Revista Tecnologia
e-ISSN: 2318-0730
10.5020/23180730.2020.11310

Serialização de dados em processamento digital de sinais: um estudo de caso

Data serialization in digital signal processing: a case study

Serialización de datos en procesamiento digital de señales: un estudio de caso

Sérialisation de données dans le traitement numérique du signal: une étude de cas

João Paulo Lemos Escola
jpescola@ifsp.edu.br
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, campus Barretos (IFSP)

Tiago Alexandre Dócusse
tad@ifsp.edu.br
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, campus Barretos (IFSP)

Resumo

Os testes em laboratório com algoritmos que empregam processamento digital de sinais costumam dispender valiosos recursos de energia e tempo por serem compostos por longas rotinas de processamento de dados que, apesar de discretizados no tempo, requerem multiplicações de matrizes que têm alto custo computacional. A serialização é uma técnica que permite aproveitar um processamento longo realizado previamente a fim de tornar o algoritmo mais rápido e econômico no tocante à energia e ao tempo. Este estudo apresenta a serialização como recurso auxiliar no trabalho de laboratório em testes envolvendo processamento digital de sinais. Os resultados mostram considerável melhoria no tempo de resposta em reprocessamento de dados utilizando a técnica de serialização, que pode ser empregada em diversos tipos de algoritmos que envolvem processamento repetitivo de dados.

Palavras-chave: Processamento de sinais. Serialização. Estruturas de dados. Sistemas de informação.

Abstract

Laboratory tests with algorithms that employ digital signal processing tend to spend valuable energy and time resources as they are composed of long data processing routines, which despite being discretized in time, require matrix multiplications that have high computational cost. Serialization is a technique that allows taking advantage of a long processing previously performed in order to make faster and more economical algorithms in terms of energy and time. This study presents serialization as an auxiliary resource in laboratory work in tests involving digital signal processing. The results show a considerable improvement in the response time in data reprocessing using the serialization technique, which can be used in several types of algorithms that involve repetitive data processing.

Keywords: Signal processing. Serialization. Data structures. Information systems.

Resumen

Los tests en laboratorio con algoritmos que emplean procesamiento digital de señales suelen gastar valiosos recursos de energía y tiempo,visto que son compuestos por largas rutinas de procesamiento de datos que, aunque discretos en el tiempo, requieren multiplicaciones de matrices que tienen alto coste computacional. La serialización es una técnica que permite aprovechar un procesamiento largo realizado con antelación con el objetivo de volver el algoritmo más rápido y económico a lo que se refiere a energía y tiempo. Este trabajo presenta la serialización como recurso auxiliar en el trabajo de laboratorio en ensayos con procesamiento digital de señales. Los resultados presentan mejoría considerable en el

tiempo de respuesta en procesamiento de datos utilizando la técnica de serialización, que puede ser empleada en distintos tipos de algoritmos con procesamiento repetitivo de datos.

Palabras-clave: Procesamiento de señales. Serialización. Estructuras de datos. Sistemas de información.

Résumé

Les tests en laboratoire avec des algorithmes qui utilisent le traitement numérique du signal ont tendance à dépenser de précieuses ressources en énergie et en temps car ils sont composés de longues routines de traitement de données qui, bien que discrétisées dans le temps, nécessitent des multiplications matricielles qui ont un coût de calcul élevé.La sérialisation est une technique qui permet de profiter d'un long traitement précédemment effectué afin de rendre l'algorithme plus rapide et plus économique en énergie et en temps. Cette étude présente la sérialisation comme une ressource auxiliaire dans les travaux de laboratoire dans les tests impliquant le traitement numérique du signal.Les résultats montrent une amélioration considérable du temps de réponse dans le retraitement des données à l'aide de la technique de sérialisation, qui peut être utilisée dans plusieurs types d'algorithmes qui ont un traitement répétitif des données.

Mots-clés: Traitement du signal Sérialisation Structures de données. Systèmes d'information.

1 Introdução

Por meio do processamento digital de sinais (PDS) é possível executar diversos tipos de análises de dados, como os oriundos de arquivos de áudio e imagens digitais. Dependendo do tamanho do sinal a ser processado, mesmo procedimentos considerados simples, como multiplicações de matrizes, podem dispender considerável recurso de processamento. Também podem ser necessárias aplicações de transformadas matemáticas, que possibilitam filtragens e separações dos sinais em sub-bandas, como a transformada discreta de fourier (*Discrete Fourier Transform* – DFT) (LATHI, 2006), e a transformada discreta *wavelet* (*DiscreteWaveletTransform* – DWT) (JENSEN; COUR-HARBO, 2001; WALKER, 2002; ADDISON, 2017; GUIDO, 2017), exemplos de técnicas empregadas em PDS para conversão de sinais do domínio do tempo para o domínio da frequência ou, ainda, para o domínio tempo-frequência.

As técnicas de laboratório, que empregam algoritmos aplicando técnicas de processamento digital de sinais, costumam tratar grandes volumes de dados, e nem sempre contam com equipamentos capazes de processar essa quantidade de dados em tempo satisfatório, fazendo com que os pesquisadores busquem por soluções que possibilitem melhoria do desempenho de seus algoritmos.

O tempo de processamento de um algoritmo é fator crucial em muitos casos e está diretamente relacionado com seu nível de complexidade. Utilizando a notação Big-O (DROZDEK, 2002; TENENBAUM, 2004; CORMEN, *et al.,* 2017) podemos conhecer a complexidade de nosso algoritmo a fim de relacionar essa informação com seu tempo de processamento, buscando, até mesmo, outras formas de solução do problema, quando possível. Segundo a notação Big-O, algoritmos de complexidade $O(n^2)$, $O(n^3)$ e $O(2^n)$ costumam ser os mais custosos, dispendendo alto recurso de processamento e podendo inviabilizar processos devido à possibilidade de lentidão no tempo de resposta.

