Computer Architecture (Assignment - 02 (Part-01))

Sreechand R (22562) , Sindhura S (22706)

November 26, 2023

Dilated Convolution

Dilated convolution, also known as *atrous convolution*, is an adaptation of the standard convolution operation in neural networks, particularly useful in computer vision tasks.

Standard Convolution

Standard convolution involves a kernel K sliding over input data (e.g., an image) to perform element-wise multiplication and summing operations. The standard convolution operation for a 2D input can be expressed as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

where I is the input feature map, K is the kernel, and S is the output feature map.

Dilated Convolution

In dilated convolution, the kernel is modified using a dilation factor d. This factor determines the spacing between the elements in the kernel. The dilated convolution operation can be represented as:

$$S(i,j) = (I *_d K)(i,j) = \sum_m \sum_n I(i+dm, j+dn)K(m,n)$$

where $*_d$ denotes the dilated convolution operation and d is the dilation factor.

Advantages

- Increased Receptive Field: Dilated convolution allows the kernel to encompass a larger receptive field without increasing the number of parameters.
- Preservation of Spatial Resolution: It helps in maintaining the spatial resolution of deeper layers in a neural network.
- Efficient Computation: Despite its larger receptive field, dilated convolution does not significantly increase computational complexity.

Applications

Dilated convolutions are widely used in deep learning architectures for tasks such as image segmentation and where detailed spatial understanding is crucial.

1 Problem Statement

In this assignment, our task is to optimize the problem of "Dilated Convolution (DC)" which involves an input matrix and a kernel matrix. The sample algorithm is pictorially depicted below. A sample single-threaded unoptimized implementation is also provided via Github —check the bottom of this document for the same. (The Github repository also has an animation depicting how the algorithm works.)

Input and Kernel Matrix

- Input Matrix I of dimensions $\text{Input\_Row} \times \text{Input\_Column}$.
- Kernel Matrix K of dimensions $\text{Kernel\_Row} \times \text{Kernel\_Column}$.

Output Matrix

The output matrix O will have dimensions $(\text{Input\_Row}-\text{Kernel\_Row}+1)\times(\text{Input\_Column}-\text{Kernel\_Column}+1)$.

Dilated Convolution Operation

The Kernel Matrix slides over the Input Matrix. Element-wise multiplication is performed between the overlapped sections and summed up to produce the Output Matrix.

Example Dimensions

- Input Matrix I: 4×4
- Kernel Matrix K: 2×2
- Output Matrix O: 3×3

Optimization Strategies

Parallel Processing

Implement multi-threading or use parallel processing libraries to speed up computation.

Vectorization

Utilize vectorized operations from libraries like NumPy for faster execution.

Reducing Memory Access

Optimize memory access patterns to leverage CPU cache effectively.

Efficient Data Structures

Use data structures that minimize overhead and improve data locality.

Algorithmic Improvements

Decompose or reorder the algorithm to minimize the number of operations.

GPU Acceleration

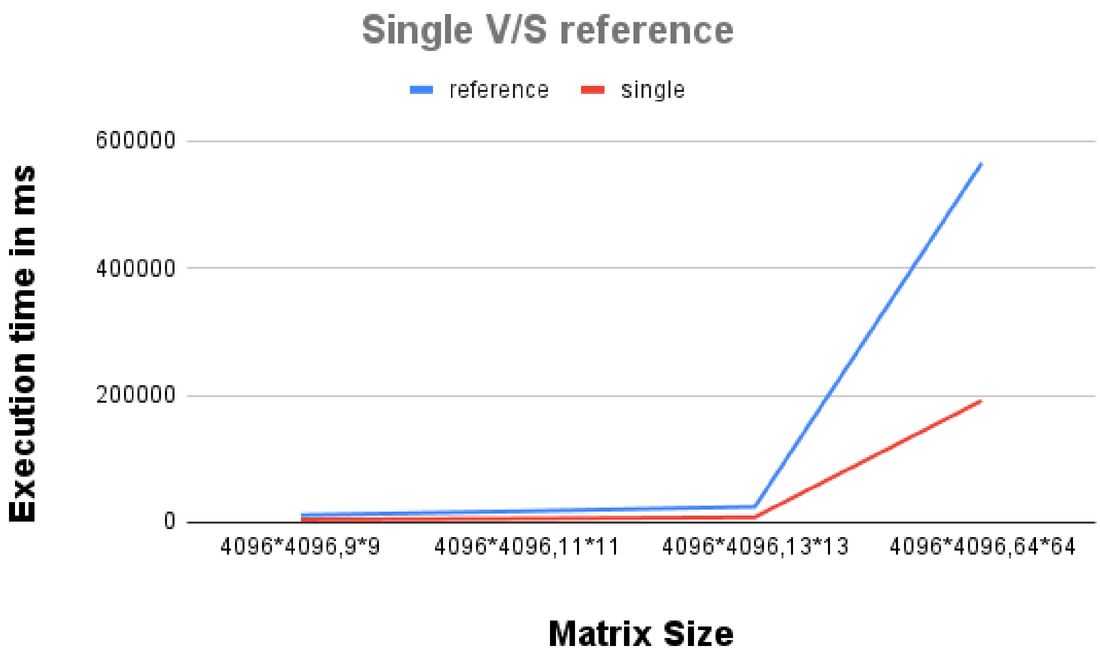
Consider using GPU acceleration for large matrix operations.

