

*Uma abordagem paralela baseada em OpenMP para  
transcrição e tradução de DNA*

Elihofni Lima  
Erick Grilo  
Max Fratane

## 1 Introdução e descrição da aplicação

A transcrição do DNA é o processo através do qual o DNA serve de modelo para a síntese de RNA feita por um ser vivo. Apenas uma cadeia de DNA é usada nesse processo, ativada pela enzima RNA-polimerase. Em uma determinada região da molécula de DNA, ocorre a separação das hélices, onde uma delas forma o RNA através do encadeamento de nucleotídeos complementares. Em suma, essa é a fase responsável por parear as bases nitrogenadas do DNA com as do RNA: A do DNA com U do RNA, T do DNA com A do RNA, C do DNA com G do Rna e G do DNA com C do RNA.

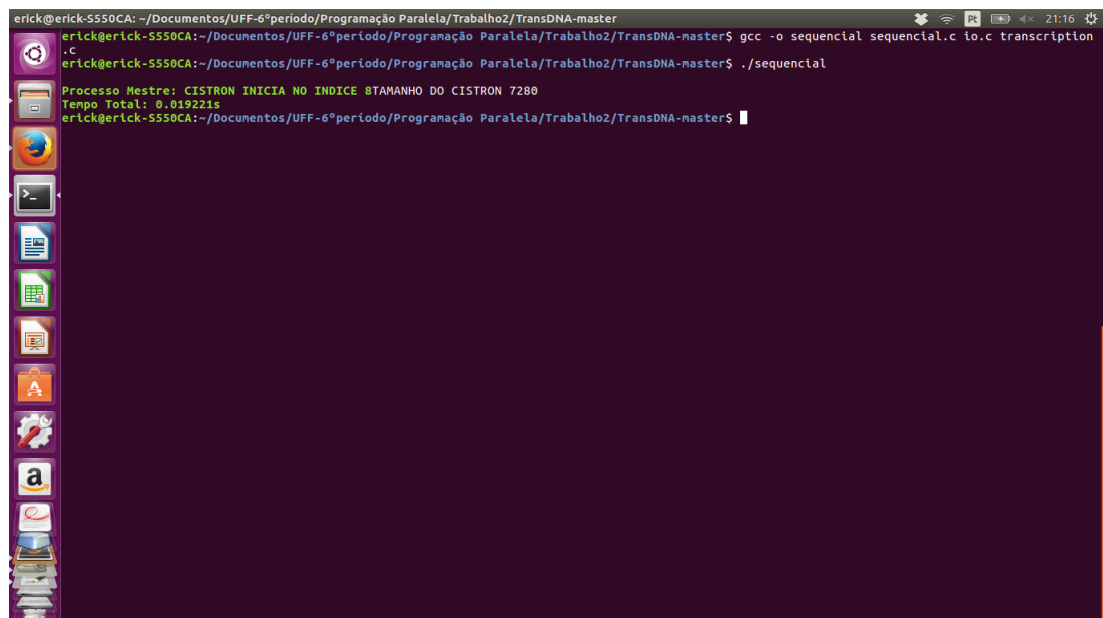
Relembrando alguns conceitos de biologia, A (adenina), C (citossina) T (timina) e G (guanina) são os nucleotídeos que compõem o DNA, e as mesmas, com exceção da timina (que vira U, de uracila, que por sua vez é uma base nitrogenada), compõem o RNA. Cada trinca de 3 dessas bases nitrogenadas é chamada de códon. Um códon codifica um aminoácido, vide Morandini et al. (2013).

