Module 11 Assignment: Convolution Modification

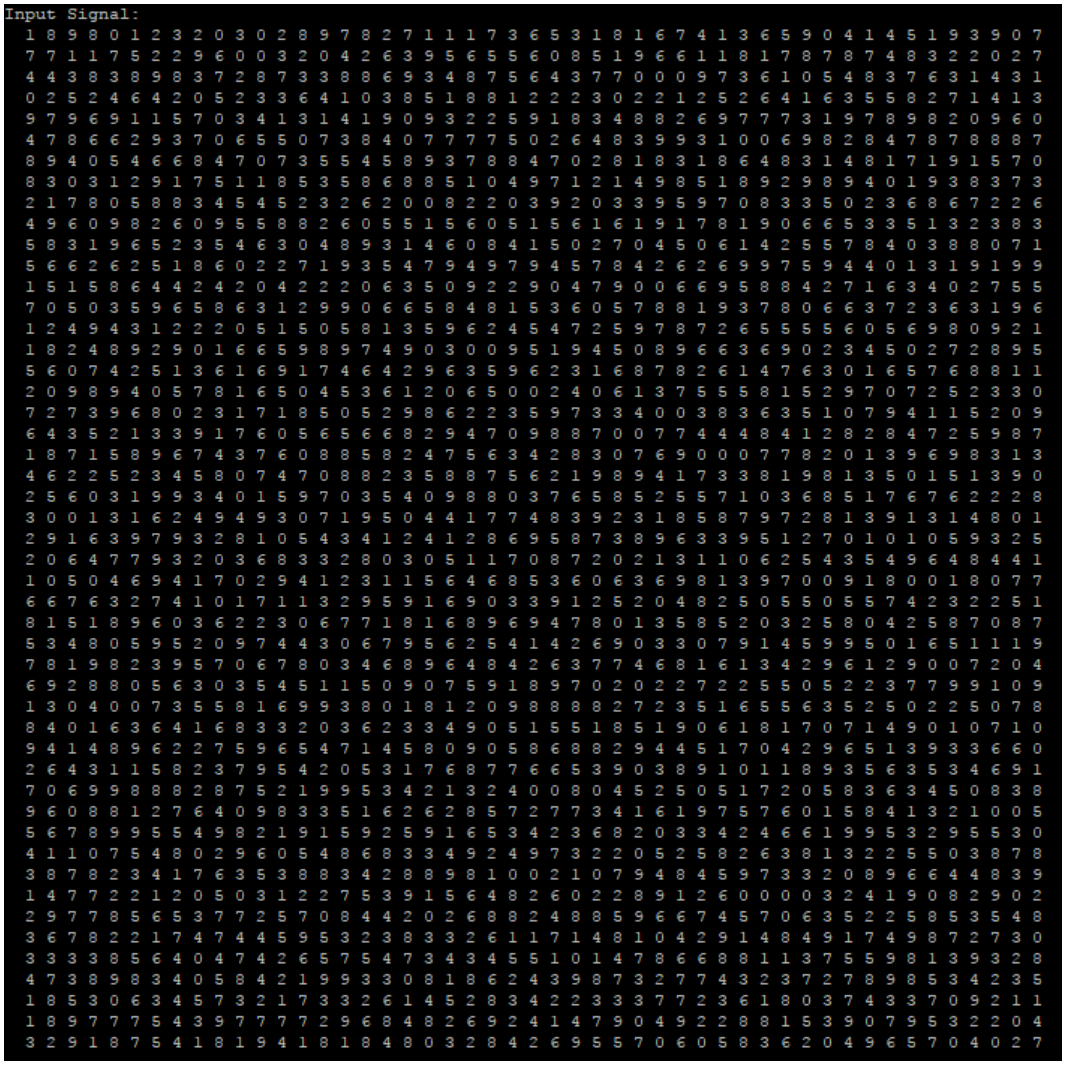
Scott Wroten
11/13/2017
Introduction to GPU Programming

