Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №1
«Распараллеливание алгоритма вычисления произведения двух матриц»
по курсу: «Разработка параллельных и распределенных программ»

Выполнил:
Студент группы ИУ9-51Б
Киселев К.А.

Проверил:
Царев А.С.

Москва, 2023

Цель: разработать алгоритм умножения матриц в нескольких потоках, измерить его производительность и сравнить с обычным алгоритмом.

Условие:

Две квадратные матрицы A и B размерности n сначала перемножить стандартным алгоритмом:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

для получения матрицы C той же размености. Замерить время вычисления, сравнить с временем при вычислении элементов матрицы C не по строкам, а по столбцам. Размер матриц подобрать таким образом, чтобы время выполнения на вашей машине было не слишком непоказательно малым (меньше нескольких минут), но и не чересчур большим (несколько часов). Использовать библиотечные функции для вычисления произведений матриц нельзя. Затем конечную матрицу C условно разделить на примерно равные прямоугольные подматрицы и распараллелить программу таким образом, чтобы каждый поток занимался вычислением своей подматрицы. Матрицы A и B для этого разделить на примерно равные группы строк и столбцов соответственно. Сделать для разного количества потоков (разных разбиений), также замерить время вычисления, сравнить с вычислениями стандартным алгоритмом. Также по окончании вычислений сравнивать получившуюся матрицу с той, что была вычислена стандартным алгоритмом, для проверки правильности вычислений (проверка во время выполнения задачи не входит)

Характеристики устройства

- **CPU**: AMD Ryzen 3 5300U
  - 4 ядра
  - 8 потоков
  - Базовая частота: 2.6 ГГц
- **GPU**: AMD ATI 03:00.0 Lucienne
- **Memory** 7262M

Решение

Функция умножения матриц

```
func MulMatrix(lhs, rhs [][]int, lhsRows, lhsCols, rhsRows, rhsCols int,
 ↪  res [][]int) {
    for i := 0; i < lhsRows; i++ {
        for j := 0; j < rhsCols; j++ {
            for k := 0; k < lhsCols; k++ {
                res[i][j] += lhs[i][k] * rhs[k][j]
            }
        }
    }
}
```

Распараллеливание умножения матриц

```
func ConcurrentMul(lhs, rhs [][]int, size int, chunksCount int, res
 ↪  [][]int, wg *sync.WaitGroup) {
    chunkSize := size / chunksCount
    for i := chunkSize; i <= size; i += chunkSize {
        rightBoundI := i
        if size-i < chunkSize && i != size {
            rightBoundI = size
        }

        for j := chunkSize; j <= size; j += chunkSize {
            rightBoundJ := j
```

```
            if size-j < chunkSize && j != size {
                rightBoundJ = size
            }

            lhs := SplitByRows(lhs, i-chunkSize, rightBoundI)
            rhs := SplitByCols(rhs, j-chunkSize, rightBoundJ)
            tmp := SplitByCols(res[i-chunkSize:rightBoundI], j-chunkSize,
 ↪  rightBoundJ)

            wg.Add(1)
            go func(i, j int) {
                defer wg.Done()
                MulMatrix(lhs, rhs, rightBoundI-i+chunkSize, size, size,
 ↪  rightBoundJ-j+chunkSize, tmp)
            }(i, j)

        }
    }
}

func SplitByCols(matrix [][]int, i, j int) [][]int {
    res := make([][]int, len(matrix))
    for k := range matrix {
        res[k] = matrix[k][i:j]
    }
    return res
}

func SplitByRows(matrix [][]int, i, j int) [][]int {
    return matrix[i:j]
}
```

