# Método computacional baseado em workflow para contabilização da frequência de repetição de *k-mers*

**Fabrício Vilasbôas[1], Carla Osthoff[1], Kary Ocaña[1], Oswaldo Trelles[2], Ana Tereza Vasconcelos [1]**

[1]Laboratório Nacional de Computação Científica (LNCC)

[2]Universidad de Málaga

***Resumo.*** *Este trabalho apresenta uma análise do desempenho do **SCFRK**, um algoritmo determinístico para uma aplicação de bioinformática computacionalmente intensiva, o k-mer, para uma arquitetura GPGPU em um ambiente de workflow científico. Nossos experimentos demonstram que o **SCFRK** é uma alternativa eficiente e de baixo custo para a contabilização de k-mers para análises em metagenoma.*

## 1. Introdução

O advento de novas tecnologias de sequenciamento na área de bioinformática está fazendo com que o gargalo migre da aquisição de dados para o processamento e interpretação dos mesmos. Por outro lado, novas tecnologias de processamento de alto desempenho de baixo custo tais como processadores gráficos, *GPU's* [Kirk and Wen-mei 2012], podem ser utilizadas para aumentar a taxa de processamento dos dados.

Em recentes trabalhos, [Vilasboas et al. 2015] e [Vilasboas et al. 2016], foi apresentado o algoritmo **CFRK**, desenvolvido para a contabilização da frequência de repetição de *k-mers* em ambiente *GPU* que apresenta bom desempenho para aplicações com valores de k menor ou igual a 5, tal como metagenoma. Neste trabalho apresentamos a avaliação de desempenho do **CFRK** em um ambiente de *workflow* cientifico, o *Swift*, um *SWfMS* (*Scientific Workflow Management Systems*) contendo *GPU's*. Na Seção 2 apresentamos os trabalhos relacionados. Na Seção 3 apresentamos a descrição do **SCFRK**. Na Seção 4 apresentamos os experimentos e os resultados. Na Seção 5 apresentamos as conclusões e os trabalhos futuros.

## 2. Trabalhos relacionados

Nesta seção relacionaremos o algoritmo **SCFRK** com o **Jellyfish**, um algoritmo desenvolvido para processamento de *k-mers* e que é considerado o estado da arte nesta tarefa.

O Jellyfish [Marçais and Kingsford 2011] é um algoritmo para contabilização da frequência de repetição de *k-mers* específico para genomas desenvolvido para processamento em memória compartilhada. Ele utiliza várias estruturas *lock-free*, que são estruturas que permitem operações atômicas sem o bloqueio da memória e, por isso, não degradam o desempenho em ambientes multiprocessáveis. A diferença mais importante entre os algoritmos **Jellyfish** e **SCFRK** é que o **SCFRK** foi desenvolvido para o processamento de metagenoma, enquanto o Jellyfish foi feito para o processamento de genoma. O

Jellyfish considera todos os *reads* como provenientes de um mesmo organismo, o **SCFRK** considera cada *read* como sendo de organismos distintos. Isso permite que o **SCFRK** seja de propósito mais geral.

## 3. SCFRK

**SCFRK** é o acrônimo de ***S**wift* - **C**ontabilizador da **F**requência de **R**epetição de **K**-mers, pois foi desenvolvido utilizando os recursos do *Swift*.

A execução do **SCFRK** é dividida em duas fases bem definidas: o pré-processamento e o processamento. Na fase de pré-processamento é realizada a divisão do arquivo que contém os *reads*. É utilizado uma aplicação em linguagem C que realiza o particionamento do arquivo em N partes iguais, onde N é um valor definido pelo usuário. Na fase de processamento, o **SCFRK** é responsável pelas chamadas ao **CFRK**. É fornecido para cada processo a localização do arquivo de entrada e do arquivo de saída. Ao final do processamento, os arquivos de saída são escritos em disco.

