Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Curso de Bacharelado em Ciência da Computação

Disciplina de Banco de Tópicos Especiais em Computação VII

Implementações NPB em Golang

Integrantes

Ariam Bartsch
Carlos Calage
João Belmonte

Pelotas, 2023

1. Introdução

O seguinte trabalho surgiu da ideia de explorar novas linguagens, as quais fornecem meios para explorar a concorrência e paralelismo, visto que as diversas soluções para a linguagem C, amplamente utilizada na introdução para o curso de ciência da computação e na indústria no geral por conta de sua robustez e velocidade, ainda carecem de melhorias na questão de facilidade de uso. Por conta disso, para a cadeira optativa, decidimos explorar a linguagem de programação Go e para testar seus limites e benefícios em relação à linguagem C, por meio da implementação de algumas instâncias do problema NPB, desenvolvido pela NASA para explorar exaustivamente os recursos.

1.1 Golang

Golang é uma linguagem de programação compilada, simultânea, com coleta de lixo e tipada estaticamente. Foi criada em 2007 pela empresa Google, após frustrações com a complexidade excessiva de outras linguagens e, portanto, modelada para ter foco em concorrência, escalabilidade e facilidade de manutenção, mas mantendo uma semelhança com C. Go é geralmente usada em back-ends de servidores web, atendendo grande quantia de requisições simultâneas — algo que frameworks como Node Js não conseguem alcançar de forma muito eficiente.

O paralelismo em Go se dá através de goroutines, que nada mais são do que funções invocadas com a palavra reservada “go” antes da chamada da assinatura da função. Essas goroutines são então executadas de forma concorrente, tendo seu escalonamento gerenciado pelo runtime do go — e não pelo sistema operacional. Dito isso, cada goroutine consome menos memória que uma thread equivalente e não é necessariamente mapeada a uma thread do sistema operacional: por exemplo, o lançamento de 100 goroutines não significa que 100 threads serão lançadas e gerenciadas pelo sistema.

2. Implementações

Neste trabalho, foram implementados dois problemas do benchmark NPB: Embarrassingly Parallel (EP) e Integer Sort (IS). No problema EP, são geradas estatísticas com base em uma grande quantidade de números aleatórios gerados. Esse problema possui grande capacidade de paralelização, o que leva a seu nome. No problema IS, o algoritmo deve ordenar um vetor aleatório de inteiros, atribuindo ranques aos valores do vetores de forma a ordenar o vetor com base nesses ranques.

2.1 EP

O algoritmo Embarrassingly Parallel foi implementado de forma paralela e serial. A versão paralela utiliza goroutines para criação do paralelismo, waitGroups para espera de goroutines, e

Mutex para evitar condição de corrida. A comunicação entre goroutines se dá por meio de memória compartilhada.

A implementação paralela do problema EP usou goroutines para simular a estrutura Parallel For do OpenMP — já que a linguagem não oferece nativamente essa estrutura. São lançados um número de goroutines igual ao número definido de processadores, cada qual realizando operações sobre um número estático de elementos do vetor. A estrutura do paralelismo pode ser vista na figura 1.

```
wg.Add(int(cores))
for c := 0; c < int(cores); c++ {
    go func(lc int) {
        var sxl float64 = 0.0
        var syl float64 = 0.0
        var qq = make([]float64, NQ)
        for jj := lc * slicesize; jj < (lc*slicesize + slicesize); jj++ {…
        }

        m2.Lock()
        sx = sxl + sx
        sy = syl + sy

        for i := 0; i <= NQ-1; i++ {
            q[i] = q[i] + qq[i]
        }
        m2.Unlock()

        defer wg.Done()

    }(c)
}
```

Figura 1: Código EP resumido

Cada laço for dentro de uma goroutine possui um limite inferior e superior de operação, que não se sobrepõem com outros goroutines. Isso simula o chunk de um Parallel For. As operações entre m2.Lock() e m2.Unlock() são operações que acumulam os resultados. Como a memória dessas variáveis é compartilhada, é necessário o uso de mutex.

A diferença entre essa implementação e a implementação serial é miníma, sendo que a implementação serial pode ser simulada no código paralelo com a definição de uma única goroutine cujo for encompassa o ínicio e fim do vetor. Mas a versão serial dispensa do uso de mutex, goroutines e waitGroups, o que evita o overhead associado a essas operações.

