Trabalho Programação Paralela e Distribuída: Implementação do algoritmo *K-means* de forma paralela utilizando o *Message Passing Interface* (MPI)

André Aranda[1], **Caio Lara**[2], **Carlos Pereira**[3]

[1]Departamento de Ciência da Computação
Universidade Federal de Lavras (UFLA) – Lavras, MG – Brasil

{andre,caio_freitas,carlospereira}@computacao.ufla.br

Abstract. *This article discribe the process of implementation of the algorithm K-means used in clustering processes for sequential and parallel using interprocess communication by means of the tool Message Passing Interface (MPI) in language C ANSI with the objective of compare time, speedup and efficiency and performance between the two types of implementation and the use of different number of process to parallel algorithm.*

Resumo. *Este artigo tem a finalidade de mostrar a implementação do algoritmo K-means utilizado em processos de clusterização de forma sequencial e de forma paralela utilizando comunicação entre os processos atravéz da ferramenta Message Passing Interface (MPI) na linguagem C ANSI com o objetivo de realizar comparações de tempo, speedup eficiência e performance entre as duas implementações e o uso de diferentes números de processos para o algoritmo implementado de forma paralela.*

1. Introdução

Com as restrições físicas que impedem a escala da frequência dos processadores [Kumar et al. 2008], e como o consumo de energia, e conseqüentemente a geração de calor, por computadores tornou-se uma preocupação nos últimos anos, a computação paralela tornou-se o paradigma dominante na arquitetura de computadores, principalmente na forma de processadores multi-core. [Asanovic et al. 2006]

O aprendizado de máquina é um campo da ciência da computação que dá aos computadores a capacidade de aprender sem serem explicitamente programados. [Samuel 1959] Atualmente há muito expectativas em relação a aplicações que utilizam aprendizado de máquina. [Forni and Meulen 2016] No campo do aprendizado de máquinas, existem algoritmos de clusterização, algoritmos que tentam agrupar dados por meio da semelhança entre os mesmo.

Neste trabalho foi desenvolvido a implementação do algoritmo de clusterização K-Means de forma paralela, de forma a obter o máximo da performance dos computadores modernos em um algoritmo robusto de aprendizado de máquina.

Este trabalho está organizado da seguinte forma, a seção 2 discute sobre o que é o problema de clusterização. Na seção 3 o algoritmo K-means é descrito como uma forma de solução para problemas de clusterização. A seção 4 traz detalhes técnicos sobre a implementação do algoritmo de forma paralela. A seção 5 possui instruções para executar

a aplicação. Na seção 6 a metodologia utilizada para o experimento é descrita. Na seção 7 são discutidos os resultados do experimento. Por fim, a seção 8 traz uma conclusão para este trabalho.

2. Problema de clusterização

Conforme [Cassiano 2015] a clusterização de Dados ou Análise de Agrupamentos é uma técnica de mineração de dados multivariados que através de métodos numéricos e a partir somente das informações das variáveis de cada caso, tem por objetivo agrupar automaticamente por aprendizado não supervisionado os *n* casos da base de dados em *k* grupos, geralmente disjuntos denominados clusters.

Ainda segundo o mesmo autor, ao contrário da classificação, a Clusterização não conta com classes predefinidas e exemplos de treinamento de classes rotuladas.

3. O algoritmo *K-means* para resolver o problema de clusterização

Algoritmos de clusterização consistem em agrupar dados em determinados grupos, chamados de clusters. Os dados devem ficar o mais parecido possível do seu *cluster*.

No algoritmo *K-means* os clusters são chamados centroides e as instâncias de pontos. Pode-se definir que cada ponto e cada centroide possui uma coordenada, que são seus dados. O algoritmo consiste nos seguintes passos básicos:

O algoritmo busca minimizar a função objetivo que é de forma geral define a distância de todos os pontos para seu centroide.

1) Definir inicialmente as coordenadas dos centroides.

2) Calcular a função objetivo.

3) Reposicionar os centroides, calcular novamente a função objetivo enquanto a diferença da função objetivo antiga para a nova for maior que um determinado threshold.

4. O algoritmo *K-means* implementado utilizando o *Message Passing Interface* (MPI)

O usuário poderá definir o número de processos n. Considere p o número do processo.

O algoritmo K-means inicialmente define um processo como *master* (mestre) e outros processos como *workers* (trabalhadores). O processo mestre será o processo de número $p = 0$.

Para leitura da base de dados em paralelo, foi levantado duas alternativas, a primeira delas é pré-processar o arquivo para determinar quais porções do arquivo seriam lidas por determinado processo p, ou converter a base para números binários e calcular o deslocamento de bytes onde cada processo deve iniciar sua leitura. Optou-se pela segunda alternativa, desenvolvendo também uma aplicação para converter uma base de dados em um arquivo binário.

Considere uma base de dados com m instâncias de tamanho s, em *bytes*. A formula $deslocamento = p * (m/n) * s$, define quanto é o deslocamento em *bytes* que a base deve sofrer para cada processo realizar a leitura, ou seja, o resultado desta formula

é o primeiro byte que será lido por determinado processo. A Figura 1 demonstra como a leitura paralela da base de dados é realizada.

