ID: 800962452 Meghana Gudaram HPC Assignment 4

Compilation: g++ -std=c++11 convol.cpp -fopenmp -o convol
Execution: ./convol 1024 768 3

2)
Flops needed to be done to compute a convolution of dimension k on an image of size n × m
= (no.of additions and multiplications x matrix size x total size of 2-D array)
= **2 × k × k × n × m**

Memory needed to be done to compute a convolution of dimension k on an image of size n × m
= Read k x k matrix + Read array + Write array
= **(size of type) × k × k + (size of type) × n × m + (size of type) × n × m**
= 4 × k × k + 4 × n × m + 4 × n × m (Assuming 32 bit machine, float and int are 4 bytes)

From Assignment 1 and 2,
Maximum number of floating operations in Mamba = 1638 GFlops / sec
Maximum number of Integer operations in Mamba = 1228 GIops / sec
In Mamba, maximum Bandwidth or Bytes fetched at a rate of (68+68 GB/s) = 136 GigaBytes/sec

Time taken for convolution of dimension 3 on an image of 1024x768 = max(FlopTime, MemTime)

For floating point arithmetic, Time = Total flops / (max flops / sec)
= 2 x 3 x 3 x 1024 x 768 / (1.7×10^{12})
= 8.642 µsec

For memory operations, Time = Total bytes / (max bytes / sec)
= 4 x (2 x 1024 x 768 + 3 x 3)/ (136×10^{9})
= 46.2 µsec

Hence, maximum time is taken by memory operation, time taken would be 46.2 µsec

Time taken for convolution of dimension 11 on an image of 1024x768= max(FlopTime, MemTime)

For floating point arithmetic, Time = Total flops / (max flops / sec)
= 2 x 11 x 11 x 1024 x 768 / (1.7×10^{12})
= 116.18 µsec

For memory operations, Time = Total flops / (max flops / sec)
= 4 x (2 x 1024 x 768 + 11 x 11)/ (136×10^{9})
= 46.2 µsec

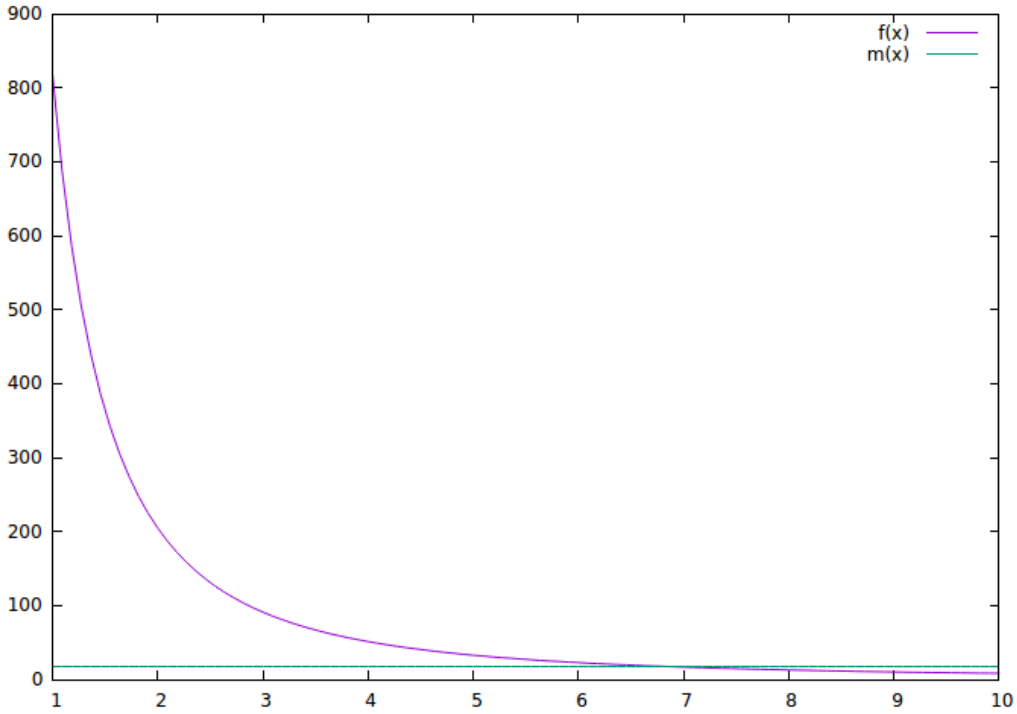
Hence, maximum time is taken by floating point operations, time taken would be 116.18 µsec

