Relatório Final
NPB em Golang

Kevin S. Pereira, Thomazio Giacobbe

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

Benchmarks Desenvolvidos

- Kernel EP - Embarrassingly Parallel
  - Implementação paralela
  - Implementação baseada na versão C++ com OpenMP do Dalvan et al.
- Kernel IS - Integer Sort, versão com buckets
  - Implementação paralela
  - Implementação baseada na versão C++ com OpenMP do Dalvan et al.
- Kernel IS - Integer Sort, versão sem buckets
  - Implementação paralela
  - Implementação baseada na versão C com OpenMP da NASA

Recursos de paralelismo

- Criação de threads paralelos
  - ➢ Goroutines
- Comunicação de dados entre threads
  - ➢ Channels
- Sincronizadores
  - ➢ sync (sync.WaitGroup)
- Instruções atômicas
  - ➢ sync/atomic (atomic.Int64)

Nossa biblioteca - “commons”

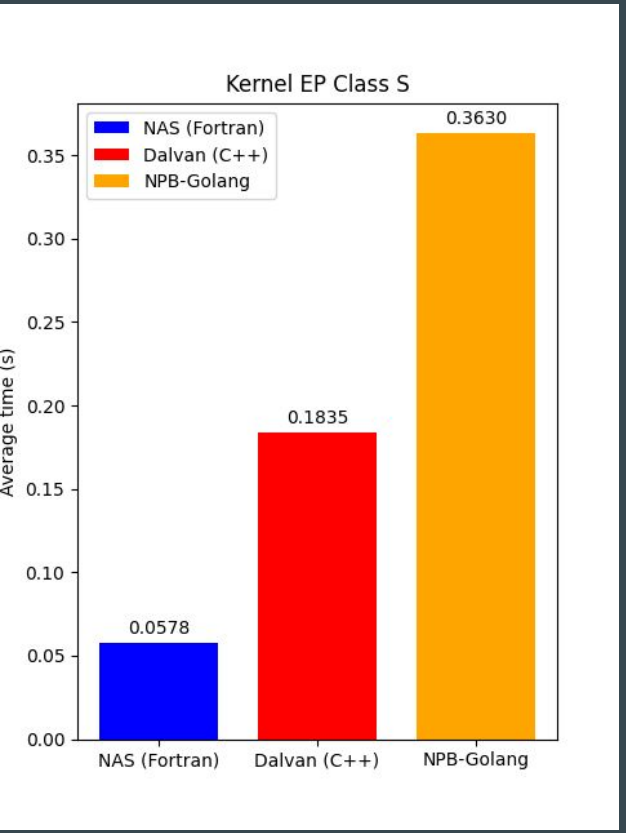
- Implementação rápida das funções Randlc e Vranlc
  - commons/rand.go
  - Aumento de desempenho em 15%
- Função que implementa um for loop paralelo
  - commons/parallel.go
  - Permite configurar o schedule como estático ou dinâmico
  - função que compõe o corpo do for loop deve receber dois parâmetros do tipo int64
- Funções para criação e limpeza de matrizes
  - commons/matrix.go
- Função para impressão de resultados
  - commons/print_results.go
  - Usa a biblioteca go-pretty (third-party)