Motivado pelo tamanho que uma cadeia de DNA pode ter (Venter et al. 2001), no mapeamento do genoma humano, por exemplo, foram encontradas aproximadamente 2.91 bilhões de pares de bases nitrogenadas) e pelo processo de transcrição ser uma tarefa repetitiva, a criação de um programa paralelo do tipo SPMD (Simple Program, Multiple Data) aparenta ser uma boa abordagem para a solução desta tarefa. Um exemplo de utilização de OpenMP é visto em Sathe & Shrimankar (2011), onde é implementado um algoritmo paralelo para o alinhamento de sequências de DNA em arquiteturas *multi-core*

O objetivo é paralelizar a transcrição e a tradução de DNA, onde a transcrição é o processo responsável por traduzir uma cadeia de DNA para RNA e a tradução consiste em identificar o aminoácido que o códon em questão representa, a partir de códon de RNA (que foram obtidos a partir da transcrição de códon de DNA para RNA). Note que o processo só se inicia efetivamente quando na cadeia original de DNA dado na entrada é identificado um códon (uma cadeia de DNA iniciada pelos códon compostos por ATC, ACT ou ATT, que possui informações para a síntese de uma proteína). Enquanto um códon não é encontrado, não há nenhum processamento efetivo. Quando um códon é encontrado, o processo se inicia. Como estamos considerando nenhuma mutação genética, o códon deve ser múltiplo de 3 para que o algoritmo funcione corretamente. Já, a cadeia original pode ter qualquer tamanho.

## 2 O método sequencial

O método sequencial consiste em tratar toda a cadeia de entrada como uma única cadeia, oriunda da leitura de um arquivo texto onde se encontra tal cadeia. A partir daí, toda a cadeia do DNA é transcrita para RNA da seguinte forma: primeiramente, é identificada o local correto para iniciar a transcrição da cadeia (onde o códon se encontra); em seguida, a cada três nucleotídeos (um códon), sua transcrição (conversão de DNA para RNA) é feita e a identificação do aminoácido que aquele códon se refere é feita em seguida. Ao término da análise de um códon, o códon DNA original, a sua transcrição para códon RNA e o aminoácido que ele representa é escrito em uma linha do arquivo e esse procedimento se repete por todo o tamanho da cadeia de entrada. Caso o códon não possui 3 nucleotídeos (o tamanho do códon não é múltiplo de 3), tal códon é ignorado.



```
erick@erick-S550CA: ~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master
erick@erick-S550CA:~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master$ gcc -o sequencial sequencial.c to.c transcription
erick@erick-S550CA:~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master$ ./sequencial
Processo Mestre: CISTRON INICIA NO INDICE 8TAMANHO DO CISTRON 7280
Tempo Total: 0.019221s
erick@erick-S550CA:~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master$
```

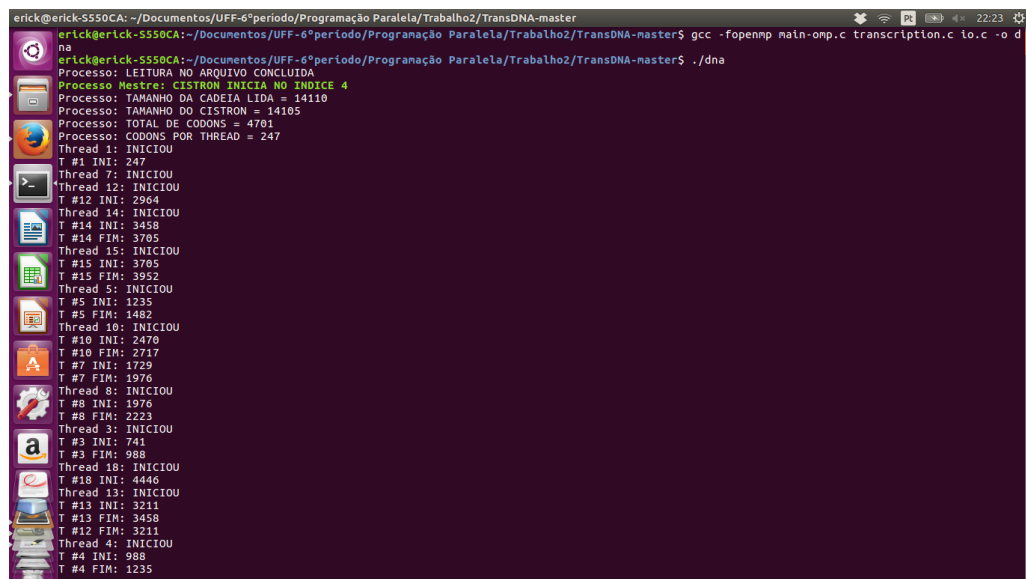
Figura 1: Exemplo de uma execução da aplicação sequencial, indicando onde o códon é encontrado na cadeia original, seu tamanho e o tempo total de execução.

