Суперкомпьютеры и параллельная обработка данных
Практическое задание

Вариант 44
Лебедев Андрей, группа 324

Оглавление

Цель работы

Научиться использовать технологии OpenMP, реализовать параллельную версию предложенного алгоритма с помощью директив for и task.

Постановка задачи

1) Для метода релаксации для решения двумерного уравнения Пуассона реализовать несколько версий параллельных программ с использованием технологии OpenMP:

а) Вариант параллельной программы с распределением витков циклов при помощи директивы for.
б) Вариант параллельной программы с использованием механизма задач (директива task).

2) Убедиться в корректности разработанных версий программ.

3) Начальные параметры для задачи должны быть подобраны таким образом, чтобы:
а) Задача помещалась в оперативную память одного узла кластера.
б) Время решения задачи было в заданном диапазоне.

4) Исследовать эффективность полученных параллельных программ на суперкомпьютере Polus. Исследовать масштабируемость полученной параллельной программы: построить графики зависимости времени выполнения параллельной программы от числа используемых ядер для различного объема входных данных.
Каждый прогон программы с новыми параметрами выполнять несколько раз с последующим усреднением результата для избавления от случайных выбросов.

5) Определить основные причины недостаточной масштабируемости программы при максимальном числе используемых ядер/процессоров.

Решение задачи

1. Реализация программ

Варианты кода для обеих реализаций приведены в конце данного отчета. Далее описываются изменения, проделанные с каждой программой.

Директива for

Модификация кода коснулась следующих функций:

1. main()
– Добавлен цикл по массиву p, который содержит различное количество нитей для параллельной обработки.
– Установка количества потоков с помощью omp_set_num_threads(p[i]).
– Замер времени выполнения с использованием omp_get_wtime().

2. init()
   – Добавлена прагма #pragma omp parallel for для распараллеливания цикла и разделения инициализации массива A между потоками.
   – Добавлена клауза private(i, j) для объявления приватных переменных внутри блока параллельной области.

3. relax()
   – Использование дополнительной матрицы B в функции relax() связано с тем, что внутри блока параллельной области (#pragma omp parallel for) несколько потоков одновременно изменяют значения элементов матрицы A. Это может привести к гонкам данных, когда несколько потоков пытаются одновременно читать и записывать значения в одни и те же ячейки памяти.
   Использование дополнительной матрицы B решает проблему гонок данных, так как каждый поток вычисляет свою часть матрицы B, используя значения из исходной матрицы A, и только после завершения всех вычислений значения копируются из B в A.
   – Добавлены клаузы private(i, j, e) и shared(A) в конструкцию #pragma omp parallel for, чтобы правильно распределить переменные и общую память.
   – Добавлена клауза reduction(max: eps) для правильной работы с переменной eps в параллельной секции.
   – Добавлена вторая параллельная область для копирования значений из массива B в массив A после завершения всех вычислений.

4. verify()
   – Добавлен #omp parallel for и reduction(+:s) для суммирования переменной s в безопасной параллельной секции.

Директива task

– double local_eps[N], double local_s[N] – эти массивы используются для сохранения локальных значений eps и s для каждого потока, чтобы избежать конфликтов записи в разделяемые переменные.
– Добавлен массив B[N][N], который используется для временного хранения значений при обновлении матрицы A.

Модификация кода коснулась следующих функций:

1. main()
   Функция аналогична той, что была в программе с директивой for.

2. init()
   – Добавлены директивы для создания параллельной области кода с использованием #pragma omp parallel.
   – Директива #pragma omp single гарантирует, что код внутри выполнится только одним потоком.
   – #pragma omp task создает задачу для каждой строки массива A.
   firstprivate(i) гарантирует, что каждая задача получит свою копию переменной i.
   private(j) указывает, что переменная j является локальной для каждой задачи.

shared(A) указывает, что массив A является общим для всех задач.