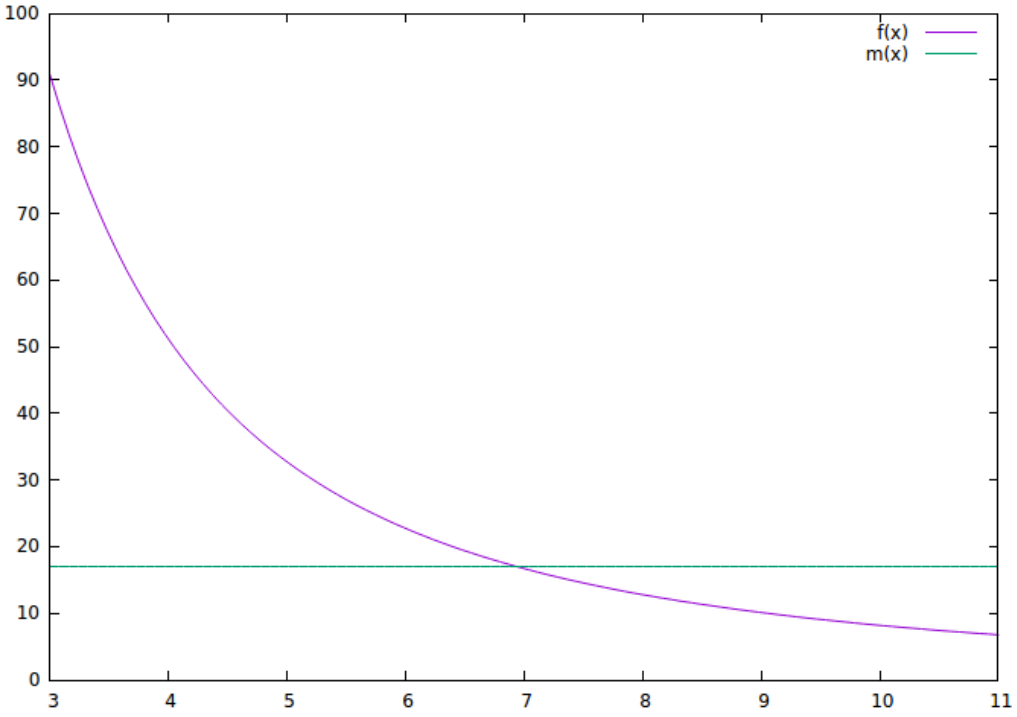
Performance is measured in pixels/sec, i.e. total pixels / time taken

Total number of pixels in m x n image (array) = m x n pixels
Time taken for floating point operations = (2 x m x n x k x k flops) / 1638 Gflops/s
Time taken for memory operations = (sizeof arrayType) x (2 x m x n + k x k) Bytes / (136 Gbytes/s)

Performance for floating point operations = (m x n) pixels / (2 x m x n x k x k flops) / 1638 Gflops/s
= 819 / k^2 Gigapixels/s
Performance for floating point operations = (m x n) pixels / 4 x (2 x m x n + k x k) / 136 Gflops/s
= 34 x m x n / (2 x m x n + k^2) Gigapixels/s

Considering m=1024 and n=768,
f(x) = 819 / x^2 Gigapixels/s
m(x) = 26738688 / (1572864 + x^2) Gigapixels/s