### 3 O método paralelo

A paralelização da aplicação sequencial é aplicada em todo o processo de transcrição e tradução do DNA em RNA e em um aminoácido, respectivamente. O processo é visto como um conjunto de  $n$  threads (1 *master thread*, a thread mestre, e as demais threads colaboradoras) onde a execução se inicia na *master thread*, que efetua a leitura da cadeia de DNA de entrada, identifica o início do códon e armazena ambas as informações em dois *arrays* diferentes na memória principal.

Em seguida, inicia-se a região de paralelismo da aplicação: são criadas as  $n-1$  threads colaboradoras, que terão acesso à todas as variáveis globais do programa (as declaradas dentro da *master thread*). Dessa forma, é possível dividir a tarefa para todas as threads acessarem a mesma região de memória (uma vez que as threads compõem um mesmo processo, isso é possível), que contém os *arrays* dos códons e do DNA original, onde cada thread é responsável por atuar sobre uma parte do vetor específica.

Após a execução, a área de memória global do programa conterá os dados já processados por cada uma das threads, restando a *master thread* mostrar os resultados na tela e escrevê-los em um arquivo.



```
erick@erick-S550CA: ~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master
erick@erick-S550CA:~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master$ gcc -fopenmp main-omp.c transcription.c io.c -o dna
na
erick@erick-S550CA:~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master$ ./dna
Processo: LEITURA NO ARQUIVO CONCLUIDA
Processo Mestre: CISTRON INICIA NO INDICE 4
Processo: TAMANHO DA CADEIA LIDA = 14118
Processo: TAMANHO DO CISTRON = 14105
Processo: TOTAL DE CODONS = 4701
Processo: CODONS POR THREAD = 247
Thread 1: INICIOU
T #1 INI: 247
Thread 7: INICIOU
Thread 12: INICIOU
T #12 INI: 2964
Thread 14: INICIOU
T #14 INI: 3458
T #14 FIM: 3705
Thread 15: INICIOU
T #15 INI: 3705
T #15 FIM: 3952
Thread 5: INICIOU
T #5 INI: 1235
T #5 FIM: 1482
Thread 10: INICIOU
T #10 INI: 2470
T #10 FIM: 2717
T #7 INI: 1729
T #7 FIM: 1976
Thread 8: INICIOU
T #8 INI: 1976
T #8 FIM: 2223
Thread 3: INICIOU
T #3 INI: 741
T #3 FIM: 988
Thread 18: INICIOU
T #18 INI: 4446
T #18 FIM: 4692
Thread 13: INICIOU
T #13 INI: 3211
T #13 FIM: 3458
T #12 FIM: 3211
Thread 4: INICIOU
T #4 INI: 988
T #4 FIM: 1235
```

Figura 2: Início da execução paralela. Note que a *master thread* mostra os valores correspondentes ao tamanho da cadeia de entrada, o tamanho do códon, o total de códons e qual o intervalo de atuação de cada thread (ini indica a posição de início no array, fim indica a posição de fim no array)

```
erick@erick-S550CA: ~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master
GTA CAU His
GCA CGU Arg
AAA UUU Phe
CGG GCC Ala
CGT GCA Ala
AGC UCG Ser
AAA UUU Phe
AAA UUU Phe
CGG GCC Ala
CGT GCA Ala
AGC UCG Ser
AAA UUU Phe
AGC UCG Cys
GCG CGC Arg
TAG AUC Leu
CAA GUU Val
AAA UUU Phe
AGC UCG Cys
GCG CGC Arg
TAG AUC Leu
CAA GUU Val
AAC UUG Leu
GGC CCG Pro
GTA CAU His
GCA CGU Arg
AAA UUU Phe
AAC UUG Leu
GGC CCG Pro
GTA CAU His
GCA CGU Arg
AAA UUU Phe
CGG GCC Ala
CGT GCA Ala
AGC UCG Ser
AAA UUU Phe
AAA UUU Phe
CGG GCC Ala
CGT GCA Ala
AGC UCG Ser
AAA UUU Phe
AGC UCG Cys
GCG CGC Arg
TAG AUC Leu
```