A principal motivação para o desenvolvimento do **SCFRK** se deve à limitação da execução do **CFRK** para arquivos de entrada com tamanho igual ou superior ao tamanho da memória principal da estação de trabalho em uso. Ao ler um arquivo de entrada de tamanho igual ou superior ao tamanho da memória principal, a estação de trabalho poderá entrar em processo de paginação ou poderá entrar em colapso se não houver espaço em disco para a paginação. Esta estratégia de paralelismo permite a divisão do arquivo de entrada em arquivos que sejam do tamanho adequado para o processamento na estação de trabalho.

## 4. Experimentos

Nesta seção serão apresentados os experimentos e resultados da execução do **SCFRK**, bem como algumas comparações com o algoritmo **Jellyfish**.

### 4.1. Ambiente de teste

Para os testes foram utilizados quatro nós da *cluster Altix-Xe* do CENAPAD/LNCC. Cada nó possui dois processadores Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz com 8 núcleos cada processador e uma *GPU Nvidia Tesla K20m* com 5 GB de memória *RAM* e 2496 núcleos. O sistema operacional utilizado foi o CetOS, *kernel* versão 2.6.32-573.7.1.el6.x86_64.

O arquivo de entrada utilizado foi uma amostra de metagenoma de um ambiente real obtida no banco de dados SRA do NCBI (*National Center for Biotechnology Information*), cujo ID nesta base de dados é SRX2021688. Este arquivo contém 8.7GB de dados e 27017895 *reads*.

### 4.2. Resultados

Os experimentos consistiram em executar os algoritmos **SCFRK** e **Jellyfish** com o mesmo conjunto de dados. Os valores de $k$ foram selecionados baseados em trabalhos anteriores [Vilasboas et al. 2015] que demonstraram que o **CFRK** apresenta ganho em relação ao **Jellyfish** para valores de $k < 5$. O tempo total de execução foi obtido através do comando *time* do linux, sendo considerado apenas o tempo real. Para este trabalho foi

considerado apenas o tempo da fase de processamento do **SCFRK**, pois o foco da análise é a execução do módulo desenvolvido utilizando o *Swift* e o **Jellyfish** não possui a etapa de divisão do arquivo. Cada processo lançado para a execução do *workflow* receberá o nome de *worker*.

Para o algoritmo **SCFRK** foram utilizados: dois nós com um *worker* por nó; três nós com um *worker* por nó; quatro nós com um *worker* por nó; e quatro nós com dois *workers* por nó. Para o **Jellyfish** foi utilizado um nó com oito *threads*.

A Figura 1 apresenta o tempo total de execução dos algoritmos **Jellyfish** e **SCFRK**. As barras em azul apresentam o tempo total de execução do **Jellyfish** para $k = 2, 3, 4, 5$. As barras em laranja apresentem o tempo de execução do **SCFRK** com 2 *workers* para $k = 2, 3, 4$. As barras em amarelo apresentem o tempo de execução do **SCFRK** com 3 *workers* para $k = 2, 3, 4, 5$. As barras em verde apresentem o tempo de execução do **SCFRK** com 4 *workers* para $k = 2, 3, 4, 5$. As barras em vinho apresentem o tempo de execução do **SCFRK** com 8 *workers* para $k = 2, 3, 4, 5$.

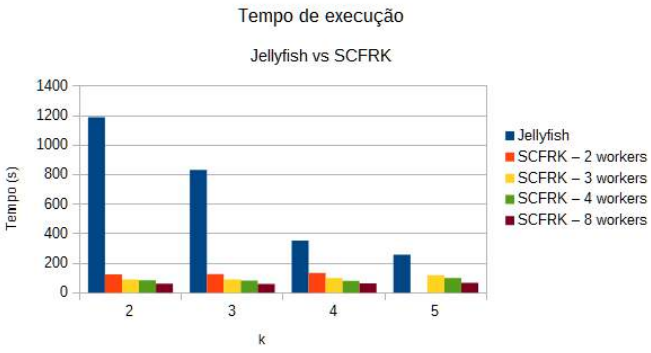


**Figure 1. Tempo de execução dos algoritmos SCFRK e Jellyfish**

A Figura 2 apresentam o ganho obtido pelo **SCFRK** em relação ao **Jellyfish**. A linha em azul apresenta o ganho do **SCFRK** utilizando 2 *workers*. A linha em laranja apresenta o ganho do **SCFRK** utilizando 3 *workers*. A linha em amarelo apresenta o ganho do **SCFRK** utilizando 4 *workers*. A linha em verde apresenta o ganho do **SCFRK** utilizando 8 *workers*.