1. Modify assignment to take in 7X7 filter. (35%)

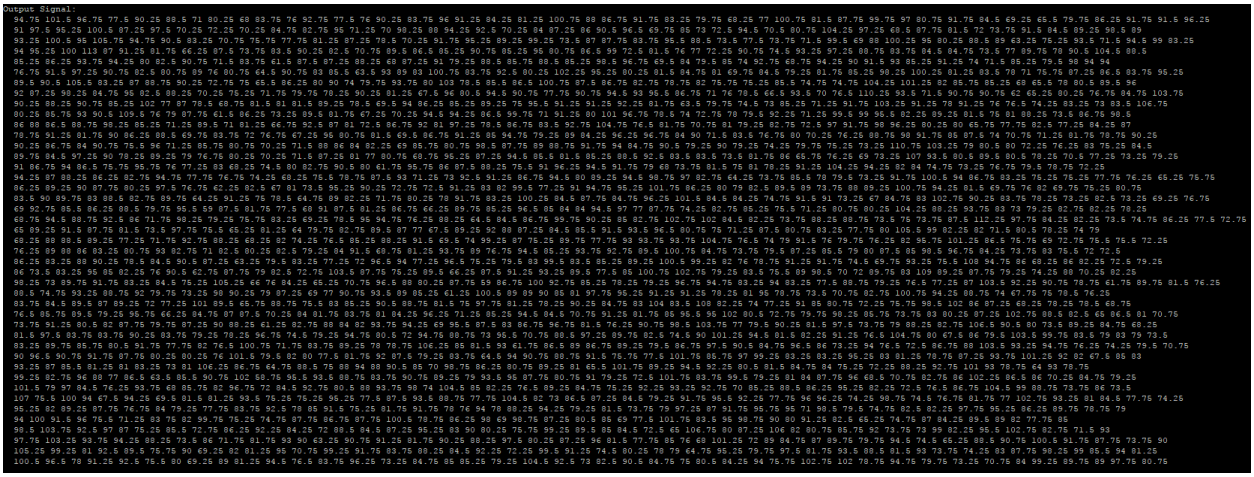
7X7 Mask Filter as gradient (100% one unit from center, 75% two units, 50% three units, and 25% four units):

```
cl_float mask[maskWidth][maskHeight] =
{
   { 0.00, 0.00, 0.25, 0.50, 0.25, 0.00, 0.00 },
   { 0.00, 0.25, 0.50, 0.75, 0.50, 0.25, 0.00 },
   { 0.25, 0.50, 0.75, 1.00, 0.75, 0.50, 0.25 },
   { 0.50, 0.75, 1.00, 0.00, 1.00, 0.75, 0.50 },
   { 0.25, 0.50, 0.75, 1.00, 0.75, 0.50, 0.25 },
   { 0.00, 0.25, 0.50, 0.75, 0.50, 0.25, 0.00 },
   { 0.00, 0.00, 0.25, 0.50, 0.25, 0.00, 0.00 },
};
```

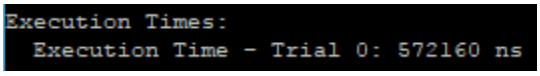
49X49 Input Signal:



43X43 Output Signal (removed 3 columns and rows from each side of input signal to get output signal that trims resulting elements where filter referenced indices outside of input signal):



Execution Time (ns):

