Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

Факультет: Информатика, искусственный интеллект и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

**Лабораторная работа №1**

**по курсу: «Разработка параллельных и распределенных программ»**

Выполнил:

Студент группы ИУ9-52Б

Терюха М.Р.

Проверил:

Царев А. С.

# Условия задачи.

Лабораторная работа №1: распараллеливание алгоритма вычисления произведения двух матриц.

* Реализовать умножение квадратных матриц стандартным алгоритмом
* Реализовать умножение квадратных матриц с помощью разбиения матрицы на подматрицы и вычисления их в разных потоках
* Сравнить результаты вычислений на совпадение (убедиться в правильности вычислений)
* Замерить и сравнить время вычисления произведения матриц 2 способами.

# **Подготовка.**

Для выполнения задачи задействованы:

* Язык golang (1.18.1 ), для распараллеливания вычислений использовались потоки и горутины.
* Утилита time, для вычисления времени потраченного на вычисление произведения матриц.
* Bash, переполнение потоков ввода\вывода (для минимизации потраченного времени на ввод\вывод данных)
* Процессор AMD Ryzen 5 5600H, для вычислений. 16 ГБ ОЗУ.

# Листинг

Для получения рандомизированной матрицы была написана программа, с помощью bash весь ее вывод был перенаправлен в файл для последующего считывания.

|  |
| --- |
| package main  import (  "bufio"  "fmt"  "math/rand"  "os"  "time"  )  func main() {  in := bufio.NewReader(os.Stdin)  out := bufio.NewWriter((os.Stdout))  var a, b, c, d int  rand.Seed(time.Now().UnixNano())  fmt.Fscanf(in, "%d %d %d %d\n", &a, &b, &c, &d)  fmt.Fprintf(out, "%d %d\n", a, b)  for ; a > 0; a-- {  for i := 0; i < b; i++ {  fmt.Fprintf(out, "%f ", rand.Float64()\*100)  }  fmt.Fprintf(out, "\n")  }  fmt.Fprintf(out, "%d %d\n", c, d)  for ; c > 0; c-- {  for i := 0; i < d; i++ {  fmt.Fprintf(out, "%f ", rand.Float64()\*100)  }  fmt.Fprintf(out, "\n")  }  out.Flush()  } |

Для вычисления матрицы стандартным способом была написана следующая программа:

|  |
| --- |
| package main  import (  "bufio"  "fmt"  "os"  )  func calculatematrix(matrix1, matrix2 [][]float64) [][]float64 {  var result [][]float64  for k := 0; k < len(matrix1); k++ {  temp := make([]float64, len(matrix2[0]), len(matrix2[0]))  result = append(result, temp)  }  for k := 0; k < len(matrix1); k++ {  for j := 0; j < len(matrix2[0]); j++ {  for i := 0; i < len(matrix2); i++ {  result[k][j] += matrix1[k][i] \* matrix2[i][j]  }  }  }  return result  }  func main() {  in := bufio.NewReader(os.Stdin)  out := bufio.NewWriter((os.Stdout))  var a, b, c, d int  var matrix1, matrix2 [][]float64  var elem float64  fmt.Fscanf(in, "%d %d\n", &a, &b)  for ; a > 0; a-- {  var temp []float64  for i := 0; i < b; i++ {  fmt.Fscanf(in, "%f", &elem)  temp = append(temp, elem)  }  matrix1 = append(matrix1, temp)  fmt.Fscanf(in, "\n")  }  fmt.Fscanf(in, "%d %d\n", &c, &d)  for ; c > 0; c-- {  var temp []float64  for i := 0; i < d; i++ {  fmt.Fscanf(in, "%f", &elem)  temp = append(temp, elem)  }  matrix2 = append(matrix2, temp)  fmt.Fscanf(in, "\n")  }  if len(matrix2) != len(matrix1[0]) {  fmt.Fprintf(out, "bad matrix\n")  out.Flush()  return  }  res := calculatematrix(matrix1, matrix2)  for i := 0; i < len(res); i++ {  for j := 0; j < len(res[0]); j++ {  fmt.Fprintf(out, "%f ", res[i][j])  }  fmt.Fprintf(out, "\n")  }  fmt.Fprintf(out, "\n")  out.Flush()  } |