3. relax()
   – Идея похожа на ту, что использовалась в программе на директиву for.
   – shared(A, local_eps) указывает, что массив A и массив local_eps являются общими для всех задач.
   – Внутри задачи происходит вычисление новых значений в массиве B и обновление локального значения eps для каждой строки.

4. verify()
   – Введен локальный массив local_s для каждого потока для хранения локальных значений s.
   – #pragma omp task создает задачу для каждой строки массива A.
   – Клауза shared(A, local_s) указывает, что массив A и массив local_s являются общими для всех задач.

2. Корректность разработанных версий

Три программы (исходная, с директивой for, с директивой task) были скомпилированы и запущены на суперкомпьютере Polus, каждая из них выдавала одинаковое значение S при одном и том же размере N матрицы A. Таким образом, стало понятно, что обе версии программы работают корректно.

3. Подбор параметров

По результатам экспериментов было выявлено, что программа удовлетворяет заявленным требованиям производительности при размерах матрицы не больше 4098 элементов.

4. Исследование эффективности на Polus

В данном разделе приведено исследование масштабируемости полученной параллельной программы и построены графики зависимости времени выполнения параллельной программы от числа используемых нитей для различного объема входных данных.

Директива for

Для каждого из размеров (N) матрицы A проведем 4 запуска на количестве нитей от 1 до 160. Измерим время на каждом запуске, а далее найдем среднее по каждому количеству нитей. Таким образом, анализ нескольких запусков поможет избежать выбросы.
Ниже приведены таблицы с соответствующими результатами, где строка соответствует конкретному запуску, а столбец количеству нитей.

1. N = 66

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0.009617	0.005099	0.003739	0.002951	0.002554	0.002489	0.002286	0.002369	0.002378
2	0.009635	0.00508	0.003776	0.00297	0.002566	0.002499	0.002271	0.00244	0.002391
3	0.009628	0.005091	0.00378	0.002967	0.002549	0.002477	0.002273	0.002408	0.002438
4	0.009621	0.005067	0.00387	0.002947	0.002541	0.002534	0.002236	0.002424	0.002422
Среднее	0.0096252	0.0050842	0.0037912	0.0029587	0.0025525	0.0024997	0.0022665	0.0024102	0.0024072

	10	**20**	**40**	**60**	**80**	**100**	**120**	**140**	**160**
1	0.002032	0.003696	0.00588	0.00697	0.009286	0.086952	0.015115	0.019536	0.075722
2	0.002042	0.003697	0.005972	0.007077	0.009671	0.168385	0.079591	0.12157	0.07412
3	0.002079	0.003723	0.006072	0.006935	0.009135	0.276268	0.017378	0.018904	0.07618
4	0.002088	0.003649	0.006004	0.006693	0.029192	0.125205	0.097091	0.019798	0.073829
Среднее	0.0020602	0.0036912	0.005982	0.0069187	0.014321	0.1642025	0.0522937	0.044952	0.0749627

2. N = 258

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0.195897	0.097976	0.071243	0.05373	0.044239	0.037012	0.032396	0.028119	0.026029
2	0.144225	0.07245	0.049171	0.037267	0.030713	0.025865	0.022663	0.020325	0.018724
3	0.144193	0.072461	0.049239	0.037316	0.030692	0.025846	0.022644	0.02036	0.018726
4	0.144199	0.072457	0.04921	0.037285	0.030696	0.025898	0.02265	0.020305	0.018657
Среднее	0.1571285	0.078836	0.0547157	0.0413995	0.034085	0.0286552	0.0250882	0.0222772	0.020534

	10	20	40	60	80	100	120	140	160
1	0.023698	0.017019	0.017657	0.019382	0.024617	0.118623	0.11003	0.134871	0.078309
2	0.017015	0.015591	0.016771	0.019716	0.024414	0.152295	0.158732	0.111022	0.08
3	0.016949	0.015585	0.016089	0.018178	0.02392	0.358507	0.038628	0.157121	0.079218
4	0.017004	0.015505	0.016323	0.021366	0.023378	0.095393	0.146745	0.081261	0.07981
Среднее	0.0186665	0.015925	0.01671	0.0196605	0.0240822	0.1812045	0.1135337	0.1210687	0.0793342

