Relatório 1: Benchmark EP Em Golang

Kevin S. Pereira, Thomazio Giacobbe

https://github.com/thomaziogiacobbe/NPB-Golang

Benchmark EP em Golang

- Baseado na implementação em C++ de Dalvan Griebler et al.
 - NPB-OMP; implementação em OpenMP
 - o https://github.com/GMAP/NPB-CPP
- Adaptações utilizando features de multithreading do Golang
 - o goroutines (criação de tarefas)
 - o channels (passagem de mensagem)
- Outras adaptações da linguagem
 - o time (biblioteca para lidar com tempo)

Bibliotecas

- math
 - Funções matemáticas
- runtime
 - Informações da CPU
- strconv
 - Conversões entre strings e outros tipos
- go-pretty (<u>https://github.com/jedib0t/go-pretty</u>)
 - Biblioteca third party para formatação dos resultados em interface

Bloco paralelo

- Função separada da main, parallelEP
- Variáveis compartilhadas declaradas no escopo da função parallelEP
- Variáveis privadas declaradas no escopo da função anônima

```
func parallelEP(
   np int,
    an float64,
    sx *float64,
    sy *float64,
    q []float64,
    tt *time.Duration,
    const k_{offset} = -1
    qTemp := [NQ]float64{}
    var result ResultData
    resultChn := make(chan ResultData, np)
    start := time.Now()
   defer getExecTime(tt, &start)
    for k := 1; k <= np; k++ \{
       go func(k int) {
            var (
                sxThis, syThis = 0.0, 0.0
                t1, t2, t3, t4, x1, x2 float64
                kk, ik, l
                                       = [NQ]float64{}, [NK_PLUS]float64{}
               qq, x
                resultThis
                                      ResultData
```

Bloco paralelo: Go

- Comando go instanciando np tarefas da função anônima
- Diferente da implementação original que usa a cláusula reduction no laço, as goroutines processam pelas variáveis privadas e comunicam seus resultados pelo channel

```
for k := 1; k <= np; k++ {
   go func(k int) {
       var (
           sxThis, syThis = 0.0, 0.0
           t1, t2, t3, t4, x1, x2 float64
           kk, ik, l
                                  = [NQ]float64{}, [NK_PLUS]float64{}
           qq, x
           resultThis
                                  ResultData
       kk = k_offset + k
       t1 = S
       t2 = an
       for i := 0; i <= 100; i++ {
           ik = kk / 2
           if (2 * ik) != kk {
               t3 = npb.Randlc(&t1, t2)
           if ik == 0 {
               break
           t3 = npb.Randlc(\&t2, t2)
           kk = ik
       npb.Vranlc(2*NK, &t1, A, x[:])
```

Bloco paralelo: Channels

- Declarado como variável compartilhada o channel para comunicação com buffer de tamanho np
- As np threads calculam seus resultados, salvam na struct, enviam para o channel e terminam

```
type ResultData struct {
    sxResult float64
    syResult float64
    qResult [NQ]float64
}
```

```
for i := 0; i < NK; i++ {
        x1 = 2.0*x[2*i] - 1.0
        x2 = 2.0*x[2*i+1] - 1.0
       t1 = math.Pow(x1, 2) + math.Pow(x2, 2)
        if t1 <= 1.0 {
            t2 = math.Sqrt(-2.0 * math.Log(t1) / t1)
            t3 = x1 * t2
            t4 = x2 * t2
            l = int(math.Max(math.Abs(t3), math.Abs(t4)))
            qq[l] += 1.0
            sxThis += t3
            syThis += t4
    resultThis.sxResult = sxThis
    resultThis.syResult = syThis
    resultThis.qResult = qq
    resultChn <- resultThis
}(k)
```

Bloco paralelo: Channels

 A thread principal na função parallelEP bloqueia aguardando resultado do channel. Quando recebe, ele lê da struct e acumula o resultado em suas variáveis

```
type ResultData struct {
    sxResult float64
    syResult float64
    qResult [NQ]float64
}
```

```
for k := 1; k <= np; k++ {
    result = <-resultChn
    *sx += result.sxResult
    *sy += result.syResult
    qTemp = result.qResult
    for j := range q {
        q[j] += qTemp[j]
```