2.2 IS

O código em C++ de Dalvan Griebler, implementado com OpenMP, apresenta duas implementações: uma usando a técnica de ordenação por buckets e outra que não o usa. A implementação do IS em Go foi feita sem a utilização de buckets, em vez disso se baseando na ordenação serial de Dalvan. A criação do vetor a ser ordenado utilizou a mesma técnica de simulação de parallel for utilizada no EP — mas aqui, as operações de limite inferior e superior são calculadas dentro da goroutine

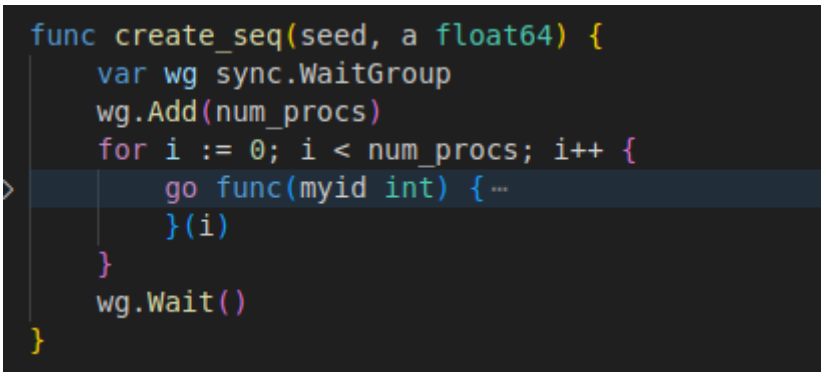

Figura 2: criação paralela do vetor a ser ordenado

As funções rank e full_verify foram traduzidas de forma um-para-um de C++ para Go, tendo como base a versão OpenMP do código sem buckets. Essas funções se mostraram impossíveis de paralelizar sem um grande retrabalho de sua lógica devido a dependência entre dados — como pode ser visto no segundo laço da figura 3, onde a posição i depende da posição i-1.

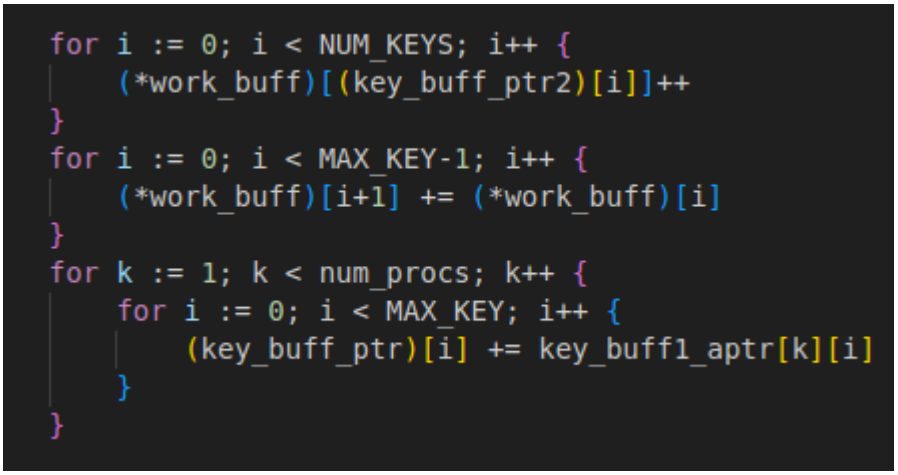

Figura 3: dependência entre dados na ordenação sem buckets.

3. Comparação de desempenho

Todas as execuções ocorreram em um computador com processador i5 de sétima geraçõ, 16gb de RAM e luz LED vermelha. Foram realizadas 30 execuções intercaladas, e os gráficos de caixa foram criados utilizando Python. O código go foi compilado com go build geral — ou seja, o programa recebe como argumento de execução a classe sobre a qual ele irá trabalhar. O código C++, por outro lado, é dedicado a executar uma única classe, definida no momento de compilação — algo que pode interferir na diferença final de tempos.

