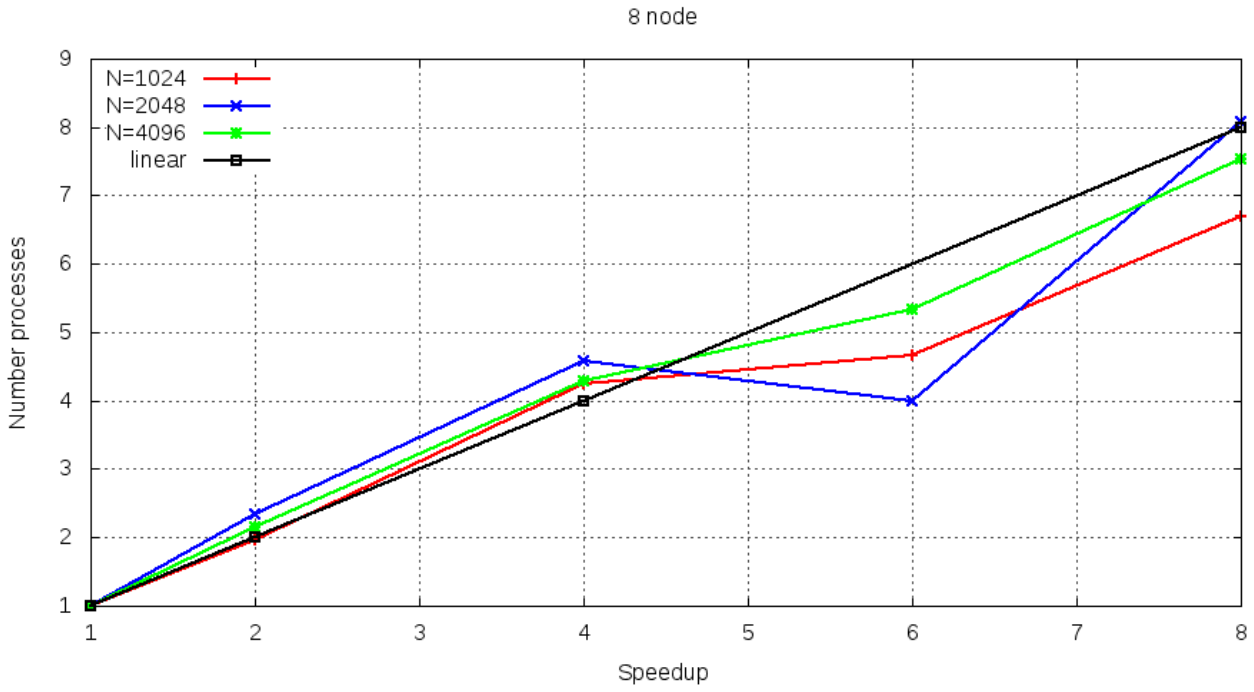
Таблица 13 Таблица времени и ускорения на восьми узлах для N=2048

N	2048				
NumCores	1	2	4	6	8
Time	86.507901	43.158811	22.622286	14.599645	11.875154
Speedup	1	2.00440881	3.824012348	5.925342774	7.284781402

Таблица 14 Таблица времени и ускорения на восьми узлах для N=4096

N	4096				
NumCores	1	2	4	6	8
Time	696.882661	347.676537	181.444636	118.917351	92.918897
Speedup	1	2.004399454	3.840745455	5.860226915	7.499902426

Рисунок 5 График зависимости коэффициента ускорения от числа процессоров



ЗАКЛЮЧЕНИЕ

Результаты показывают хорошую эффективность распараллеливания, близкую к линейной на количестве узлов 1 и 2, но на 4 и 8 с 4 активными ядрами графики показывают суперскалярное ускорение. Такая ситуация возникает при распределении 1:1, то есть один процесс на один узел, что дает увеличение объема, используемого одним процессом кэша. Кэши второго и третьего уровней инклюзивные, а в ситуации, когда используется лишь одно ядро, происходит меньше кэш-промахов, так как большую часть кэшей занимают данные активного ядра. Обратная ситуация при 4 и более активных процессах на количестве узлов 4 и 8. Дополнительные задержки появляются в результате затрат на создание нового процесса и увеличенном количестве пересылок данных между узлами по довольно медленному интерфейсу.