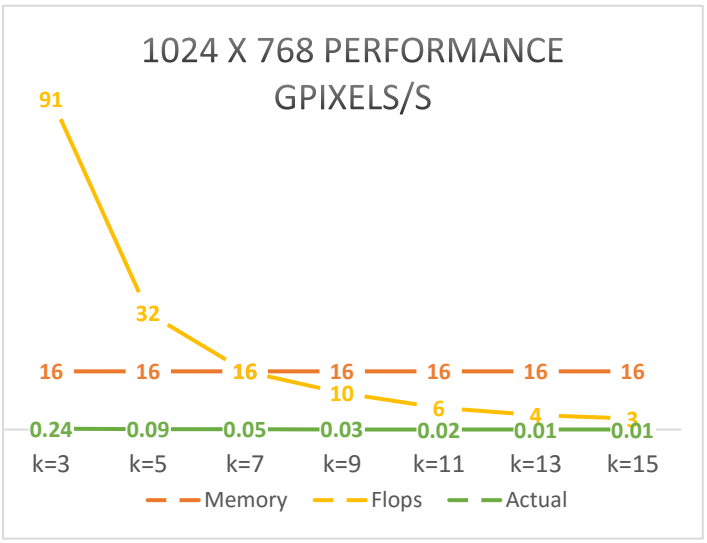


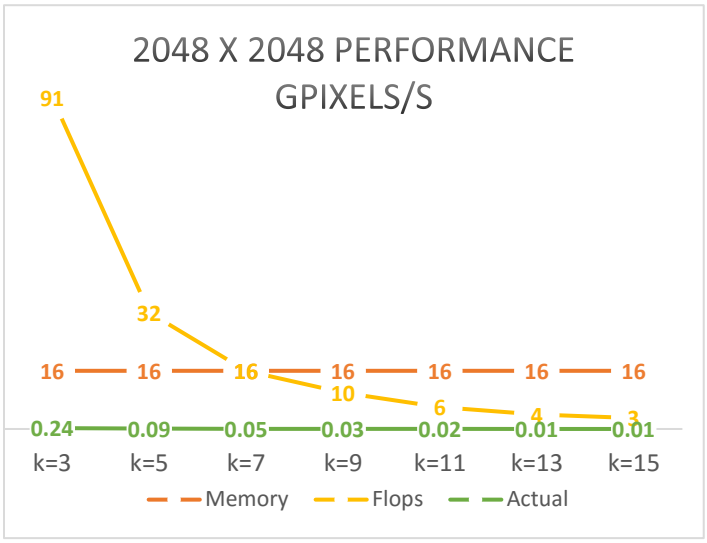


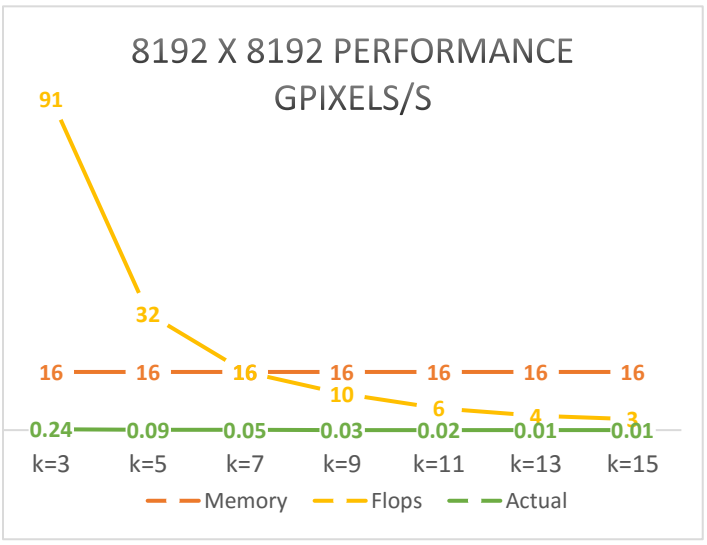
Basic.cpp

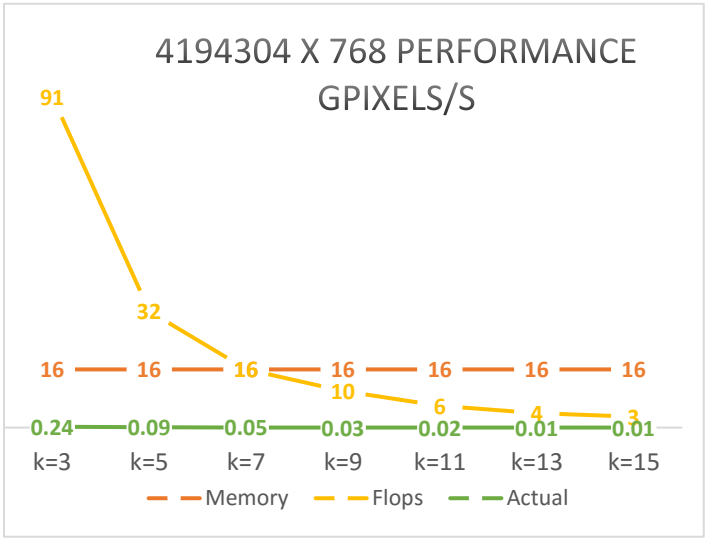
```
#include "compact.hpp"
#include <iostream>
#include <omp.h>
#include <immintrin.h>
#include<chrono>
#include<ctime>

using namespace std;
using namespace std::chrono;

int main (int argc, char **argv)
{
        if(argc < 4)
        {
                printf("Run parameters as follows:\nExecutable ImgHeight ImgWidth k\n");
        }
        else
        {
                long ImageHeight=atoi(argv[1]), ImageWidth=atoi(argv[2]);
                int k=atoi(argv[3]);
                util::Compact2D<int> array(ImageHeight,ImageWidth);
                util::Compact2D<int> Earray(ImageHeight,ImageWidth);
                int convArray[k][k];
                for(int i=0; i < ImageHeight; i++)
                {
                        for(int j=0; j < ImageWidth; j++)
                        {
                                array[i][j]=1;
                        }
                }

                for(int convR=0; convR < k; convR++)
                {
                        for(int convC=0; convC < k; convC++)
                        {
                                convArray[convR][convC]=1;
                        }
                }
                high_resolution_clock::time_point t1 = high_resolution_clock::now();
                for(int m=0;m<100;m++)
                {
        #pragma omp parallel for collapse(2)
                for(int imgH=0; imgH < ImageHeight; imgH++)
                {
                        for(int imgW=0; imgW < ImageWidth; imgW++)
