Московский государственный университет имени М.В.Ломоносова
Факультет вычислительно математики и кибернетики

Отчет о выполнении задания на вычислительном комплексе IBM Regatta

Выполнил
студент 513 группы
кафедры Оптимального Управления
Ахрамеев Павел

2014

Численное решение задачи Дирихле.
Метод попеременных направлений.
Разработка параллельной программы и исследование ее эффективности.

Постановка задачи.

Дана последовательная программа, реализующая метод попеременных направлений.

Требуется разработать параллельную программу с использованием технологии OpenMP и провести исследование ее эффективности.

Цель.

Получить навыки распараллеливания существующих программ на языке Си с использованием технологии OpenMP.

Распараллеливание осуществляется с помощью анализа последовательной программы, аналогично анализу распараллеливающего компилятора. Поэтому не предполагается знания указанного алгоритма.

Требуется.

1. Разработать параллельную версию программы с использованием технологии OpenMP
2. Исследовать время выполнения разработанной программы в зависимости от размера сетки и количества используемых потоков на вычислительном комплексе IBM Regatta.
3. Построить графики - зависимость ускорения от количества процессоров для разных размеров сетки.
4. Подготовить отчет о выполнении задания, включающий таблицу с временами, графики, текст программы. Сделать выводы по полученным результатам (объяснить убывание или возрастание производительности параллельной программы при увеличении числа используемых процессоров, сравнить поведение параллельной программы в зависимости от размера сетки).

Результаты.

Ниже представлена таблица, по которой можно оценить ускорение времени вычисления параллельной программы в зависимости от количества процессоров и размера сетки.

Размер сетки	Последовательный алгоритм -O2	Параллельный алгоритм							
		1 процессор		2 процессора		4 процессора		8 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение	Время	Ускорение
256x256x256	68,566	66,3721	1,0330546	34,4107	1,9925779	18,8399	3,6394036	9,70675	7,0637443
384x384x384	399,79	239,098	1,6720759	120,566	3,3159431	62,7336	6,3728209	35,4974	11,262515
512x512x512	672,965	577,393	1,1655233	283,429	2,3743689	163,6	4,113478	87,6649	7,6765615

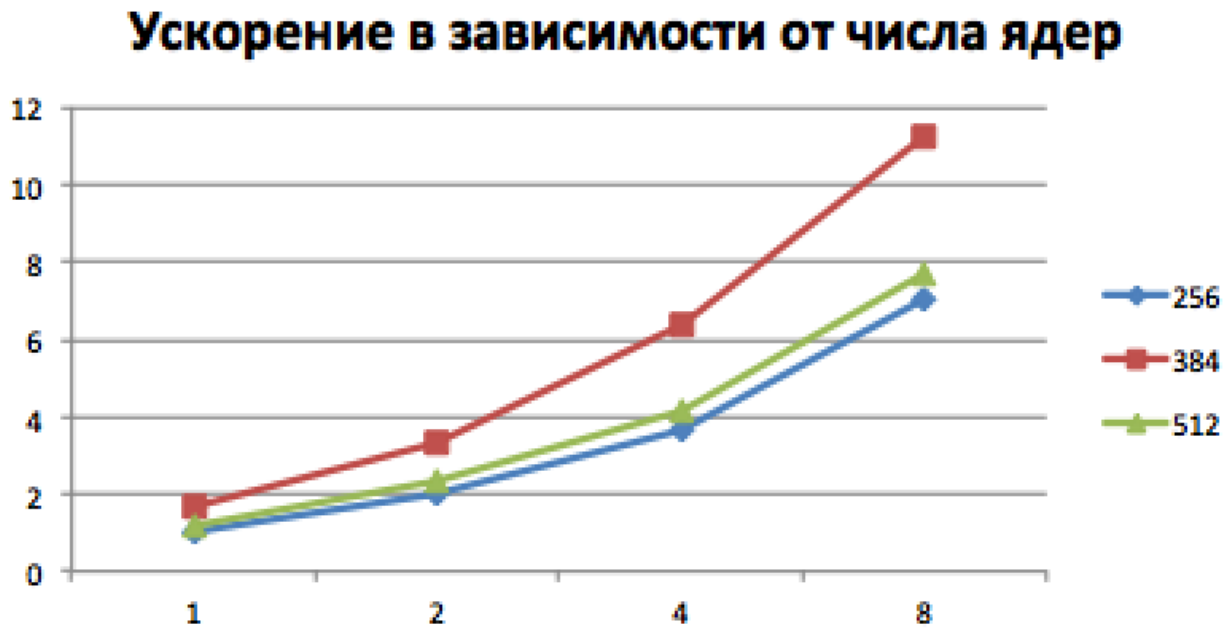
Ускорение (speedup), получаемое при использовании параллельного алгоритма для p процессоров, определяется величиной:

