МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по курсу

"Суперкомпьютеры и параллельная обработка данных"

Разработка параллельной версии программы для задачи Gauss

ОТЧЕТ

о выполненном задании

студентки 321 учебной группы факультета ВМК МГУ

Задорожной Юлии Андреевны

Москва
2021

Содержание

1 Постановка задачи

1. Рассмотреть код программы и оптимизировать его для выполнения в многопоточном режиме с применением технологии OpenMP
2. Исследовать зависимость времени выполнения программы от различных способов распараллеливания кода программы
3. Исследовать зависимость времени выполнения программы от размера входной матрицы и числа используемых потоков и ядер
4. Визуализировать полученные результаты при помощи трехмерных графиков, обосновать вывод

2 Описание алгоритма и код программы

Параллельный алгоритм Гаусса представлен ниже:

```
    #include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <omp.h>

void prt1a(char *t1, double *v, int n,char *t2);
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;
}

int N;
double *A;
#define A(i,j) A[(i)*(N+1)+(j)]
double *X;

int main(int argc,char **argv) {
  double time0, time1;
  FILE *in;
  int i, j, k;
  int threads = omp_get_num_procs();
  in=fopen("data.in","r");
  if(in==NULL) {
    printf("Can not open 'data.in' "); exit(1);
  }
  i=fscanf(in,"%d", &N);
  if(i<1) {
    printf("Wrong 'data.in' (N ...)"); exit(2);
  }

  omp_set_num_threads(threads);
  /* create arrays */
  A=(double *)malloc(N*(N+1)*sizeof(double));
  X=(double *)malloc(N*sizeof(double));
  printf("GAUSS %dx%d\n--------------------------------\n",N,N);
  /* initialize array A*/

    #pragma omp parallel for shared(A, N) private(i, j) schedule(static)
    for(i=0; i <= N-1; i++)
    for(j=0; j <= N; j++)
      if (i==j || j==N)
        A(i,j) = 1.f;
      else
        A(i,j)=0.f;
  wtime(&time0);

```

```
51
52    /* elimination */
53    double m;
54    omp_set_nested(1);
55      for (i=0; i<N-1; i++) {
56        #pragma omp parallel for shared(N, A, i) private(m, j,k) schedule(dynamic)
57      for (k=i+1; k <= N-1; k++)
58        m = A(k,i) / A(i,i);
59        for (j=i+1; j <= N; j++)
60          A(k,j) = A(k,j)- (m * A(i,j));
61    /* reverse substitution */
62    }
63
64      X[N-1] = A(N-1,N)/A(N-1,N-1);
65      #pragma omp parallel for shared(A, X) private(k,j) schedule(dynamic)
66      for (j=N-2; j>=0; j--) {
67      for (k=0; k <= j; k++)
68        A(k,N) = A(k,N)-A(k,j+1)*X[j+1];
69      X[j]=A(j,N)/A(j,j);
70    }
71    wtime(&time1);
72    printf("Time in seconds=%gs\n",time1-time0);
73    prt1a("X=(", X,N>9?9:N,"...)\n");
74    free(A);
75    free(X);
76    return 0;
77  }
78
79  void prt1a(char * t1, double *v, int n,char *t2) {
80    int j;
81    printf("%s",t1);
82      for(j=0;j<n;j++)
83      printf("%.4g%s",v[j], j%10==9? "\n": ", ");
84    printf("%s",t2);
85  }
```

Этот алгоритм изначально являлся итеративным и был ориентирован на последовательное заполнение и вычисление строк матрицы A. Количество выполненных операций каждого цикла имеет порядок $O(n^3)$.

Оптимальная параллельная программа

Из определения метод Гаусса следует, что он реккурентно, итеративно исключает переменные с помощью элементарных преобразований. Для уменьшения времени выполнения программы была использована попытка распараллелить все из имеющихся циклов с использованием добавленной клаузы pragma omp parallel for, private, schedule и shared. Поскольку ни один из циклов не содержит зависимостей, это можно было сделать. Для решения возникших сложностей алгоритма из-за цикла был применен алгоритм разбиения матрицы на строки, с последующим вычислением каждой новой строки и всех вместе, а также был изменен порядок обхода матрицы с обхода по столбцам на обход по строкам.