                        {
                                int sum=0;
                                for(int k1=0; k1 < k; k1++)
                                {
                                        for(int k2=0; k2 < k; k2++)
                                        {
                                                if((imgH+k1-1) >= 0 && (imgW+k2-1) >= 0 && (imgH+k1-1) < ImageHeight && (imgW+k2-
1) < ImageWidth)
                                                {
                                                        sum = sum + array[imgH+k1-1][imgW+k2-1];
                                                }
                                        }
                                 }
                                Earray[imgH][imgW]=sum;
                        }
                }
                }
                high_resolution_clock::time_point t2 = high_resolution_clock::now();
                duration<double> time_span = duration_cast<duration<double>> (t2 - t1);

                printf(" Time taken for operations of array[%d][%d] with k as %d : %lf\n",ImageHeight, ImageWidth,
time_span.count()/100, k);

                float fma=819/(k*k);
                float Mem=(34*ImageHeight*ImageWidth)/((2*ImageHeight*ImageWidth) + (k*k));
                printf("Performance in GigaPixels/s by FMA: Expected= %0.2f\n",fma);
                printf("Performance in GigaPixels/s by Memory: Expected= %0.2f\n", Mem);
                double expected=fma>Mem?fma:Mem;
                printf("From above expected max performance= %0.2f, actual= %0.2f\n", expected, ImageHeight*Image-
Width/(10000000*time_span.count()));

        }
  return 0;
}
```