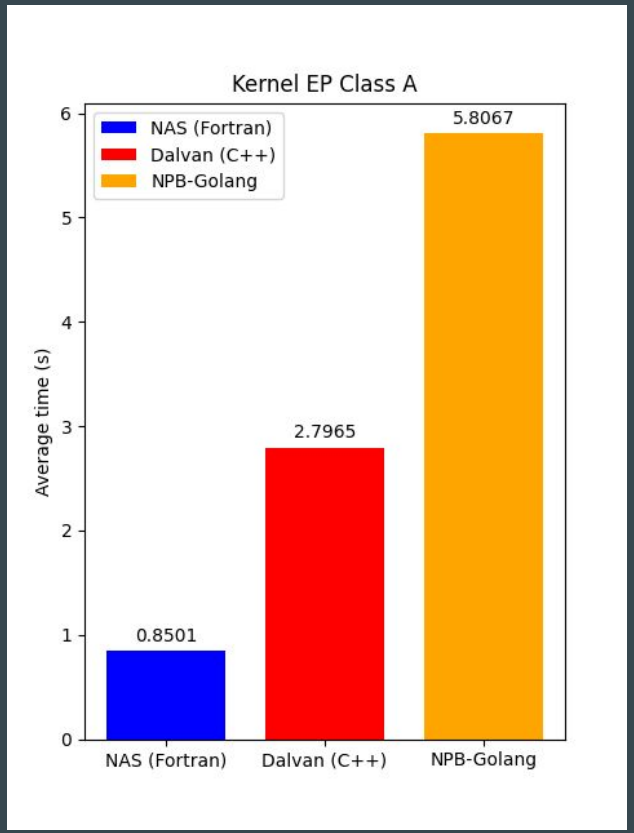
Benchmarks - Experimentos

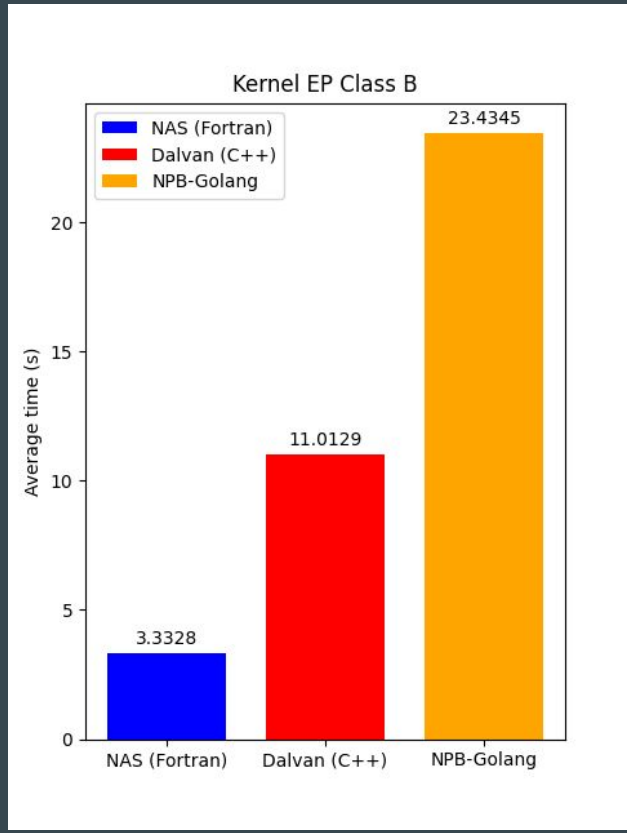
- Kernel EP - Embarrassingly Parallel
  - Classes: S, A, B
- Kernel IS - Integer Sort, versão com buckets
  - Classes: S, W, A, B
- Kernel IS - Integer Sort, versão com buckets
  - Classes: S, W, A, B

- Dados coletados são os tempos de execução, em segundos
- Representação dos dados em gráfico de barras
- Não houve experimentos com nenhuma versão serial, logo não calculamos o SpeedUp
- Resultados disponíveis no repositório em docs
- Todos os experimentos foram executados em um Intel i7 7700 que possui 4 núcleos físicos e 8 threads