3. N = 1026

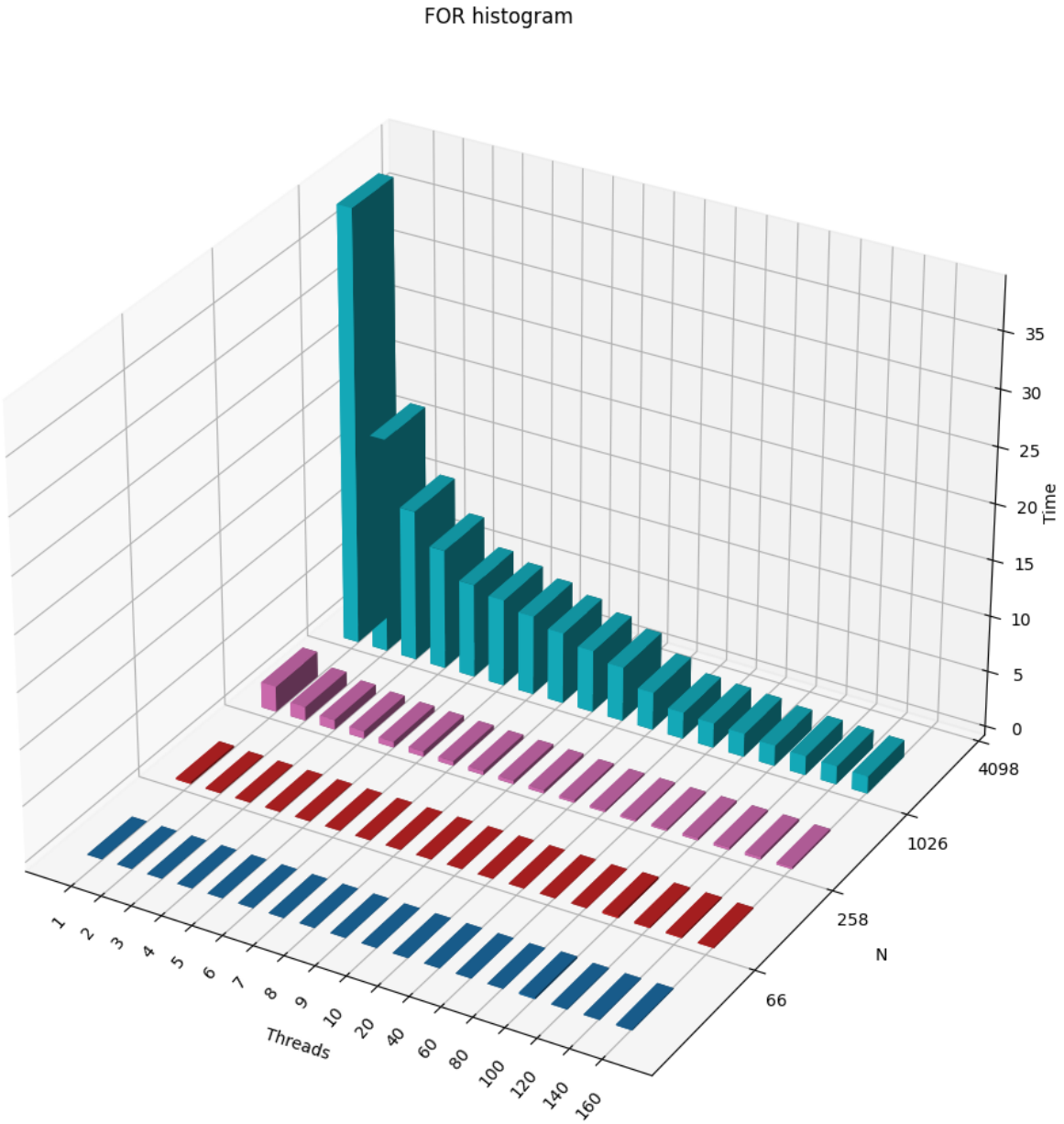
	1	2	3	4	5	6	7	8	9
1	2.281635	1.141405	1.055916	0.796901	0.639632	0.660897	0.567749	0.496801	0.445298
2	2.281629	1.503065	0.852037	0.653473	0.49609	0.385128	0.331609	0.28947	0.260416
3	2.281272	1.141222	0.764627	0.578	0.465074	0.388029	0.336311	0.295852	0.26542
4	2.281182	1.141241	0.764915	0.578056	0.463883	0.388009	0.336739	0.29587	0.265634
Среднее	2.2814295	1.2317332	0.8593737	0.6516075	0.5161697	0.4555157	0.393102	0.3444982	0.309192