Figura 3: Em algum ponto entre o início e o fim do processamento. Exibe-se na tela cada códon na cadeia de DNA original, sua transcrição para RNA, sua tradução em um aminoácido e escreve em um arquivo estas mesmas informações.

```
erick@erick-S550CA: ~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master
GCG CGC Arg
CAT GUA Val
GCG CGC Ala
TGC ACG Thr
ATC UAG S/N
GAC CUG Leu
TAG AUC Leu
ATC UAG S/N
GAT CUA Leu
CAG GUC Val
CTA GAU Asp
GCG CGC Arg
CAT GUA Val
CGC CGC Ala
TGC ACG Thr
ATC UAG S/N
GAC CUG Leu
TAG AUC Leu
ATC UAG S/N
GAT CUA Leu
CAG GUC Val
CTA GAU Asp
GCG CGC Arg
CAT GUA Val
GCG CGC Ala
TGC ACG Thr
ATC UAG S/N
GAC CUG Leu
TAG AUC Leu
ATC UAG S/N
GAT CUA Leu
CAG GUC Val
CTA GAU Asp
GCA CGU Arg
TCG AGC Ser
ATG UAC Tyr
CAT GUA Val
GCA CGU Arg
TGC ACG Thr
ATG UAC Tyr

Processo: ESCRITA DE RESULTADOS NO ARQUIVO CONCLUIDA
Tempo total: 0.062242s
erick@erick-S550CA:~/Documentos/UFF-6ºperíodo/Programação Paralela/Trabalho2/TransDNA-master$
```

Figura 4: Fim da execução paralela. Exibe-se na tela cada códon na cadeia de DNA original, sua transcrição para RNA, sua tradução em um aminoácido e escreve em um arquivo estas mesmas informações. Exibe-se também na tela uma mensagem indicando o fim da execução, seguida do tempo total de execução

## 4 A implementação paralela

A abordagem paralela consiste no uso do OpenMP (Dagum & Menon 1998), uma API para a programação *multi-threaded* fazendo uso de memória compartilhada. Nesta implementação, a paralelização é feita no momento de efetuar a transcrição de DNA em RNA e o de efetuar a tradução de um códon para um aminoácido. O programa então é iniciado com uma única thread, chamada de *master thread*, que executa de forma sequencial até encontrar a diretiva que indica a primeira construção de uma região paralela (a diretiva `#pragma omp parallel`).

A abordagem adotada pelo OpenMP segue a ideia do modelo *fork-join*, onde o *fork* consiste na criação de threads paralelas, as quais possuem suas variáveis locais e acesso à todas as variáveis inicializadas na *master thread* (memória compartilhada). Dessa forma, como as threads possuem acesso à mesma área de memória compartilhada, é possível ajustá-las a fim de trabalharem em paralelo: no caso da transcrição e na tradução do DNA, a *master thread* efetua a leitura da cadeia de DNA do arquivo dado como entrada, armazena a cadeia de DNA lida, a partir da cadeia lida a *master thread* encontra o início do códon e armazena todo o códon (que é um subconjunto da cadeia lida como entrada, podendo ser a própria).

Neste ponto de execução, a master thread cria as demais threads, que são responsáveis por efetuar a tradução e a transcrição de uma parcela do códon original. Cada thread possui uma área operacional, determinada pela quantidade de códons que será repassado para cada thread (que é dado pela quantidade de códons dividido pela quantidade de threads) multiplicada pelo seu id. Dessa forma, a thread 1 é responsável pela área do vetor que contém o códon de 0 até o número de quantidade de códons por thread - 1, a thread 2 é responsável pela área do vetor dada pelo intervalo que compreende de quantidade de códons por thread até duas vezes a quantidade de códons por thread - 1, até a enésima thread.

