Pedro Inácio Rodrigues Pontes

Prática 10: Otimização com Array Pool

Belo Horizonte, Brasil

2025

1 Introdução

O objetivo do presente trabalho foi a otimização de um código que simulava o processamento de imagens e a aplicação de um filtro *blur* nelas. O código possuía 2 classes - Program e ImageProcessor -, além de uma struct PixelRGB. O componente no qual foi objetivada alteração era a classe ImageProcessor.

2 Desenvolvimento

2.1 Versão Não Otimizada

Antes de ser otimizado, o código apresentava uma criação excessiva de arrays, que poderiam simplesmente ser reutilizados usando array pool.

```
public static void ProcessImages()
{
    Console.WriteLine("Iniciando processamento de imagens
       (vers o trivial)...");

    for (int imageIndex = 0; imageIndex < TOTAL_IMAGES;
       imageIndex++)
    {
        PixelRGB[,] originalImage =
           GenerateSyntheticImage(imageIndex);
        PixelRGB[,] blurredImage =
           ApplyBlurFilter(originalImage);
        SaveImage(blurredImage,
           $"processed_{imageIndex}.jpg");
    }
    [...]
}

private static PixelRGB[,] GenerateSyntheticImage(int seed)
{
    var image = new PixelRGB[IMAGE_HEIGHT, IMAGE_WIDTH];
    var random = new Random(seed);

    for (int y = 0; y < IMAGE_HEIGHT; y++)
    {
        for (int x = 0; x < IMAGE_WIDTH; x++)
```

```
        {
            image[y, x] = new PixelRGB(
                (byte)random.Next(256),
                (byte)random.Next(256),
                (byte)random.Next(256)
            );
        }
    }
    return image;
}

private static PixelRGB[,] ApplyBlurFilter(PixelRGB[,]
   original)
{
    int height = original.GetLength(0);
    int width = original.GetLength(1);
    var blurred = new PixelRGB[height, width];

    for (int y = 0; y < height - 1; y++)
    {
        for (int x = 0; x < width - 1; x++)
        {
            blurred[y, x] = PixelRGB.Average(
                original[y, x],
                original[y, x + 1],
                original[y + 1, x],
                original[y + 1, x + 1]
            );
        }
    }
    return blurred;
}
```

2.2 Versão Otimizada

Os arrays reutilizados via ArrayPool foram implementados para GenerateSyntheticImage e ApplyBlurFilter, os quais geravam uma quantidade enorme de arrays temporários, sobrecarregando o GC (garbage collector).

