CMPT365

Project 2 Report

Question 1.

Background

- The created program supports samples with the maximum bit depth of two bytes per sample (for both mono and stereo channel wav files).
- The program assumes that all samples are PCM encoded and represented in a little-endian format.
- The algorithm uses first-order linear prediction.
- To compress and decompress data samples, the java.util.zip.Deflater library was utilized.

1. The results of compression for the given three samples

Note: the original size represents the total size of the file's sampled data (in bits) extracted by the program (not the size of the actual file).

Samples	Original size	Compressed data	Compression rate
mono-sample-65536.wav	1048592	865256	1.211886424364581
stereo-sample-1-48798.wav	1561536	780640	2.0003279360524697
stereo-sample-2-48536.wav	1553152	1304024	1.1910455635785844

The table below compares the actual sizes of the sample wav files with the generated compressed files (in bytes).

Samples	Original size	Compressed data	Compression rate
mono-sample-65536.wav	131,118	108,157	1.21229324038
stereo-sample-1-48798.wav	195,236	97,580	2.00077884812
stereo-sample-2-48536.wav	194,188	163,003	1.19131549726

The obtained sizes of compressed Mid and Side channels for the stereo samples:

stereo-sample-1-48798.wav

- Compressed Mid-Channel size=419720 bits
- Compressed Side-Channel size=360920 bits

stereo-sample-2-48536.wav

- Compressed Mid-Channel size=676160 bits
- Compressed Side-Channel size =627864 bits

2. The table below compares compressed Wav files using FLAC encoding (using AIMP software at the best possible compression setting) with the project's algorithms.

Samples	Compression rate obtained the program	Compression rate using FLAC encoding
mono-sample-65536.wav	1.211886424364581	1.56547255166
stereo-sample-1-48798.wav	2.0003279360524697	6.78505338078
stereo-sample-2-48536.wav	1.1910455635785844	1.71868769259

Results for the generated FLAC files by AIMP:

mono-sample-65536.wav

Input: 128.04 KB

Output: 81.79 KB

Saved: 46.25 KB (36.1 %)

stereo-sample-1-48798.wav

Input: 190.66 KB

Output: 28.10 KB

Saved: 162.56 KB (85.3 %)

stereo-sample-2-48536.wav

Input: 189.64 KB

Output: 110.34 KB

Saved: 79.30 KB (41.8 %)

2. Discussion

The algorithm uses first-order linear prediction (DPCM) to decrease the sample sizes to be encoded and remove redundancy for mono channel files. For the stereo channel files, the algorithm used the mid/side encoding to reduce left/right channel correlation and decrease the size of samples. Then Deflate compression was used to the transformed data samples provided by the java.util.zip.Deflater library. The Deflate compression uses a combination of dictionary coding and Huffman coding. The Deflate encoding is used as a general-purpose compression tool while FLAC is specifically designed and optimized for audio file compression.

mono-sample-65536.wav

The program's compression algorithm performs less effectively for the *mono-sample-65536.wav* file than the FLAC encoding. The possible reason for that might be that the FLAC compression uses higher-order linear predictions. If we look at the data samples of the file, we can observe that there is a strong correlation between sets of samples and less correlation between neighboring samples. Therefore, FLAC was able to reduce the sample sizes further than the project's algorithm.

The first 10 samples for the *mono-sample-65536.wav*:

443, -598, -524, 339, 317, -561, -1381, -1514, -1418, -1773…

stereo-sample-1-48798.wav

Both encodings achieved high compression ratios for the stereo-sample-1-48798.wav sample file. The possible for that is that is that the left/right channels have strongly correlated samples.