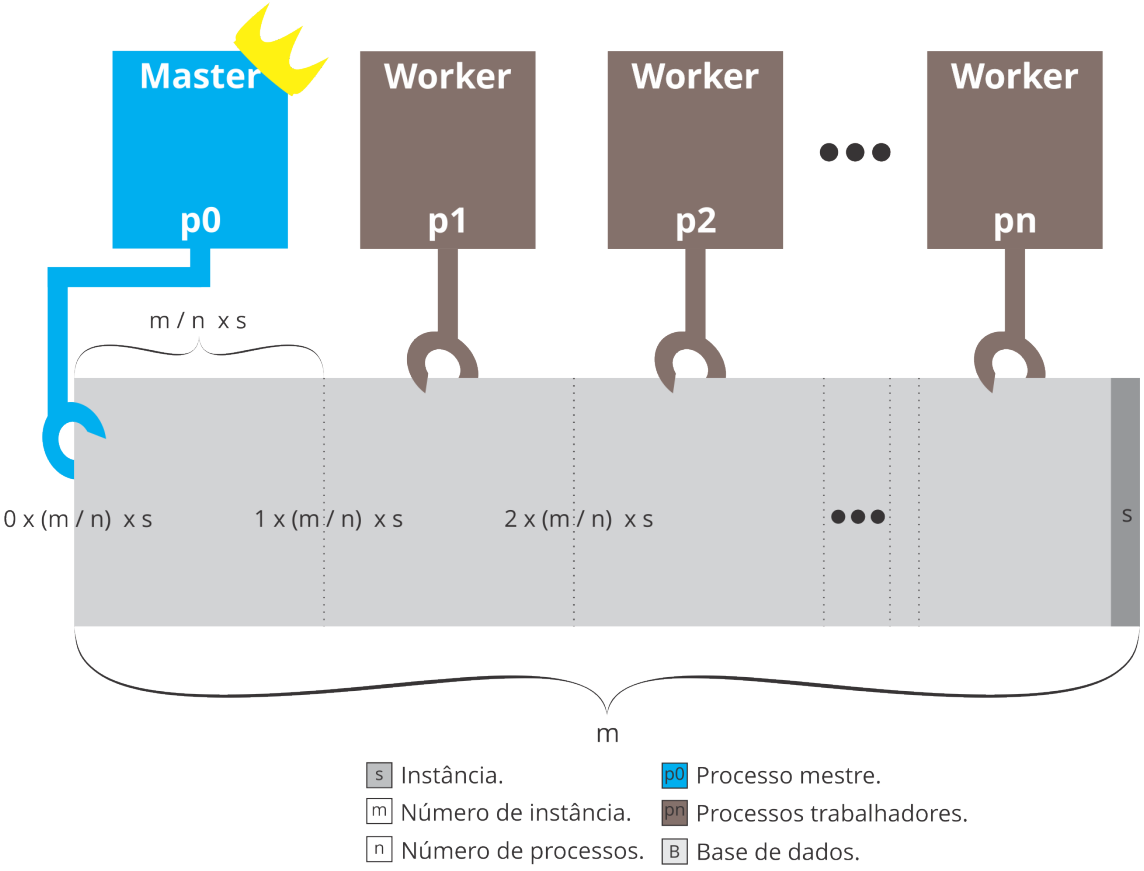

Figura 1. Leitura paralela da base de dados

O algoritmo se inicia, com o processo mestre definindo os índices das instâncias que serão definidos como os centroides iniciais. Os processos *workers* estarão aguardando mensagens. Caso o processo mestre não tenha em sua memória alguma instância definida como centroide inicial, ele requisita a instância ao processo que a possui.

Após esta operação os processos trabalhadores que receberam a solicitação de uma instância, enviam a instância para o mestre, que captura esta resposta.

Após possuir todos os centroides iniciais, o processo mestre envia por broadcast a informação dos centroides para todos os processos trabalhadores. A Figura 2 e o Algoritmo 1 demostram esse procedimento de definição dos centroides iniciais.

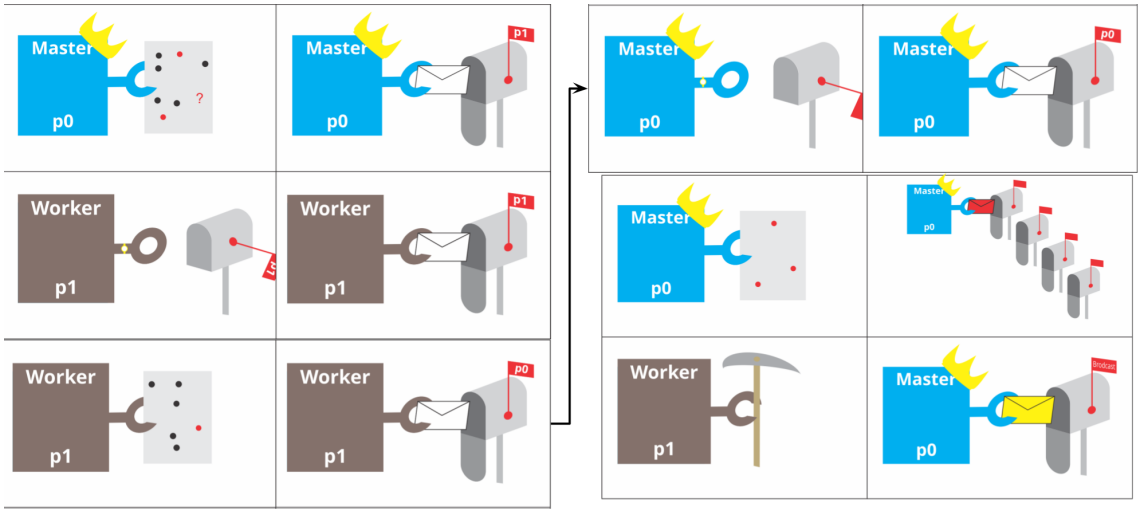

Figura 2. Definição dos centroides iniciais

Algoritmo 1 Definir centroides iniciais

```
1:  procedure CompStartCentroids
2:      if MESTRE then
3:          determinar instâncias como centroides iniciais
4:          for cada instância do centroid inicial do
5:              if não possui instância then
6:                  requisita instância para processo que a mantém
7:          informa fim de requisições aos processos trabalhadores
8:      else
9:          while houver requisições do
10:             envia instância requisitada ao processo mestre
```

Cada processo define à qual cluster suas instâncias pertencem, elas pertencem ao cluster do centroide que possui a menor distância para a determinada instância. Então, o processo calcula sua função objetivo que é definida como: $\sum_{i=1}^{k} \sum_{j=1}^{m} \|X_j - CE_i\|^2 \mid X_j \in C_i$, onde k é a quantidade de clusters, m é a quantidade de instâncias, X_j é a instância j, CE_i é o centroide que pertence ao cluster i, e C_i é o cluster i. O Algoritmo 2 e a Figura 3 demonstram esse cálculo.