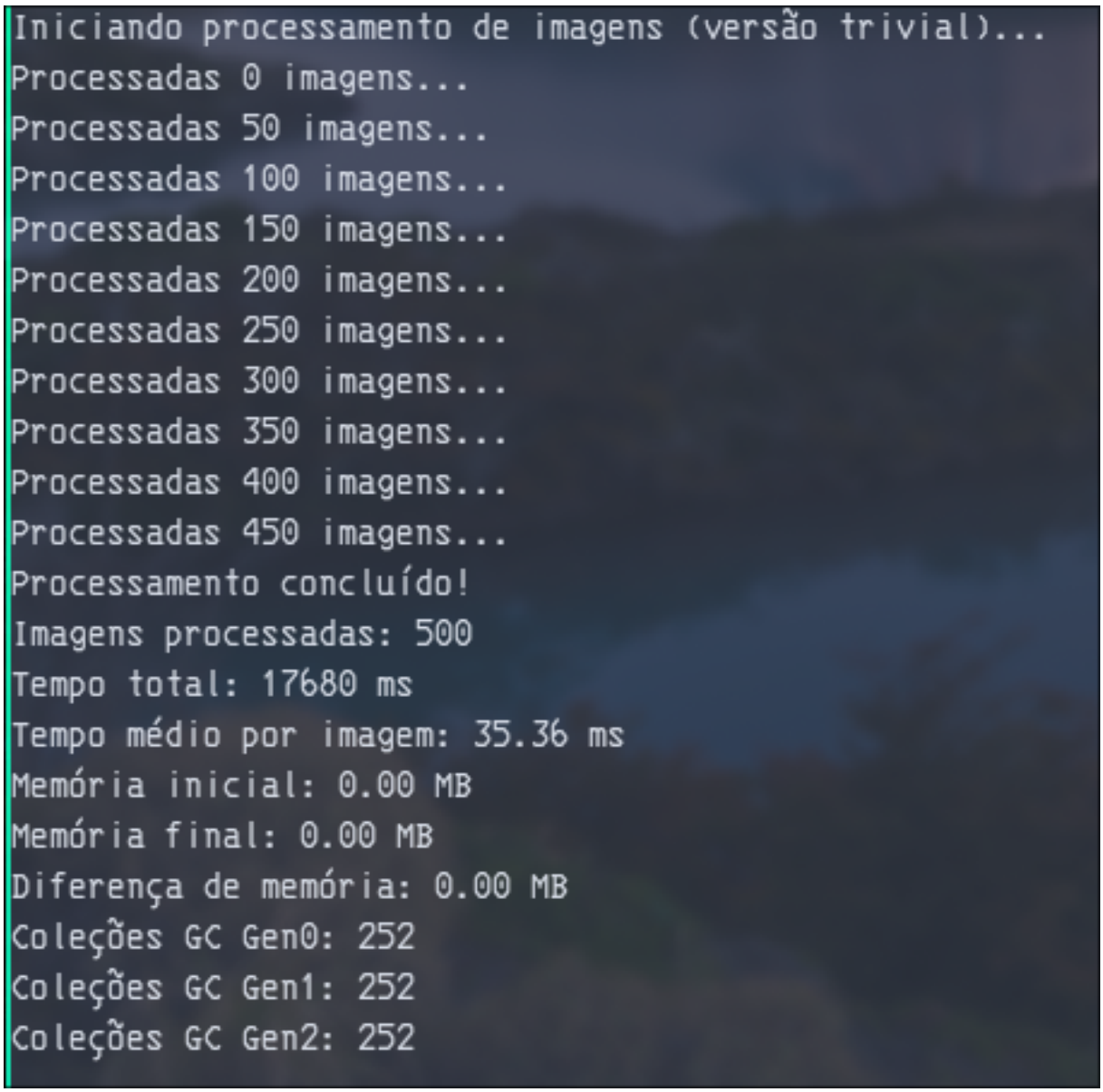
Após a correção e aplicação das devidas alterações (ArrayPool não suporta arrays