The first 10 samples of the stereo-sample-1-48798.wav:

Left channel: -12, -13, -12, -12, -14, -12, -12, -13, -10, -13…

Right channel: 11, 13, 11, 13, 13, 10, 14, 11, 13, 12…

However, FLAC encoding can achieve a compression ratio of 6. The FLAC compression is three times higher than the project's algorithm. The possible reason for that might be that FLAC encoding uses an additional layer of compression – Run Length Encoding (RLE). RLE groups a stream of samples, if they are of the same value, into one sample. If we take a look at the values of the byte streams of Mid and Side channels, we can observe that most of the data can be grouped and encoded as a single sample.

Mid channel byte stream:

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 0, -1, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, -1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0, 0, 0, …]

Side channel byte stream:

[-11, -1, -13, -1, -11, -1, -12, -1, -13, -1, -11, -1, -13, -1, -12, -1, -11, -1, -12, -1, -12, -1, -12, -1, -11, -1, -12, -1, -12, -1, -10, -1, -13, -1, -11, -1, -12, -1, -11, -1, -12, -1, -12, -1, -12, -1, -11, -1, -10, -1, -12, -1, -11, -1, -12, -1, -11, -1, -11, -1, -12, -1, -11, -1…]

stereo-sample-2-48536.wav

The compressions for the *stereo-sample-2-48536.wav* are much less efficient than the compressions on the *stereo-sample-1-48798.wav* file. The possible reason for that might be that the samples in the file are less correlated in the right and left channels.

The first 10 samples of the stereo-sample-2-48536.wav

Left channel samples: -1899, -2125, -957, -471, -179, 340, 580, 662, -48, 28

Right channel samples: -2196, -1960, -1407, -884, -394, -178, 123, -146, -622, -510

However, FLAC encoding was able to compress the file more efficiently than the project's algorithm. The possible reasons might be that FLAC uses RLE, higher-order linear predictions, and optimization.

Possible improvements:

The algorithm introduced round-off errors after decompression on stereo channels. The reason is the use of floating-point arithmetic introduced by the Mid/Side encoding.

Results of decompression on the *stereo-sample-1-48798.wav file:*

The first 20 elements of the original data:

-12 -1 11 0 -13 -1 13 0 -12 -1 11 0 -12 -1 13 0 -14 -1 13 0

The first 20 elements of the uncompressed data (after transforming the uncompressed data to original input):

-11 -1 11 0 -13 -1 13 0 -11 -1 11 0 -12 -1 12 0 -13 -1 13 0

Results of decompression on the *stereo-sample-2-48536.wav* file:

The first 20 elements of the original data:

-107 -8 108 -9 -77 -9 88 -8 67 -4 -127 -6 41 -2 -116 -4 77 -1 118 -2

The first 20 elements of the uncompressed data (after transforming the uncompressed data to original input):