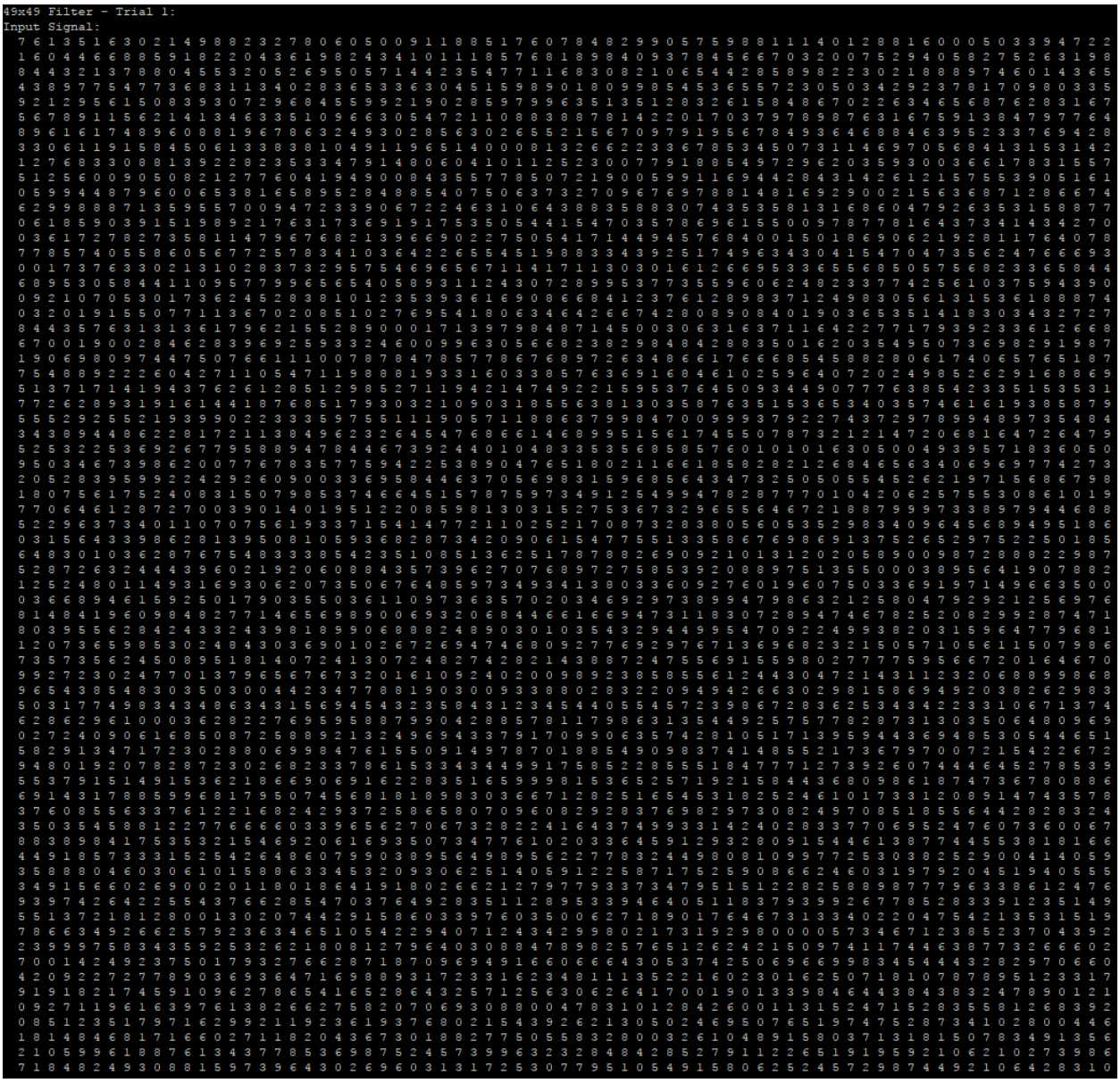


2. Modify assignment to take in 49X49 filter. (35%)

49X49 Mask Filter as gradient (100% one unit from center, 75% two units, 50% three units, and 25% four units):

```
cl_float fortyNineByFortyNineMask[fortyNineByFortyNineMaskWidth][fortyNineByFortyNineMaskHeight] =
{
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.75,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.75,1.00,0.75,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.50,0.75,1.00,0,1.00,0.75,0.50,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.75,1.00,0.75,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.75,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
};
```

73X73 Input Signal:



49X49 Output Signal (removed 24 columns and rows from each side of input signal to get output signal that trims resulting elements where filter referenced indices outside of input signal):

Execution Time (ns):

```
49x49 Filter - Execution Times:
  Execution Time - Trial 0: 22700000 ns
```

3. Output timing or other metrics for comparison of different datasets (10%).

Ran 3 trials for 7x7 Filter to get execution time for each Input Signal being filtered to get resulting Output Signal:

```
Execution Times:
  Execution Time - Trial 0: 572160 ns
  Execution Time - Trial 1: 590960 ns
  Execution Time - Trial 2: 582000 ns
```