При таком подходе достигнутый уровень параллелизма является в большинстве случаев избыточным. Обычно при проведении практических расчетов такое количество сформированных подзадач превышает число имеющихся процессоров и делает неизбежным этап укрупнения базовых задач. В этом плане может оказаться полезной агрегация вычислений уже на шаге выделения базовых подзадач. Такой подход приводит к снижению общего количества подзадач до величины N.

Модификация кода сводится к добавлению нескольких клауз.

```
1   omp_set_num_threads(threads);
2   /* create arrays */
3   A=(double *)malloc(N*(N+1)*sizeof(double));
4   X=(double *)malloc(N*sizeof(double));
5   printf("GAUSS %dx%d\n--------------------------------\n",N,N);
```

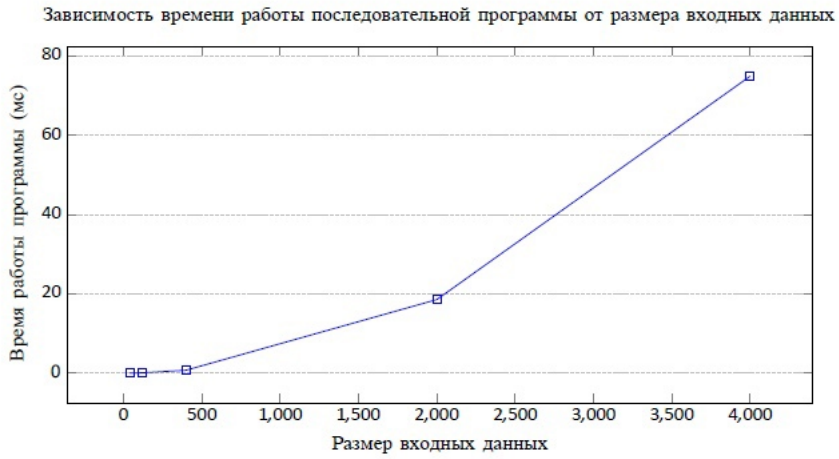
```
/* initialize array A*/

  #pragma omp parallel for shared(A, N) private(i, j) schedule(static)
  for(i=0; i <= N-1; i++)
  for(j=0; j <= N; j++)
    if (i==j || j==N)
      A(i,j) = 1.f;
    else
      A(i,j)=0.f;
wtime(&time0);


/* elimination */
double m;
omp_set_nested(1);
  for (i=0; i<N-1; i++) {
    #pragma omp parallel for shared(N, A, i) private(m, j,k) schedule(dynamic)
  for (k=i+1; k <= N-1; k++)
    m = A(k,i) / A(i,i);
    for (j=i+1; j <= N; j++)
      A(k,j) = A(k,j)- (m * A(i,j));
/* reverse substitution */
}

  X[N-1] = A(N-1,N)/A(N-1,N-1);
  #pragma omp parallel for shared(A, X) private(k,j) schedule(dynamic)
  for (j=N-2; j>=0; j--) {
  for (k=0; k <= j; k++)
    A(k,N) = A(k,N)-A(k,j+1)*X[j+1];
  X[j]=A(j,N)/A(j,j);
}
```

Реализованный алгоритм проверялся на корректность и совпадение по результатам с последовательным алгоритмом.



3 Результаты тестирования программы

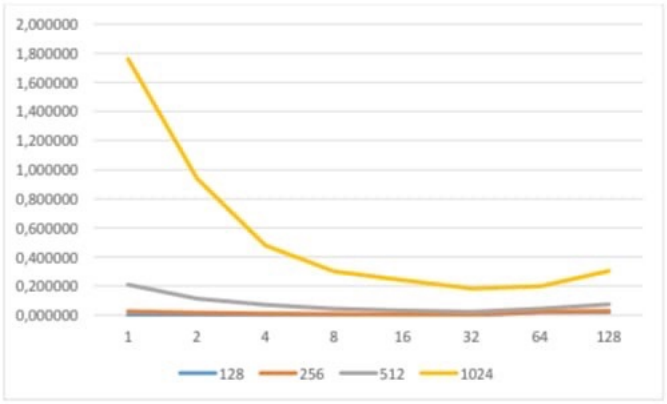
Результаты былли получены на вычислительном комплексе Polus. Для тестирования были использованы две версии программы – исходная и с распараллеливанием.
Входные данные для тестирования:

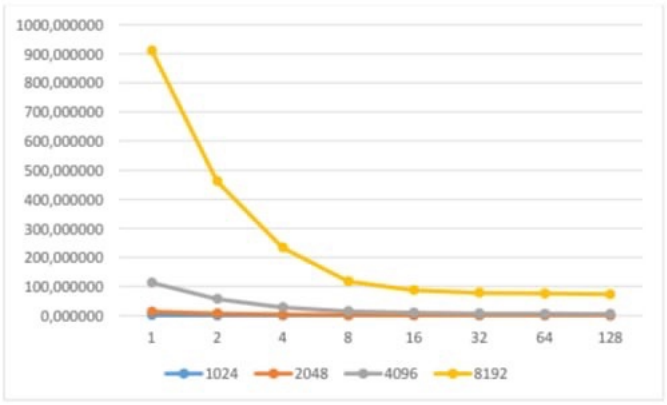
1. Число используемых потоков было выбрано 1, 2, 4, 6, 8, 16, 32, 64, 160 и тд потоков
2. Размер матрицы в data.in: 8, 64, 128

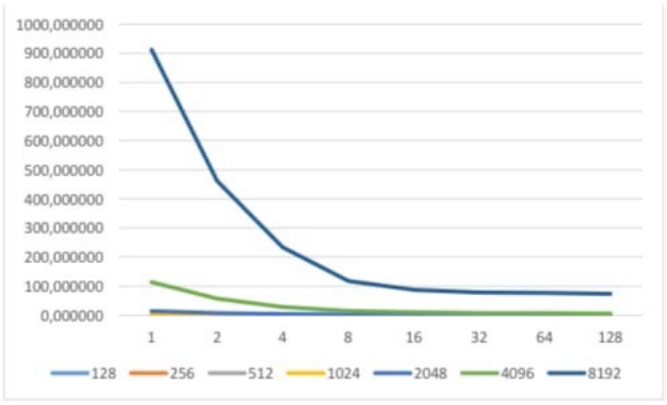
Каждая конфигурация была запущена 5 раз. Ниже приведены усредненные результаты.

Результаты замеров времени выполнения

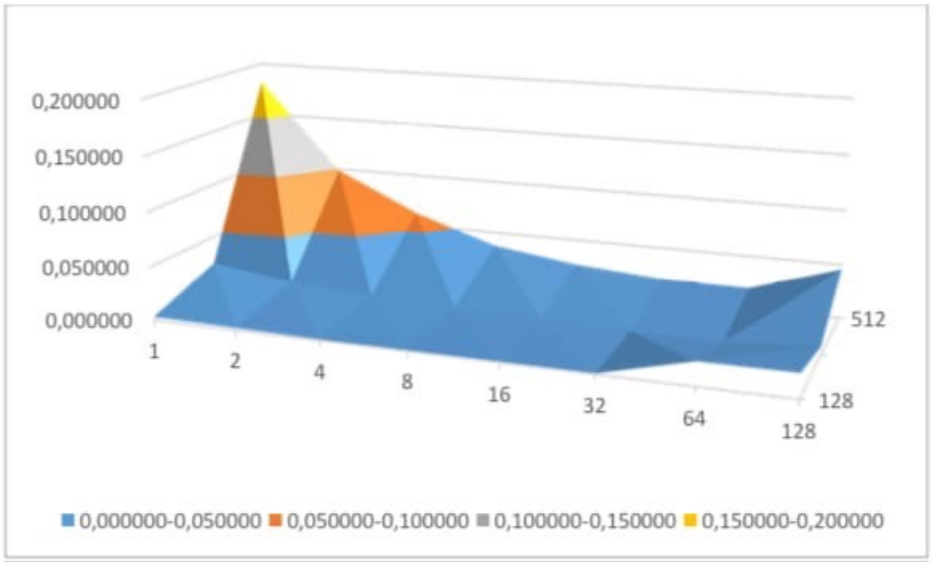
Нити/Размер матрицы	128	256	512	1024	2048	4096	8192
1	0.003115	0.024327	0.181999	1.55047	12.4399	99.4217	798.166
2	0.002	0.01417	0.098851	0.825769	6.29858	50.3307	404.434
4	0.00106	0.00887	0.062314	0.407289	3.20299	25.175	204.9
8	0.000789	0.006204	0.036908	0.256996	1.62157	12.8999	103.242
16	0.000731	0.004142	0.025983	0.209367	1.31767	8.59211	77.0492
32	0.000719	0.002738	0.020072	0.159401	1.13686	5.85083	71.5973
64	0.021115	0.002709	0.019805	0.155255	1.13403	5.11938	69.6801
128	0.021586	0.00789	0.045674	0.228739	0.723675	4.69169	67.7701