Algoritmo 2 Calcular função objetivo e define cluster de cada instância

```
1:  function CompFuncObj
2:      sum_square_distance ← 0
3:      for cada instância do
4:          coloca a instância no cluster com a menor distância para seu centroide
5:          sum_square_distance += quadrado da distância da instância para centroide
    do cluster que pertence
        return sum_square_distance
```

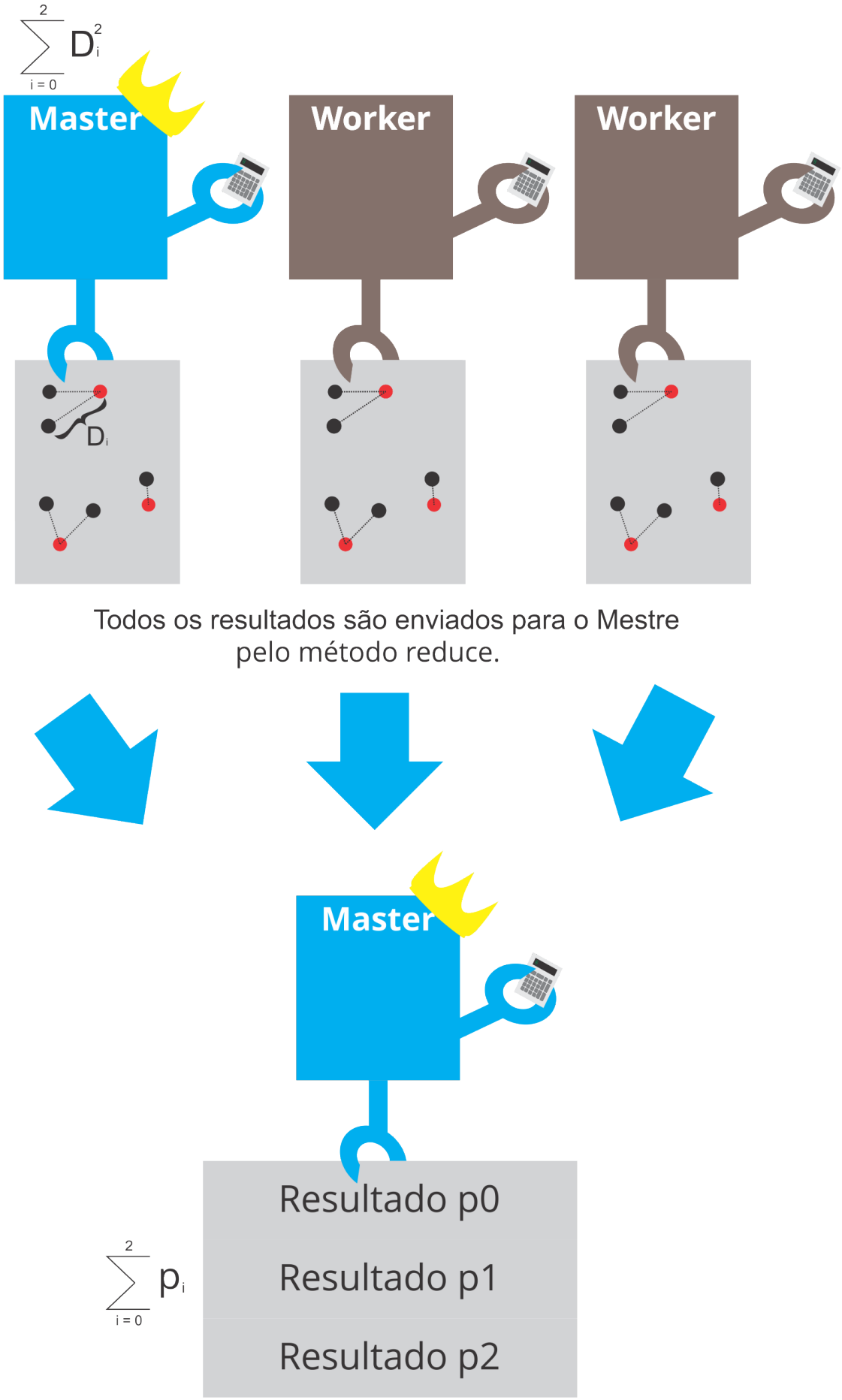

Figura 3. Os processos realizam o somatório das instâncias e fazem um reduce para o processo mestre.

Com as funções objetivo locais os processos reduzem para todos processos o somatório das funções objetivo locais que representam a função objetivo global.

Após esses procedimentos, repete-se os próximos passos até que a diferença da nova função objetivo para a atual seja não seja maior que um determinado *threshold*. Esses próximos passos são: calcular os centroides de cada cluster, definir à qual cluster cada instância pertence, calcular a nova função objetivo, e reduzir o somatório o somatório das funções objetivo para uma global em todos os processos.

Destes passos, apenas calcular os centroides de cada cluster não foi detalhado, portanto para realizar esse cálculo cada processo cálcula o somatório dos elementos de cada cluster local e quantidade de elementos em cada cluster local. Então, reduz o somatório destas duas listas para o processo mestre, o processo mestre agora possui o somatório dos elementos de cada cluster global e a quantidade de elementos em cada cluster global. Com isso, o processo mestre cálcula os novos centroides (ponto médio) dos clusters. Para os clusters que estão vazios, o mestre sorteia uma instância para definir como centroide, caso ele não possua essa instância ele a requisita ao processo que a mantém. Por fim, todos processos recebem os novos centroides do processo mestre via broadcast. Esse procedimento está detalhado no Algoritmo 3 e nas Figuras 4 e 5.