Ran 3 trials for 49x49 Filter to get execution time for each Input Signal being filtered to get resulting Output Signal:

```
49x49 Filter - Execution Times:
  Execution Time - Trial 0: 22700000 ns
  Execution Time - Trial 1: 22721680 ns
  Execution Time - Trial 2: 22695600 ns
```

```cpp
//
// Book:      OpenCL(R) Programming Guide
// Authors:   Aaftab Munshi, Benedict Gaster, Timothy Mattson, James Fung, Dan Ginsburg
// ISBN-10:   0-321-74964-2
// ISBN-13:   978-0-321-74964-2
// Publisher: Addison-Wesley Professional
// URLs:      http://safari.informit.com/9780132488006/
//            http://www.openclprogrammingguide.com
//

// Convolution.cpp
//
//    This is a simple example that demonstrates OpenCL platform, device, and context
//    use.

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>

#ifdef __APPLE__
#include <OpenCL/cl.h>
#else
#include <CL/cl.h>
#endif

#if !defined(CL_CALLBACK)
#define CL_CALLBACK
#endif

// Constants
const unsigned int inputSignalWidth  = 73;
const unsigned int inputSignalHeight = 73;
const unsigned int outputSignalWidth  = 49;
const unsigned int outputSignalHeight = 49;
const unsigned int sevenBySevenMaskWidth = 7;
const unsigned int sevenBySevenMaskHeight = 7;
const unsigned int fortyNineByFortyNineMaskWidth = 49;
const unsigned int fortyNineByFortyNineMaskHeight = 49;

cl_float sevenBySevenMask[sevenBySevenMaskWidth][sevenBySevenMaskHeight] =
{
   { 0.00, 0.00, 0.25, 0.50, 0.25, 0.00, 0.00 },
   { 0.00, 0.25, 0.50, 0.75, 0.50, 0.25, 0.00 },
   { 0.25, 0.50, 0.75, 1.00, 0.75, 0.50, 0.25 },
   { 0.50, 0.75, 1.00, 0.00, 1.00, 0.75, 0.50 },
   { 0.25, 0.50, 0.75, 1.00, 0.75, 0.50, 0.25 },
   { 0.00, 0.25, 0.50, 0.75, 0.50, 0.25, 0.00 },
   { 0.00, 0.00, 0.25, 0.50, 0.25, 0.00, 0.00 },
};

cl_float fortyNineByFortyNineMask[fortyNineByFortyNineMaskWidth][fortyNineByFortyNineMaskHeight] =
{
   { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
   { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
   { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
   { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
   { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
   { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
```

```
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0 },
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.75,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0 },
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.75,1.00,0.75,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0 },
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.50,0.75,1.00,0,1.00,0.75,0.50,0,0,0,0,0,0,0,0,0,0,0,0
 ,0,0,0,0,0,0,0,0,0 },
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.75,1.00,0.75,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0 },
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.75,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0 },
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.25,0.50,0.25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0 },
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
 ,
{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
```

```
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    ,
    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
};


// Function to check and handle OpenCL errors
inline void
checkErr(cl_int err, const char * name)
{
    if (err != CL_SUCCESS) {
        std::cerr << "ERROR: " <<  name << " (" << err << ")" << std::endl;
        exit(EXIT_FAILURE);
    }
}

void CL_CALLBACK contextCallback(
   const char * errInfo,
   const void * private_info,
   size_t cb,
   void * user_data)
{
   std::cout << "Error occured during context use: " << errInfo << std::endl;
   // should really perform any clearup and so on at this point
   // but for simplicitly just exit.
   exit(1);
}

cl_ulong RunKernelForInputSignalWithSevenBySevenFilter()
{
   // Initialize Variables
   cl_uint index;
   cl_int errNum;
   cl_kernel kernel;
   cl_mem maskBuffer;
   cl_uint numDevices;
   cl_program program;
   size_t totalLength;
   cl_uint numPlatforms;
   cl_ulong stopTime = 0;
   cl_ulong startTime = 0;
   cl_command_queue queue;
   cl_ulong kernelTime = 0;
   cl_mem inputSignalBuffer;
   cl_context context = NULL;
   cl_mem outputSignalBuffer;