Тесты

Код

```
func TestConncurentMul(t *testing.T) {
    size := 10
    matrixLhs := make([][]int, size)
    matrixRhs := make([][]int, size)

    for i := 0; i < size; i++ {
        matrixLhs[i] = make([]int, size)
        matrixRhs[i] = make([]int, size)
        for j := 0; j < size; j++ {
            matrixLhs[i][j] = rand.Intn(size)
            matrixRhs[i][j] = rand.Intn(size)
        }
    }

    wg := &sync.WaitGroup{}

    connResult := make([][]int, size)
    defResult := make([][]int, size)

    for i := 0; i < size; i++ {
        connResult[i] = make([]int, size)
        defResult[i] = make([]int, size)
    }

    t.Run("2 chunks", func(t *testing.T) {
```

```
        matrixops.ConcurrentMul(matrixLhs, matrixRhs, size, 2,
 ↪  connResult, wg)
        wg.Wait()
        matrixops.MulMatrix(matrixLhs, matrixRhs, size, size, size, size,
 ↪  defResult)
        for i := 0; i < size; i++ {
            for j := 0; j < size; j++ {
                assert.Equal(t, defResult[i][j], connResult[i][j])
            }
        }
    })

    t.Run("3 chunks", func(t *testing.T) {
        matrixops.ConcurrentMul(matrixLhs, matrixRhs, size, 3,
 ↪  connResult, wg)
        wg.Wait()
        matrixops.MulMatrix(matrixLhs, matrixRhs, size, size, size, size,
 ↪  defResult)
        for i := 0; i < size; i++ {
            for j := 0; j < size; j++ {
                assert.Equal(t, defResult[i][j], connResult[i][j])
            }
        }
    })
}
```

Результаты

```
$ go test -v ./...
=== RUN   TestConncurentMul
=== RUN   TestConncurentMul/#00
=== RUN   TestConncurentMul/#01
--- PASS: TestConncurentMul (0.00s)
    --- PASS: TestConncurentMul/#00 (0.00s)
    --- PASS: TestConncurentMul/#01 (0.00s)
PASS
ok      github.com/t1d333/concurrency/lab1/internal/matrixops   0.005s
```

Производительность

Тест производительности

```
func BenchmarkConcurrentMul(b *testing.B) {
    size := 1000
    lhs := make([][]int, size)
    rhs := make([][]int, size)
    res := make([][]int, size)
    for i := 0; i < size; i++ {
        res[i] = make([]int, size)
        lhs[i] = make([]int, size)
        rhs[i] = make([]int, size)
        for j := 0; j < size; j++ {
            lhs[i][j] = rand.Intn(size)
            rhs[i][j] = rand.Intn(size)
        }
    }

    b.Run("default mul", func(b *testing.B) {
        b.ResetTimer()
        matrixops.ConcurrentMul(lhs, rhs, size, 1, res)
    })
```

```
    b.Run("two chunks", func(b *testing.B) {
        b.ResetTimer()
        matrixops.ConcurrentMul(lhs, rhs, size, 2, res)
    })

    b.Run("four chunks", func(b *testing.B) {
        b.ResetTimer()
        matrixops.ConcurrentMul(lhs, rhs, size, 4, res)
    })

    b.Run("six chunks", func(b *testing.B) {
        b.ResetTimer()
        matrixops.ConcurrentMul(lhs, rhs, size, 4, res)
    })
}
```

Результат

```
$ make bench
goos: linux
goarch: amd64
pkg: github.com/t1d333/concurrency/lab1/internal/matrixops
cpu: AMD Ryzen 3 5300U with Radeon Graphics
BenchmarkConcurrentMul/default_mul-8                 1 14292385860 ns/op
BenchmarkConcurrentMul/two_chunks-8                  1 5952078100 ns/op
BenchmarkConcurrentMul/four_chunks-8                 1 3902838566 ns/op
BenchmarkConcurrentMul/six_chunks-8                  1 3835036261 ns/op
BenchmarkConcurrentMul/eight_chunks-8                1 3784158661 ns/op
PASS
ok      github.com/t1d333/concurrency/lab1/internal/matrixops   35.593s
```

Кол-во потоков	время, с
1	14,3
4	6,0
16	3,9
36	3,8
64	3,8