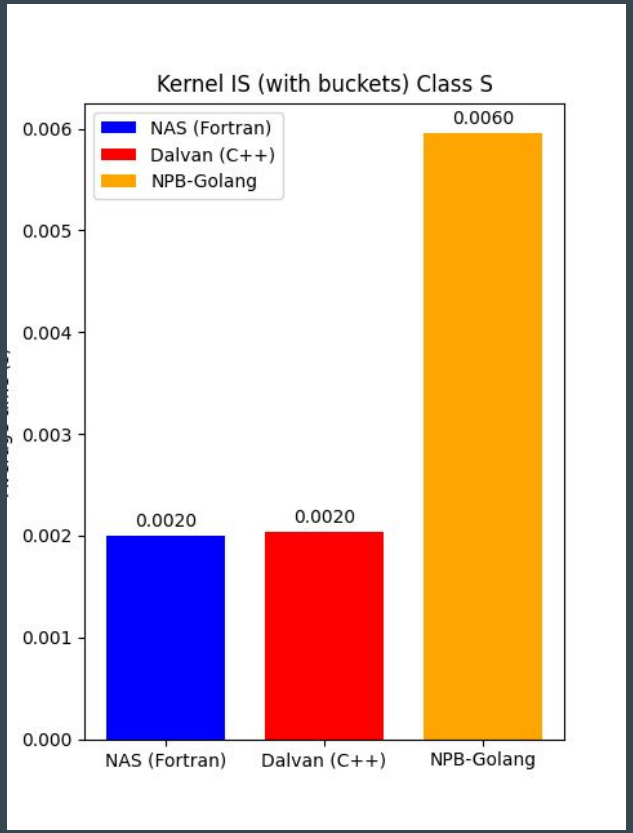
	Kernel EP	Kernel IS com buckets	Kernel IS sem buckets
Go	✓	✓	✓
Dalvan (OpenMP)	✓	✓	✗
NAS (OpenMP)	✓	✓	✓
Serial (Qualquer)	✗	✗	✗

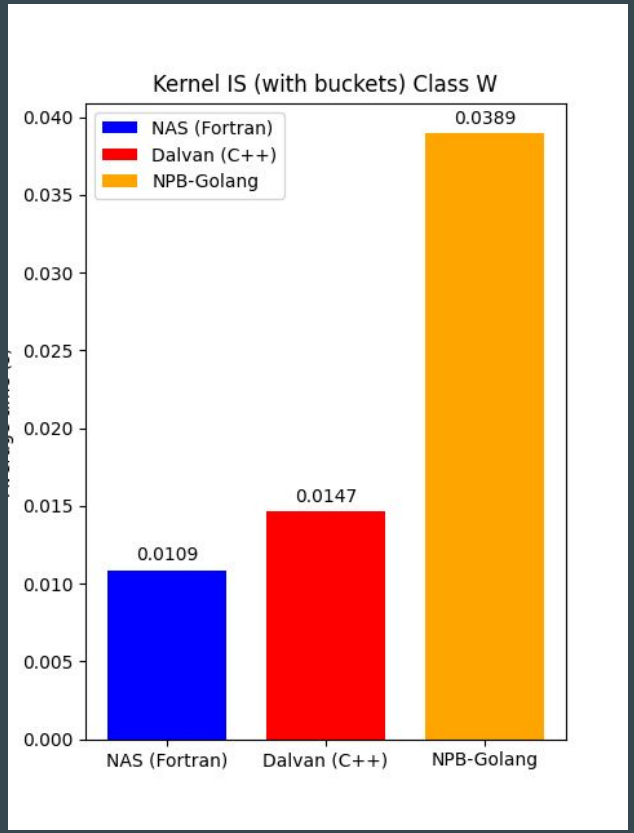

Kernel EP Class S
NAS (Fortran)
Dalvan (C++)
NPB-Golang
Average time (s)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.0578
0.1835
0.3630
NAS (Fortran)
Dalvan (C++)
NPB-Golang


Kernel EP Class A
NAS (Fortran)
Dalvan (C++)
NPB-Golang
Average time (s)
0
1
2
3
4
5
6
0.8501
2.7965
5.8067
NAS (Fortran)
Dalvan (C++)
NPB-Golang