	10	20	40	60	80	100	120	140	160
1	0.40449	0.2191	0.180859	0.177509	0.169743	0.213701	0.178346	0.1992	0.213676
2	0.234888	0.199224	0.16055	0.167771	0.172732	0.171645	0.298298	0.372572	0.165864
3	0.241719	0.221211	0.156091	0.161915	0.169907	0.169393	0.192692	0.199897	0.181039
4	0.242186	0.198369	0.156187	0.161473	0.163896	0.25488	0.191065	0.185911	0.18441
Среднее	0.2808207	0.209476	0.1634217	0.167167	0.1690695	0.2024047	0.2151002	0.239395	0.1862472

4. N = 4098

	1	2	3	4	5	6	7	8	9
1	37.21708	21.029009	12.539427	10.572106	7.362821	6.812924	6.332498	6.615423	5.027166
2	38.26322	19.56325	16.168892	10.863602	8.396675	8.281345	7.851273	7.208232	5.705259
3	38.589531	18.235994	13.222122	12.352536	10.461631	9.856152	8.761837	6.822203	6.703567
4	41.567364	18.239651	12.186727	9.223242	7.423861	6.188789	5.400739	4.778709	4.7382
Среднее	38.909298	19.266976	13.529292	10.752871	8.411247	7.7848025	7.0865867	6.3561417	5.543548

	10	20	40	60	80	100	120	140	160
1	4.121471	3.490397	2.554921	2.229801	2.08317	1.999024	1.757118	1.641404	1.527383
2	5.235136	3.721735	2.319269	2.146524	2.115996	1.833297	1.755967	1.656922	1.539483
3	6.000061	3.099377	2.256547	2.198304	2.057704	1.741533	1.658861	1.591162	1.547447
4	3.97855	3.088287	2.252764	2.189044	2.057598	1.727956	1.619574	1.571691	1.499741
Среднее	4.8338045	3.349949	2.3458752	2.1909182	2.078617	1.8254525	1.69788	1.6152947	1.5285135



Директива task

1. N = 66

	1	2	3	4	5	6
1	0,004692	0,012835	0,020768	0,025672	0,029557	0,032060
2	0,004663	0,012877	0,020900	0,025141	0,029125	0,034072
3	0,005862	0,013470	0,020272	0,025416	0,029657	0,032615
4	0,004719	0,012844	0,020927	0,025184	0,025492	0,031884
Среднее	0.004984	0.013007	0.020717	0.025353	0.028458	0.032658

	7	8	9	10	20	40
1	0,037527	0,045246	0,048104	0,050753	0,108459	0,245684
2	0,037923	0,045495	0,049570	0,055917	0,113954	0,237094
3	0,034383	0,042942	0,051449	0,053422	0,115620	0,240032
4	0,037364	0,046255	0,054493	0,056288	0,112280	0,240717
Среднее	0.036799	0.044985	0.050904	0.054095	0.042578	0.040882

2. N = 258

	1	2	3	4	5	6
1	0,035088	0,050715	0,070928	0,100874	0,119892	0,140960
2	0,023675	0,042576	0,071998	0,096949	0,112679	0,122631
3	0,023576	0,042610	0,071238	0,096332	0,112138	0,123102
4	0,023760	0,051366	0,072367	0,100507	0,117671	0,125430
Среднее	0.026525	0.046817	0.071633	0.098666	0.115595	0.128031

