Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа №2

по курсу «Разработка параллельных и распределенных программ»

« Параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI»

Студент группы ИУ9-51Б Киселев К.А.

Преподаватель Царев А.С.

Москва 2023

Содержание

1 Постановка задачи

1. Написать программу, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b в соответствии с выбранным вариантом. Здесь A – матрица размером N×N, x и b – векторы длины N. Тип элементов – double.

2. Программу распараллелить с помощью MPI с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части. Соседние строки матрицы должны располагаться в одном или в соседних MPI-процессах. Реализовать два варианта программы: 1: векторы x и b дублируются в каждом MPI-процессе, 2: векторы x и b разрезаются между MPI-процессами аналогично матрице A. (только для сдающих после срока) Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом).

3. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: 1,2, 4, 8, 16. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ε подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд. Также параметр N разрешено подобрать таким образом, чтобы он нацело делился на на 1,2,4,8 и 16.

4. На основании полученных результатов сделать вывод о целесообразности использования одного или второго варианта программы.

2 Практическая реализация

Код программы для решение СЛАУ методом простых итераций с делением матрицы A между процессами

```
func (s *Solver) FindSolution() *mat.VecDense {
  res := mat.NewVecDense(s.size, nil)
  metric := math.MaxFloat64

  prev := metric

  buff := make([]float64, s.size)
  for metric > s.eps {
    s.comm.AllGatherF64(buff, s.iterateSolution(res).RawVector().Data)
    tmp := mat.NewVecDense(s.size, buff)

    res.SubVec(res, tmp)

    prev = metric
    metric = s.calcMetric(res)

    if prev < metric {
      s.tau *= -1
    }

  }

  return res
}

func (s *Solver) iterateSolution(chunk *mat.VecDense) *mat.VecDense {
  tmp := mat.NewVecDense(s.end-s.start, nil)

  tmp.MulVec(s.coeffs, chunk)
  tmp.SubVec(tmp, s.freeCoeffs.SliceVec(s.start, s.end))
  tmp.ScaleVec(s.tau, tmp)

  return tmp
}

func (s *Solver) calcMetric(xVec *mat.VecDense) float64 {
  tmp := mat.NewVecDense(s.end-s.start, nil)
  tmp.MulVec(s.coeffs, xVec)
  tmp.SubVec(tmp, s.freeCoeffs.SliceVec(s.start, s.end))

  frac := make([]float64, 2)
  num := float64(0)
  den := float64(0)

```

```
45    for _, v := range tmp.RawVector().Data {
46      num += v * v
47    }
48
49    for _, v := range s.freeCoeffs.RawVector().Data {
50      den += v * v
51    }
52
53    s.comm.AllReduceF64(mpi.OpSum, frac, []float64{num, den})
54
55    return math.Sqrt(frac[0]) / math.Sqrt(frac[1])
56  }
```