   cl_event startStopEvent = 0;
   cl_platform_id * platformIDs;
```

```
   cl_device_id *deviceIDs = NULL;
   cl_float inputSignal[inputSignalWidth][inputSignalHeight];
   cl_float outputSignal[outputSignalWidth][outputSignalHeight];

   // Initialize Input Signal to Random Values
   for (int i = 0; i < inputSignalHeight; i++)
   {
      for (int j = 0; j < inputSignalWidth; j++)
      {
         inputSignal[i][j] = rand() % 10;
      }
   }

   // First, select an OpenCL platform to run on.
   errNum = clGetPlatformIDs(0, NULL, &numPlatforms);
   checkErr((errNum != CL_SUCCESS) ? errNum :
      (numPlatforms <= 0 ? -1 : CL_SUCCESS), "clGetPlatformIDs");
   platformIDs = (cl_platform_id *)alloca(sizeof(cl_platform_id) * numPlatforms);
   errNum = clGetPlatformIDs(numPlatforms, platformIDs, NULL);
   checkErr((errNum != CL_SUCCESS) ? errNum :
      (numPlatforms <= 0 ? -1 : CL_SUCCESS), "clGetPlatformIDs");

   // Iterate through the list of platforms until we find one that supports
   // a CPU device, otherwise fail with an error.
   for (index = 0; index < numPlatforms; index++)
   {
      errNum = clGetDeviceIDs(platformIDs[index],
         CL_DEVICE_TYPE_GPU, 0, NULL, &numDevices);

      if (errNum != CL_SUCCESS && errNum != CL_DEVICE_NOT_FOUND)
      {
         checkErr(errNum, "clGetDeviceIDs");
      }
      else if (numDevices > 0)
      {
         deviceIDs = (cl_device_id *)alloca(sizeof(cl_device_id) * numDevices);
         errNum = clGetDeviceIDs(platformIDs[index],
            CL_DEVICE_TYPE_GPU, numDevices, &deviceIDs[0], NULL);
         checkErr(errNum, "clGetDeviceIDs");

         break;
      }
   }

   // Next, create an OpenCL context on the selected platform.
   cl_context_properties contextProperties[] =
   {
      CL_CONTEXT_PLATFORM,
      (cl_context_properties)platformIDs[index],
      0
   };

   context = clCreateContext(contextProperties, numDevices,
      deviceIDs, &contextCallback, NULL, &errNum);
   checkErr(errNum, "clCreateContext");

   std::ifstream srcFile("Convolution.cl");
   checkErr(srcFile.is_open() ? CL_SUCCESS : -1, "reading Convolution.cl");

   std::string srcProg(std::istreambuf_iterator<char>(srcFile),
      (std::istreambuf_iterator<char>()));

   const char *src = srcProg.c_str();
   totalLength = srcProg.length();
```

```
    // Create program from source
    program = clCreateProgramWithSource(context, 1, &src, &totalLength, &errNum);
    checkErr(errNum, "clCreateProgramWithSource");

    // Build program
    errNum = clBuildProgram(program, numDevices, deviceIDs, NULL, NULL, NULL);
    if (errNum != CL_SUCCESS)
    {
        // Determine the reason for the error
        char buildLog[16384];
        clGetProgramBuildInfo(program, deviceIDs[0], CL_PROGRAM_BUILD_LOG,
            sizeof(buildLog), buildLog, NULL);

        std::cerr << "Error in kernel: " << std::endl;
        std::cerr << buildLog;

        checkErr(errNum, "clBuildProgram");
    }

    // Create kernel object
    kernel = clCreateKernel(program, "convolve", &errNum);
    checkErr(errNum, "clCreateKernel");

    // Now allocate buffers
    inputSignalBuffer = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
        sizeof(cl_float) * inputSignalHeight * inputSignalWidth,
        static_cast<void *>(inputSignal),
        &errNum);
    checkErr(errNum, "clCreateBuffer(inputSignal)");

    maskBuffer = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
        sizeof(cl_float) * sevenBySevenMaskHeight * sevenBySevenMaskWidth, static_cast<void *>
         (sevenBySevenMask), &errNum);
    checkErr(errNum, "clCreateBuffer(mask)");

    outputSignalBuffer = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
        sizeof(cl_float) * outputSignalHeight * outputSignalWidth, NULL, &errNum);
    checkErr(errNum, "clCreateBuffer(outputSignal)");

    // Pick the first device and create command queue.
    queue = clCreateCommandQueue(context, deviceIDs[0],
        CL_QUEUE_PROFILING_ENABLE, &errNum);
    checkErr(errNum, "clCreateCommandQueue");

    errNum = clSetKernelArg(kernel, 0, sizeof(cl_mem), &inputSignalBuffer);
    errNum |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &maskBuffer);
    errNum |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &outputSignalBuffer);