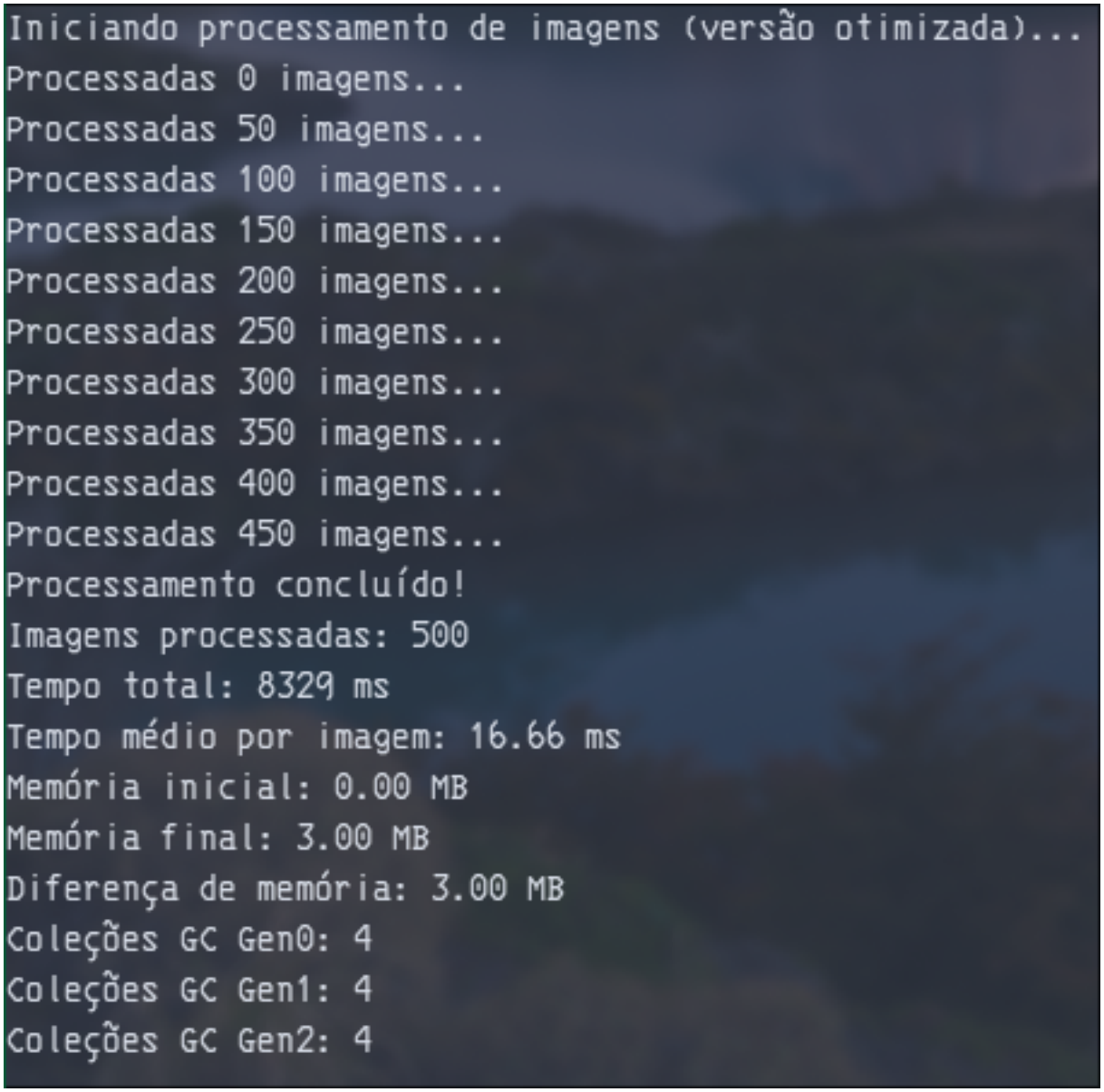
bidimensionais), o código ficou desta forma:

```
public static void ProcessImages()
{
    Console.WriteLine("Iniciando processamento de imagens
       (vers o otimizada)...");

    var pool = ArrayPool<PixelRGB>.Shared;
    PixelRGB[] blurred = pool.Rent(TOTAL_PIXELS);
    PixelRGB[] imageArray = pool.Rent(TOTAL_PIXELS);

    try
    {
        for (int imageIndex = 0; imageIndex < TOTAL_IMAGES;
           imageIndex++)
        {
            GenerateSyntheticImage(imageIndex, imageArray);
            ApplyBlurFilter(imageArray, blurred);
            SaveImage(blurred,
               $"processed_{imageIndex}.jpg");
        }
    }
    finally
    {
        pool.Return(blurred);
        pool.Return(imageArray);
    }
    [...]
}

private static void GenerateSyntheticImage(int seed,
   PixelRGB[] image)
{
    var random = new Random(seed);
    for (int y = 0; y < IMAGE_HEIGHT - 1; y++)
    {
        int rowStart = y * IMAGE_WIDTH;
        for (int x = 0; x < IMAGE_WIDTH - 1; x++)
        {
            image[rowStart + x] = new PixelRGB(
```

```
                (byte)random.Next(256),
                (byte)random.Next(256),
                (byte)random.Next(256)
            );
        }
    }
}

private static void ApplyBlurFilter(PixelRGB[] original,
   PixelRGB[] blurred)
{
    for (int y = 0; y < IMAGE_HEIGHT - 1; y++)
    {
        int rowStart = y * IMAGE_WIDTH;
        int nextRowStart = (y + 1) * IMAGE_WIDTH;
        for (int x = 0; x < IMAGE_WIDTH - 1; x++)
        {
            blurred[y * IMAGE_WIDTH + x] = PixelRGB.Average(
                original[rowStart + x],
                original[rowStart + x + 1],
                original[nextRowStart + x],
                original[nextRowStart + x + 1]
            );
        }
    }
}
```

3 Resultados





4 Conclusão

A otimização foi alcançada. As alterações de memória inicial e final foram insignificantes nas duas situações. O mais relevante foram a diferença do total de coleções realizadas pelo garbage collector, que passaram de 252 para 4, e o tempo de execução, que caiu pela metade.