Bloco paralelo: Observações

- Optamos por channel e não mutex para evitar bloqueios e sincronismos entre as threads
- O buffer do channel é importante para evitar bloqueios parecidos com mutex
- Explorar comunicação assíncrona

```
for i := 0; i < NK; i++ {
        x1 = 2.0 \times x[2 \times i] - 1.0
        x2 = 2.0*x[2*i+1] - 1.0
        t1 = math.Pow(x1, 2) + math.Pow(x2, 2)
        if t1 <= 1.0 {
            t2 = math.Sqrt(-2.0 * math.Log(t1) / t1)
            t3 = x1 * t2
            t4 = x2 * t2
            l = int(math.Max(math.Abs(t3), math.Abs(t4)))
            qq[l] += 1.0
            sxThis += t3
            syThis += t4
    reductionMutex.Lock()
    *sx += sxThis
    *sy += syThis
    reductionMutex.Unlock()
    arrayResultMutex.Lock()
    for i := 0; i < NQ; i++ {
        q[i] += qq[i]
    arrayResultMutex.Unlock()
}(k)
```

Resultados

- Mesmos resultados numéricos da implementação em C++ de Dalvan Griebler et al.
- Resultados diferentes da implementação original em Fortran (RNR Technical Report RNR-94-007)
- Desempenho foi aproximadamente 2 vezes pior do que a implementação em C++ e 6 vezes pior do que a implementação em Fortran

 Os benchmarks foram executados em um Intel i7 7700, que possui 4 núcleos físicos e 8 threads

$$Mop/s = rac{2^{M+1} \cdot 1 imes 10^6}{t}$$

go build run NAS Parallel Benchmarks 4.1 Parallel Golang version - EP Benchmark Number of random numbers generated: 3.3554432e+07 EP BENCHMARK RESULTS CPU Time No. Gaussian Pairs 1.3176389e+07 -3247.8346520346136 -6958.407078382822 6.140517e+06 5.8653e+06 1.100361e+06 EP BENCHMARK COMPLETED 33554432 Number of available threads Number of iterations Time in seconds 417.528852ms Mop/s total 80.36434330051999 Operation type Random numbers generated NPB Version

Process finished with the exit code 0

Resultados: S

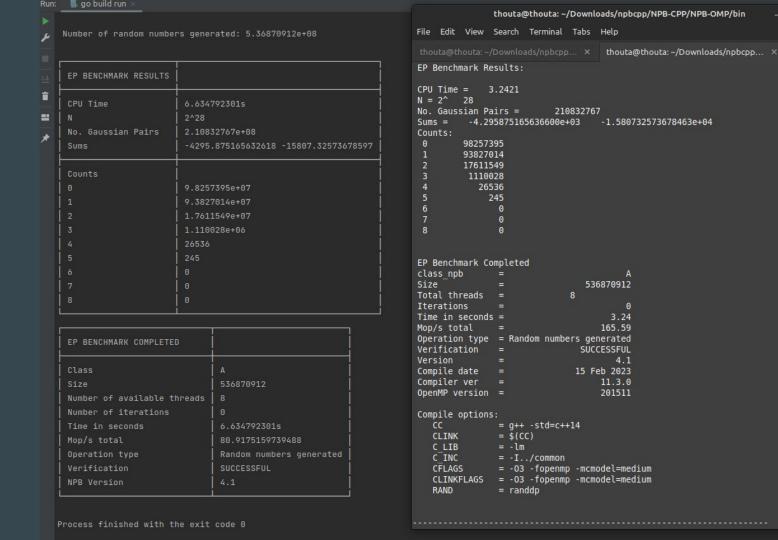
```
EP Benchmark Results:
CPU Time =
             0.2448
N = 2^{2}
No. Gaussian Pairs =
                           13176389
Sums =
         -3.247834652034389e+03
                                   -6.958407078382825e+03
Counts:
         6140517
         5865300
          1100361
            68546
             1648
EP Benchmark Completed
class npb
Size
                                 33554432
Total threads =
Iterations
                                     0.24
Time in seconds =
Mop/s total
                                   137.07
Operation type = Random numbers generated
Verification =
                               SUCCESSFUL
Version
                                     4.1
Compile date =
                              15 Feb 2023
Compiler ver =
                                   11.3.0
OpenMP version =
                                   201511
Compile options:
               = g++ -std=c++14
   CLINK
               = $(CC)
   C LIB
   C INC
               = -I../common
               = -03 -fopenmp -mcmodel=medium
   CFLAGS
               = -03 -fopenmp -mcmodel=medium
   CLINKFLAGS
   RAND
               = randdp
```