-107 -8 109 -9 -76 -9 88 -8 67 -4 -127 -6 41 -2 -115 -4 77 -1 119 -2

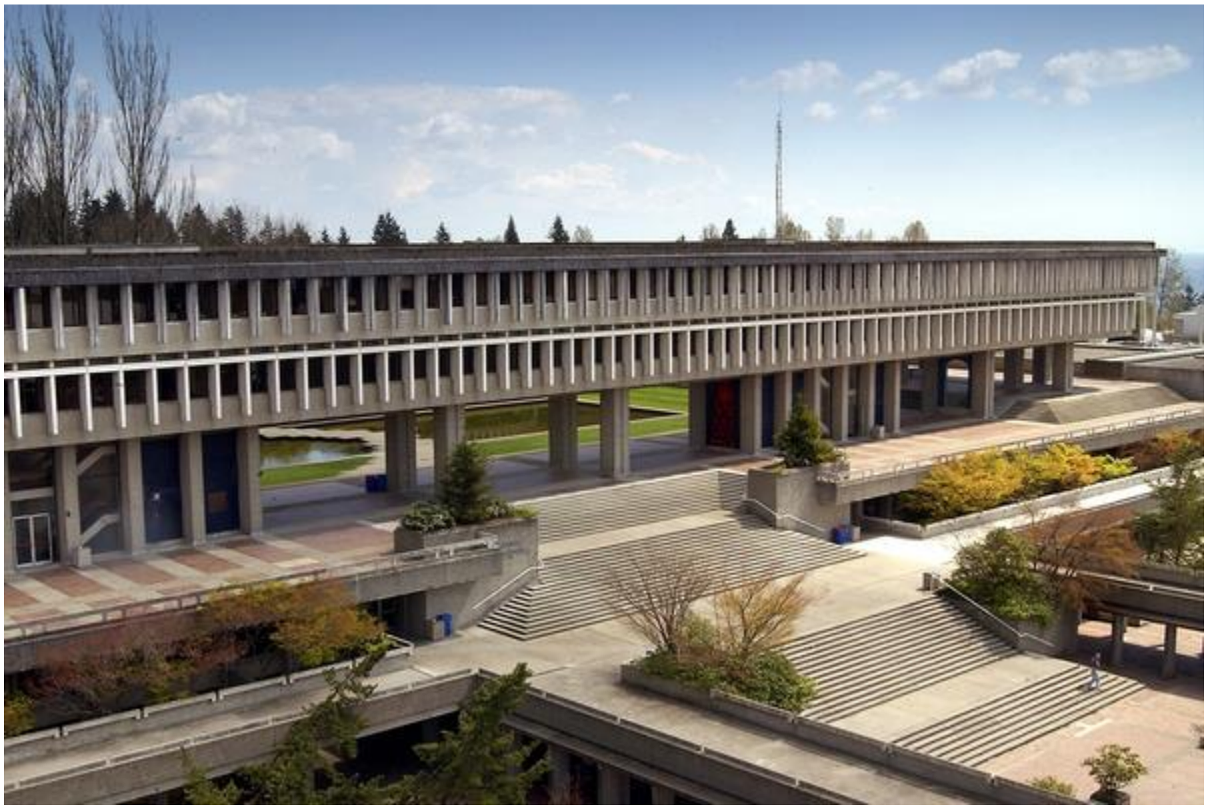
Possible modifications that can improve the program's performance in compression:

- Use higher-order linear predictions
- Use an additional layer of compression (Run Length Encoding)
- Optimization for round-off errors
- Optimization techniques for array manipulation (Loop unrolling)
- Perform less access to the main memory for storing results of buffers.