    errNum |= clSetKernelArg(kernel, 3, sizeof(cl_uint), &inputSignalWidth);
    errNum |= clSetKernelArg(kernel, 4, sizeof(cl_uint), &sevenBySevenMaskWidth);
    checkErr(errNum, "clSetKernelArg");

    const size_t globalWorkSize[1] = { outputSignalWidth * outputSignalHeight };
    const size_t localWorkSize[1] = { 1 };

    // Queue the kernel up for execution across the array
    errNum = clEnqueueNDRangeKernel(queue, kernel, 1,
        NULL, globalWorkSize, localWorkSize, 0, NULL, &startStopEvent);
    checkErr(errNum, "clEnqueueNDRangeKernel");

    // Wait for Events to End to Get Start and Stop Time
    errNum = clWaitForEvents(1, &startStopEvent);
    checkErr(errNum, "clWaitForEvents");
    errNum = clGetEventProfilingInfo(startStopEvent,
        CL_PROFILING_COMMAND_START, sizeof(cl_ulong), &startTime, NULL);
    checkErr(errNum, "clGetEventProfilingInfo");
```

```
   errNum = clGetEventProfilingInfo(startStopEvent,
      CL_PROFILING_COMMAND_END, sizeof(cl_ulong), &stopTime, NULL);
   checkErr(errNum, "clGetEventProfilingInfo");

   // Compute Kernel Time
   kernelTime = (stopTime - startTime);

   // Read output buffer
   errNum = clEnqueueReadBuffer(queue, outputSignalBuffer, CL_TRUE,
      0, sizeof(cl_uint) * outputSignalWidth * outputSignalHeight,
      outputSignal, 0, NULL, NULL);
   checkErr(errNum, "clEnqueueReadBuffer");

   // Write Input Signal
   std::cout << "Input Signal:" << std::endl;

   // Write each element of input signal
   for (int y = 0; y < inputSignalHeight; y++)
   {
      // Add Spacing
      std::cout << "  ";

      // Step through and output each element in signal
      for (int x = 0; x < inputSignalWidth; x++)
      {
         std::cout << inputSignal[x][y] << " ";
      }

      // Add new line for next row
      std::cout << std::endl;
   }

   // Add New Line for Spacing
   std::cout << std::endl;

   // Write Output Signal
   std::cout << "Output Signal:" << std::endl;

   // Output the result buffer
   for (int y = 0; y < outputSignalHeight; y++)
   {
      // Add Spacing
      std::cout << "  ";

      // Step through and output each element in signal
      for (int x = 0; x < outputSignalWidth; x++)
      {
         std::cout << outputSignal[x][y] << " ";
      }

      // Add new line for next row
      std::cout << std::endl;
   }

   // Return total kernel time
   return kernelTime;
}

cl_ulong RunKernelForInputSignalWithFortyNineByFortyNineFilter()
{
   // Initialize Variables
   cl_uint index;
   cl_int errNum;
   cl_kernel kernel;
   cl_mem maskBuffer;
```

```
    cl_uint numDevices;
    cl_program program;
    size_t totalLength;
    cl_uint numPlatforms;
    cl_ulong stopTime = 0;
    cl_ulong startTime = 0;
    cl_command_queue queue;
    cl_ulong kernelTime = 0;
    cl_mem inputSignalBuffer;
    cl_context context = NULL;
    cl_mem outputSignalBuffer;
    cl_event startStopEvent = 0;
    cl_platform_id * platformIDs;
    cl_device_id *deviceIDs = NULL;
    cl_float inputSignal[inputSignalWidth][inputSignalHeight];
    cl_float outputSignal[outputSignalWidth][outputSignalHeight];

    // Initialize Input Signal to Random Values
    for (int i = 0; i < inputSignalHeight; i++)
    {
       for (int j = 0; j < inputSignalWidth; j++)
       {
          inputSignal[i][j] = rand() % 10;
       }
    }

    // First, select an OpenCL platform to run on.
    errNum = clGetPlatformIDs(0, NULL, &numPlatforms);
    checkErr((errNum != CL_SUCCESS) ? errNum :
       (numPlatforms <= 0 ? -1 : CL_SUCCESS), "clGetPlatformIDs");
    platformIDs = (cl_platform_id *)alloca(sizeof(cl_platform_id) * numPlatforms);
    errNum = clGetPlatformIDs(numPlatforms, platformIDs, NULL);
    checkErr((errNum != CL_SUCCESS) ? errNum :
       (numPlatforms <= 0 ? -1 : CL_SUCCESS), "clGetPlatformIDs");

    // Iterate through the list of platforms until we find one that supports
    // a CPU device, otherwise fail with an error.
    for (index = 0; index < numPlatforms; index++)
    {
       errNum = clGetDeviceIDs(platformIDs[index],
          CL_DEVICE_TYPE_GPU, 0, NULL, &numDevices);

       if (errNum != CL_SUCCESS && errNum != CL_DEVICE_NOT_FOUND)
       {
          checkErr(errNum, "clGetDeviceIDs");
       }