Algoritmo 3 Calcular centroide de cada clusters

```
1:  procedure CalcClusterCentroids
2:      for para cada cluster C_i do
3:          sum_cluster_local[i] ← Σ_{j=1}^{m} X_j | X_j ∈ C_i, onde m é a quantidade de instâncias, X_j é a instância j
4:          count_cluster_local[i] ← quantidade de elementos do cluster i
5:      sum_cluster_global ← redução do somatório de sum_cluster_local
6:      count_cluster_global ← redução do somatório de count_cluster_local
7:      if MESTRE then
8:          for para cada cluster C_i do
9:              if C_i != ∅ then
10:                 centroid[i] ← sum_cluster_global[i] ÷ count_cluster_global[i]
11:             else
12:                 centroid[i] ← uma instância definida pelo mestre. Se não possui a instância, requisita ao processo que possui
13:         informa fim de requisições aos processos trabalhadores
14:     else
15:         while houver requisições do
16:             envia instância requisitada ao processo mestre
17:     envio dos novos centroides para todos processos via broadcast
```

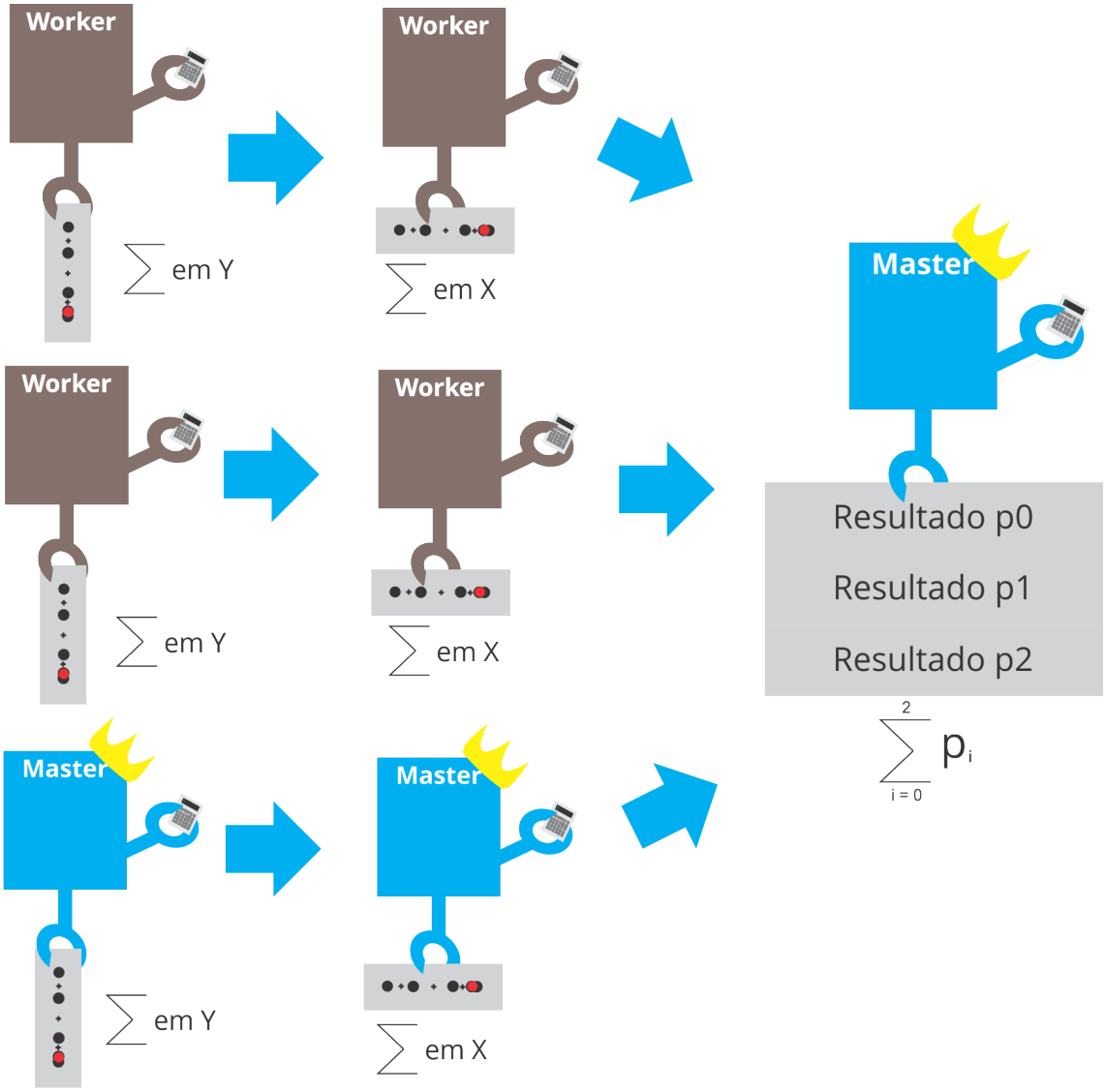

Figura 4. Os processos realizam o somatório das instâncias e fazem um reduce para o processo mestre.