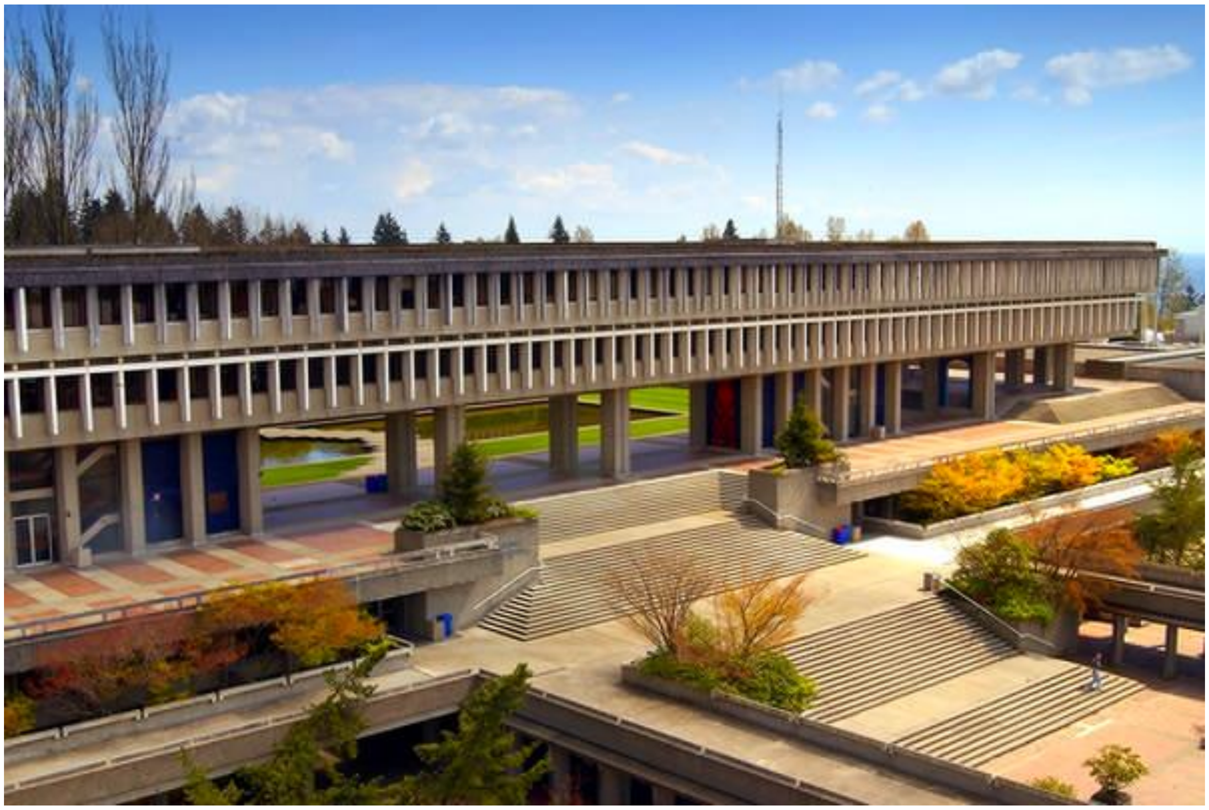
Question 2

1. Display original images and the resultant images.

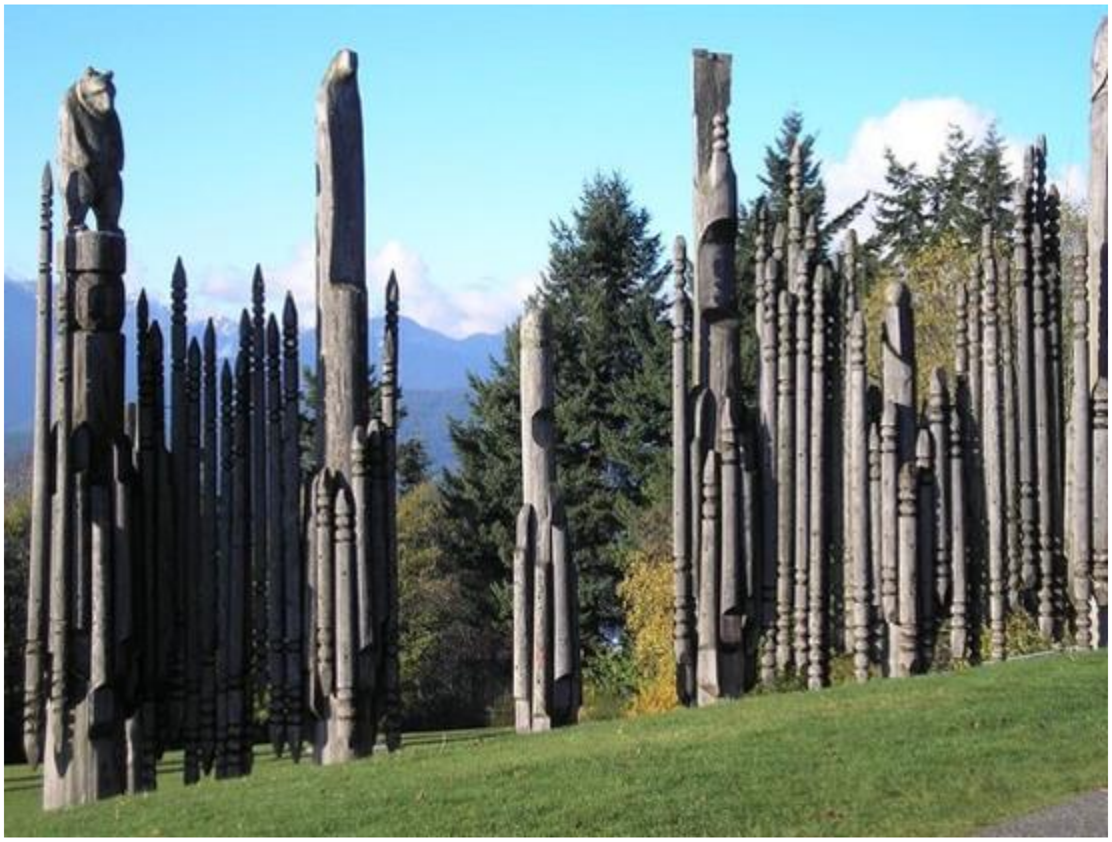
Original image *bmp-sample-3.bmp*



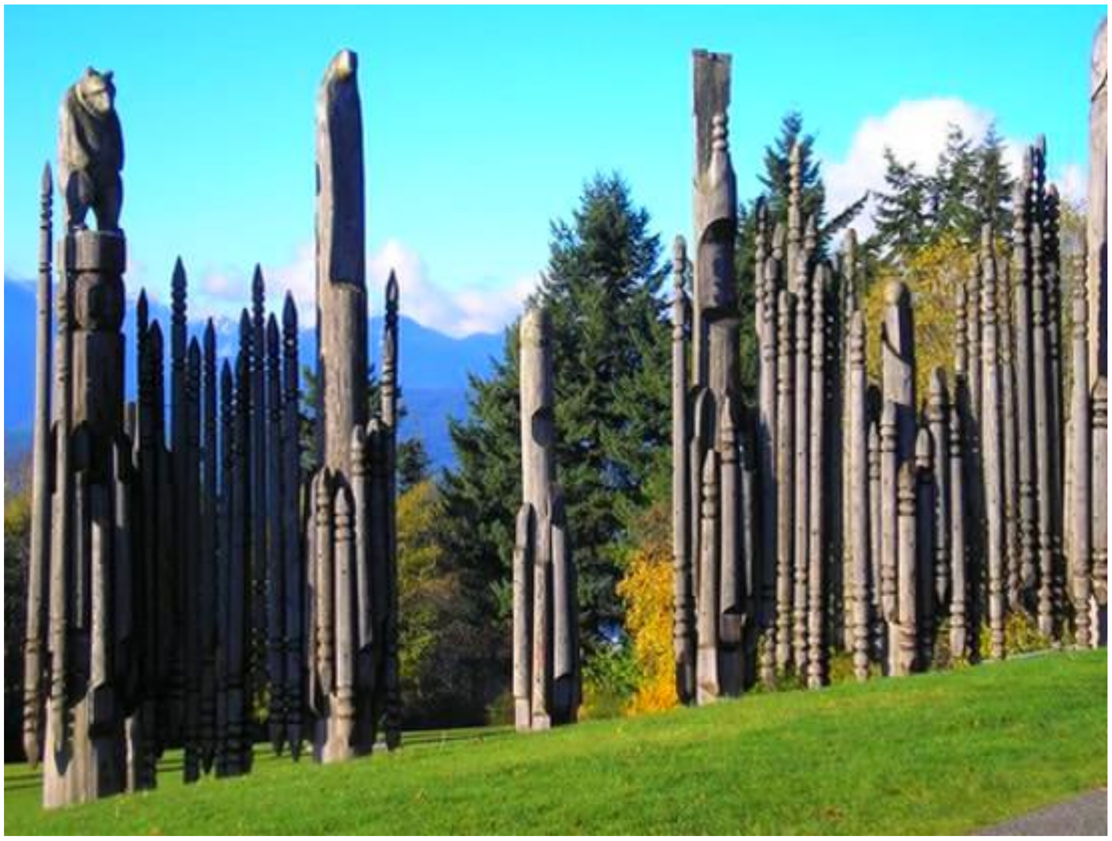
Resultant *bmp-sample-3.bmp*



Original
bmp-sample-2.bmp

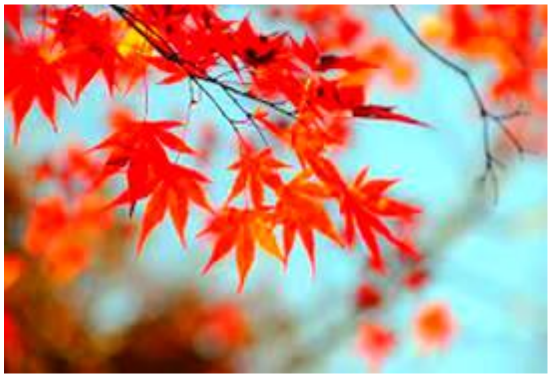


Resultant
bmp-sample-2.bmp



Original image *bmp-sample-1.bmp*

Resultant image *bmp-sample-1.bmp*



2. Quantized results

The first 8x8 block entries of the quantized coefficients matrix of the *bmp-sample-1.bmp* sample

642:1530:-839	642:1530:-839	27:-90:9	-11:6:-2	0:-4:3	0:1:-1	0:0:0	0:0:0
-44:-4:-30	-10:163:-68	7:26:-18	6:0:-4	0:0:0	(0:0:0)	0:0:0	0:0:0
0:-25:18	0:-28:24	0:-27:30	0:2:-1	0:0:0	0:0:0	0:0:0	0:0:0
-4:5:-2	0:4:-3	0:4:-3	0:-1:1	0:0:0	0:0:0	0:0:0	0:0:0
0:-1:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0

The first 8x8 block entries of the quantized coefficients DCT matrix of the *bmp-sample-2.bmp* sample

620:-955:134	-2:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
-7:-12:-1	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