	7	8	9	10	20	40
1	0,162149	0,193424	0,134411	0,174553	0,143914	0,110492
2	0,143718	0,159149	0,183214	0,131456	0,183497	0,359339
3	0,150472	0,145198	0,161145	0,135168	0,148772	0,003809
4	0,147335	0,181471	0,102492	0,108776	0,186198	0,934229
Среднее	0.150919	0.169811	0.165316	0.167488	0.140595	0.131967

3. N = 514

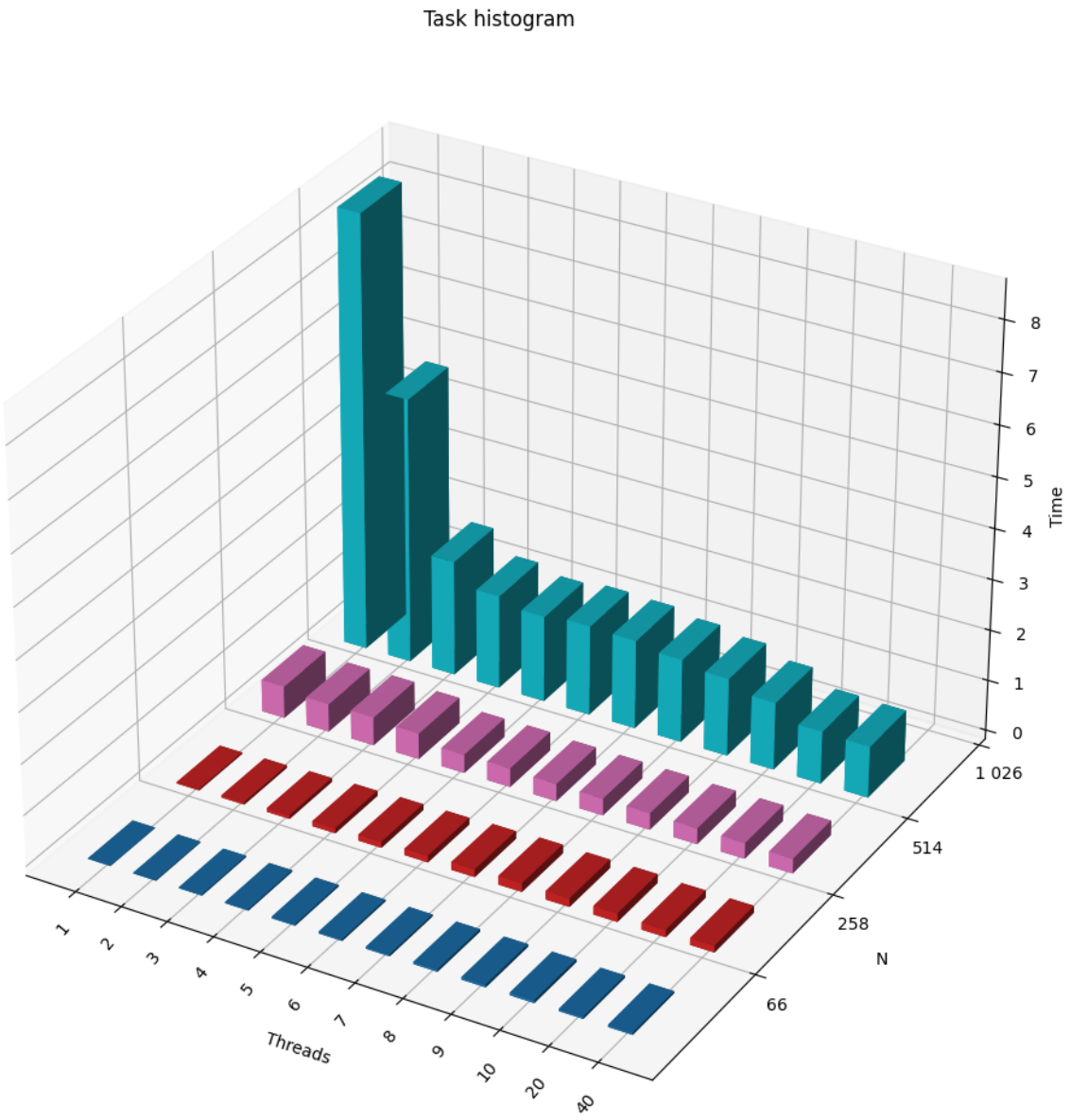
	1	2	3	4	5	6
1	0,684307	0,542364	0,530199	0,389917	0,350605	0,359873
2	0,608926	0,561471	0,572727	0,420118	0,359720	0,387500
3	0,609780	0,561754	0,567433	0,416287	0,352235	0,379538
4	0,611079	0,521904	0,551811	0,498816	0,382521	0,343310
Среднее	0.628520	0.546873	0.555543	0.406285	0.36127	0.367555