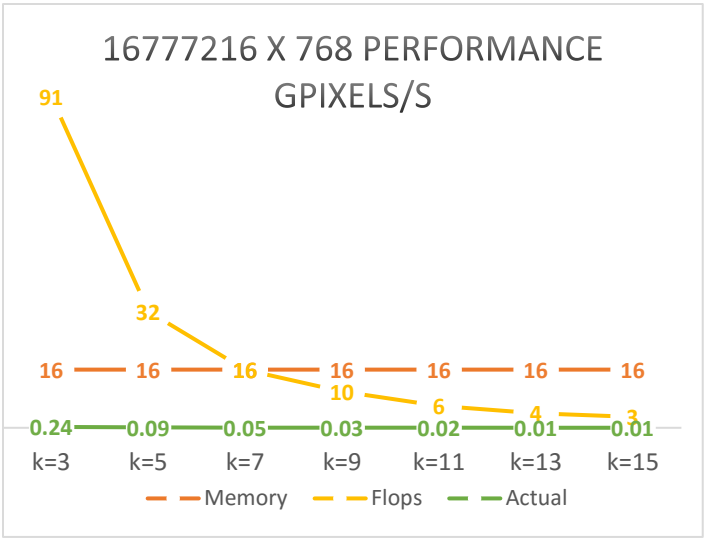
Results from Basic code are as follows:











Values for 1024 x 768 and 2048 x 2048 matrix

Time taken for operations of array[1024][768] with k as 3 : 0.003255
Performance in GigaPixels/s by FMA: Expected= 91.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 91.00, actual= 0.24

Time taken for operations of array[1024][768] with k as 5 : 0.009041
Performance in GigaPixels/s by FMA: Expected= 32.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 32.00, actual= 0.09
Time taken for operations of array[1024][768] with k as 7 : 0.016637
Performance in GigaPixels/s by FMA: Expected= 16.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.05

Time taken for operations of array[1024][768] with k as 9 : 0.026820
Performance in GigaPixels/s by FMA: Expected= 10.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.03

Time taken for operations of array[1024][768] with k as 11 : 0.043278
Performance in GigaPixels/s by FMA: Expected= 6.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.02

Time taken for operations of array[1024][768] with k as 13 : 0.054616
Performance in GigaPixels/s by FMA: Expected= 4.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.01

Time taken for operations of array[1024][768] with k as 15 : 0.074244
Performance in GigaPixels/s by FMA: Expected= 3.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.01

Time taken for operations of array[2048][2048] with k as 3 : 0.017300
Performance in GigaPixels/s by FMA: Expected= 91.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 91.00, actual= 0.24

Time taken for operations of array[2048][2048] with k as 5 : 0.048214
Performance in GigaPixels/s by FMA: Expected= 32.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 32.00, actual= 0.09
Time taken for operations of array[2048][2048] with k as 7 : 0.088742
Performance in GigaPixels/s by FMA: Expected= 16.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.05

Time taken for operations of array[2048][2048] with k as 9 : 0.144247
Performance in GigaPixels/s by FMA: Expected= 10.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.03

Time taken for operations of array[2048][2048] with k as 11 : 0.212176
Performance in GigaPixels/s by FMA: Expected= 6.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.02

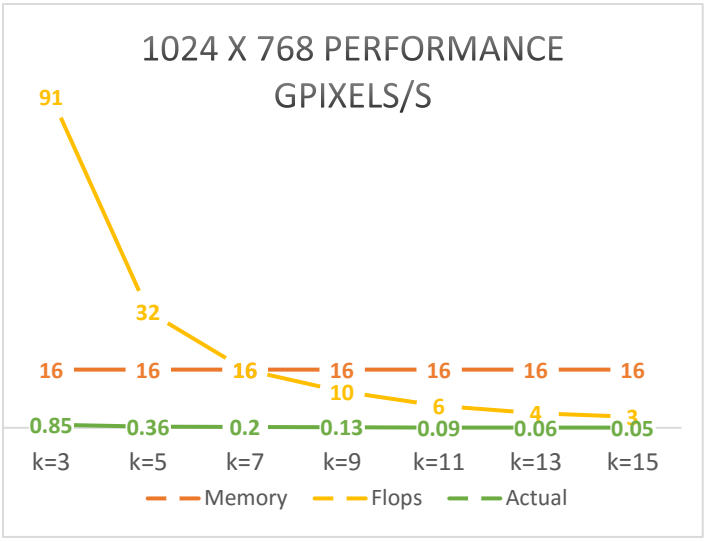
Time taken for operations of array[2048][2048] with k as 13 : 0.292782
Performance in GigaPixels/s by FMA: Expected= 4.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.01