Данный алгоритм имеет зависимость от подсистемы памяти, поэтому, чтобы минимизировать задержки здесь применяется параллельная инициализация данных. Кэш процессоров «прогретый», то есть уже частично имеет данные необходимые для дальнейшей обработки. Так же умножение происходит типом строка-строка, а не строка-столбец, что дает эффективнее использовать кэш. А благодаря библиотеке MPI данный алгоритм имеет хорошую масштабируемость, которую можно проследить по таблицам и графикам.

Даже такая, довольно нетривиальная задача, показывает производительность, гибкость и масштабируемость данной библиотеки.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Курносов М.Г. К93 Введение в структуры и алгоритмы обработки данных. – Новосибирск: Автограф, 2015. – 179 с. ISBN 978-5-9906983-4-5

ПРИЛОЖЕНИЯ

1. Листинг

```
#include <stdio.h>
#include <mpi.h>
#include <time.h>
#include <sys/time.h>

enum {
    N = 1024, /* 2048, 4096 */
    NREPS = 5
};

double A[N * N], B[N * N], C[N * N];

void dgemm_parallel(double *a, double *b, double *c, int n, int st, int fn)
{
    int i, j, k;

    for (i = st; i < fn; i++) {
        for (k = 0; k < n; k++) {
            for (j = 0; j < n; j++) {
                *(c + i * n + j) += *(a + i * n + k) * *(b + k * n + j);
            }
        }
    }
}

void init_matrix(double *a, double *b, double *c, int n)
{
    int i, j, k;

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            for (k = 0; k < n; k++) {
                *(a + i * n + j) = 1.0;
                *(b + i * n + j) = 2.0;
                *(c + i * n + j) = 0.0;
            }
        }
    }
}

void printMatrix(double *a, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%.2lf ", *(a + i * n + j));
        }
```

```
47.             printf("\n");
48.         }
49.         printf("\n");
50.     }
51.
52.     double wtime()
53.     {
54.         struct timeval t;
55.         gettimeofday(&t, NULL);
56.         return (double)t.tv_sec + (double)t.tv_usec * 1E-6;
57.     }
58.
59.     int main(int argc, char *argv[])
60.     {
61.         MPI_Init(&argc, &argv);
62.
63.         double t;
64.
65.         int rank;
66.         int size;
67.
68.         MPI_Comm_rank(MPI_COMM_WORLD, &rank);
69.         MPI_Comm_size(MPI_COMM_WORLD, &size);
70.
71.         double t_all[size];
72.
73.         init_matrix(A, B, C, N);
74.
75.         int st = (N / size) * rank;
76.         int fn = ((rank + 1) == size) ? N : (st + (N / size));
77.
78.         t -= wtime();
79.
80.         for (int i = 0; i < NREPS; i++) {
81.             dgemm_parallel(A, B, C, N, st, fn);
82.         }
83.
84.         t += wtime();
85.         t /= NREPS;
86.
87.         MPI_Barrier(MPI_COMM_WORLD);
88.         if (rank != 0) {
89.             MPI_Send(C + (st * N), (fn - st) * N, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
90.         }
91.         MPI_Status status;
92.         if (rank == 0) {
93.             for (int i = 1; i < size; i++) {
94.                 int st_loc = (N / size) * i;
95.                 int fn_loc = ((i + 1) == size) ? N : (st_loc + (N / size));
```

```
96.                 MPI_Recv(C + (st_loc * N), (fn_loc - st_loc) * N, MPI_DOUBLE, i, 0, MPI_COMM_WORLD,
&status);
97.             }
98.         }
99.         MPI_Barrier(MPI_COMM_WORLD);
100.
101.        if (rank != 0) {
102.            MPI_Send(&t, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
103.        }
104.        if (rank == 0) {
105.            for (int i = 1; i < size; i++) {
106.                MPI_Recv(&t_all[i], 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD, &status);
107.            }
108.        }
109.
110.        if (rank == 0) {
111.            t_all[0] = t;
112.        }
113.
114.        double max = -1;
115.        for (int i = 0; i < size; i++) {
116.            if (t_all[i] > max) {
117.                max = t_all[i];
118.            }
119.        }
120.        if (rank == 0) {
121.            printf("time = %.6f\n", max);
122.        }
123.
124.        MPI_Finalize();
125.
126.
127.        return 0;
128.    }
```