	7	8	9	10	20	40
1	0,326290	0,317928	0,353254	0,342664	0,312482	0,305400
2	0,351913	0,367310	0,314441	0,299604	0,289154	0,215626
3	0,348004	0,383157	0,301975	0,382816	0,350166	0,244571
4	0,319899	0,384916	0,347062	0,378168	0,347759	0,247085
Среднее	0.336527	0.328328	0.324183	0.310813	0.307039	0.288171

4. N = 1026

	1	2	3	4	5	6
1	8,299092	5,509145	2,459388	1,953123	1,719422	1,788185
2	8,199145	5,866609	2,243366	1,753866	1,594497	1,759141
3	9,165556	5,246439	2,319316	1,988156	1,861448	1,760562
4	8,685452	5,299988	2,073327	1,676392	1,613044	1,770185
Среднее	8.587311	5.730545	2.273849	1.842884	1.697103	1.769518

	7	8	9	10	20	40
1	1,818950	1,946571	1,148479	1,340189	0,861532	1,131618
2	1,861528	1,927982	1,422031	1,742935	1,227886	1,134227
3	1,558419	1,422408	1,694428	1,405956	1,131326	1,085337
4	1,950799	2,177344	1,461286	1,211787	1,002571	0,756969
Среднее	1.747424	1.638576	1.531556	1.325217	1.028329	1.007038



5. Причины недостаточной масштабируемости

Недостаточная масштабируемость программы при максимальном числе используемых ядер/процессоров может быть вызвана различными причинами:

1. Избыточная синхронизация. Программы часто требуют синхронизации между потоками/процессами для корректного выполнения. Избыточная синхронизация может привести к ожиданию доступа к общим ресурсам и уменьшению эффективности параллельных вычислений.
2. Неравномерное распределение нагрузки. Некоторые потоки или процессы могут выполняться дольше или требовать больше ресурсов, что приводит к неравномерному распределению нагрузки между ядрами/процессорами.

Код программ

Директива for

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

#define  Max(a,b) ((a)>(b)?(a):(b))

#define  N   (64+2)
double   maxeps = 0.1e-7;
int itmax = 100;
int i,j;
double eps;

double A [N][N];
double B [N][N];

void relax();
void init();
void verify();

int main(int an, char **as)
{
    int p[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 40, 60, 80, 100, 120, 140, 160};
    double time_start;
    double time_end;
    int it;

    for(i = 0; i < 18; i++)
    {
        omp_set_num_threads(p[i]);
        time_start = omp_get_wtime();
        init();
        for(it=1; it<=itmax; it++)
        {
            eps = 0.;
            relax();
            if (eps < maxeps) break;
        }
        verify();
        time_end = omp_get_wtime();
        printf("%10f, ", time_end - time_start);
    }

    printf("\n");
    return 0;
}


void init()
{
```

```
    #pragma omp parallel for shared(A) private(i, j)
    for(j=0; j<=N-1; j++)
    {
        for(i=0; i<=N-1; i++)
        {
            if(i==0 || i==N-1 || j==0 || j==N-1) A[i][j]= 0.;
            else A[i][j]= ( 1. + i + j );
        }
    }
}


void relax()
{
    double e;
    #pragma omp parallel for private(i, j, e) shared(A) reduction(max: eps) collapse(2)
    for (i = 1; i <= N - 2; i++)
    {
        for (j = 1; j <= N - 2; j++)
        {
            e = A[i][j];
            B[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i][j - 1] + A[i][j + 1]) / 4.;
            eps = Max(eps, fabs(e - A[i][j]));
        }
    }