Uma alternativa para melhoria do tempo de processamento de algoritmos computacionais, como nos casos de PDS, pode ser a implementação de técnicas de programação paralela. Entretanto, nesse caso, é necessário adaptar o algoritmo desenvolvido, utilizando as rotinas da biblioteca de paralelização definida, obrigando o pesquisador a adaptar seu código e aumentando sua complexidade (SATO, 1996; TORELLI, 2004).

Como alternativa para a melhoria do desempenho do algoritmo e, até mesmo, para o caso de implementação conjunta com programação paralela, uma alternativa pode ser a expansão dos recursos de *hardware*, incluindo memória RAM (*Random Access Memory* – Memória de Acesso Aleatório) ou adquirindo novos equipamentos, mais modernos, o que pode inviabilizar a pesquisa (MORIMOTO, 2002; VASCONCELOS, 2007).

As memórias do tipo RAM armazenam os dados que são diretamente acessados pelo processador, possuem custo por capacidade de armazenamento maior e estão disponíveis comercialmente, nos tempos atuais, na casa dos *gigabytes* para computadores pessoais. Já as memórias de armazenamento secundário, como os discos rígidos, possuem custo por capacidade de armazenamento menor e são utilizadas para armazenamento de arquivos ou dados que não são diretamente acessados pelo processador, ou seja, podem ser acessados mediante requisição de leitura pelo sistema operacional (MORIMOTO, 2002; VASCONCELOS, 2007). Estas estão disponíveis comercialmente na casa dos *terabytes* atualmente. Esse fator faz com que a serialização se apresente como uma alternativa viável para economia de recursos computacionais, além de possibilitar ganho de desempenho.

Nesse contexto, a técnica da serialização de dados (PHILIPPSEN *et al.,* 1999; MIREKU, 2007; DALCÍN et al., 2008; MAEDA, 2012) pode ser uma alternativa para reduzir o custo computacional dos algoritmos nos procedimentos de análise de dados, além de poder ser aplicada em qualquer algoritmo em que exista a possibilidade de aproveitamento de processamento de dados prévios, buscando reduzir o tempo de resposta. Na técnica da serialização, os dados previamente processados são armazenados em disco no formato empregado no algoritmo (variável ou objeto), provendo uma forma de acesso rápido aos dados em disco sem a necessidade de reprocessamento.

Para exemplificar o conceito de serialização, podemos referenciar o termo *cache* (SMITH, 1982; FRIGO, 1999), muito utilizado em diversas tecnologias, como navegadores de *internet,* e dispositivos, como discos rígidos. Essa tecnologia consiste no armazenamento temporário de dados para consulta posterior, procurando reduzir o tempo de resposta e economizar recursos. Assim como as memórias *cache*, a serialização busca viabilizar ao programador uma memória de dados para aumentar o desempenho de seu algoritmo, decrescendo seu tempo de resposta.

No artigo de Philippsen *et al.* (1999)são apresentados os detalhes de um estudo voltado ao uso da serialização de objetos remotos implementados em *Java*, em que os autores desenvolvem melhorias em algumas bibliotecas de serialização buscando aprimorar seu desempenho. Segundo os autores, a serialização é um procedimento que vale a pena ser implementado, visto que seu tempo de leitura em disco demanda de 25% a 50% do tempo que levaria para recuperá-lo de um servidor remoto na Internet.

Em Mireku (2007) temos uma patente de uma tecnologia desenvolvida para serialização e preservação de objetos. O intuito do trabalho é definir um padrão de objetos que sejam serializados em formato de marcação, como XML, e possam ser recuperados sem a necessidade de utilizar uma linguagem de programação, permitindo que as definições das classes dos objetos serializados sejam recuperadas posteriormente.

Segundo Dalcín *et al.* (2008), que estudam o padrão MPI (*MessagePassing Interface*) da linguagem *Python*, a serialização é um importante recurso para essa e outras técnicas de desenvolvimento de *software*. Segundo eles, nessa linguagem, a exemplo de outras, existem ainda diversos mecanismos de persistência de dados em disco rígido que possibilitam armazenamento temporário de variáveis e objetos em padrão ASCII ou binário.

No trabalho de Maeda (2012) são comparadas diversas bibliotecas de serialização, em diversos formatos, medindo o tempo de processamento e o tamanho do arquivo resultante. Segundo o autor, quando aplicamos a serialização, há prejuízos em relação ao desempenho do algoritmo, entretanto se mostra uma solução compensatória para diversas aplicações. O artigo relata também que,entre as bibliotecas estudadas, apenas a biblioteca *Apache Avro* tem suporte à linguagem de programação C, adotada no presente trabalho.

O objetivo do presente trabalho é apresentar um estudo comparativo do uso de serialização em algoritmos de alta complexidade de processamento digital de sinais, buscando auxiliar pesquisadores e cientistas da computação na procura de algoritmos mais rápidos para uso em testes de PDS em laboratório. Para isso, foi desenvolvido um conjunto de algoritmos que utilizam a técnica de serialização e efetuados testes de desempenho calculando seu tempo de execução.

Nos tópicos a seguir, apresentaremos a metodologia empregada no presente trabalho e, em seguida, apresentaremos e discutiremos os resultados obtidos, passando pelas conclusões e finalizando com possíveis trabalhos futuros.

2 Metodologia

Os algoritmos desenvolvidos neste trabalho foram escritos na linguagem de programação *C*, pois se trata de uma das linguagens mais populares da atualidade e é amplamente empregada em pesquisas científicas envolvendo processamento digital de sinais (EMBREE, 1991; TOMPKINS, 1993; SUNG *et al.,* 1995). Além disso, é uma linguagem compilada, cujo código-fonte gerado é nativo do sistema operacional para o qual a compilação foi realizada, não sendo necessário interpretar esse código por meio de interpretadores ou máquinas virtuais, o que acarretaria em perda de desempenho dos algoritmos escritos. Foram desenvolvidos três algoritmos com o intuito de simular uma situação real de uso em laboratório, buscando imitar a possível complexidade computacional que os pesquisadores e cientistas de computação podem se deparar.