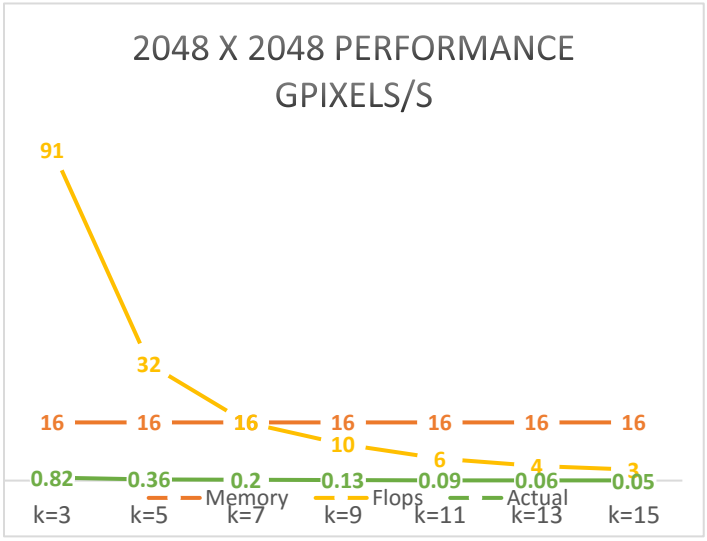
Time taken for operations of array[2048][2048] with k as 15 : 0.387162
Performance in GigaPixels/s by FMA: Expected= 3.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected max performance= 16.00, actual= 0.01

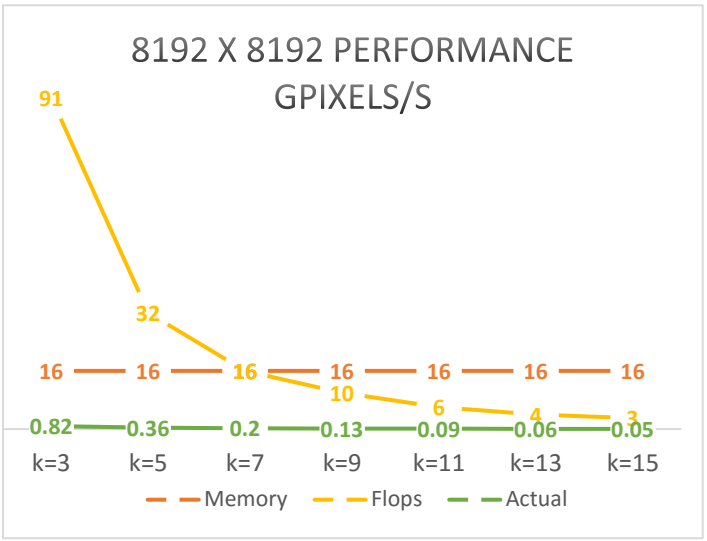
```
#include "compact.hpp"
#include <iostream>
#include <omp.h>
#include <immintrin.h>
#include<chrono>
#include<ctime>

using namespace std;
using namespace std::chrono;

int main (int argc, char **argv)
{
        if(argc < 5)
        {
                printf("Run parameters as follows:\nExecutable ImgHeight ImgWidth k block\n");
        }
        else
        {
                long ImageHeight=atoi(argv[1]), ImageWidth=atoi(argv[2]);
                int k=atoi(argv[3]), block=atoi(argv[4]);
                int kcen=k/2;
                //printf("kcen: %d",kcen);
                util::Compact2D<float> array(ImageHeight,ImageWidth);
                util::Compact2D<float> Earray(ImageHeight,ImageWidth);
                util::Compact2D<float> convArray(k, k);
                for(int i=0; i < ImageHeight; i++)
                {
                        for(int j=0; j < ImageWidth; j++)
                        {
                                array[i][j]=1;
                        }
                }

                for(int convR=0; convR < k; convR++)
                {
                        for(int convC=0; convC < k; convC++)
                        {
                                convArray[convR][convC]=1;
                        }
                }
                int count=0;
                high_resolution_clock::time_point t1 = high_resolution_clock::now();
                for(int m=0;m<100;m++)
                {
                        #pragma omp parallel for schedule(runtime) collapse(2)
                        for(int imgH=0; imgH < ImageHeight; imgH=imgH+block)
                        {
                                for(int imgW=0; imgW < ImageWidth; imgW=imgW+block)
                                {
                                       // #pragma omp parallel for schedule(dynamic, 1000) collapse(2)
                                        for(int imgHB=imgH; imgHB < imgH+block; imgHB++)
                                        {
                                                for(int imgWB=imgW; imgWB < imgW+block; imgWB=imgWB+8)
                                                {
                                                        if(imgHB < ImageHeight && imgWB < ImageWidth)
                                                        {
                                                                __m256 sum = _mm256_set_ps(0,0,0,0,0,0,0,0);

                                                                __m256 Widthvect = _mm256_loadu_ps(&array[imgHB][imgWB]);
                                                                for(int k1=0; k1 < k; k1++)
                                                                {
                                                                        for(int k2=0; k2 < k; k2++)
                                                                        {

                                                                                if((imgHB+k1-kcen) >= 0 && (imgWB+k2+7-kcen) >= 0 &&
(imgHB+k1-kcen) < ImageHeight && (imgWB+k2+7-kcen) < ImageWidth)

                                                                                {
                                                                                        sum = _mm256_add_ps (sum,
                                                                                _mm256_mul_ps(Widthvect, _mm256_set1_ps(convAr-
                                                                                ray[k1][k2])));

                                                                                }
                                                                        }
                                                                }
                                                                _mm256_storeu_ps((float *)&Earray[imgHB][imgWB],sum);

                                                        }
                                                }
                                        }
                                }
```