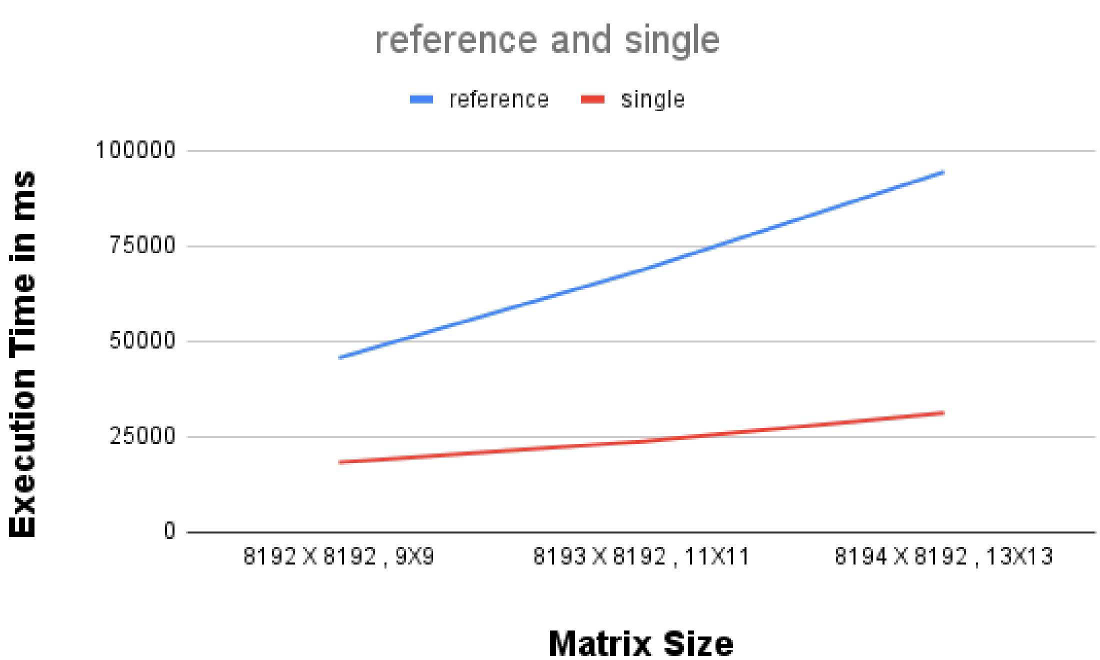
Profiling and Bottleneck Analysis

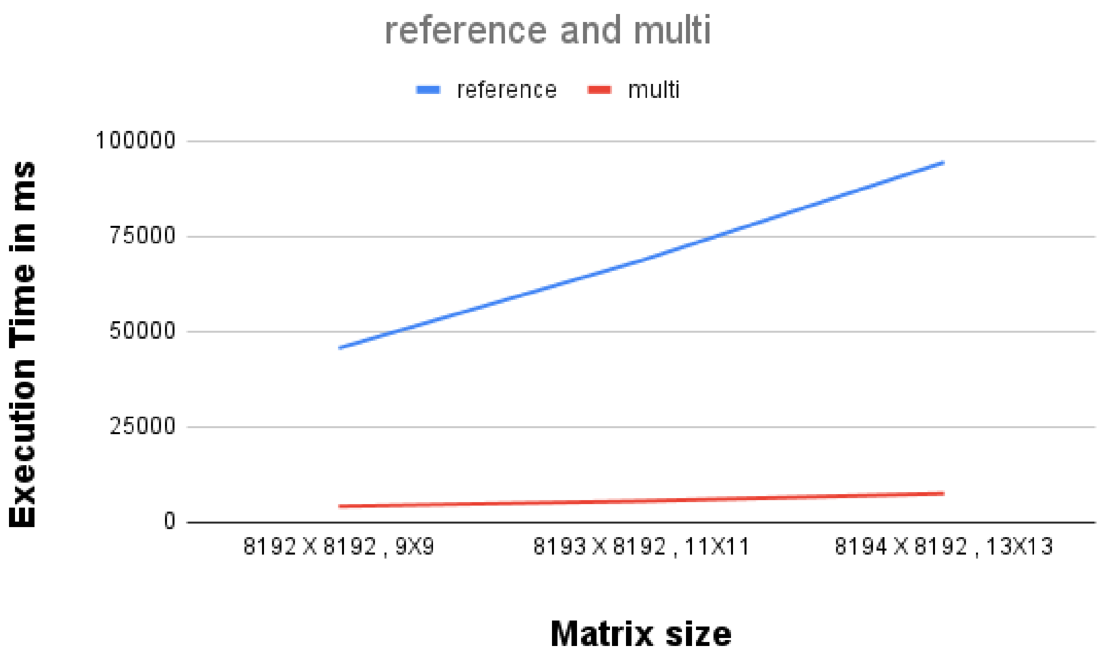
Use profiling tools to identify and optimize bottlenecks in the implementation.