-1:1:-1	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0

The first 8x8 block entries of the quantized coefficients DCT matrix of the *bmp-sample-3.bmp* sample

1068:-514:-41	7:-7:-4	-15:7:0	2:0:0	-2:0:0	0:0:0	0:0:0	0:0:0
-1:-12:1	1:-6:-5	4:0:0	-1:0:0	0:0:0	0:0:0	0:0:0	0:0:0
-1:1:-1	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0
0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0	0:0:0

Note: The tuple (x1:x2:x3) represents quantized results on the YCoCg channel planes (at a particular pixel index). x1 represents quantized result on Y channel, x2 represents quantized result on Co channel, and x3 represents quantized result on Cg channel. The full table for the quantized results can be found in the target directory of the program.

3. Observations

Background:

The algorithm converts raw RGB samples to their corresponding YCoCg values. Then each of the YCoCg planes was transformed into the DCT frequency domain using matrix-vector multiplication. Afterward, the results were quantized by the quantization table provided in the lectures slides. The resultant image is the transform of the quantized results.

Results:

The resultant images visually show a loss in the texture of high-frequency blocks. The recovered images appear to be brighter in the Y channel. The recovered images show less differentiation between neighboring pixel blocks and visually appear to be blurred. That might have resulted from round-off errors and quantization.

Running speeds for applying compression and transforming back to RGB values:

Sample	Running time (in milliseconds)
bmp-sample-1.bmp	53
bmp-sample-2.bmp	138
bmp-sample-3.bmp	145

Potential improvements:

The algorithm’s core is to perform transformations from one domain to another. Therefore, the algorithm’s speed and precision are strongly dependent on floating-point operations.

Possible improvements for the algorithm:

- Use iterative matrix multiplication
- Use lifting networks to reduce the number of floating-point operations
- Perform Co and Cg subsampling
- Optimization techniques for operations on arrays (Loop unrolling)
- Perform less access to the main memory for storing results of buffers.