Код программы для решение СЛАУ методом простых итераций с делением матрицы A и векторов x, b между процессами

```
func (s *SolverWithVecSeparation) FindSolution() *mat.VecDense {
  res := mat.NewVecDense(s.end-s.start, nil)
  metric := math.MaxFloat64

  prev := metric

  buff := make([]float64, s.size)
  for metric > s.eps {
    chunk := s.iterateSolution(res).RawVector().Data

    s.comm.AllGatherF64(buff, chunk)
    tmp := mat.NewVecDense(s.size, buff)

    res.SubVec(res, tmp.SliceVec(s.start, s.end))

    prev = metric
    metric = s.calcMetric(res)

    if prev < metric {
      s.tau *= -1
    }

  }

  return res
}

func (s *SolverWithVecSeparation) iterateSolution(chunk *mat.VecDense) *mat.VecDense {
  buff := make([]float64, s.size)
  s.comm.AllGatherF64(buff, chunk.RawVector().Data)

  xvec := mat.NewVecDense(s.size, buff)

  tmp := mat.NewVecDense(s.end-s.start, nil)

  tmp.MulVec(s.coeffs, xvec)
  tmp.SubVec(tmp, s.freeCoeffs)
  tmp.ScaleVec(s.tau, tmp)

  return tmp
}

func (s *SolverWithVecSeparation) calcMetric(chunk *mat.VecDense) float64 {
  buff := make([]float64, s.size)

  s.comm.AllGatherF64(buff, chunk.RawVector().Data)
```

```
47
48    xvec := mat.NewVecDense(s.size, buff)
49
50    tmp := mat.NewVecDense(s.end-s.start, nil)
51
52    tmp.MulVec(s.coeffs, xvec)
53    tmp.SubVec(tmp, s.freeCoeffs)
54
55    frac := make([]float64, 2)
56    num := float64(0)
57    den := float64(0)
58
59    for _, v := range tmp.RawVector().Data {
60      num += v * v
61    }
62
63    for _, v := range s.freeCoeffs.RawVector().Data {
64      den += v * v
65    }
66
67    s.comm.AllReduceF64(mpi.OpSum, frac, []float64{num, den})
68
69    return math.Sqrt(frac[0]) / math.Sqrt(frac[1])
70  }
```

3 Результаты сравнения производительности

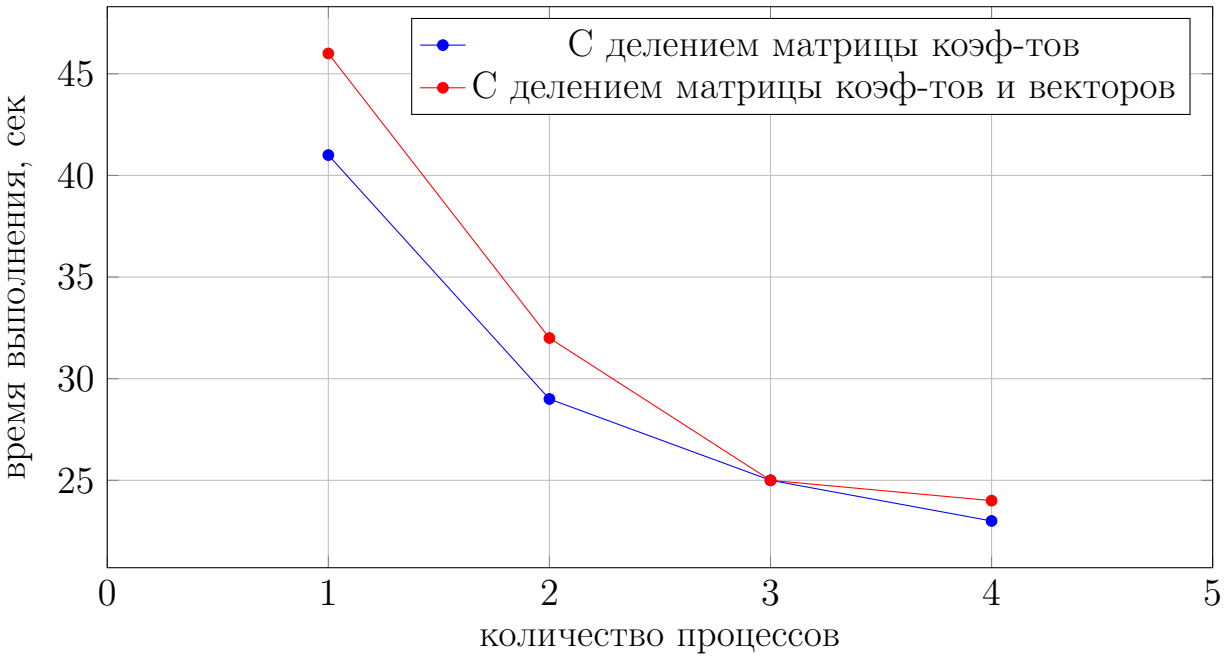

Рис. 1: зависимость времени выполнения от количества процессов(n=36000)