Os algoritmos implementam cálculos de alta complexidade utilizando laços de repetição aninhados de multiplicação de matrizes nos formatos 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, 2048x2048, 4096x4096 e 8192x8192, respectivamente. Esse conjunto de matrizes busca simular diferentes resoluções espaciais de imagens monocromáticas para o processamento dos algoritmos de testes, detalhados a seguir. Além desses, um algoritmo que implementa o processo simplificado de escrita e leitura de dados em disco foi desenvolvido para comparar o tempo de resposta de leitura/escrita de dados com o processamento de dados em si, buscando corroborar a aplicabilidade da serialização no processo. Não foi utilizado nenhum algoritmo de compressão de dados, como os utilizados para salvar imagens em disco, para que as diferenças entre esses algoritmos não interferissem nos resultados obtidos.

A primeira etapa do presente trabalho, chamada E0, é composta de três algoritmos, que simulam altas complexidades, a fim de realizarmos a medição do tempo de processamento e compará-lo com o tempo de leitura dos dados em disco.

Em seguida, foram realizados os testes de escrita/leitura em disco para medir o tempo necessário para a leitura e a escrita dos arquivos em disco, considerando as matrizes do conjunto de testes, sendo chamada de etapa E1. Essa medição é importante de ser feita, pois, para que a serialização seja realizada, é necessário gravar os dados desejados no disco na primeira vez que a operação é realizada, e, em todas as operações seguintes, é necessário realizar a leitura desses dados previamente gravados no disco, ao invés de refazer os cálculos.

A etapa seguinte consistiu na execução de três algoritmos de processamento digital de sinais em imagens, chamada etapa E2. Nessa, o algoritmo “binarização” realiza o processo de redução dos níveis de intensidade de uma imagem digital, e, para uma imagem em níveis de cinza, ou seja, uma imagem que possui intensidade de seus *pixels* variando do preto ao branco em diferentes tonalidades da cor cinza, a transforma em uma imagem com apenas dois níveis de cinza, normalmente o preto e o branco, de acordo com um limiar definido. Os algoritmos “filtragem da média” e “filtragem da mediana” realizam o processamento da imagem, filtrando possíveis ruídos, sendo este para ruídos em geral e aquele para ruídos do tipo *salt-and-pepper* ou salpicado (LIM, 1990; GONZALES e WOODS, 2010).

Nos três algoritmos da etapa E2, todos os *pixels* das matrizes que representam as imagens de entrada são analisados, e determinadas ações são tomadas, de acordo com a operação desejada para cada *pixel* da imagem. Como as imagens de teste são quadradas, ou seja, possuem a mesma quantidade de linhas e colunas, seja N o valor da altura e da largura de uma imagem, percebe-se que as operações serão realizadas nos N^2 *pixels* da imagem de entrada. Dessa forma, os algoritmos podem ser classificados como pertencentes à classe de algoritmos quadráticos, de notação $O(n^2)$, isto é, ao aumentar o valor de N, o número de *pixels* a serem calculados aumenta de forma quadrática, tornando o algoritmo mais custoso computacionalmente.

O algoritmo da binarização é um algoritmo que trabalha apenas com o valor de cada *pixel* da imagem, classificado como uma operação ponto a ponto (GONZALES e WOODS, 2010). Os algoritmos da filtragem da média e da mediana trabalham com, além do valor do próprio *pixel*, o valor dos *pixels* pertencentes à vizinhança desse *pixel*, sendo classificados como operações sobre vizinhança de *pixels* (GONZALES e WOODS, 2010), tendo que realizar mais operações que o algoritmo de binarização, pois devem calcular a média e a mediana de um agrupamento de *pixels* para todos os *pixels* da imagem.

Nos testes realizados foram simulados tamanhos diferentes de máscaras, que indicam a quantidade de *pixels* da imagem que serão processados ao analisar cada *pixel*. Na filtragem da média, para cada *pixel* da imagem, calcula-se a média de seus k^2 vizinhos, para uma máscara de tamanho k x k, atribuindo o valor da média calculada na mesma posição da imagem resultante dessa filtragem. Já para a filtragem da mediana, calcula-se, para cada *pixel* da imagem, a mediana entre os k^2 vizinhos desse *pixel*, para uma máscara de tamanho k x k, e o valor mediano calculado é o valor armazenado na imagem resultante referente ao *pixel* analisado. Neste trabalho, utilizamos o algoritmo de ordenação *Quicksort* (CORMEN *et al.,* 2017) para ordenar os valores da vizinhança de um *pixel* e possibilitar encontrar o valor mediano na operação de filtragem da mediana. Na Figura 1 é exibido um exemplo ilustrando como são realizadas a filtragem da média e a filtragem da mediana, sendo que: em (a), destaca-se o *pixel* central da imagem (em vermelho) e seus vizinhos (em azul) ao utilizar uma máscara de tamanho 2x2; em (b), exibe-se o resultado da filtragem da média para o *pixel* central de (a) utilizando uma máscara 2x2; em (c), exibe-se o resultado da filtragem da mediana para o *pixel* central de (a) utilizando uma máscara 2x2.

Figura 1 – Exemplo de cálculo de filtragem da média e mediana

1	2	4	6	8
1	3	9	27	81
1	4	16	64	99
1	1	3	5	8
1	1	2	2	4

(a) Imagem de entrada exibindo os vizinhos do *pixel* central utilizando uma máscara 2x2.

		22		

(b) Valor calculado na filtragem da média para o *pixel* central de (a) utilizando uma máscara 2x2.

		11		

(c) Valor calculado na filtragem da mediana para o *pixel* central de (a) utilizando uma máscara 2x2.

Fonte: Elaboração própria (2020).