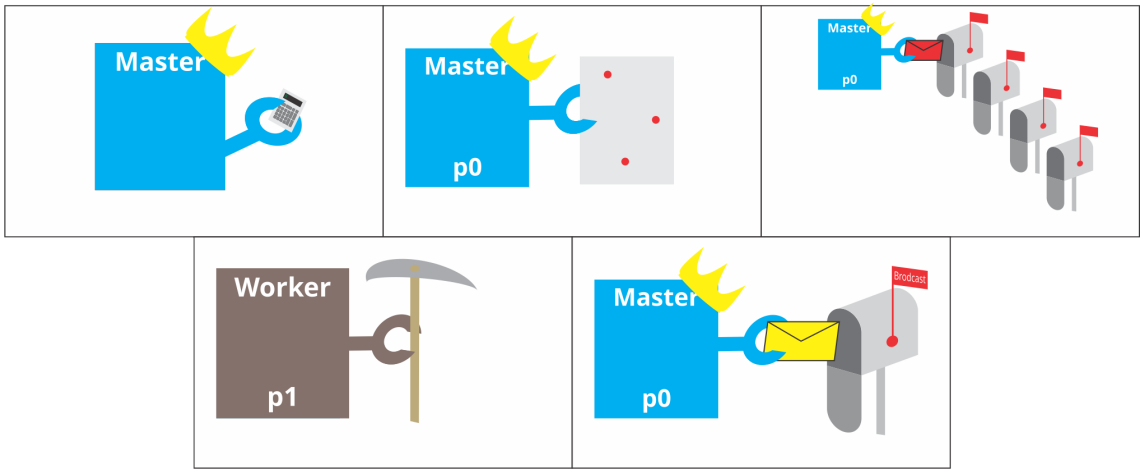

Figura 5. O processo mestre calcula os centroides e envia por broadcast para os trabalhadores.

Após o algoritmo convergir, a diferença das funções objetivos for menor que o threshold, é realizada a escrita do relatório que informa os centroides e elementos de cada cluster. O algoritmo paralelo da escrita do relatório está definido em Algoritmo **??**.

Algoritmo 4 Escrever relatório

```
1:  procedure WRITEREPORT
2:      if MESTRE then
3:          for para cada cluster C_i do
4:              escreve o centroide do cluster i
5:              escreve os índices das instâncias que pertencem ao cluster i
6:              for para cada processo p do
7:                  solicita os índices das instâncias que pertencem ao cluster i ao processo p
8:                  escreve os índices enviados pelo processo p
9:          informa fim de requisições aos processos trabalhadores
10:     else
11:         while houver requisições do
12:             envia índice das instâncias do cluster requisitado para o processo mestre
```

O algoritmo geral do k-means paralelo com MPI está definido no Algoritmo 5.

5. Instruções para execução da aplicação

Nesta seção será informado como compilar e executar as aplicações do projeto, que estão disponíveis no GitHub, o link é https://github.com/CarlosPereira27/k-means-mpi.

5.1. Compilando o projeto

Para compilar o projeto utilize o comando:

```
# make
```

Então, serão gerados 3 executáveis no diretório bin:

Algoritmo 5 K-means MPI

1: cada processo p lê uma parte da base de dados
2: define os centroides iniciais com o procedimento *CompStartCentroids* do Algoritmo 1
3: calcula a função objetivo e atualiza o cluster de cada instância com a função *CompFuncObj* do Algoritmo 2
4: $func\_obj' \leftarrow$ reduz para todos processos o somatório das funções objetivo locais em uma função objetivo global
5: **repeat**
6: $\quad func\_obj \leftarrow func\_obj'$
7: $\quad$ calcula os centroides de cada cluster com o procedimento *CalcClusterCentroids* do Algoritmo 3
8: $\quad$ calcula a função objetivo e atualiza o cluster de cada instância com a função *CompFuncObj* do Algoritmo 2
9: $\quad func\_obj' \leftarrow$ reduz para todos processos o somatório das funções objetivo locais em uma função objetivo global
10: **until** $func\_obj - func\_obj' <= THRESHOLD$
11: escreve o relatório com o procedimento *WriteReport* do Algoritmo 4

- `bin_database_writer` - converter uma base de dados em csv em uma base de dados binária
- `kmeans_mpi_app` - implementação multi-processos do algoritmo *k-means* usando com a biblioteca MPI
- `kmeans_seq_app` - implementação sequencial do algoritmo *k-means*

5.2. Gerando base de dados binária

Para gerar uma base de dados binária com a aplicação `bin_database_writer`, execute-a da seguinte forma:

```
# ./bin_database_writer <path_database> <class_column>
```

onde:

- `<path_database>` - é o caminho onde a base de dados csv está salva
- `<class_column>` - é um inteiro que indica o índice da coluna onde a classe da instância está localizada

LIMITAÇÃO - Essa aplicação só funciona para base de dados onde todas *features* das instâncias são numéricas, exceto a classe.

5.3. Executando o k-means

A aplicação `kmeans_mpi_app` executa o `k-means` com multiprocessos usando MPI. Para executar essa aplicação utilize o comando abaixo:

```
# mpirun -np <num_process> ./kmeans_mpi_app
<path_database_bin> <num_features> <k> <debug_mode>
```

onde:

- `<num_process>` - quantidade de processos que a aplicação deve utilizar para executar
- `<path_database_bin>` - caminho onde a base de dados binária está salva

- <num_features> - quantidade de *features* que as instâncias da base de dados possui
- <k> - quantidade de clusters em que o algoritmo deve agrupar as instâncias
- <debug_mode> - parâmetro opcional. Valores válidos para esse parâmetro são -d"ou --debug". Se executar com este parâmetro, executará no modo de debug, onde a cada iteração irá mostrar os centroides atuais e a função objetivo.

Essa aplicação irá gerar um relatório no arquivo com o nome <path_database_bin>.clusters que contém informações sobre os clusters gerados pela aplicação, onde cada cluster terá o seu centroide e também os índices das instâncias que estão contidas nesse cluster.