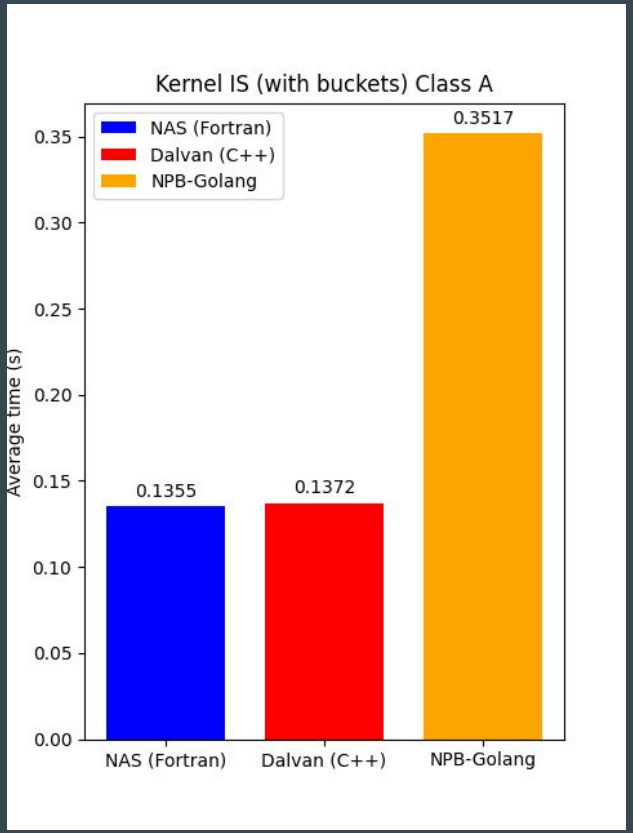

Kernel EP Class B
NAS (Fortran)
Dalvan (C++)
NPB-Golang
Average time (s)
0
5
10
15
20
3.3328
11.0129
23.4345
NAS (Fortran)
Dalvan (C++)
NPB-Golang