Na Figura 2 exibimos um pseudocódigo simplificado do algoritmo de binarização implementado. Na figura é possível perceber que esse pseudocódigo possui dois laços aninhados, ambos sendo executados para todos os valores inteiros presentes no intervalo [1, n], sendo que $n$representa o número de linhas e colunas da imagem processada. É possível dizermos que, nesse algoritmo, o código presente na linha 3 será executado n^2 vezes, e a soma da quantidade de vezes que os códigos presentes nas linhas 4 e 6 serão executados também será n^2, pois dependem do resultado da avaliação da expressão lógica presente na linha 3.

Figura 2 – Pseudocódigo do algoritmo de binarização

```
1 para i de 1 aténpasso 1 faça
2         para j de 1 aténpasso 1 faça
3                 se imagem[i,j] < limiar então
4                         novaimagem[i,j] ← 0;
5                 senão
6                         novaimagem[i,j] ← 255;
7                 fimse
8         fimpara
9 fimpara
```

Fonte: Elaboração própria (2020).

Na Figura 3 exibimos um pseudocódigo simplificado do algoritmo de filtragem da média para uma máscara com quantidade de linhas e colunas par, sem levar em consideração o tipo de convolução realizada nos *pixels* das bordas da imagem de entrada (GONZALES e WOODS, 2010). Nessa figura, *n* representa o número de linhas e colunas da imagem de entrada e *k* representa a quantidade de linhas e colunas da máscara utilizada, preenchida em todas as suas posições com o valor , de forma a calcular a média da região sobre a qual a máscara é posicionada. É possível visualizar por essa figura que a operação presente na linha 5 será executada um total de k^2 vezes, dependendo do tamanho da máscara, pra cada *pixel* presente na imagem. Uma vez que a imagem possui n^2 *pixels*, conforme pode ser visto pelo código dos laços presentes nas linhas 1 e 2, podemos dizer então que a operação presente na linha 5 será executada um total de n^2k^2 vezes, ao processar uma imagem com *n* linhas e *n* colunas e ao utilizar uma máscara com *k* linhas e *k* colunas, sendo *k* um número inteiro par.

Figura 3 – Pseudocódigo do algoritmo de filtragem da média para uma máscara par

```
1 para i de 1 aténpasso 1 faça
2         para j de 1 aténpasso 1 faça
3                 para x de 1 até k passo 1 faça
4                         para y de 1 até k passo 1 faça
5                                 novaimagem[i,j] ← novaimagem[i,j] + imagem[i+x,j+y] * mascara[x,y];
6                         fimpara
7                 fimpara                                                                    b
8         fimpara
9 fimpara
```

Fonte: Elaboração própria (2020).

Na Figura 4 exibimos um pseudocódigo simplificado do algoritmo de filtragem da mediana para uma máscara com quantidade par de linhas e colunas, sem levar em consideração o tipo de convolução realizada nos *pixels* das bordas da imagem de entrada (GONZALES e WOODS, 2010). Nessa figura, *n* representa o número de linhas e colunas da imagem de entrada e *k* representa a quantidade de linhas e colunas da máscara utilizada. Por essa figura é possível visualizar que os códigos presentes nas linhas 6 e 7 serão executados um total de k^2 vezes para cada *pixel* da imagem de entrada, dependendo do tamanho da máscara utilizada. Como a imagem de entrada possui n^2 *pixels*, podemos dizer que as operações presentes nas linhas 6 e 7 serão executadas um total de n^2k^2 vezes durante todo o processamento da imagem de entrada.

Já as operações presentes nas linhas 10 e 11 serão executadas um total de n^2 vezes, uma vez para cada *pixel* da imagem de entrada.

Para ordenar o vetor criado com o valor dos vizinhos de cada *pixel* da imagem, de forma a encontrar o valor mediano presente nesse vetor, utilizamos o algoritmo *Quicksort*, representado na linha 10, que possui complexidade no caso médio de O(m log m), em que m representa a quantidade de elementos presentes no vetor a serem ordenados pelo algoritmo (CORMEN *et al.,* 2017). Como o vetor que queremos ordenar possui a mesma quantidade de elementos que a máscara possui, ou seja, k^2 elementos, podemos dizer que o algoritmo *Quicksort* apresenta, para esse caso, complexidade $k^2\log(k^2)$. A notação Big-O nos dá uma representação da complexidade de um algoritmo, e não o total de operações realizadas por ele. Porém, apenas para efeito de comparação, vamos simplificar e considerar que, no caso da filtragem da mediana, a linha de código mais executada do algoritmo *Quicksort*é executada um total aproximado de $k^2\log(k^2)$ vezes para cada *pixel* presente na imagem de entrada. Assim, se a imagem de entrada possui n^2 *pixels*, podemos dizer que o algoritmo da filtragem da mediana, utilizando máscara par, possuirá, para efeito de comparação com os demais algoritmos implementados, aproximadamente $n^2(k^2\log(k^2))$ operações executadas em sua linha de código que apresenta maior quantidade de execução.

Figura 4 – Pseudocódigo do algoritmo de filtragem da mediana para máscara par

```
1  para i de 1 aténpasso 1 faça
2          para j de 1 aténpasso 1 faça
3                  posição ← 1;
4                  para x de 1 até k passo 1 faça
5                          para y de 1 até k passo 1 faça
6                                  vetor[posição] ← imagem[i+x,j+y];
7                                  posição ← posição + 1;
8                          fimpara
9                  fimpara
10                         quicksort(vetor);
11                         novaimagem[i,j] ← (vetor[(posição-1)/2] + vetor[(posição+1)/2])/2;
12                 fimpara
13         fimpara
```

Fonte: Elaboração própria (2020).