A aplicação kmeans_seq_app executa o k-means de forma sequencial. Para executar essa aplicação utilize o comando abaixo:

```
# ./kmeans_seq_app <path_database> <class_column> <k> <debug_mode>
```

onde:

- <path_database> - é o caminho onde a base de dados csv está salva
- <class_column> - é um inteiro que indica o índice da coluna onde a classe da instância está localizada
- <k> - quantidade de clusters em que o algoritmo deve agrupar as instâncias
- <debug_mode> - parâmetro opcional. Valores válidos para esse parâmetro são -d"ou --debug". Se executar com este parâmetro, executará no modo de debug, onde a cada iteração irá mostrar os centroides atuais e a função objetivo.

Essa aplicação irá gerar um relatório no arquivo com o nome <path_database>.clusters que contém informações sobre os clusters gerados pela aplicação, onde cada cluster terá o seu centroide e também os índices das instâncias que estão contidas nesse cluster.

6. Metodologia

Nesta seção é descrito a metodologia dos experimentos realizados. Identificando o ambiente onde os experimentos foram executados, as bases de dados utilizadas no experimento e o procedimento do experimento.

6.1. Ambiente do experimento

Como o algoritmo exige muito processamento, para realização do experimento foi alocada uma máquina virtual na nuvem, atravéz da plataforma Google Cloud com um processador Intel Xeon de 24 núcleos e 24 GB de memória RAM, o sistema operacional utilizado foi a distribuição Linux Ubuntu 16.04.3 LTS x86_64 GNU/Linux (xenial) com o kernel do linux versão 4.13.0-1008-gcp. No sistema foi instalado o Open MPI versão 1.10.2 com MPICH versão 3.2.

6.2. Base de dados

As bases de dados *Image Segmentation Data Set* e *Letter Recognition Data Set* do repositório [Lichman 2013] foram utilizadas para o experimento. No entanto para aumentar a robustez do experimento, decidiu-se criar uma versão dez vezes maior de cada base de dados. Para isso, foi desenvolvimento um script em linguagem Python para gerar instâncias

aleatórias, porém estas instâncias estavam dentro do padrão, isto é valor máximo e mínimo das features das instâncias da base de dados original. As instâncias de base dados *Image Segmentation Data Set* contém informações sobre desenhos aleatórios de 7 imagens de uma determinada base de dados. As instâncias da base de dados *Letter Recognition Data Set* contém informações sobre imagens de caracteres a fim de identificar letras. A Tabela 1 contém informações sobre as bases de dados utilizadas.

Tabela 1. Informação das base de dados

nome	qtd_atributos	qtd_instâncias	tamanho
Image Segmentation	19	2310	379,2 kB
Letter Recognition	16	20000	712,6 kB
Image Segmentation 10 Times	19	23100	7,6 MB
Letter Recognition 10 Times	16	200000	7,9 MB

6.3. Procedimento do experimento

Para elaboração do experimento foram definidos os seguintes números de processos: 1, 2, 3, 4, 6, 8, 12, 16, 20, 24, 32, 40, onde para cada uma das base de dados foram executadas dez vezes. O motivo de se executar dez vezes foi para ganhar confiança, caso em algum momento, algo atrapalhe o processamento da máquina. Utilizou-se dois cenários de números de processos maiores do que a capacidade da máquina, para analisar o quanto a troca de contexto de processos pode atrapalhar no processamento.

7. Resultados

Na Figura 6 nota-se o gráfico da relação entre o logaritmo do tempo médio de execução, em microsegundos, pela quantidade de processos. Este gráfico, contém uma linha para cada base de dados, o uso do logaritmo foi utilizado para ficar claro que o comportamento do gráfico é o muito próximo para todas as bases de dados, mesmo as bases tendo escala de tempos diferentes, como pode ser visualizado nas tabelas com os resultados no Anexo A - Resultados.

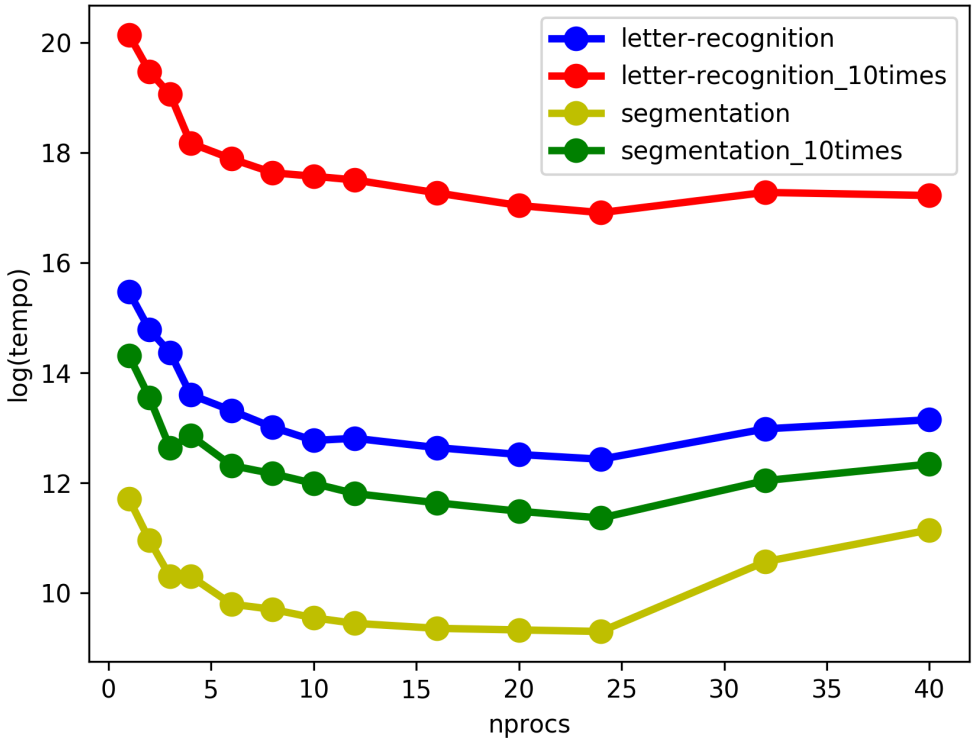

Figura 6. Logaritmo do tempo médio de execução

O *speedup*, é a relação de ganho de tempo de execução paralelo em relação ao sequencial do mesmo algoritmo.