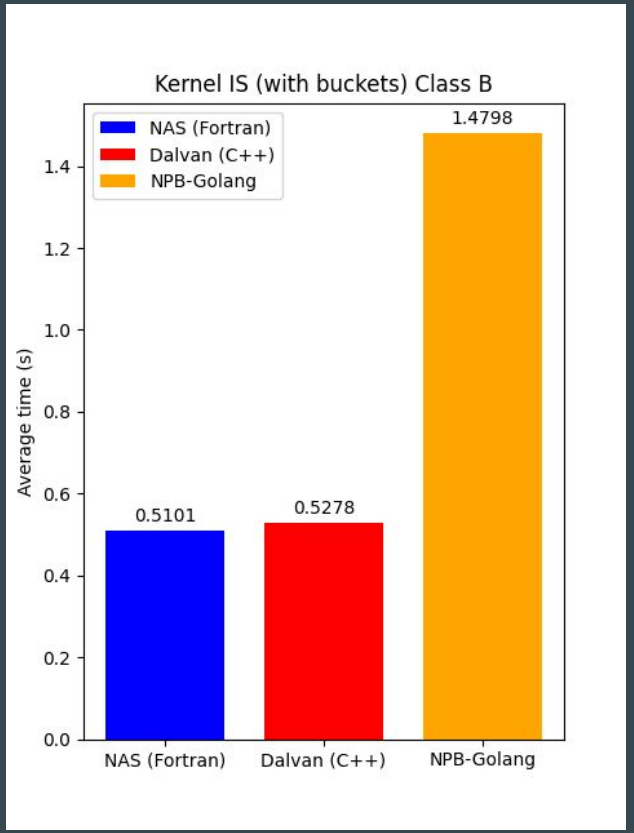
Kernel EP



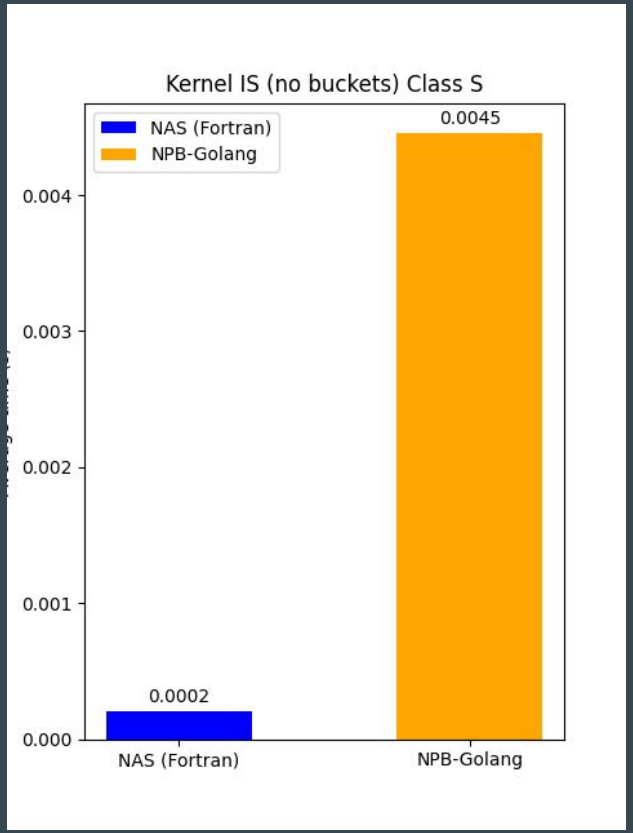


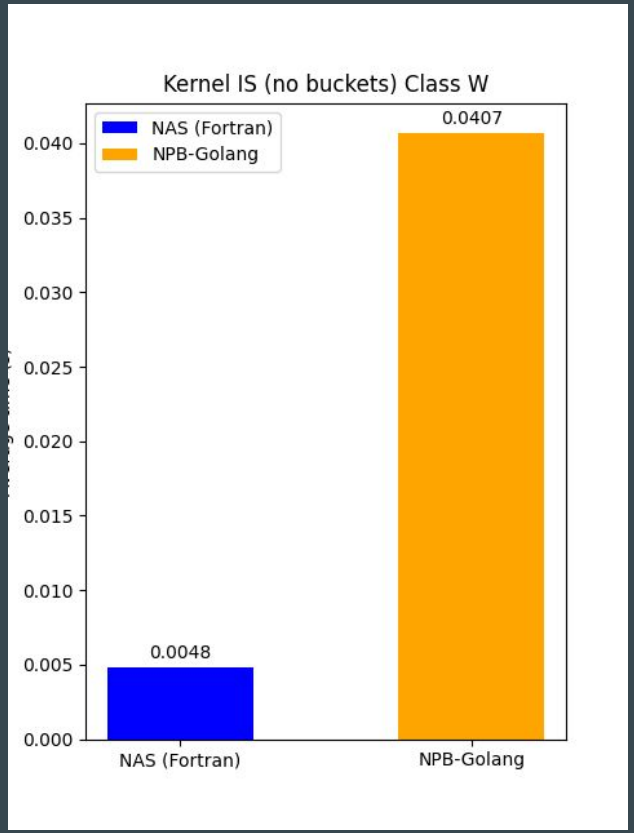
Kernel IS com buckets



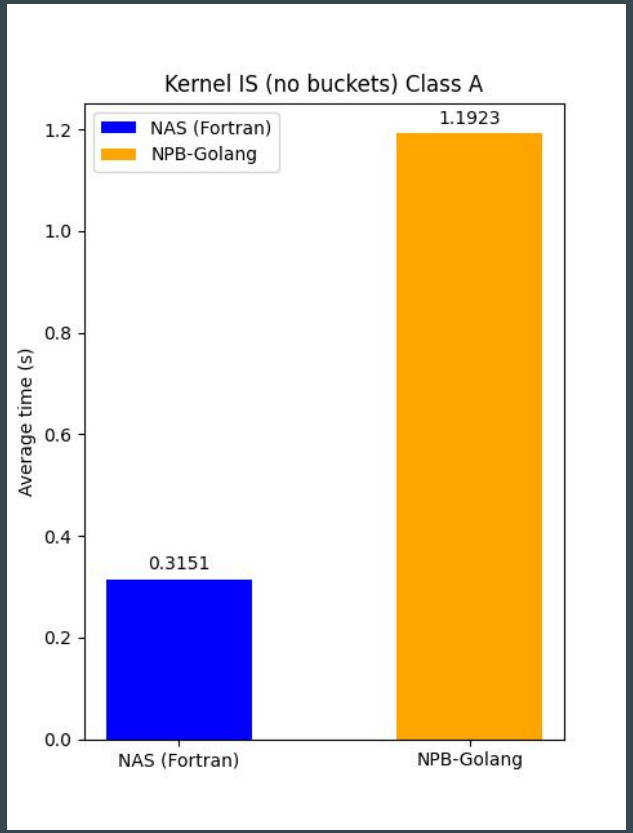


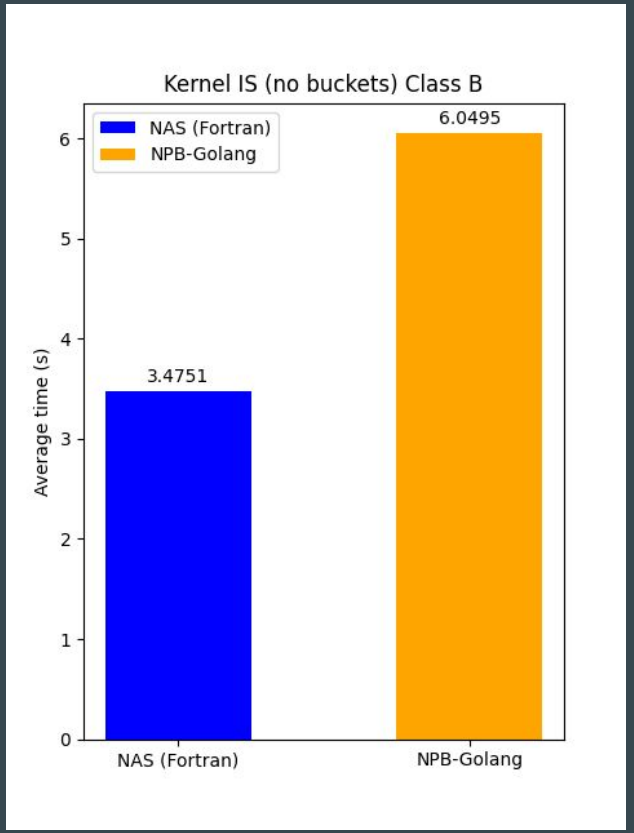
Kernel IS com buckets





Kernel IS sem buckets





Kernel IS sem buckets

Sobre a implementação dos benchmarks

Vantagens

- Ferramentas de paralelismo são simples de serem utilizadas
- Escalonador de Goroutines é eficiente

Desvantagens

- Poucas construções paralelas já definidas
- Desempenho insatisfatório para esse tipo de aplicação

Acreditamos que a implementação ficou boa dentro dos limites da linguagem. É compensador explorar o uso das ferramentas de paralelismo, como goroutines e channels, pois o runtime do Go trabalha de forma eficiente com esses recursos. O maior contraponto foi necessitar de implementações manuais das construções paralelas que no OpenMP são cláusulas pré definidas, causando um overhead que não pode ser evitado. Devido a isso, o desempenho foi insatisfatório.

Sobre a linguagem Golang

Vantagens

- Simplicidade da linguagem
- Facilidade de criar código que executa em paralelo

Desvantagens

- Pouco controle sobre paralelismo
- Decisões de design da linguagem atrapalham em explorar o paralelismo