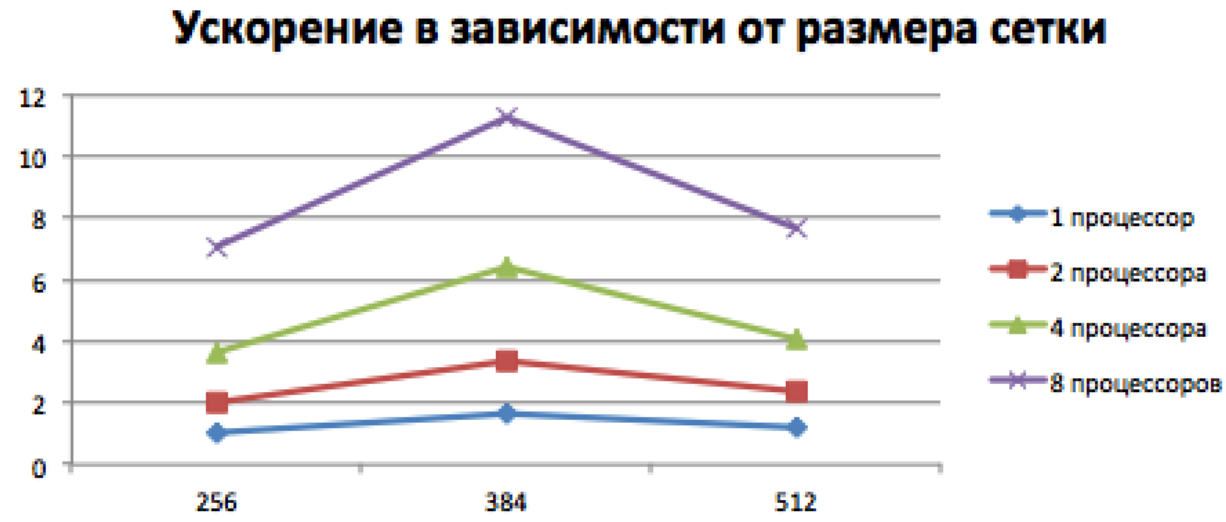
Speedup(n) = T1(n)/Tp(n),

где T1(n)- время последовательного выполнения задачи,

Tp(n)- время параллельного выполнения задачи при использовании p процессоров.

По данным из этой таблицы можно построить графики ускорения программы в зависимости от количества ядер распараллеливания, а также посмотреть, как размер сетки влияет на ускорение программы. Графики представлены ниже.





Выводы.

Таким образом, можно сделать вывод о том, что ускорение программы в зависимости от количества ядер. Удивительно, но в моей реализации программы на полученных статистических данных наибольшее ускорение удалось получить на размере сетки в 384.

Отмечу, что время выполнения программы для нескольких запусков одной и той же программы (одного и того же бинарного файла) отличалось до 6%. Графики приведены для моих первых запусков программы с указанными параметрами.

Текст параллельной программы.

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

#define OMP 1

#if (OMP)
#include <omp.h>
#endif

#define Max(a,b) ((a)>(b)?(a):(b))
#define N 256

double maxeps = 0.1e-7;
int itmax = 100;
int i,j,k;
double eps;
double A [N][N][N];
void relax();
void init();
void verify();
void wtime();
int main(int an, char **as) {
      int it;
      double time0, time1;
      init();
#if (OMP)
      time0=omp_get_wtime ();
#else
      wtime(&time0);
#endif
      for(it=1; it<=itmax; it++) {
            eps = 0.;
            relax();
            printf( "it=%4i eps=%f\n", it,eps);
            if (eps < maxeps) {
                  break;
            }
      }
#if (OMP)
      time1=omp_get_wtime ();
#else
      wtime(&time1);
#endif
      printf("Time(sec)=%gs\t",time1-time0);
      verify();
      return 0;
}
void init() {
#if (OMP)
      #pragma omp parallel for private(i,j,k)
#endif
      for(i=0; i<=N-1; i++) {
            for(j=0; j<=N-1; j++) {
                  for(k=0; k<=N-1; k++) {
                        if(i==0 || i==N-1 || j==0 || j==N-1 || k==0 || k==N-1)
{
                              A[i][j][k]= 0.;
                        } else {
                              A[i][j][k]= ( 4. + i + j + k);
                        }
```

```
                }
            }
        }
}
void relax() {
#pragma omp parallel
        {
        double eps_internal = eps;
        #pragma omp for private(i,j,k)
        for(i=1; i<=N-2; i++) {
            for(j=1; j<=N-2; j++) {
                for(k=1; k<=N-2; k++) {
                    A[i][j][k] = (A[i-1][j][k]+A[i+1][j][k])/2.;
                }
            }
        }
        #pragma omp for private(i,j,k)
        for(i=1; i<=N-2; i++) {
            for(j=1; j<=N-2; j++) {
                for(k=1; k<=N-2; k++) {
                    A[i][j][k] =(A[i][j-1][k]+A[i][j+1][k])/2.;
                }
            }
        }
        #pragma omp for private(i,j,k) nowait
        for(i=1; i<=N-2; i++) {
            for(j=1; j<=N-2; j++) {
                for(k=1; k<=N-2; k++) {
                    double e;
                    e=A[i][j][k];
                    A[i][j][k] = (A[i][j][k-1]+A[i][j][k+1])/2.;
                    eps_internal=Max(eps_internal,fabs(e-A[i][j][k]));
                }
            }
        }
        #pragma omp critical
            eps = Max(eps, eps_internal);
        #pragma omp barrier
        {}
        //pragma close
        }
}
void verify() {
        double s;
        s=0.;
        #pragma omp parallel for private (i, j, k) reduction (+:s)
        for(i=0; i<=N-1; i++) {
            for(j=0; j<=N-1; j++) {
                for(k=0; k<=N-1; k++) {
                    s=s+A[i][j][k]*(i+1)*(j+1)*(k+1)/(N*N*N);
                }
            }
        }
        printf(" S = %f\n",s);
}
void wtime(double *t) {
        static int sec = -1;
        struct timeval tv;
        gettimeofday(&tv, (void *)0);
        if (sec < 0) {
            sec = tv.tv_sec;
        }
        *t = (tv.tv_sec - sec) + 1.0e-6*tv.tv_usec;
}
```