Como pode-se observar na Figura 7, o *speedup* no gráfico se eleva até a execução com 24 processos e após este momento passa a cair. Em vários pontos do gráfico podemos observar o *speedup* superlinear, isto é o *speedup* está superior a quantidade de processos em determinados tempos. Além das linhas de *speedup* de cada base de dados, têm-se a linha do *speedup* ideal.

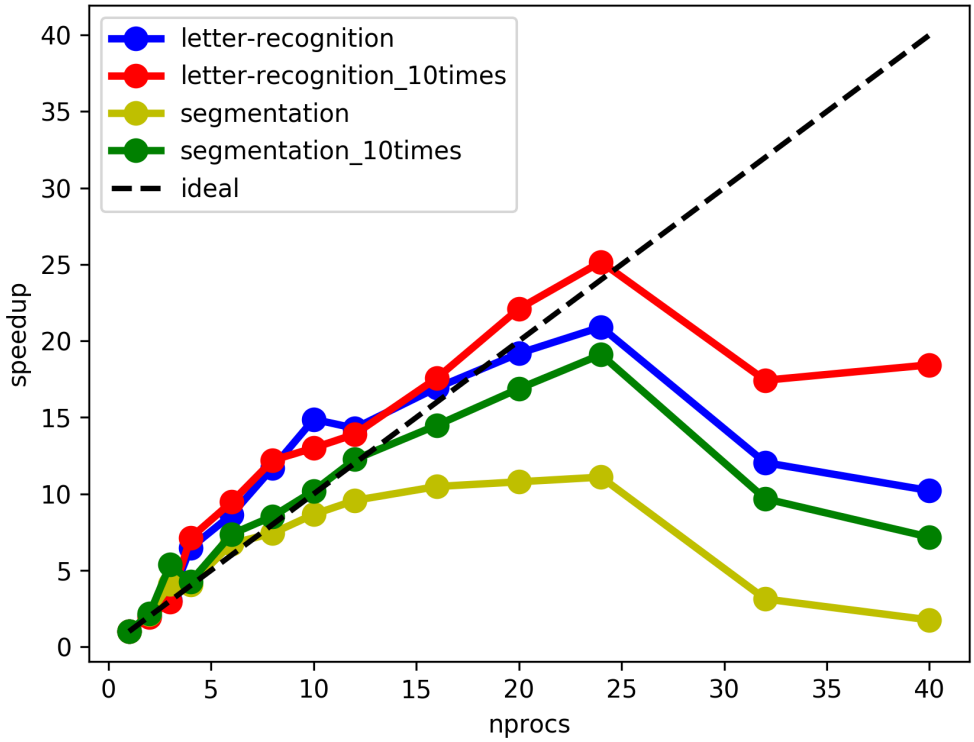

Figura 7. *speedup* do algoritmo paralelo

Na Figura 8, nota-se que o gráfico tem as linhas de eficiência pela quantidade de processos para cada base de dados e também tem a linha da eficiência ideal. Como o *speedup* superlinear foi alcançado pelo algoritmo, pode-se perceber que a eficiência passa dos 100% em grande parte do gráfico.

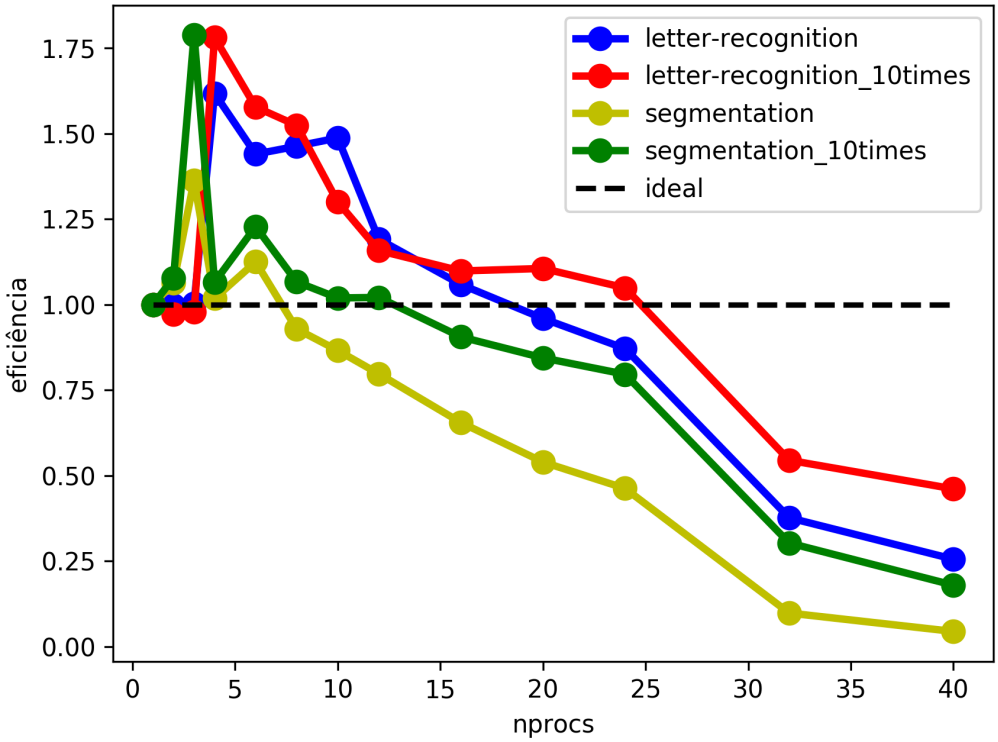

Figura 8. Eficiência do algoritmo paralelo

No Anexo A - Resultados estão as tabelas com os resultados dos experimentos executados, onde as tabelas informam o tempo médio de execução das 10 execuções para cada processo, também é informado o intervalo de confiança com um grau de confiança de 95%, o *speedup* e a eficiência do algoritmo paralelo.