       else if (numDevices > 0)
       {
          deviceIDs = (cl_device_id *)alloca(sizeof(cl_device_id) * numDevices);
          errNum = clGetDeviceIDs(platformIDs[index],
             CL_DEVICE_TYPE_GPU, numDevices, &deviceIDs[0], NULL);
          checkErr(errNum, "clGetDeviceIDs");

          break;
       }
    }

    // Next, create an OpenCL context on the selected platform.
    cl_context_properties contextProperties[] =
    {
       CL_CONTEXT_PLATFORM,
       (cl_context_properties)platformIDs[index],
       0
    };
```

```
   context = clCreateContext(contextProperties, numDevices,
      deviceIDs, &contextCallback, NULL, &errNum);
   checkErr(errNum, "clCreateContext");

   std::ifstream srcFile("Convolution.cl");
   checkErr(srcFile.is_open() ? CL_SUCCESS : -1, "reading Convolution.cl");

   std::string srcProg(std::istreambuf_iterator<char>(srcFile),
      (std::istreambuf_iterator<char>()));

   const char *src = srcProg.c_str();
   totalLength = srcProg.length();

   // Create program from source
   program = clCreateProgramWithSource(context, 1, &src, &totalLength, &errNum);
   checkErr(errNum, "clCreateProgramWithSource");

   // Build program
   errNum = clBuildProgram(program, numDevices, deviceIDs, NULL, NULL, NULL);
   if (errNum != CL_SUCCESS)
   {
      // Determine the reason for the error
      char buildLog[16384];
      clGetProgramBuildInfo(program, deviceIDs[0], CL_PROGRAM_BUILD_LOG,
         sizeof(buildLog), buildLog, NULL);

      std::cerr << "Error in kernel: " << std::endl;
      std::cerr << buildLog;

      checkErr(errNum, "clBuildProgram");
   }

   // Create kernel object
   kernel = clCreateKernel(program, "convolve", &errNum);
   checkErr(errNum, "clCreateKernel");

   // Now allocate buffers
   inputSignalBuffer = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
      sizeof(cl_float) * inputSignalHeight * inputSignalWidth,
      static_cast<void *>(inputSignal),
      &errNum);
   checkErr(errNum, "clCreateBuffer(inputSignal)");

   maskBuffer = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
      sizeof(cl_float) * fortyNineByFortyNineMaskHeight * fortyNineByFortyNineMaskWidth,
       static_cast<void *>(fortyNineByFortyNineMask), &errNum);
   checkErr(errNum, "clCreateBuffer(mask)");

   outputSignalBuffer = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
      sizeof(cl_float) * outputSignalHeight * outputSignalWidth, NULL, &errNum);
   checkErr(errNum, "clCreateBuffer(outputSignal)");

   // Pick the first device and create command queue.
   queue = clCreateCommandQueue(context, deviceIDs[0],
      CL_QUEUE_PROFILING_ENABLE, &errNum);
   checkErr(errNum, "clCreateCommandQueue");

   errNum = clSetKernelArg(kernel, 0, sizeof(cl_mem), &inputSignalBuffer);
   errNum |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &maskBuffer);
   errNum |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &outputSignalBuffer);
   errNum |= clSetKernelArg(kernel, 3, sizeof(cl_uint), &inputSignalWidth);
   errNum |= clSetKernelArg(kernel, 4, sizeof(cl_uint), &fortyNineByFortyNineMaskWidth);
   checkErr(errNum, "clSetKernelArg");

   const size_t globalWorkSize[1] = { outputSignalWidth * outputSignalHeight };
```

```
    const size_t localWorkSize[1] = { 1 };

    // Queue the kernel up for execution across the array
    errNum = clEnqueueNDRangeKernel(queue, kernel, 1,
       NULL, globalWorkSize, localWorkSize, 0, NULL, &startStopEvent);
    checkErr(errNum, "clEnqueueNDRangeKernel");

    // Wait for Events to End to Get Start and Stop Time
    errNum = clWaitForEvents(1, &startStopEvent);
    checkErr(errNum, "clWaitForEvents");
    errNum = clGetEventProfilingInfo(startStopEvent,
       CL_PROFILING_COMMAND_START, sizeof(cl_ulong), &startTime, NULL);
    checkErr(errNum, "clGetEventProfilingInfo");
    errNum = clGetEventProfilingInfo(startStopEvent,
       CL_PROFILING_COMMAND_END, sizeof(cl_ulong), &stopTime, NULL);
    checkErr(errNum, "clGetEventProfilingInfo");

    // Compute Kernel Time
    kernelTime = (stopTime - startTime);

    // Read output buffer
    errNum = clEnqueueReadBuffer(queue, outputSignalBuffer, CL_TRUE,
       0, sizeof(cl_uint) * outputSignalWidth * outputSignalHeight,
       outputSignal, 0, NULL, NULL);