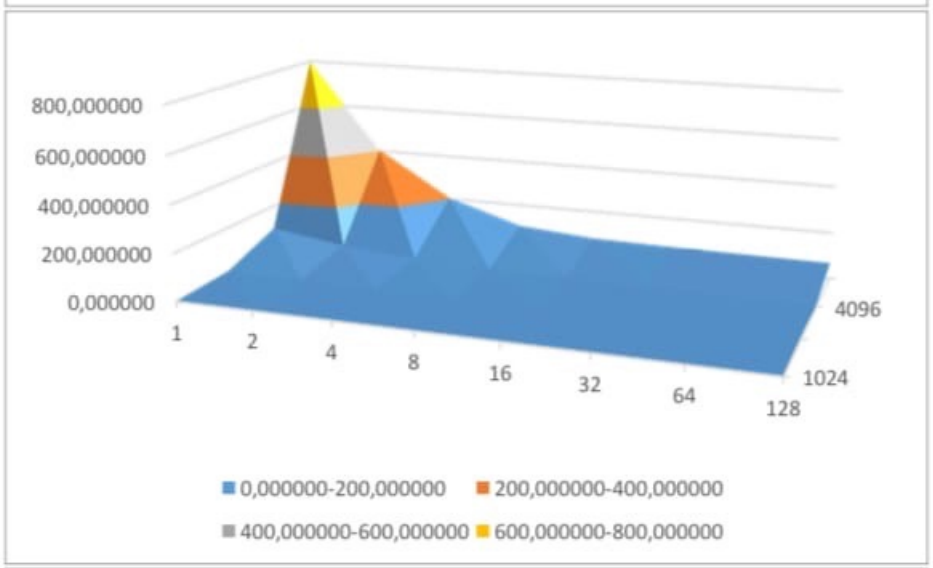


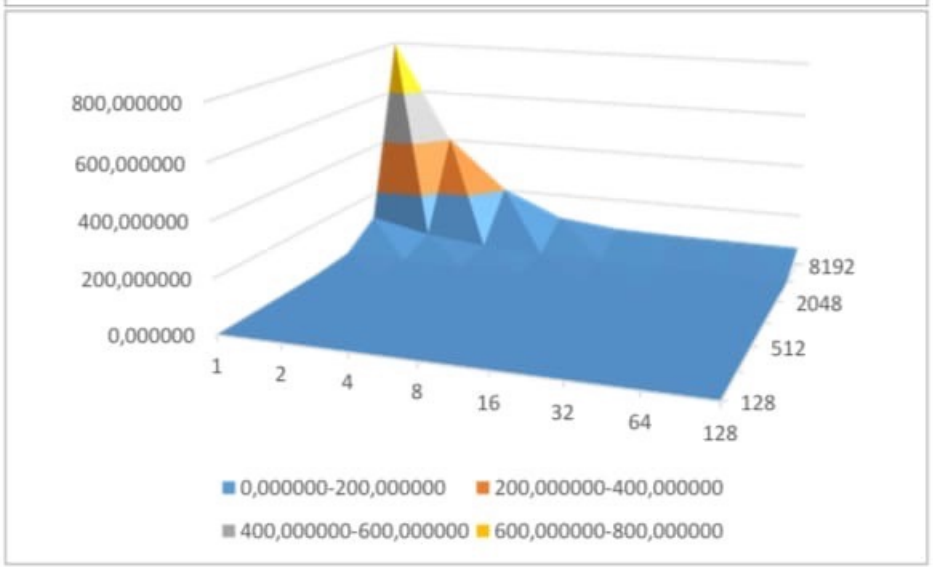




График, отражающий зависимость времени выполнения программы от различных входных данных и числа потоков.