thouta@thouta: ~/Downloads/npbcpp/NPB-CPP/NPB-OMP/bin

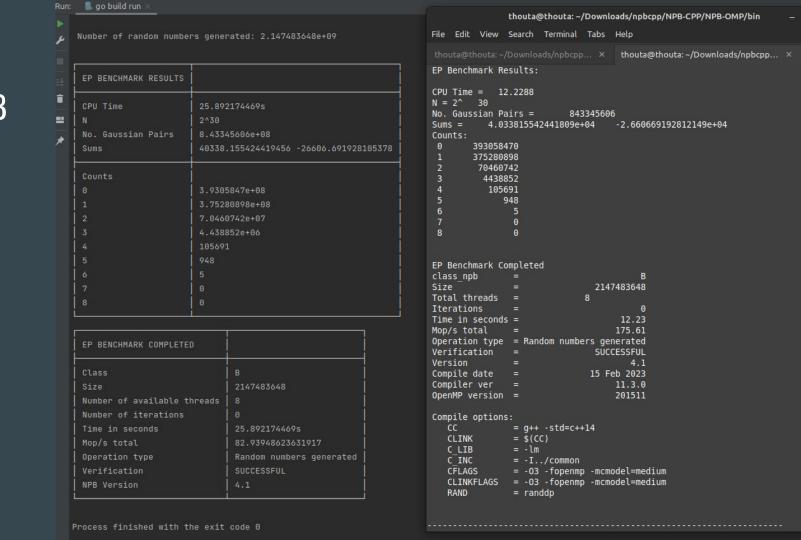
thouta@thouta: ~/Downloads/npbcpp... × thouta@thouta: ~/Downloads/npbcpp... ×

File Edit View Search Terminal Tabs Help

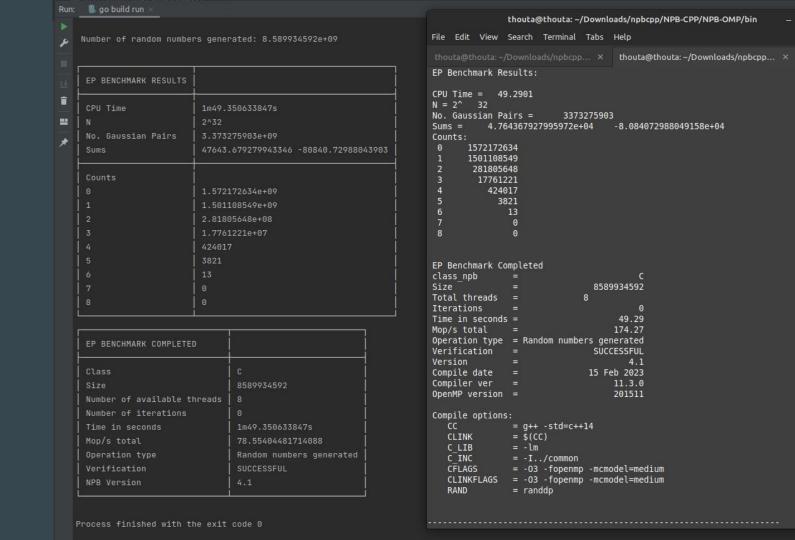
Resultados: A



Resultados: B



Resultados: C



Relatório 1: Benchmark EP Em Golang

Kevin S. Pereira, Thomazio Giacobbe

https://github.com/thomaziogiacobbe/NPB-Golang