A Tabela 1 apresenta um resumo da quantidade aproximada de operações realizadas pela linha de código mais executada dos algoritmos desenvolvidos. A partir dessa tabela podemos perceber que, apesar de todos serem classificados como algoritmos quadráticos, o algoritmo da filtragem da mediana é o mais complexo, seguido do algoritmo da filtragem da média e do algoritmo de binarização, o de menor custo computacional. Isso se deve ao processamento realizado para cada *pixel*. Enquanto na operação de binarização apenas o valor do *pixel* é verificado, nas operações da filtragem da média e da filtragem da mediana, a vizinhança do *pixel* deve ser analisada, o que faz com que o número de operações aumente e, consequentemente, aumente o tempo decorrido para o seu processamento. É possível perceber também que, conforme o tamanho da máscara aumenta, mais operações o algoritmo deverá realizar, por isso esperamos que o algoritmo de serialização seja mais vantajoso conforme o tamanho da imagem e da máscara utilizada aumentem.

Tabela 1 – Quantidade aproximada de repetições da linha de código mais executada por algoritmo desenvolvido

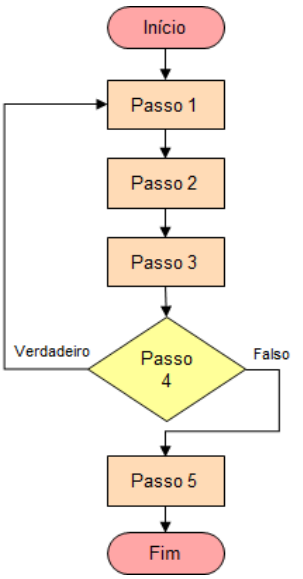
Algoritmo	Quantidade aproximada de repetições da linha de código mais executada
Binarização	N^2
Filtragem da média para máscara par	N^2K^2
Filtragem da mediana para máscara par	$N^2K^2\log(K^2)$

Fonte: Elaboração própria (2020).

Na Figura 5 ilustramos as etapas aplicadas pelos algoritmos propostos em E2, conforme descrito:

- Passo 1: inicializa a matriz da imagem e a matriz da máscara (no caso da filtragem da média e da mediana);
- Passo 2: realiza o processo de binarização, média ou mediana de acordo com o parâmetro especificado na execução;
- Passo 3: serializa a matriz resultante;
- Passo 4: verifica se ainda existem matrizes para processar;
- Passo 5: apresenta o resultado;

Figura 5 –Diagrama estrutural dos algoritmos da etapa E2



Fonte: Elaboração própria (2020).

Para cada matriz de entrada em E2 são apresentados, no passo 5, as dimensões da matriz processada, o tempo de processamento e o tempo de escrita da imagem, respectivamente, para que seja possível comparar o tempo de processamento do algoritmo sem serialização e o tempo de execução com serialização.

A biblioteca *time.h* da linguagem C foi empregada nos algoritmos para computar o tempo de execução dos processamentos realizados e comparar com os tempos de leitura e escrita. Por meio da função *clock* é possível capturar o tempo, em milissegundos, do sistema operacional para capturar o instante T1 de início da rotina e o instante T2 ao final da rotina, de forma a saber o seu tempo total de execução (FEOFILOFF, 2009).

A serialização dos vetores de dados em linguagem C foi realizada por meio das funções *fwrite* e *fread* da biblioteca *stdio.h*. Apesar de existirem bibliotecas específicas para esse fim, como (ÅSTRÖM, 2013) e

(LÖFGREN, 2014), visam encapsular a rotina para simplificar o processo, por isso optou-se por desenvolver uma rotina própria de serialização, buscando maior domínio do processo.

Foram realizadas duas baterias de testes (B1 e B2): a primeira, utilizando um microcomputador com processador Intel(R) Core(TM) i7-4800MQ, 2.70GHz, 32GB de memória RAM e disco rígido SSD de 1TB; e a segunda, em um microcomputador com processador Intel Core i5 4590, 3.30GHz, 8GB de memória RAM e disco rígido HDD de 1TB, em ambiente *Linux*.

Um algoritmo em *shellscript* (JARGAS, 2008; SHOTTS, 2012) foi criado para compilar os algoritmos e automatizar os testes do presente trabalho, a fim de executar consecutivamente os algoritmos das etapas E0, E1 e E2, utilizando as matrizes definidas no conjunto de testes, no caso de E1 e E2, buscando garantir a mesma ordem de execução e as mesmas condições para B1 e B2.

3 Resultados e discussão

Na Tabela 2 apresentamos um teste inicial de análise de tempo de processamento comparado com o tempo de leitura dos dados serializados em disco. Nessa etapa, chamada E0, temos três algoritmos com códigos que implementam rotinas aninhadas de multiplicação de matrizes. O algoritmo A1 contém uma rotina de complexidade O(n), o algoritmo A2 contém uma rotina de complexidade O(n^2) e o algoritmo A3 contém uma rotina de complexidade exponencial, ou seja, O(2^n).

Analisando a Tabela 2, podemos constatar que, conforme a complexidade do algoritmo aumenta, também aumenta o tempo necessário para processamento. Também é possível concluir que não é viável implementação de serialização para algoritmos de complexidade O(n), pois o tempo de leitura dos dados em disco é maior que o tempo de processamento.

Tabela 2 – Etapa de testes E0: simulação de complexidade

Algoritmo	*N*	Complexidade	Proc. B1 (segundos)	Proc. B2 (segundos)	Leitura B1 (segundos)	Leitura B2 (segundos)
A1	5000	O(n)	0,00	0,00	0,03	0,05
A2	5000	O(n^2)	0,06	0,05	0,04	0,05
A3	5000	O(n^3)	208,43	251,95	0,05	0,08

Fonte: Elaboração própria (2020).