    #pragma omp parallel for private(i, j) shared(A, B)
    for (i = 1; i <= N - 2; i++)
    {
        for (j = 1; j <= N - 2; j++)
        {
            A[i][j] = B[i][j];
        }
    }
}


void verify()
{
    double s;
    s=0.;

    #pragma omp parallel for shared(A) private(i, j) reduction(+:s)
    for(j=0; j<=N-1; j++)
    {
        for(i=0; i<=N-1; i++)
        {
            s=s+A[i][j]*(i+1)*(j+1)/(N*N);
        }
    }
    //printf("  S = %f\n",s);
}
```

Директива task

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

#define Max(a, b) ((a) > (b) ? (a) : (b))

#define N (4096 + 2)
double maxeps = 0.1e-7;
int itmax = 100;
int i, j;
double eps;

double A[N][N];
double B[N][N];

double local_eps[N];
double local_s[N];

void relax();
void init();
void verify();

int main(int an, char **as)
{
        int p[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 40, 60, 80, 100, 120, 140, 160};
    double time_start;
    double time_end;
    int it;

    for(k = 0; k < 18; k++) {
      omp_set_num_threads(p[k]);
        time_start = omp_get_wtime();
        init();
        for(it=1; it<=itmax; it++)
        {
            eps = 0.;
            relax();
            if (eps < maxeps) break;
        }
        verify();
        time_end = omp_get_wtime();
        printf("%10f, ", time_end - time_start);
    }
    printf("\n");

        return 0;
}

void init()
{
    #pragma omp parallel
    #pragma omp single
    for (i = 0; i <= N - 1; i++) {
```

```
        #pragma omp task firstprivate(i) private(j) shared(A)
        for (j = 0; j <= N - 1; j++)
        {
            if (i == 0 || i == N - 1 || j == 0 || j == N - 1)
                A[i][j] = 0.;
            else
                A[i][j] = (1. + i + j);
        }
    }
}


void relax()
{
    #pragma omp parallel
    #pragma omp single
    for (i = 1; i <= N - 2; i++)
    {
        local_eps[i] = eps;
        #pragma omp task firstprivate(i) private(j) shared(A, local_eps)
        for (j = 1; j <= N - 2; j++)
        {
            B[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i][j - 1] + A[i][j + 1]) / 4.;
            double e = fabs(A[i][j] - B[i][j]);
            local_eps[i] = Max(local_eps[i], e);
        }
    }

    for (i = 1; i <= N-2; ++i)
    {
        eps = Max(local_eps[i], eps);
    }

    #pragma omp parallel
    #pragma omp single
    for (i = 1; i <= N - 2; i++)
    {
        #pragma omp task firstprivate(i) private(j) shared(A, B)
        for (j = 1; j <= N - 2; j++)
        {
            A[i][j] = B[i][j];
        }
    }
}


void verify()
{
    for (i = 0; i <= N - 1; i++)
    {
        local_s[i] = 0.0;
    }

    #pragma omp parallel
    #pragma omp single
    for (i = 0; i <= N - 1; i++) {
```

```
        #pragma omp task firstprivate(i) private(j) shared(A, local_s)
        for (j = 0; j <= N - 1; j++)
        {
            local_s[i] += A[i][j] * (i + 1) * (j + 1) / (N * N);
        }
    }

    double s;
    s = 0.;

    for (i = 0; i <= N - 1; i++)
    {
        s += local_s[i];
    }
}
```