Для непоследовательного вычисления матрицы была написана программа:

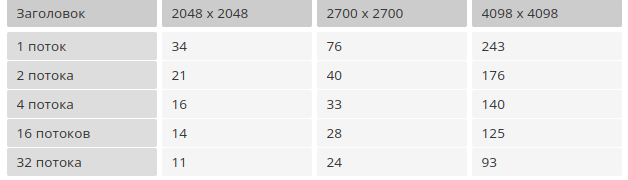
|  |
| --- |
| package main  import (  "bufio"  "fmt"  "os"  )  var result [][]float64  var out \*bufio.Writer  func summatrix(matrix1, matrix2, matrix3, matrix4, matrix5, matrix6, matrix7, matrix8 [][]float64) [][]float64 {  var result1 [][]float64  z := len(matrix1)  for k := 0; k < z\*2; k++ {  temp := make([]float64, z\*2, z\*2)  result1 = append(result1, temp)  }  for i := 0; i < z; i++ {  for j := 0; j < len(matrix1[i]); j++ {  result1[i][j] += matrix2[i][j] + matrix1[i][j]  result1[i][z+j] += matrix3[i][j] + matrix4[i][j]  result1[z+i][j] += matrix5[i][j] + matrix6[i][j]  result1[z+i][z+j] += matrix7[i][j] + matrix8[i][j]  }  }  return result1  }  func cutmatrix(in [][]float64, starta, stopa, startb, stopb int) [][]float64 {  var temp [][]float64  var z float64  for i := starta; i < stopa; i++ {  var tempp []float64  for j := startb; j < stopb; j++ {  z = in[i][j]  tempp = append(tempp, z)  }  temp = append(temp, tempp)  }  return temp  }  func printmatrix(res [][]float64) {  for i := 0; i < len(res); i++ {  for j := 0; j < len(res[i]); j++ {  fmt.Fprintf(out, "%f ", res[i][j])  }  fmt.Fprintf(out, "\n")  }  fmt.Fprintf(out, "\n")  }  func calculatematrixnew(matrix1, matrix2 [][]float64) [][]float64 {  var result1 [][]float64  for k := 0; k < len(matrix1); k++ {  temp := make([]float64, len(matrix2[0]), len(matrix2[0]))  result1 = append(result1, temp)  }  for k := 0; k < len(matrix1); k++ {  for j := 0; j < len(matrix2[0]); j++ {  for i := 0; i < len(matrix2); i++ {  result1[k][j] += matrix1[k][i] \* matrix2[i][j]  }  }  }  //printmatrix(result1)  return result1  }  func newcalculatematrix(matrix1, matrix2 [][]float64) [][]float64 {  var result1 [][]float64  for k := 0; k < len(matrix1); k++ {  temp := make([]float64, len(matrix2[0]), len(matrix2[0]))  result1 = append(result1, temp)  }  if len(matrix1) > 65 {  aa := cutmatrix(matrix1, 0, len(matrix1)/2, 0, len(matrix1[0])/2)  ab := cutmatrix(matrix1, 0, len(matrix1)/2, len(matrix1[0])/2, len(matrix1[0]))  ac := cutmatrix(matrix1, len(matrix1)/2, len(matrix1), 0, len(matrix1[0])/2)  ad := cutmatrix(matrix1, len(matrix1)/2, len(matrix1), len(matrix1[0])/2, len(matrix1[0]))  ba := cutmatrix(matrix2, 0, len(matrix2)/2, 0, len(matrix2[0])/2)  bb := cutmatrix(matrix2, 0, len(matrix2)/2, len(matrix2[0])/2, len(matrix2[0]))  bc := cutmatrix(matrix2, len(matrix2)/2, len(matrix2), 0, len(matrix2[0])/2)  bd := cutmatrix(matrix2, len(matrix2)/2, len(matrix2), len(matrix2[0])/2, len(matrix2[0]))  t1 := newcalculatematrix(aa, ba)  n1 := newcalculatematrix(ab, bc)  t2 := newcalculatematrix(aa, bb)  n2 := newcalculatematrix(ab, bd)  t3 := newcalculatematrix(ac, ba)  n3 := newcalculatematrix(ad, bc)  t4 := newcalculatematrix(ad, bd)  n4 := newcalculatematrix(ac, bb)  result1 = summatrix(t1, n1, t2, n2, t3, n3, t4, n4)  } else {  result1 = calculatematrixnew(matrix1, matrix2)  }  return result1  }  func main() {  in := bufio.NewReader(os.Stdin)  out = bufio.NewWriter((os.Stdout))  var a, b, c, d int  var matrix1, matrix2 [][]float64  var elem float64  fmt.Fscanf(in, "%d %d\n", &a, &b)  for ; a > 0; a-- {  var temp []float64  for i := 0; i < b; i++ {  fmt.Fscanf(in, "%f", &elem)  temp = append(temp, elem)  }  matrix1 = append(matrix1, temp)  fmt.Fscanf(in, "\n")  }  fmt.Fscanf(in, "%d %d\n", &c, &d)  for ; c > 0; c-- {  var temp []float64  for i := 0; i < d; i++ {  fmt.Fscanf(in, "%f", &elem)  temp = append(temp, elem)  }  matrix2 = append(matrix2, temp)  fmt.Fscanf(in, "\n")  }  if len(matrix2) != len(matrix1[0]) {  fmt.Fprintf(out, "bad matrix\n")  out.Flush()  return  }  for k := 0; k < len(matrix1); k++ {  temp := make([]float64, len(matrix2[0]), len(matrix2[0]))  result = append(result, temp)  }  printmatrix(newcalculatematrix(matrix1, matrix2))  out.Flush()  } |