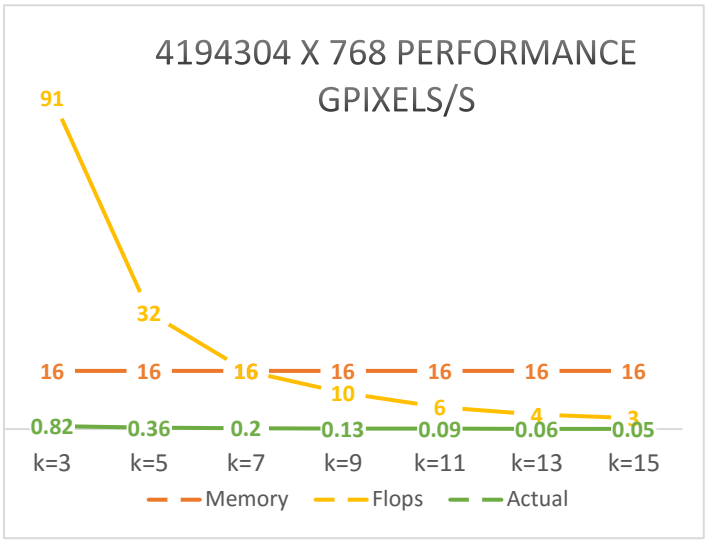
```
                        }
                }
                high_resolution_clock::time_point t2 = high_resolution_clock::now();
                duration<double> time_span = duration_cast<duration<double>> (t2 - t1);
                printf("Time taken for operations of array[%d][%d] with k as %d : %lf\n",ImageHeight, ImageWidth,
time_span.count()/100, k);

                float fma=819/(k*k);
                float Mem=(34*ImageHeight*ImageWidth)/((2*ImageHeight*ImageWidth) + (k*k));
                printf("a[%d][%d] = %f\n", ImageHeight-1, ImageWidth-1, Earray[ImageHeight-1][ImageWidth-1]);
                printf("Performance in GigaPixels/s by FMA: Expected= %0.2f\n",fma);
                printf("Performance in GigaPixels/s by Memory: Expected= %0.2f\n", Mem);
                double expected=fma>Mem?Mem:fma;
                printf("From above expected min performance= %0.2f, actual= %0.2f\n", expected, ImageHeight*Image-
Width/(10000000*time_span.count()));
        }


  return 0;
}
```