Após cada thread efetuar a transcrição e a tradução da parcela correspondente do vetor do códon, a *master thread* então exibe na tela cada códon da cadeia de DNA lida originalmente, sua transcrição em RNA e sua tradução em um aminoácido, escrevendo todos esses dados em um arquivo texto.

## 5 Descrição dos experimentos computacionais

A máquina física onde as aplicações paralela e sequencial foram testadas é um computador com 8 GB de memória RAM, processador Intel i7-3517U @ 2.00 GHz e S.O Ubuntu 16.04 LTS 64 bits. Foram executados 6 casos de teste: uma entrada com a cadeia de DNA contendo um códon de tamanho 60, uma entrada com a cadeia de DNA contendo um códon de tamanho 140, uma entrada com a cadeia de DNA contendo um códon de tamanho 837, uma entrada com a cadeia de DNA contendo um códon de tamanho 4437, uma entrada com a cadeia de DNA contendo um códon de tamanho 7280 e uma entrada com a cadeia de DNA contendo um códon de tamanho 14105. Ambas as implementações (sequencial e paralela) podem ser encontradas em <https://github.com/elihofni/TransDNA>.

Observe que a quantidade de códons na entrada pode diferir do tamanho da cadeia de DNA de entrada original (veja figura 4). A cada bateria de teste, houve uma pausa de três minutos e entre duas execuções, houve uma pausa de um minuto. Tais pausas foram tomadas a fim de tentar contornar eventuais interferências de outros programas que poderiam estar executando em *background*. Nessas condições, os seguintes resultados foram obtidos:

### 5.1 Entrada com códon de tamanho 60

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.000955s	0.000760s	0.000912s	0.000876s

Tabela 1: Tabela da execução com entrada de um códon com 60 nucleotídeos para a aplicação sequencial

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.000727s	0.000792s	0.000625s	0,000715s
2 threads	0.001395s	0.001461s	0.001331s	0.001396s
4 threads	0.001188s	0.003726s	0.002771s	0,002562s
5 threads	0.001389s	0.000409s	0.001968s	0,001255s
8 threads	0.001595s	0.001360s	0.001403s	0,001453s
16 threads	0.001809s	0.002400s	0.002594s	0,002268s
32 threads	0.002322s	0.004192s	0.003813s	0,003442s

Tabela 2: Tabela da execução com entrada de um códon com 60 nucleotídeos para a aplicação paralela

Pelos valores obtidos acima, a paralelização com um códon muito pequeno acaba por não compensar: o custo de manipulação das threads acaba não levando a resultados melhores do que os obtidos na aplicação sequencial.

### 5.2 Entrada com códon de tamanho 140

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.001994s	0.002162s	0.001967s	0,002041s

Tabela 3: Tabela da execução com entrada de um cístron com 140 nucleotídeos para a aplicação sequencial

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.001133s	0.001392s	0.001218s	0,001058s
2 threads	0.001310s	0.001500s	0.001081s	0.001297s
4 threads	0,005501s	0.005497s	0.006926s	0,005975s
5 threads	0.001692s	0.002032s	0.002045s	0,001923s
8 threads	0.002699s	0.002381s	0.002053s	0,002378s
16 threads	0.002994s	0.002700s	0.002340s	0,002678s
32 threads	0.002558s	0.002937s	0.002112s	0,002536s
46 threads	0.003620s	0.004677s	0.006654s	0,004984s
64 threads	0.006507s	0.007083s	0.006785s	0,006792s
69 threads	0.007037s	0.008164s	0.007912s	0,007704s

Tabela 4: Tabela da execução com entrada de um cístron com 140 nucleotídeos para a aplicação paralela

### 5.3 Entrada com cístron de tamanho 837

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.008252s	0.008714s	0.008954s	0,008640s