Позже она была адаптированна для распределенного вычисления.

|  |
| --- |
| package main  import (  "bufio"  "fmt"  "os"  )  var result [][]float64  var out \*bufio.Writer  func summatrix(matrix1, matrix2, matrix3, matrix4, matrix5, matrix6, matrix7, matrix8 [][]float64) [][]float64 {  var result1 [][]float64  z := len(matrix1)  for k := 0; k < z\*2; k++ {  temp := make([]float64, z\*2, z\*2)  result1 = append(result1, temp)  }  for i := 0; i < z; i++ {  for j := 0; j < len(matrix1[i]); j++ {  result1[i][j] += matrix2[i][j] + matrix1[i][j]  result1[i][z+j] += matrix3[i][j] + matrix4[i][j]  result1[z+i][j] += matrix5[i][j] + matrix6[i][j]  result1[z+i][z+j] += matrix7[i][j] + matrix8[i][j]  }  }  return result1  }  func cutmatrix(in [][]float64, starta, stopa, startb, stopb int) [][]float64 {  var temp [][]float64  var z float64  for i := starta; i < stopa; i++ {  var tempp []float64  for j := startb; j < stopb; j++ {  z = in[i][j]  tempp = append(tempp, z)  }  temp = append(temp, tempp)  }  return temp  }  func printmatrix(res [][]float64) {  for i := 0; i < len(res); i++ {  for j := 0; j < len(res[i]); j++ {  fmt.Fprintf(out, "%f ", res[i][j])  }  fmt.Fprintf(out, "\n")  }  fmt.Fprintf(out, "\n")  }  func calculatematrixnew(matrix1, matrix2 [][]float64) [][]float64 {  var result1 [][]float64  for k := 0; k < len(matrix1); k++ {  temp := make([]float64, len(matrix2[0]), len(matrix2[0]))  result1 = append(result1, temp)  }  for k := 0; k < len(matrix1); k++ {  for j := 0; j < len(matrix2[0]); j++ {  for i := 0; i < len(matrix2); i++ {  result1[k][j] += matrix1[k][i] \* matrix2[i][j]  }  }  }  //printmatrix(result1)  return result1  }  func newcalculatematrix(rec int, matrix1, matrix2 [][]float64, c chan [][]float64) {  var result1 [][]float64  for k := 0; k < len(matrix1); k++ {  temp := make([]float64, len(matrix2[0]), len(matrix2[0]))  result1 = append(result1, temp)  }  if rec < 1 {  aa := cutmatrix(matrix1, 0, len(matrix1)/2, 0, len(matrix1[0])/2)  ab := cutmatrix(matrix1, 0, len(matrix1)/2, len(matrix1[0])/2, len(matrix1[0]))  ac := cutmatrix(matrix1, len(matrix1)/2, len(matrix1), 0, len(matrix1[0])/2)  ad := cutmatrix(matrix1, len(matrix1)/2, len(matrix1), len(matrix1[0])/2, len(matrix1[0]))  