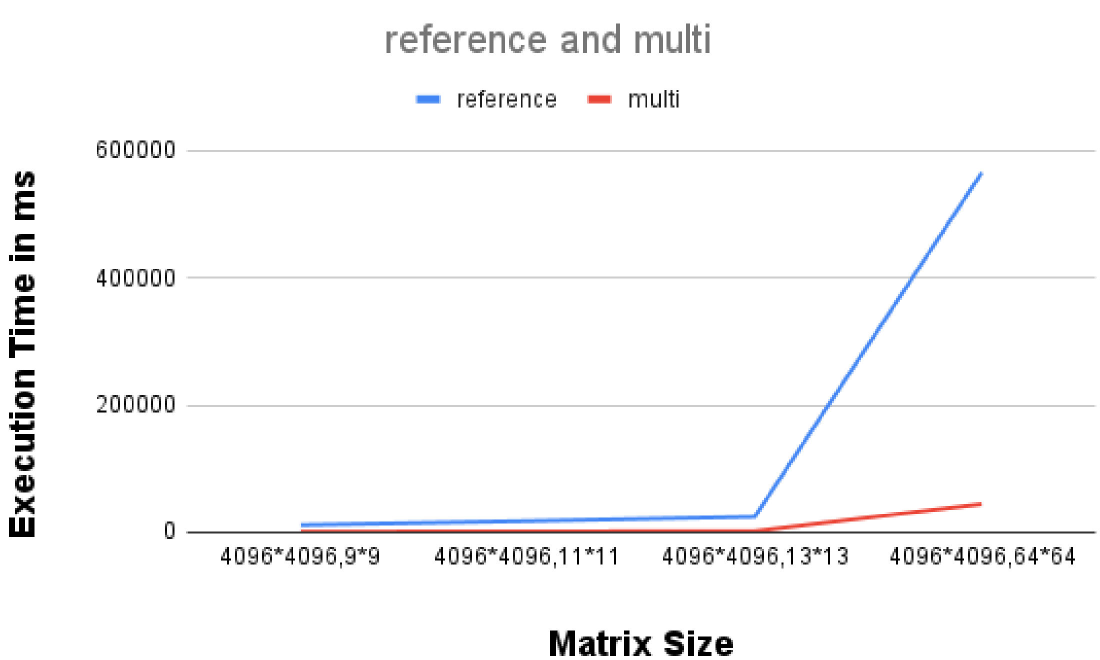
Batch Processing

Perform operations on multiple data points simultaneously if multiple convolutions are needed.


Single V/S reference
reference
single
600000
400000
200000
0
Execution time in ms
4096*4096,9*9
4096*4096,11*11
4096*4096,13*13
4096*4096,64*64
Matrix Size


reference and single
reference
single
100000
75000
50000
25000
0
Execution Time in ms
8192 X 8192 , 9X9
8193 X 8192 , 11X11
8194 X 8192 , 13X13
Matrix Size


reference and multi
reference
multi
Execution Time in ms
100000
75000
50000
25000
0
8192 X 8192 , 9X9
8193 X 8192 , 11X11
8194 X 8192 , 13X13
Matrix size


reference and multi
reference
multi
Execution Time in ms
600000
400000
200000
0
4096*4096,9*9
4096*4096,11*11
4096*4096,13*13
4096*4096,64*64
Matrix Size

Conclusion

- speed up = reference / single thread execution
- The speed up for the following matrix in single thread execution are
  4096*4096, 9*9 = 2.49
  4096*4096, 11*11 = 2.96
  4096*4096, 13*13 = 3.03
  4096*4096, 64*64 = 2.95
  8192*8129, 9*9 = 2.49
  8192*8192, 11*11 = 2.89
  8192*8192, 13*13 = 3.02
- The average speed up = 2.83

- speed up = reference / multi thread execution
- The speed up for the following matrix in multi thread execution are
  4096*4096, 9*9 = 10.89
  4096*4096, 11*11 = 12.54
  4096*4096, 13*13 = 12.80
  4096*4096, 64*64 = 12.85
  8192*8129, 9*9 = 10.96
  8192*8192, 11*11 = 12.29
  8192*8192, 13*13 = 12.85
- The average speed up = 12.16