    checkErr(errNum, "clEnqueueReadBuffer");

    // Write Input Signal
    std::cout << "Input Signal:" << std::endl;

    // Write each element of input signal
    for (int y = 0; y < inputSignalHeight; y++)
    {
       // Add Spacing
       std::cout << "  ";

       // Step through and output each element in signal
       for (int x = 0; x < inputSignalWidth; x++)
       {
          std::cout << inputSignal[x][y] << " ";
       }

       // Add new line for next row
       std::cout << std::endl;
    }

    // Add New Line for Spacing
    std::cout << std::endl;

    // Write Output Signal
    std::cout << "Output Signal:" << std::endl;

    // Output the result buffer
    for (int y = 0; y < outputSignalHeight; y++)
    {
       // Add Spacing
       std::cout << "  ";

       // Step through and output each element in signal
       for (int x = 0; x < outputSignalWidth; x++)
       {
          std::cout << outputSignal[x][y] << " ";
       }

       // Add new line for next row
       std::cout << std::endl;
```

```
    }

    // Return total kernel time
    return kernelTime;
}

// main() for Convoloution example
int main(int argc, char** argv)
{
    // Initialize Variables
    const int numberOfTrials = 3;
    cl_ulong executionTimesForEachSevenBySevenTrial[numberOfTrials];
    cl_ulong executionTimesForEachFortyNineByFortyNineTrial[numberOfTrials];

    // Execute Trial for 7x7 Filter
    for (int i = 0; i < numberOfTrials; i++)
    {
        // Write header for starting trial
        std::cout << "7x7 Filter - Trial " << i << ":" << std::endl;

        // Run Trial and Record total time to execute
        executionTimesForEachSevenBySevenTrial[i] = RunKernelForInputSignalWithSevenBySevenFilter();

        // Add Line for Spacing
        std::cout << std::endl;
    }

    // Output Execution Time
    std::cout << std::endl;
    std::cout << "7x7 Filter - Execution Times:" << std::endl;

    // Write out execution time for each
    for (int i = 0; i < numberOfTrials; i++)
    {
        std::cout << "  Execution Time - Trial " << i << ": "
            << executionTimesForEachFortyNineByFortyNineTrial[i] << " ns" << std::endl;
    }

    // Output Execution Time
    std::cout << std::endl;

    // Execute Trial for 49x49 Filter
    for (int i = 0; i < numberOfTrials; i++)
    {
        // Write header for starting trial
        std::cout << "49x49 Filter - Trial " << i << ":" << std::endl;

        // Run Trial and Record total time to execute
        executionTimesForEachFortyNineByFortyNineTrial[i] =
         RunKernelForInputSignalWithFortyNineByFortyNineFilter();

        // Add Line for Spacing
        std::cout << std::endl;
    }

    // Output Execution Time
    std::cout << std::endl;
    std::cout << "49x49 Filter - Execution Times:" << std::endl;

    // Write out execution time for each
    for (int i = 0; i < numberOfTrials; i++)
    {
        std::cout << "  Execution Time - Trial " << i << ": "
            << executionTimesForEachFortyNineByFortyNineTrial[i] << " ns" << std::endl;
    }
```

```
    // Write to Console Success
    std::cout << std::endl;
    std::cout << "Executed program successfully." << std::endl;
    std::cout << std::endl;

    // Return Success
    return 0;
}
```

```
//
// Book:      OpenCL(R) Programming Guide
// Authors:   Aaftab Munshi, Benedict Gaster, Timothy Mattson, James Fung, Dan Ginsburg
// ISBN-10:   0-321-74964-2
// ISBN-13:   978-0-321-74964-2
// Publisher: Addison-Wesley Professional
// URLs:      http://safari.informit.com/9780132488006/
//            http://www.openclprogrammingguide.com
//

// Convolution.cl
//
//    This is a simple kernel performing convolution.

__kernel void convolve(const __global float * const input,
   __constant float * const mask,
   __global  float * const output,
   const int inputWidth,
   const int maskWidth)
{
   // Initialize Variables
   float sum = 0;
   const int x = get_global_id(0);
   const int y = get_global_id(1);

   for (int r = 0; r < maskWidth; r++)
   {
      const int idxIntmp = (y + r) * inputWidth + x;

      for (int c = 0; c < maskWidth; c++)
      {
         sum += mask[(r * maskWidth)  + c] * input[idxIntmp + c];
      }
   }

   output[y * get_global_size(0) + x] = sum;
}
```