ba := cutmatrix(matrix2, 0, len(matrix2)/2, 0, len(matrix2[0])/2)  bb := cutmatrix(matrix2, 0, len(matrix2)/2, len(matrix2[0])/2, len(matrix2[0]))  bc := cutmatrix(matrix2, len(matrix2)/2, len(matrix2), 0, len(matrix2[0])/2)  bd := cutmatrix(matrix2, len(matrix2)/2, len(matrix2), len(matrix2[0])/2, len(matrix2[0]))  t1, t2, t3, t4, n1, n2, n3, n4 := make(chan [][]float64), make(chan [][]float64), make(chan [][]float64), make(chan [][]float64), make(chan [][]float64), make(chan [][]float64), make(chan [][]float64), make(chan [][]float64)  go newcalculatematrix(rec+1, aa, ba, t1)  go newcalculatematrix(rec+1, ab, bc, n1)  go newcalculatematrix(rec+1, aa, bb, t2)  go newcalculatematrix(rec+1, ab, bd, n2)  //N1, N2, N3, N4 := <-t1, <-n1, <-t2, <-n2  go newcalculatematrix(rec+1, ac, ba, t3)  go newcalculatematrix(rec+1, ad, bc, n3)  go newcalculatematrix(rec+1, ad, bd, t4)  go newcalculatematrix(rec+1, ac, bb, n4)  result1 = summatrix(<-t1, <-n1, <-t2, <-n2, <-t3, <-n3, <-t4, <-n4)  } else {  result1 = calculatematrixnew(matrix1, matrix2)  }  c <- result1  }  func main() {  in := bufio.NewReader(os.Stdin)  out = bufio.NewWriter((os.Stdout))  var a, b, c, d int  var matrix1, matrix2 [][]float64  var elem float64  fmt.Fscanf(in, "%d %d\n", &a, &b)  for ; a > 0; a-- {  var temp []float64  for i := 0; i < b; i++ {  fmt.Fscanf(in, "%f", &elem)  temp = append(temp, elem)  }  matrix1 = append(matrix1, temp)  fmt.Fscanf(in, "\n")  }  fmt.Fscanf(in, "%d %d\n", &c, &d)  for ; c > 0; c-- {  var temp []float64  for i := 0; i < d; i++ {  fmt.Fscanf(in, "%f", &elem)  temp = append(temp, elem)  }  matrix2 = append(matrix2, temp)  fmt.Fscanf(in, "\n")  }  if len(matrix2) != len(matrix1[0]) {  fmt.Fprintf(out, "bad matrix\n")  out.Flush()  return  }  for k := 0; k < len(matrix1); k++ {  temp := make([]float64, len(matrix2[0]), len(matrix2[0]))  result = append(result, temp)  }  res := make(chan [][]float64)  go newcalculatematrix(0, matrix1, matrix2, res)  printmatrix(<-res)  out.Flush()  } |

# Результаты тестирования

Результаты тестирования при различных матрицах были занесены в таблицу, время измерялось утилитой time.



Все результаты вычислений проверялись с помощью утилиты diff (проверка правильности вычислений).