**Figure 2. Ganho obtido pelo SCFRK em relação ao Jellyfish**

### 4.3. Discussão do resultado

Ao observar a Figura 1 vemos que o **SCFRK** apresenta um menor tempo de execução em relação ao **Jellyfish** para todos os casos. Observamos que o tempo de execução de **SCFRK** se mantém praticamente constante com o aumento do valor de $k$. Isso acontece por conta da característica do algoritmo **CFRK** que processa todas as combinações de forma independente e, dessa forma, mais *threads* são ativadas e aumenta a ocupação da *GPU*. Outra observação é em relação ao tempo de execução do **Jellyfish**. Vemos que a medida que o valor de $k$ aumenta o tempo de execução decai. Isto acontece porque o **Jellyfish** foi desenvolvido para o processamento de um genoma, ou seja, ele considera todos os *reads* como pertencentes a um mesmo organismo. O **Jellyfish** aloca um único vetor global para contabilizar a frequência de repetição dos *k-mers* de todos os *reads*. Portanto, a medida que o valor de $k$ aumenta ele manipula menos a memória, dado que o número de possíveis combinações decai com o aumento do valor de $k$.

A Figura 2 mostra que para $k = 2$ o menor ganho do **SCFRK** foi de $9.4$ vezes com 2 *workers* e o maior ganho foi de $19.4$ vezes com 8 *workers*. Para $k = 3$ observamos que o menor ganho foi de $6.5$ vezes com 2 *workers* e o maior ganho foi de $13.9$ com 8 *workers*. Para $k = 4$ temos que o menor ganho foi de $2.6$ vezes com 2 *workers* e o maior ganho foi de $5.5$ vezes com 8 *workers*. Para $k = 5$ o menor ganho foi de $2.2$ vezes com 3 *workers* e o maior ganho foi de $3.8$ vezes com 8 *workers*.

## 5. Conclusão e trabalhos futuros

Neste trabalho foi apresentado o **SCFRK**, uma variação do algoritmo **CFRK** utilizando o gerenciador de *workflow* cientifico *Swift*. Mostramos que o **SCFRK** apresenta bom desempenho com ganho máximo de $19.4$ vezes para $k = 2$ com 8 *workers* e ganho mínimo de $2.2$ vezes com 3 *workers* para $k = 5$ em relação ao **Jellyfish** e que é uma opção viável para a contabilização de *k-mers* menores que 5 em aplicações tais como de metagenoma em ambientes de *workflow* cientifico com *GPU's*. Como trabalhos futuros pretendemos avaliar o desempenho com grandes bases de dados em ambientes computacionais que permitam implementar um grande número de *workers*, tal como o supercomputador *SDumont*.

## References

Foster, I. (1994). Task parallelism and high-performance languages. *IEEE Parallel & Distributed Technology: Systems & Technology*, 2(3):27–36.

Kirk, D. B. and Wen-mei, W. H. (2012). *Programming massively parallel processors: a hands-on approach*. Newnes.

Marçais, G. and Kingsford, C. (2011). A fast, lock-free approach for efficient parallel counting ofoccurrences of k-mers. *Bioinformatics (Oxford, England)*, 27(6):764–70.

Vilasboas, F., Osthoff, C., Trelles, O., and Vasconcelos, A. T. (2015). Desenvolvimento de um algoritmo paralelo para contabilização da repetição de k-mers. *3ª Conferência Ibero Americana de Computação Aplicada 2015*.

Vilasboas, F., Osthoff, C., Trelles, O., and Vasconcelos, A. T. (2016). Otimização de um algoritmo paralelo para contabilização da repetição de k-mers. *II Escola Regional de Computação de Alto Desempenho do Rio de Janeiro*.