A estrutura do shell script que realiza a compilação e execução das implementações pode ser vista na figura 4. A compilação é feita antes da execução, e a variável $KERNEL é definida por entrada do usuário.

```
# Running All The Classes
for CLASS in S W A; do
    echo $CLASS

    for i in $(seq 1 $N); do
        echo $i

    echo C
        start=$(date +%s%N)
        ./GMAP/NPB-OMP/bin/is.$CLASS
        end=$(date +%s%N)
        time=$((end-start))
        echo "${KERNEL},${CLASS},C,${time}" >> results/log.csv

        echo GO
        start=$(date +%s%N)
        ./${KERNEL}/${KERNEL} ${CLASS} ${NUM_CORES}
        end=$(date +%s%N)
        time=$((end-start))
        echo "${KERNEL},${CLASS},GO,${time}" >> results/log.csv
    done
done
```

Figura 4: shell script de execução

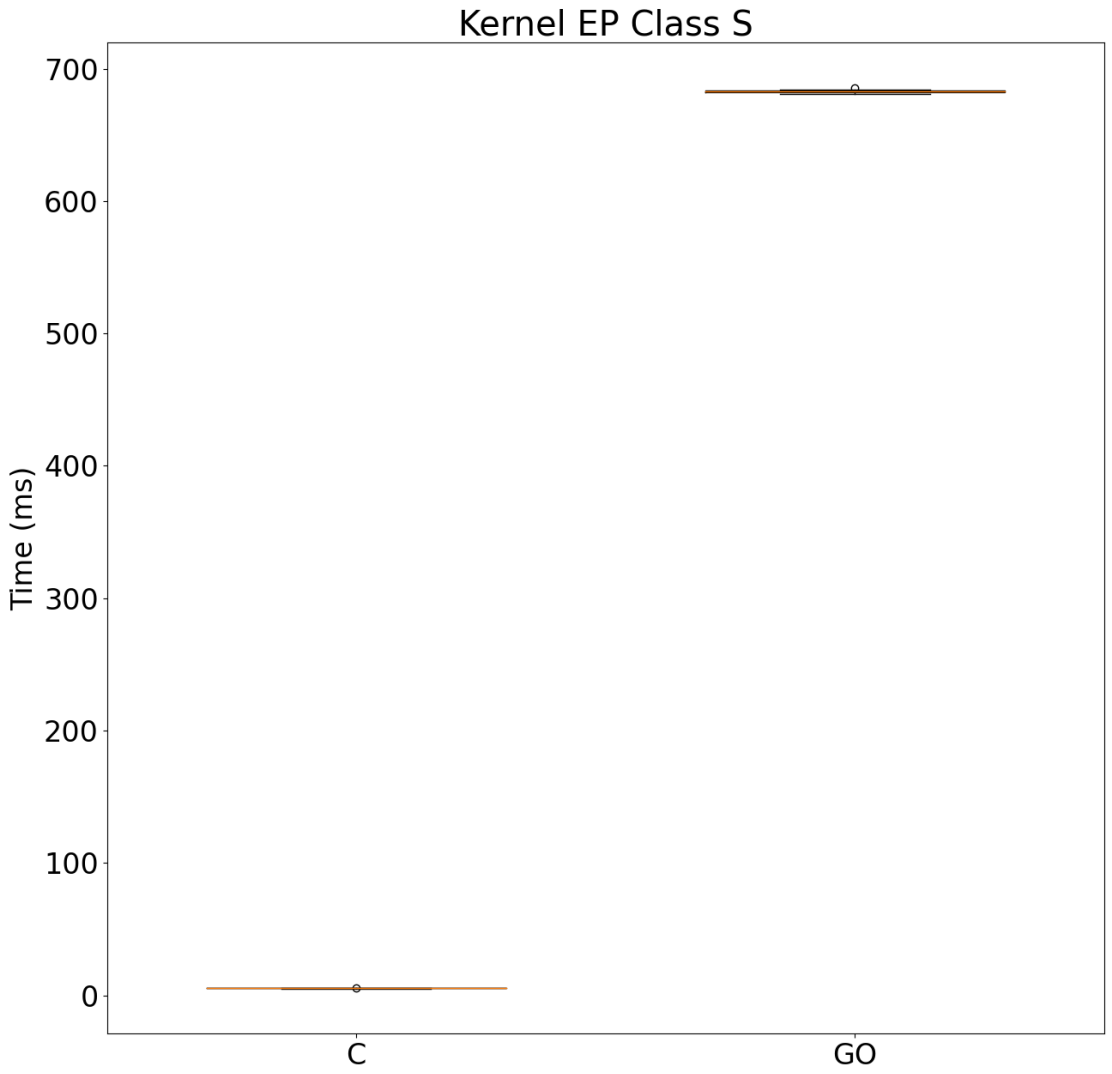

figura 5: Comparação entre EP em Go e EP em c++, classe S

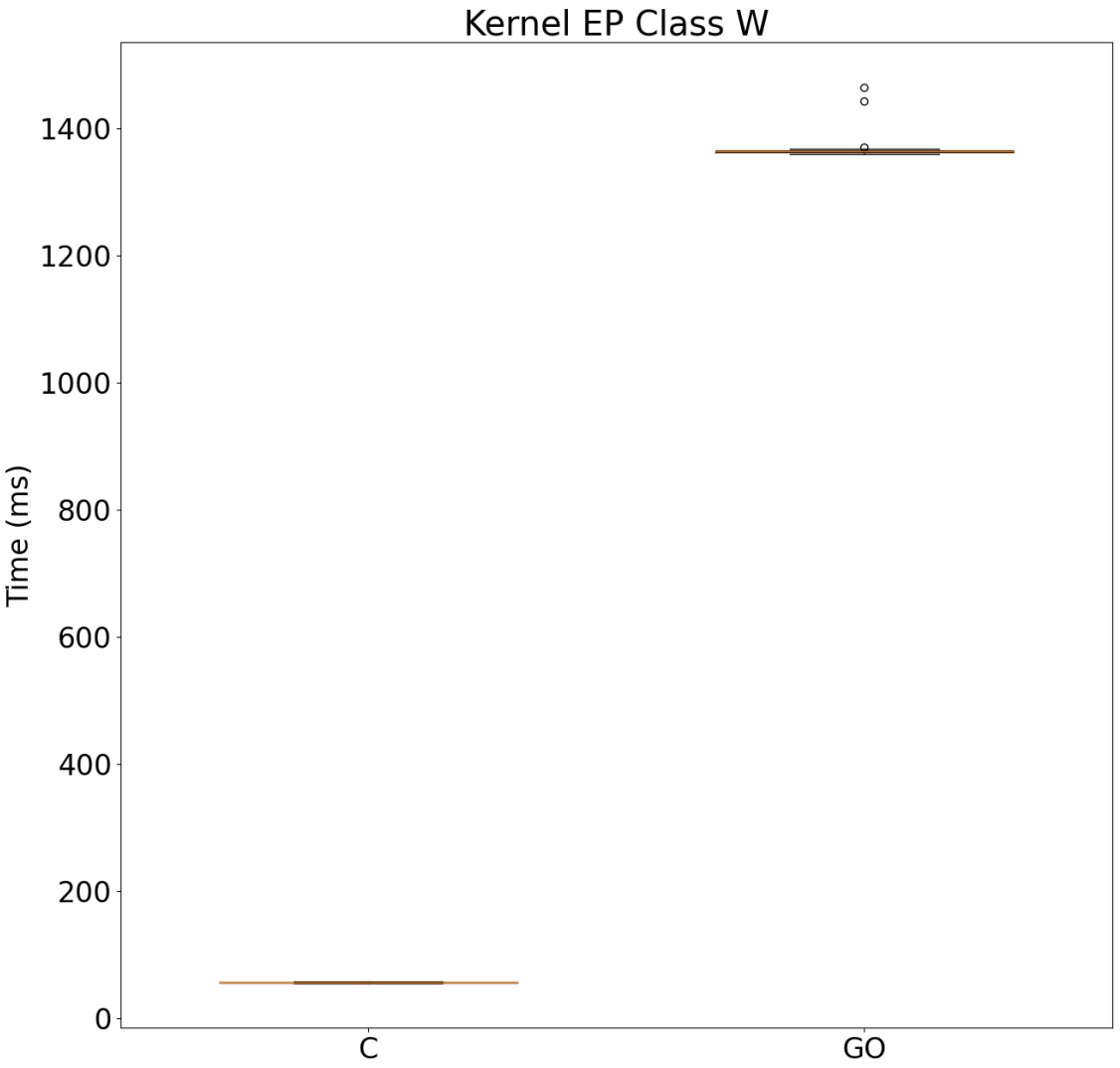