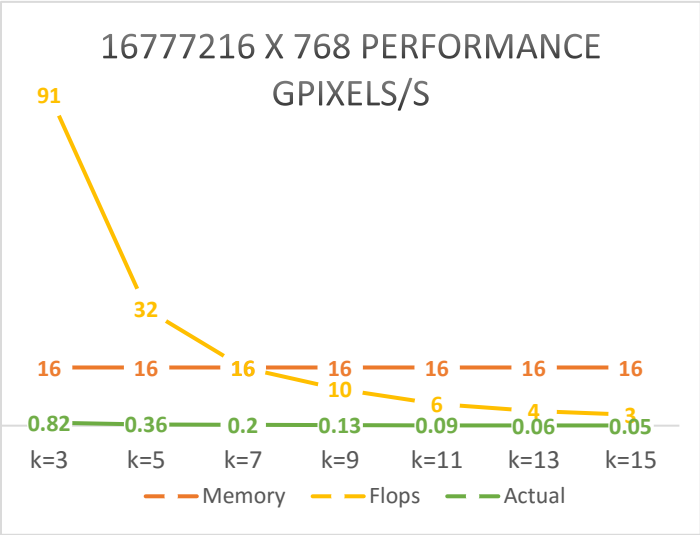
Results from Basic code are as follows: Performance increased from 0.2 GPixels/s to 0.85 GPixels/s or 5.3 %











Values for 1024 x 768 and 2048 x 2048 matrix

Time taken for operations of array[1024][768] with k as 3 : 0.000928
a[1023][767] = 0.000000
Performance in GigaPixels/s by FMA: Expected= 91.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 16.00, actual= 0.85

Time taken for operations of array[1024][768] with k as 5 : 0.002155
a[1023][767] = 0.000000
Performance in GigaPixels/s by FMA: Expected= 32.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 16.00, actual= 0.36

Time taken for operations of array[1024][768] with k as 7 : 0.003903
a[1023][767] = 0.000000
Performance in GigaPixels/s by FMA: Expected= 16.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 16.00, actual= 0.20

Time taken for operations of array[1024][768] with k as 9 : 0.006257
a[1023][767] = 5.000000
Performance in GigaPixels/s by FMA: Expected= 10.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 10.00, actual= 0.13

Time taken for operations of array[1024][768] with k as 11 : 0.009118
a[1023][767] = 12.000000
Performance in GigaPixels/s by FMA: Expected= 6.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 6.00, actual= 0.09

Time taken for operations of array[1024][768] with k as 13 : 0.012668
a[1023][767] = 21.000000
Performance in GigaPixels/s by FMA: Expected= 4.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 4.00, actual= 0.06

Time taken for operations of array[1024][768] with k as 15 : 0.016587
a[1023][767] = 32.000000
Performance in GigaPixels/s by FMA: Expected= 3.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 3.00, actual= 0.05

Time taken for operations of array[2048][2048] with k as 3 : 0.005092
a[2047][2047] = 4.000000
Performance in GigaPixels/s by FMA: Expected= 91.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 16.00, actual= 0.82

Time taken for operations of array[2048][2048] with k as 5 : 0.011569
a[2047][2047] = 9.000000
Performance in GigaPixels/s by FMA: Expected= 32.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 16.00, actual= 0.36

Time taken for operations of array[2048][2048] with k as 7 : 0.020815
a[2047][2047] = 16.000000
Performance in GigaPixels/s by FMA: Expected= 16.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 16.00, actual= 0.20

Time taken for operations of array[2048][2048] with k as 9 : 0.033375
a[2047][2047] = 25.000000
Performance in GigaPixels/s by FMA: Expected= 10.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 10.00, actual= 0.13

Time taken for operations of array[2048][2048] with k as 11 : 0.048418
a[2047][2047] = 36.000000
Performance in GigaPixels/s by FMA: Expected= 6.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 6.00, actual= 0.09

Time taken for operations of array[2048][2048] with k as 13 : 0.066545
a[2047][2047] = 49.000000
Performance in GigaPixels/s by FMA: Expected= 4.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 4.00, actual= 0.06

Time taken for operations of array[2048][2048] with k as 15 : 0.087776
a[2047][2047] = 64.000000
Performance in GigaPixels/s by FMA: Expected= 3.00
Performance in GigaPixels/s by Memory: Expected= 16.00
From above expected min performance= 3.00, actual= 0.05