Tabela 5: Tabela da execução com entrada de um cístron com 837 nucleotídeos para a aplicação sequencial

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.003569s	0.002371s	0.003984s	0,003308s
2 threads	0.003887s	0.003295s	0.004221s	0.003801s
4 threads	0.008824s	0.006690s	0.005883s	0,007102s
5 threads	0.005173s	0.003483s	0.002739s	0,003798s
8 threads	0.006520s	0.004901s	0.004805s	0,005409s
16 threads	0.005482s	0.004895s	0.005394s	0,005257s
32 threads	0.007081s	0.007203s	0.007819s	0,007368s
46 threads	0.008515s	0.007432s	0.008203s	0,008050s
64 threads	0.008989s	0.010908s	0.009554s	0,009817s
69 threads	0.010717s	0.015701s	0.014559s	0,013659s

Tabela 6: Tabela da execução com entrada de um cístron com 837 nucleotídeos para a aplicação paralela



## 5.4 Entrada com cístron de tamanho 4437

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.021126s	0.028207s	0.020000s	0,023111s

Tabela 7: Tabela da execução com entrada de um cístron com 4437 nucleotídeos para a aplicação sequencial

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.020119s	0.021027s	0.019998s	0,020381s
2 threads	0.010064s	0.014493s	0.012741s	0.012433s
4 threads	0.023904s	0.021841s	0.023590s	0,023145s
5 threads	0.016946s	0.018255s	0.017123s	0,017441s
8 threads	0.019390s	0.017777s	0.019421s	0,018863s
16 threads	0.016674s	0.018119s	0.016497s	0,017097s
32 threads	0.019792s	0.021998s	0.017228s	0,019673s
46 threads	0.018374s	0.021735s	0.022240s	0,020783s
64 threads	0.019999s	0.020692s	0.020441s	0,020377s
69 threads	0.019450s	0.024876s	0.019639s	0,021632s

Tabela 8: Tabela da execução com entrada de um cístron com 4437 nucleotídeos para a aplicação paralela

## 5.5 Entrada com cístron de tamanho 7280

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.053332s	0.046465s	0.046110s	0,048636s

Tabela 9: Tabela da execução com entrada de um cístron com 7280 nucleotídeos para a aplicação sequencial

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.043076s	0.042638s	0.029132s	0,038282s
2 threads	0.029382s	0.029757s	0.028939s	0,029359s
4 threads	0.032608s	0.039423s	0.034612s	0,035548s
5 threads	0.031019s	0.028282s	0.027870s	0,029057s
8 threads	0.025235s	0.025188s	0.020623s	0,021698s
16 threads	0.028186s	0.038771s	0.032606s	0,033188s
32 threads	0.036162s	0.028468s	0.030128s	0,031586s
46 threads	0.031687s	0.032003s	0.036529s	0,033406s
64 threads	0.041742s	0.042371s	0.047419s	0,043844s
69 threads	0.039208s	0.035987s	0.037311s	0,037502s

Tabela 10: Tabela da execução com entrada de um cístron com 7280 nucleotídeos para a aplicação paralela

## 5.6 Entrada com cístron de tamanho 14105

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.090016s	0.080204s	0.079517s	0,083246s

Tabela 11: Tabela da execução com entrada de um cístron com 14105 nucleotídeos para a aplicação sequencial

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.059459s	0.064403s	0.069250s	0,064371s
2 threads	0.072735s	0.065109s	0.064392s	0,067412s
4 threads	0.041539s	0.042256s	0.042319s	0,042140s
5 threads	0.063214s	0.066446s	0.058199s	0,062620s
8 threads	0.059713s	0.064468s	0.068252s	0,064144s
16 threads	0.037551s	0.069197s	0.054230s	0,053660s
32 threads	0.056415s	0.064820s	0.067551s	0,062929s
46 threads	0.067774s	0.061960s	0.062744s	0,064159s
64 threads	0.066161s	0.066357s	0.068598s	0,067039s
69 threads	0.062844s	0.073508s	0.067113s	0,067821s

Tabela 12: Tabela da execução com entrada de um cístron com 14105 nucleotídeos para a aplicação paralela

## 5.7 Exemplo real: gene humano para a proteína GLA do osso

O gene que identifica a proteína GLA (também conhecida como osteocalcina) é um cítron de tamanho 74 <sup>1</sup>. Através da execução da aplicação, é possível identificar os códons que compõem esse gene, a sua transcrição para RNA e a tradução para um aminoácido.