figura 6: Comparação entre EP em Go e EP em c++, classe W

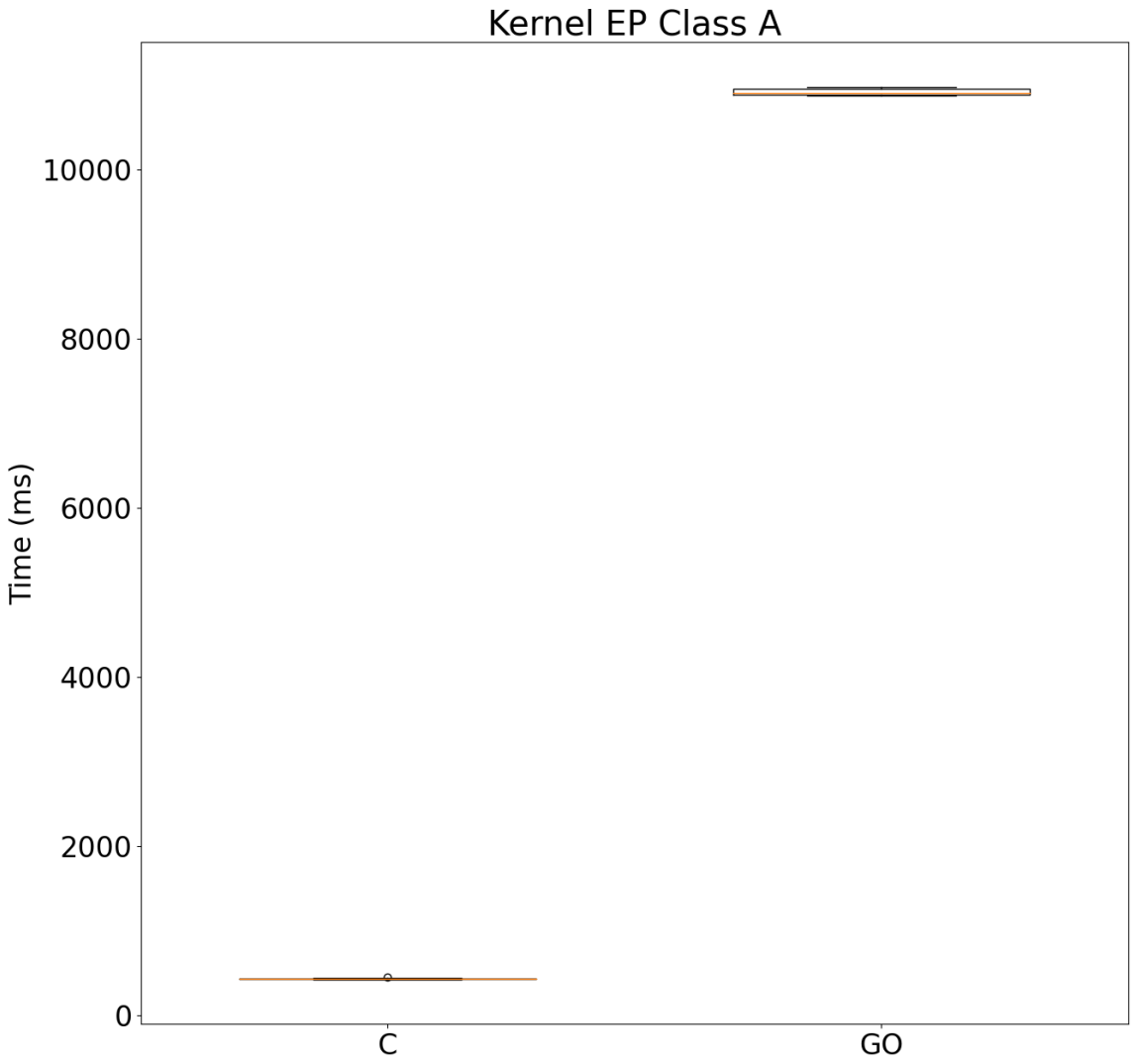

figura 7: Comparação entre EP em Go e EP em c++, classe A

As figuras 4, 5 e 6 mostram a comparação entre o código C++ e Go usando as classes S, W e A. Em todos os casos, o código se em Go se mostrou mais lento que o código C. Aqui não é mostrada a comparação entre as versões seriais do algoritmo EP — mas o código C++ também se mostrou ser o mais veloz entre as duas versões.