0,200000
0,150000
0,100000
0,050000
0,000000
1
2
4
8
16
32
64
128
512
128
0,000000-0,050000
0,050000-0,100000
0,100000-0,150000
0,150000-0,200000


800,000000
600,000000
400,000000
200,000000
0,000000
1
2
4
8
16
32
64
128
4096
1024
0,000000-200,000000
200,000000-400,000000
400,000000-600,000000
600,000000-800,000000


800,000000
600,000000
400,000000
200,000000
0,000000
1
2
4
8
16
32
64
128
8192
2048
512
128
0,000000-200,000000
200,000000-400,000000
400,000000-600,000000
600,000000-800,000000

4 Вывод программы

```
OMP_NUM_THREADS=1 mpiexec ./a.out
OMP_NUM_THREADS=2 mpiexec ./a.out
OMP_NUM_THREADS=4 mpiexec ./a.out
OMP_NUM_THREADS=8 mpiexec ./a.out
OMP_NUM_THREADS=16 mpiexec ./a.out
OMP_NUM_THREADS=32 mpiexec ./a.out
OMP_NUM_THREADS=64 mpiexec ./a.out

GAUSS 2000x2000
---------------------------------
Time in seconds=253.63s
X=(1, 1, 1, 1, 1, 1, 1, 1, 1, ...)
GAUSS 2000x2000
---------------------------------
Time in seconds=171.75s
X=(1, 1, 1, 1, 1, 1, 1, 1, 1, ...)
GAUSS 2000x2000
---------------------------------
Time in seconds=73.1096s
X=(1, 1, 1, 1, 1, 1, 1, 1, 1, ...)
GAUSS 2000x2000
---------------------------------
Time in seconds=11.7977s
X=(1, 1, 1, 1, 1, 1, 1, 1, 1, ...)
GAUSS 2000x2000
---------------------------------
Time in seconds=3.12045s
X=(1, 1, 1, 1, 1, 1, 1, 1, 1, ...)
GAUSS 2000x2000
---------------------------------
Time in seconds=3.31593s
X=(1, 1, 1, 1, 1, 1, 1, 1, 1, ...)
GAUSS 2000x2000
---------------------------------
Time in seconds=2.92982s
X=(1, 1, 1, 1, 1, 1, 1, 1, 1, ...)
  }
```

Как мы видим: с ростом количества нитей время работы уменьшается.

5 Анализ результатов

На основании результатов измерений можно сделать следующие выводы: Время выполнения программы перестает линейно уменьшаться с ростом числа потоков на которых выполняется программа, потому что накладные расходы на создание порции нитей значительно увеличиваются, и, проанализировав результаты выполнения, можно заметить, что при меньших размерах данных программа перестает ускоряться при меньшем числе потоков. Это подтверждает идею о том, что накладные расходы на создание нитей - основная причина недостаточной масштабируемости программы. Такой алгоритм позволяет очень хорошо ускорить программу и избежать зависимости по данным при параллельном выполнении программы. Анализируя графики и находя минимумы времени при различных наборах параметров, можно установить оптимальное число потоков для данного размера данных. Например, для размера данных равного 256, можно заметить, что минимум времени достигается при 64 потоках. Также из полученных данных видно, что произведенная оптимизация последовательной программы значительно ускоряет работу.

6 Выводы

Работа по улучшению и разработке параллельной версии программы при помощи средств OpenMP позволила значительно ускорить начальную последовательную версию программы и провести ряд экспериментов на таких вычислительных ресурсах факультета ВМК, как Polus. OpenMP - это очень удобная в использовании технология, которая позволяет быстро получить качественный результат и достичь значительного прироста в производительности как на домашних машинах, так и на суперкомпьютерах, но к сожалению, рост нитей не всегда пропорционален уменьшению времени работы программы.