Na Tabela 3 apresentamos os resultados dos testes realizados em laboratório com os algoritmos propostos para a etapa E2. Na primeira coluna temos o nome do algoritmo; na segunda coluna numeramos os testes por ordem decrescente de custo de processamento, em segundos, com intuito de ilustrar os dados contidos no gráfico da Figura 6; na terceira coluna temos o tamanho da máscara utilizada no processo de filtragem da mediana e de filtragem da média; na quarta temos a resolução espacial da imagem empregada no teste; na quinta coluna temos o resultado do tempo de processamento, em segundos, durante a bateria de testes B1, com o referido algoritmo; na sexta temos o tempo de processamento, em segundos, na bateria de testes B2; na penúltima coluna temos o tempo, em segundos, de leitura dos dados serializados em B1; e na última coluna temos o tempo, em segundos, de leitura do referido teste na bateria de testes B2. Todos os resultados apresentados na Tabela 3 foram obtidos a partir da execução de um *shell script*, que compila o código-fonte para o sistema alvo e executa os algoritmos para as entradas informadas.

Tabela 3 – Resultados dos testes em B1 e B2

Operação	Teste	Máscara	Imagem	Proc. B1 (segundos)	Proc. B2 (segundos)	Leitura B1 (segundos)	Leitura B2 (segundos)
Mediana	T1	8x8	8192x8192	365,75	368,98	0,41	0,42
Mediana	T2	6x6	8192x8192	136,87	135,56	0,41	0,42
Mediana	T3	8x8	4096x4096	91,48	91,83	0,09	0,06
Mediana	T4	6x6	4096x4096	35,44	33,62	0,09	0,06
Mediana	T5	4x4	8192x8192	31,55	31,71	0,41	0,42
Mediana	T6	4x4	8192x8192	31,55	31,71	0,41	0,42
Média	T7	8x8	8192x8192	24,53	25,09	0,41	0,42
Média	T8	8x8	8192x8192	24,53	25,09	0,41	0,42
Mediana	T9	8x8	2048x2048	22,96	22,64	0,02	0,03
Média	T10	6x6	8192x8192	14,08	14,27	0,41	0,42
Média	T11	6x6	8192x8192	14,08	14,27	0,41	0,42
Mediana	T12	6x6	2048x2048	8,48	8,38	0,02	0,03
Mediana	T13	4x4	4096x4096	8,16	7,89	0,09	0,06
Média	T14	4x4	8192x8192	6,26	6,32	0,41	0,42
Média	T15	4x4	8192x8192	6,26	6,32	0,41	0,42
Média	T16	8x8	4096x4096	6,01	6,07	0,09	0,06
Mediana	T17	2x2	8192x8192	4,88	5,03	0,41	0,42
Média	T18	6x6	4096x4096	3,42	3,45	0,09	0,06
Mediana	T19	4x4	2048x2048	1,93	1,97	0,02	0,03
Média	T20	2x2	8192x8192	1,72	1,76	0,41	0,42
Média	T21	4x4	4096x4096	1,54	1,57	0,09	0,06
Média	T22	8x8	2048x2048	1,49	1,51	0,02	0,03
Mediana	T23	2x2	4096x4096	1,19	1,21	0,09	0,06
Média	T24	6x6	2048x2048	0,85	0,86	0,02	0,03
Média	T25	2x2	4096x4096	0,43	0,44	0,09	0,06
Média	T26	4x4	2048x2048	0,38	0,39	0,02	0,03
Mediana	T27	2x2	2048x2048	0,3	0,3	0,02	0,03
Binarização	T28	Não	8192x8192	0,17	0,17	0,41	0,42
Média	T29	2x2	2048x2048	0,11	0,11	0,02	0,03
Binarização	T30	Não	4096x4096	0,04	0,04	0,09	0,06
Binarização	T31	Não	2048x2048	0,01	0,01	0,02	0,03

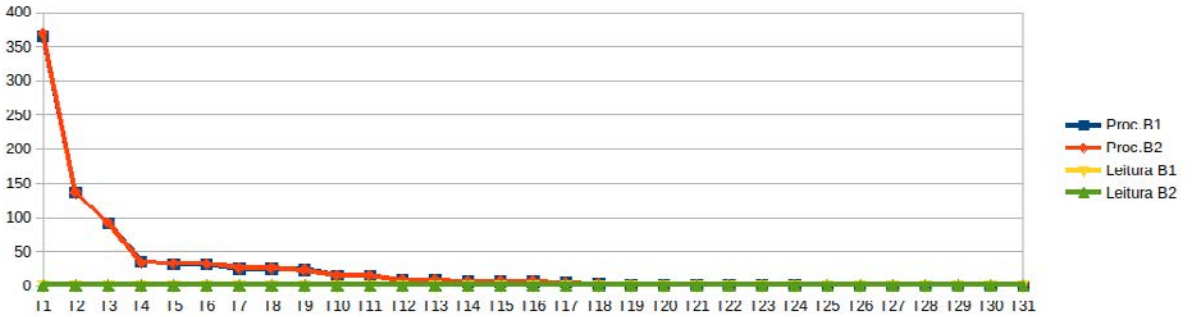
Fonte: Elaboração própria (2020).

Conforme podemos verificar na Tabela 3, há uma considerável redução de tempo de resposta no algoritmo quando se aplica a serialização, pois, como vimos, ao invés de refazer o processamento, recuperam-se os dados resultantes que estão devidamente armazenados em disco. Uma exceção está no processo de binarização de imagens, conforme podemos verificar na Tabela 3, em T28, T30 e T31, pois não há melhora do desempenho do algoritmo com a serialização. Esse fenômeno ocorre por conta do tempo de leitura em disco ser maior que o tempo de processamento devido à menor quantidade de operações que esse algoritmo realiza em comparação com os demais, conforme descrito na Tabela 1.

Com intuito de condensar a Tabela 3, não são apresentados os resultados em que o tempo de processamento ficou abaixo de 0,01 segundos, casos em que a serialização não se mostra necessária, pois o tempo de processamento é equivalente ao tempo de leitura dos dados.

Na Figura 6 temos um gráfico comparativo entre os tempos de resposta das baterias de testes B1 e B2, em segundos. Podemos verificar que não houve relevante melhoria do desempenho comparando-se os equipamentos que têm famílias de processadores, tipos de disco rígido e montante de memória RAM diferentes.