Go não é uma boa linguagem para este tipo de aplicação. Enquanto em OpenMP usamos uma cláusula para utilizar construções como reduction e parallel for, em Go foi necessário fazer a implementação manualmente. Além disso, o Go usa seu próprio escalonador para as goroutines e não permite controle delas, o que dificultou para o caso de precisarmos de escalonamento especializado, como o estático. As decisões de design da linguagem incentivam o uso de ferramentas específicas (goroutines, sync, channels) mas desincentivam o uso de outras (atomic), perdendo desempenho caso utilizadas.

Conclusão

Go é uma linguagem de programação muito simples e que tem como objetivo simplificar a criação de aplicações paralelas. Porém, essa simplificação tem suas desvantagens:

- Falta de controle do paralelismo. Tentar ir contra o padrão de design indicado pela linguagem e seu escalonador gera situações diversas, como resultados inferiores e até grande variação entre execuções.
- O desempenho. Embora seja eficiente em usar suas ferramentas de multithreading, ao realizar tarefas pesadas, Go se mostra bem inferior a outras linguagens de nível de abstração menor com execuções mais lentas.

Com isso, concluímos que Go é uma excelente linguagem quando precisamos executar tarefas em paralelo. Porém, quando se trata de uma aplicação pesada que necessita de um controle forte nas ferramentas de paralelismo, ela não é uma boa opção.

Relatório Final
NPB em Golang

Kevin S. Pereira, Thomazio Giacobbe

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