8. Conclusão e trabalhos futuros

Com este trabalho podemos concluir que a implementação do algoritmo K-means de forma paralela utilizando MPI se beneficia do uso de multi-processos, o ganho em utilizar o algoritmo de forma paralela é maior que as perdas por comunicação entre processos. Podemos observar também que além do uso de vários processos, o algoritmo se beneficia de outras caracteríticas dos processadores com mais de um núcleo, pois o *speedup* (ganho) atingido foi superlinear. Esse *speedup* superlinear mostra que o ganho com localização de memória, por meio dos níveis mais baixos de *cache*, resultando em menos *miss*, é muito maior do que a perda em ralação a comunicação, pois só assim esse algoritmo conseguiu atingir esse *speedup* superlinear. Outro ponto a se analisar, é que quando executa-se o algoritmo com mais processos do que a capacidade da máquina, o desempenho piora, devido as trocas de contexto de processos.

Referências

Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W., et al. (2006). The landscape of

parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

Cassiano, K. M. (2015). Análise de séries temporais usando análise espectral singular (ssa) e clusterização de suas componentes baseada em desindade. pages 59–96. PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO - PUC-RIO.

Forni, A. and Meulen, R. (2016). Gartner's 2016 hype cycle for emerging technologies identifies three key trends that organizations must track to gain competitive advantage.

Kumar, R., Marinov, D., Padua, D., Parthasarathy, M., Patel, S., Roth, D., Snir, M., and Torrellas, J. (2008). Parallel computing research at illinois the upcrc agenda.

Lichman, M. (2013). UCI machine learning repository.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

9. Anexo A - Resultados

Tabela 2. letter-recognition

nprocs	média_tempo	intervalo_de_confiança	speedup	eficiência
1	5243274	[5204762 - 5281786]	1.000000	1.000000
2	2628204	[2620774 - 2635634]	1.995003	0.997501
3	1742152	[1737852 - 1746452]	3.009654	1.003218
4	810471	[804226 - 816716]	6.469416	1.617354
6	606821	[522205 - 691437]	8.640561	1.440094
8	448020	[397322 - 498718]	11.703214	1.462902
10	352538	[329443 - 375633]	14.872933	1.487293
12	366864	[330508 - 403220]	14.292146	1.191012
16	309528	[300905 - 318151]	16.939579	1.058724
20	273309	[259235 - 287383]	19.184418	0.959221
24	250787	[241935 - 259639]	20.907280	0.871137
32	435399	[392360 - 478438]	12.042458	0.376327
40	513567	[465067 - 562067]	10.209523	0.255238

Tabela 3. letter-recognition - 10times

nprocs	média_tempo	intervalo_de_confiança	speedup	eficiência
1	556245969	[555394437 - 557097501]	1.000000	1.000000
2	286028735	[284200993 - 287856477]	1.944721	0.972360
3	189540316	[188701942 - 190378690]	2.934711	0.978237
4	78093868	[77177227 - 79010509]	7.122787	1.780697
6	58762040	[50822758 - 66701322]	9.466077	1.577679
8	45626456	[44505535 - 46747377]	12.191303	1.523913
10	42783667	[42130202 - 43437132]	13.001363	1.300136
12	40029602	[36199159 - 43860045]	13.895866	1.157989
16	31656583	[30102764 - 33210402]	17.571257	1.098204
20	25158317	[23968075 - 26348559]	22.109824	1.105491
24	22101935	[20712136 - 23491734]	25.167297	1.048637
32	31919388	[31247789 - 32590987]	17.426586	0.544581
40	30172949	[29879665 - 30466233]	18.435254	0.460881

Tabela 4. segmantation

nprocs	média_tempo	intervalo_de_confiança	speedup	eficiência
1	121779	[120208 - 123350]	1.000000	1.000000
2	57282	[56542 - 58022]	2.125956	1.062978
3	29794	[26427 - 33161]	4.087367	1.362456
4	29914	[28720 - 31108]	4.070970	1.017743
6	18043	[16822 - 19264]	6.749376	1.124896
8	16397	[16122 - 16672]	7.426907	0.928363
10	14067	[13362 - 14772]	8.657070	0.865707
12	12726	[12043 - 13409]	9.569307	0.797442
16	11622	[11111 - 12133]	10.478317	0.654895
20	11289	[10506 - 12072]	10.787404	0.539370
24	10979	[10706 - 11252]	11.091994	0.462166
32	38960	[30554 - 47366]	3.125744	0.097680
40	69440	[56443 - 82437]	1.753730	0.043843

Tabela 5. segmantation - 10times

nprocs	média_tempo	intervalo_de_confiança	speedup	eficiência
1	1646147	[1634030 - 1658264]	1.000000	1.000000
2	763687	[761994 - 765380]	2.155526	1.077763
3	306869	[300135 - 313603]	5.364331	1.788110
4	386188	[384140 - 388236]	4.262553	1.065638
6	223431	[205927 - 240935]	7.367586	1.227931
8	192998	[180474 - 205522]	8.529347	1.066168
10	161501	[155191 - 167811]	10.192798	1.019280
12	134389	[127772 - 141006]	12.249120	1.020760
16	113639	[110243 - 117035]	14.485758	0.905360
20	97520	[94253 - 100787]	16.880096	0.844005
24	86162	[82762 - 89562]	19.105255	0.796052
32	170020	[157741 - 182299]	9.682079	0.302565
40	229648	[205404 - 253892]	7.168131	0.179203