Figura 6 – Gráfico dos resultados dos testes em B1 e B2 (em segundos)



Fonte: Elaboração própria (2020).

É possível destacar, ainda, que, no caso do teste T1, o tempo de resposta 892 vezes menor de leitura em relação ao processamento comprova a importância do uso de serialização para processos em que o arquivo, demandando considerável recurso computacional, precisa ser repetidamente processado e, consequentemente, carregado na memória. A Tabela 4 apresenta um comparativo de tempo para uma situação hipotética de um conjunto de testes utilizando processamento de múltiplos arquivos, considerando o tempo de T1. Na tabela, para a situação com a serialização, consideramos que o processamento foi realizado a fim de armazenar as informações no arquivo serializado ao menos uma vez.

Também é importante destacar que, conforme vemos na Figura 6, para binarização e para filtragens de imagens pequenas, o tempo de processamento é equivalente ou menor ao tempo de leitura dos dados serializados em disco, inviabilizando ou tornando-se desnecessária a inclusão da rotina de serialização.

Tabela 4 – Comparativo de custo de tempo para processamento de T1 com múltiplos arquivos

Quantidade de arquivos	Custo sem serialização	Custo com serialização	Diferença
1	≅ 6,10 min	≅ 6,10 min	Nenhuma
10	≅ 1,02 horas	≅ 6,16 min	> 55 min
100	≅ 10,16 horas	≅ 6,78 min	> 10 horas
1000	≅ 4,23 dias	≅ 12,93 min	> 4 dias

Fonte: Elaboração própria (2020).

Conforme podemos visualizar na Tabela 4, a utilização da serialização no processamento de um conjunto de mil arquivos de imagens, considerando o teste T1, pode economizar mais de quatro dias de processamento, além dos custos com energia elétrica com alimentação e refrigeração dos equipamentos.

4 Conclusão

Após discorrer uma breve introdução, apresentamos um estudo de caso, no qual algoritmos de alta complexidade que empregam serialização são submetidos para análise de desempenho de tempo de processamento em laboratório. Este estudo objetiva auxiliar pesquisadores no desenvolvimento de algoritmos mais rápidos para processamento de sinais em testes de laboratório utilizando a técnica da serialização.

Diversos testes foram realizados, com algoritmos desenvolvidos em linguagem C. Na etapa E0 desenvolvemos algoritmos que simulam diferentes complexidades para medir o tempo de processamento e o tempo de leitura dos dados serializados. Na etapa E1, foram realizados testes de leitura em disco, no qual matrizes de dados que simulam imagens foram gravadas em disco e recuperadas a fim de medir o seu tempo de leitura antes mesmo de qualquer tipo de processamento ou filtragem. Na última etapa, chamada E2, realizamos os testes com os algoritmos de binarização, filtragem da média e filtragem da mediana para medir o tempo de processamento. Nessa última etapa, as matrizes de dados foram serializadas e o tempo de leitura

contabilizado e apresentado. Todos os algoritmos das etapas E0, E1 e E2 foram testados em duas baterias de testes com microcomputadores diferentes, chamadas B1 e B2, em que os resultados são apresentados, comprovando a vantagem da implementação da serialização.

Os testes demonstraram que, para filtragem de imagens, a serialização se mostra importante aliada na busca de economia de recursos e de tempo de resposta, entretanto vimos que, no caso da binarização, a serialização não se mostrou vantajosa, mesmo para imagens maiores, pois o tempo de resposta do processamento é menor que o tempo de leitura em disco.

Comparando-se os resultados de B1 e B2, vemos que não há considerável diferença nos resultados obtidos, mesmo com a significativa diferença de capacidade de processamento e de memória RAM, mostrando que a aplicação da serialização se mostra mais atrativa, pois permite aumentar o desempenho do sistema sem investimento adicional no equipamento.

Apesar de não terem sido utilizadas transformadas, como *wavelet* e de Fourier,nesse momento, foi possível alcançar o nível desejado de complexidade nos testes com a implementação de rotinas aninhadas de multiplicação de matrizes, simulando operações com imagens digitais, como a filtragem da média e da mediana, além da binarização, a fim de verificar a viabilidade da solução.

É importante salientar que a viabilidade no uso da técnica da serialização não é exclusiva dos algoritmos de processamento de sinais, mas de todo algoritmo que programe rotinas de processamento que são comumente retroalimentados, como leituras de dados em serviços *web* e buscas de dados remotas, mostrando-se uma importante aliada para o pesquisador ou para o desenvolvedor de *software*, visto que os dados previamente processados são armazenados em disco e ficam disponíveis para futuras consultas, que substituem o reprocessamento, que poderia demandar esforço computacional e desperdício de energia repetidas vezes durante o processo.

Para trabalhos futuros, almejamos testar os algoritmos aplicando transformadas *wavelet* discretas e de Fourier em arquivos de áudio e imagens digitais. Também é desejado aprofundar os estudos de análises técnicas que forneçam melhorias nos processos em atividades envolvendo algoritmos de processamentos de sinais.

Referências

ADDISON, Paul S. **The illustrated wavelet transform handbook:** introductory theory and applications in science, engineering, medicine and finance. Boca Raton: CRC press, 2017.

ÅSTRÖM, Ulf. **C serialization library**. [*S.l.:s. n*], 2013. Disponível em: http://www.happyponyland.net/cserialization/readme.html. Acesso em: 3 jul. 2020.

CORMEN, Thomas; LEISERSON, Charles; RIVEST, Ronald. **Algoritmos**. [*S. l.*]: Elsevier Brasil, 2017.

DALCÍN, Lisandro *et al*. MPI for python: performance improvements and MPI-2 extensions. **Journal of Parallel and Distributed Computing**, [*S. l.*], v. 68, n. 5, p. 655-662, 2008.