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.001432s	0.001116s	0.001136s	0,001228s

Tabela 13: Tabela da execução com entrada de um cítron com 74 nucleotídeos para a aplicação sequencial

Nº threads	1ª execução	2ª execução	3ª execução	Média
1 thread	0.000702s	0.000867s	0.000851s	0,000807s
2 threads	0.000916s	0.000887s	0.000964s	0,000922s
4 threads	0.003565s	0.005370s	0.004052s	0,004329s
5 threads	0.001339s	0.001485s	0.001382s	0,001402s
8 threads	0.001997s	0.001918s	0.001614s	0,001843s
16 threads	0.002568s	0.002019s	0.002423s	0,002337s
32 threads	0.003701s	0.003038s	0.003102s	0,003280s
46 threads	0.004302s	0.004253s	0.004118s	0,004224s
64 threads	0.007357s	0.006561s	0.006648s	0,006855s
69 threads	0.007551s	0.007729s	0.007627s	0,007636s

Tabela 14: Tabela da execução com entrada de um cítron com 74 nucleotídeos para a aplicação paralela

---

<sup>1</sup>Esse exemplo pode ser encontrado em <http://www.cbs.dtu.dk/services/NetGene2/fasta.php>.

## 6 Conclusão

Frente os resultados obtidos, é possível notar que o desempenho da aplicação paralela quase sempre se mostrou melhor ou igual ao desempenho da aplicação sequencial. Nos experimentos com uma cadeia pequena (os com o cístron de tamanho 60, o cístron de tamanho 74 e o cístron de tamanho 140) é possível notar que o desempenho da aplicação sequencial foi melhor do que a da aplicação paralela na maioria dos casos. Nesses casos, quanto maior o número de threads para atacar o problema, a tendência do tempo de execução é aumentar. Isso pode ser explicado por fatores como o custo do gerenciamento das threads, pois nesses casos, acaba por não compensar a criação de muitas threads para lidar com um problema relativamente pequeno, e o desbalanceamento de carga (uma divisão não justa de trabalho), que pode fazer com que uma ou mais threads tenham que trabalhar mais que as demais.

Nas execuções com entradas maiores, temos que a aplicação paralela se desempenhou melhor do que a sequencial. No caso de execução da aplicação no cístron de tamanho 14015, é possível notar que todas as execuções obtiveram tempos melhores que a aplicação sequencial. Os melhores tempos de execução se deram nas execuções onde houve uma melhor divisão da carga de trabalho entre as threads, considerando fatores como a quantidade de *cores* do processador e o tamanho do cístron, a fim de evitar uma sobrecarga muito alta para certas threads. Ao mesmo tempo, é possível concluir que não é útil várias threads executando em um problema pequeno (vide a execução com o cístron de tamanho 60), pois o custo de lidar com a gerência de threads acaba sendo muito alto.

## Referências

- Dagum, L. & Menon, R. (1998), ‘Openmp: an industry standard api for shared-memory programming’, *IEEE computational science and engineering* **5**(1), 46–55.
- Morandini, C., Bellinello, L. C. & Constantino, C. (2013), *Coleção Objetivo - Apostila Resumo Biologia 1*.
- Sathe, S. & Shrimankar, D. (2011), Parallelization of dna sequence alignment using openmp, in ‘Proceedings of the 2011 International Conference on Communication, Computing & Security’, ACM, pp. 200–203.
- Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A. et al. (2001), ‘The sequence of the human genome’, *science* **291**(5507), 1304–1351.