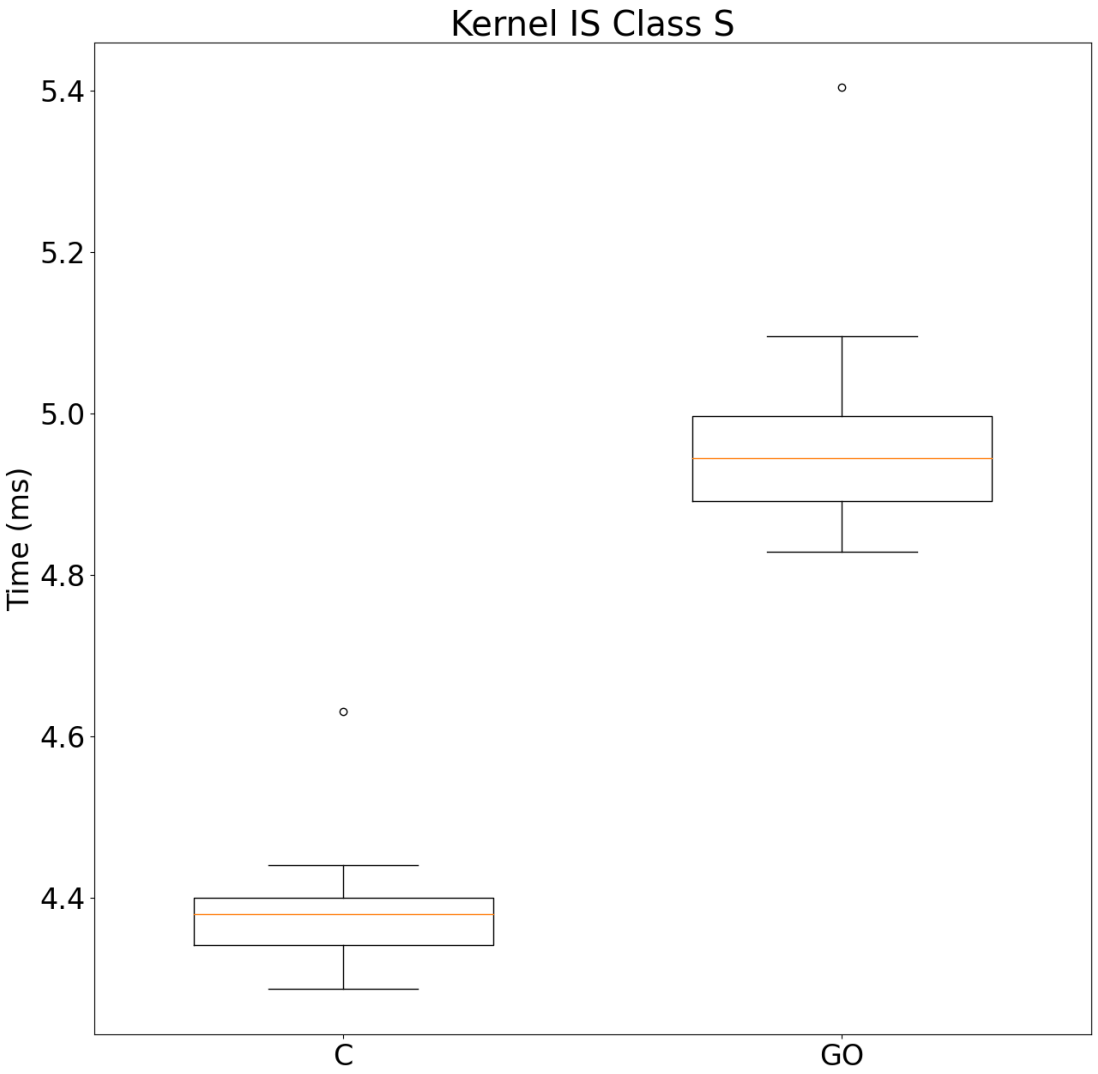

figura 8: Comparação entre IS em Go e IS em c++, classe S

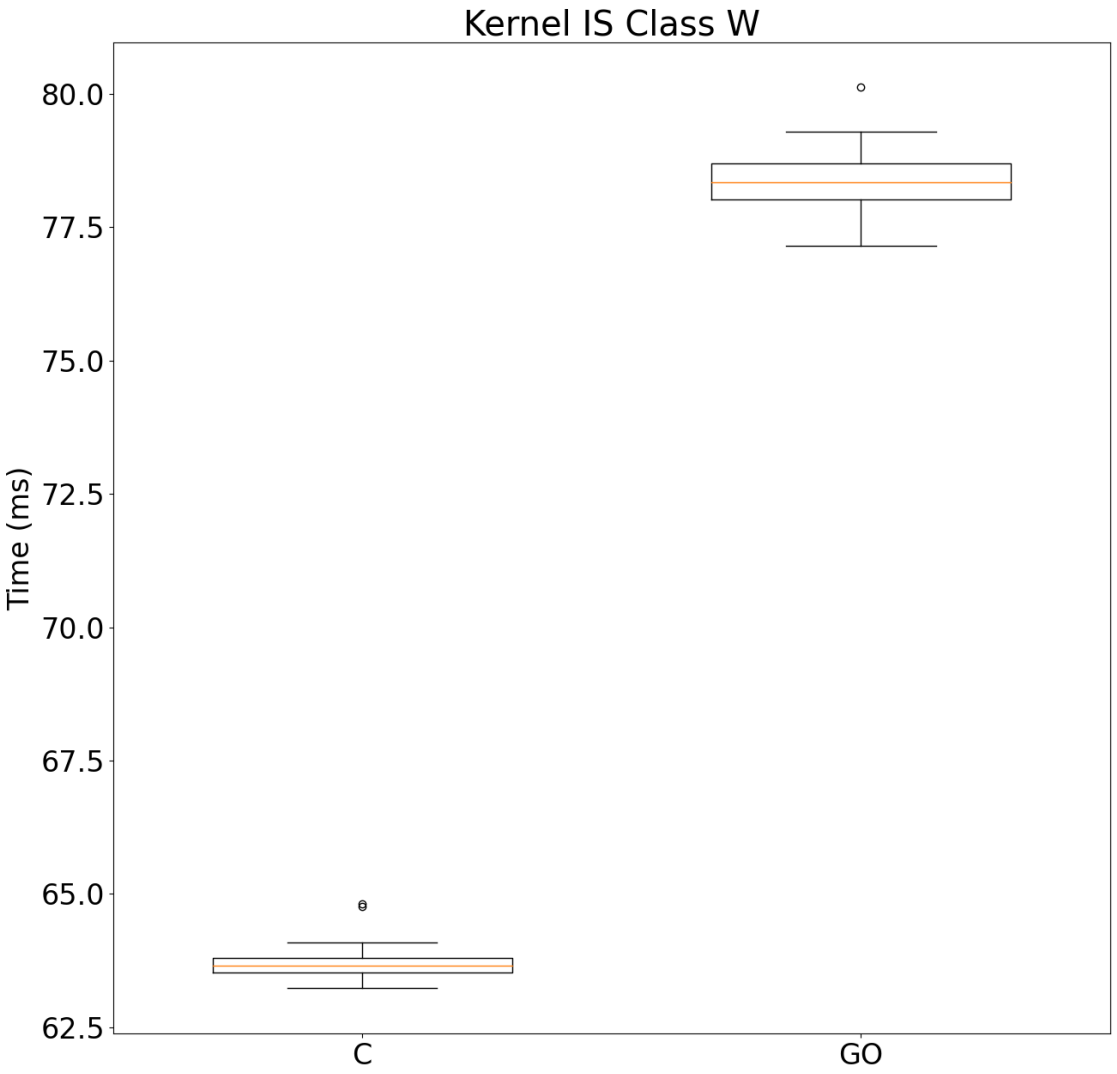

figura 9: Comparação entre IS em Go e IS em c++, classe W

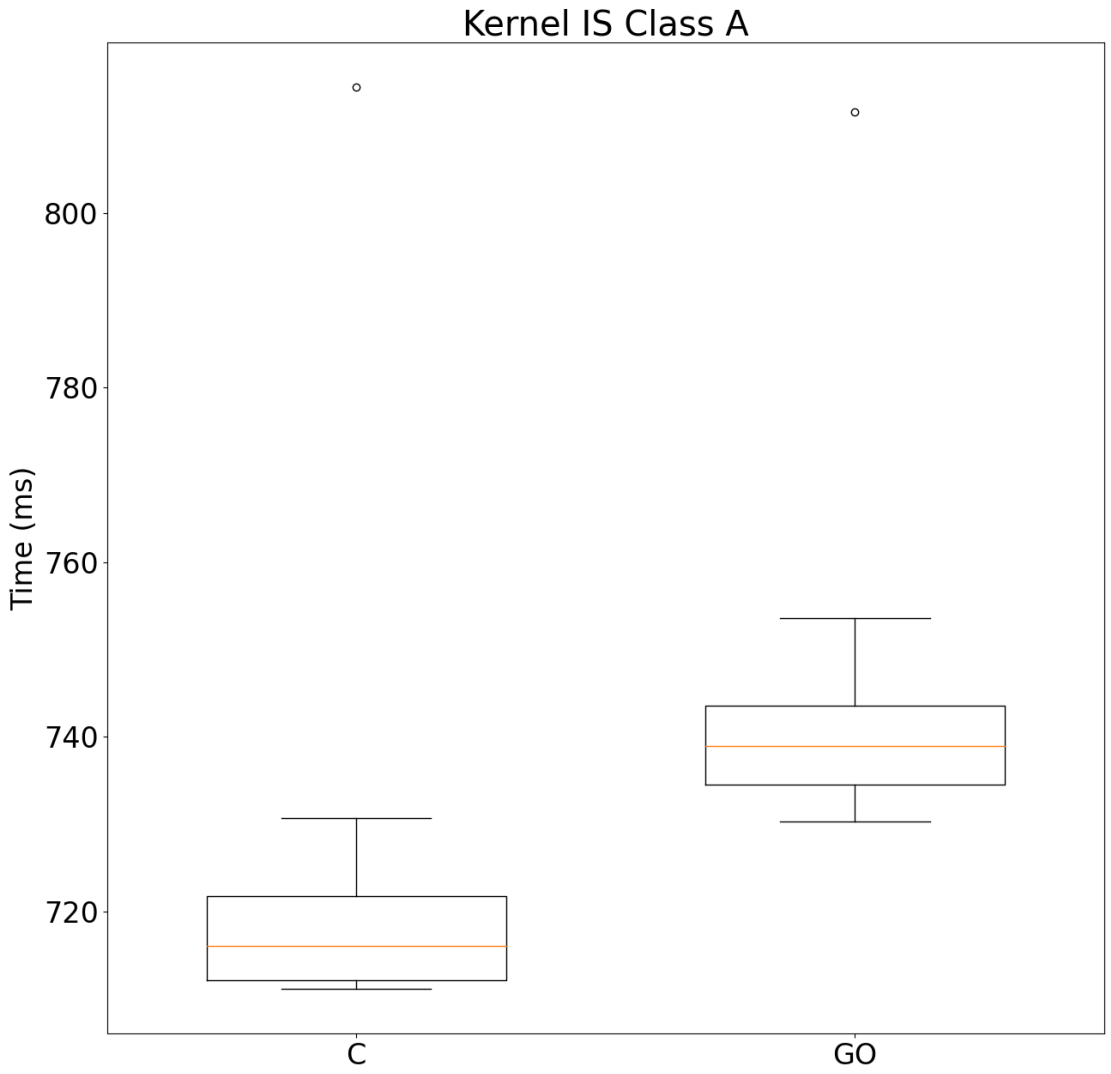

figura 10: Comparação entre IS em Go e IS em c++, classe A

As figuras 7, 8 e 9 mostram uma comparação de tempos entre a implementação C e a implementação Go usando as classes S, W e A. As diferenças de tempo não foram tão grandes entre cada execução — parecendo acentuadas dada a escala de visualização — mas a implementação em C++ se mostrou ser mais veloz.

4. Considerações Finais

A linguagem Go não pareceu ser muito adequada aos problemas apresentados, talvez pela sua natureza de buscar fácil escalabilidade e facilidade de programação paralela em vez de buscar alto desempenho em problemas que requerem muito processamento. Em termos de

facilidade de programação, as estruturas de paralelismo do Go se mostraram bastante acessíveis, salvo a falta de uma estrutura dedicada para for paralelo.