DROZDEK, Adam. **Estrutura de dados e algoritmos em C++**. [*S. l.*]: Pioneira Thomson Learning, 2002.

EMBREE, Paul M.; KIMBLE, Bruce; BARTRAM, James F**.** C language algorithms for digital signal processing.[*S. l.*], **TheJournaloftheAcousticalSocietyofAmerica**, v. 90, n.1, p. 618.

FEOFILOFF, Paulo. **Algoritmos em linguagem C**. [*S. l.*]: Elsevier Brasil, 2009.

FRIGO, Matteo*et al*. Cache-obliviousalgorithms. *In*: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 40., 1999,New York. **Proceedings**[…]. New York: IEEE, 1999. p. 285-297.

GUIDO, Rodrigo Capobianco. Effectively interpreting discrete wavelet transformed signals. **IEEE Signal Processing Magazine**, [S. l.], v. 34, n. 3, p. 89-100, 2017. Lecture notes.

GONZALES, R. C., WOODS, R. E. **Processamento digital de imagens**. 3. ed. São Paulo: Pearson Prentice Hall. 2010. 624 p.

JARGAS, Aurélio Marinho. **Shell script profissional**.São Paulo: Novatec Editora, 2008.

JENSEN, Arne; LA COUR-HARBO, Anders. **Ripples in mathematics**: the discrete wavelet transform. [S. l.]: Springer Science & Business Media, 2001.

LATHI, Bhagwandas Pannalal. **Sinais e sistemas lineares-2**.Porto Alegre: Bookman, 2006.

LIM, Jae S. **Two-dimensional signal and image processing**. Upper Saddle River: Printice Hall, 1990.

LÖFGREN, Anton. **Protobuf-C**. [*S. l.: s.n.*], 2014. Disponível em: http://https://github.com/protobuf-c. Acesso em: 3 jul. 2020.

MAEDA, Kazuaki. Performance evaluation of object serialization libraries in XML, JSON and binary formats. In: INTERNATIONAL CONFERENCE ON DIGITAL INFORMATION AND COMMUNICATION TECHNOLOGY AND IT'S APPLICATIONS,2.,2012, Bangkok, **Proceedings**[…].Bangkok: IEEE, 2012. p. 177-182.

MIREKU, Kwabena A. **Serialization and preservation of objects**. Depositante: Kwabena Mireku. US 2005O108627A1. Depósito: 19 maio 2005. Concessão: 17 abr. 2007.

MORIMOTO, Carlos Eduardo. **Hardware**:manual completo. [*S. l.*]: GDH Press, 2002.

PHILIPPSEN, Michael; HAUMACHER, Bernhard. More efficient object serialization. *In*: INTERNATIONAL PARALLEL PROCESSING SYMPOSIUM, 13.,1999, Proceedings […]. Berlin:ACM,1999. p. 718-732.

SATO, Liria Matsumoto; MIDORIKAWA, Edson Toshimi; SENGER, Hermes. Introdução a programação paralela e distribuída. JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA,15., 1996, Recife,**Anais** [...]. Recife: EDITORA, 1996. p. 1-56, 1996.

SHOTTS JR, William E. **The Linux command line:** a complete introduction. San Francisco: No Starch Press, 2012.

SMITH, Alan Jay. Cache memories. **ACM Computing Surveys (CSUR)**, [*S. l.*], v. 14, n. 3, p. 473-530, 1982.

SUNG, Wonyong; KANG, Jiyang. Fixed-point C language for digital signal processing. *In*: CONFERENCE RECORD OF THE TWENTY-NINTH ASILOMAR CONFERENCE ON SIGNALS, SYSTEMS AND COMPUTERS,29., 1995,Pacific Grove. **Proceedings**[…].Pacific Grove: IEEE, 1995. p. 816-820.

TENENBAUM, Aaron M.; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. **Estruturas de dados usando C**. São Paulo: Pearson Makron Books, 2004.

TOMPKINS, Willis J. Biomedical digital signal processing. [*S. l.*]: **Editorial Prentice Hall**, 1993.

TORELLI, Julio Cesar; BRUNO, Odemir Martinez. Programação paralela em SMPs com OpenMP e POSIX Threads: um estudo comparativo. *In*: CONGRESSO BRASILEIRO DE COMPUTAÇAO, 4., 2004, Itajaí. **Anais** [...].Itajaí: SBPC, 2004.p. 486-491.

VASCONCELOS, Laércio. **Hardware na prática**. Rio de Janeiro: LaércioVasconcelos, 2007.

WALKER, James S. **A primer on wavelets and their scientific applications**.Boca Raton: CRC press, 2008.

Sobre os autores

João Paulo Lemos Escola
Doutorando da Escola de Engenharia de São Carlos - Universidade de São Paulo (USP). Mestre em Ciências, com ênfase em Física Aplicada Computacional, pelo Instituto de Física de São Carlos/USP (2014). Graduado em Tecnologia em Informática pela Faculdade de Tecnologia de São José do Rio Preto (2007). É revisor e parecerista da revista Sinergia. Atualmente, é professor efetivo do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (*campus* Barretos) em regime de dedicação exclusiva. Atua, principalmente, nas áreas de processamento de sinais, tecnologias de *software* livre e *open source*, desenvolvimento de *software* PHP, Java e Android.

Tiago Alexandre Dócusse
Doutor em Engenharia Elétrica pela Universidade Estadual Paulista Júlio de Mesquita Filho (2014). Mestre em Ciência da Computação pela Universidade Estadual Paulista Júlio de Mesquita Filho (2008). Bacharel em Ciência da Computação pela Universidade Estadual Paulista Júlio de Mesquita Filho (2005). Atualmente, é professor do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo. Tem experiência na área de Ciência da Computação, com ênfase em processamento de imagens; atuando, principalmente, nos seguintes temas: processamento de imagens, mamografias digitais e *wavelets*.

Recebido em: 13.07.2020
Aceito em: 21.08.2020