**CHAPTER 1**

**INTRODUCTION**

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

In image filtering there are lot of computations involved which leads to consumption of time since it involved serial computing. To make it fast we go for parallel computation ,we assign kernels to each task and passes to the workgroups ,which actually increases the time speed. For parallel processing of image we use OpenCL platform .

OpenCL is a heterogeneous world ,where a model platform consists of one or more GPU'S ,CPU'S and optional accelerators like DSP'S.OPENCL platform model consists of computational resources,compute unit ,host,processing unit,compute device.

Anatomy of OPENCL consists of HOST code and DEVICE code.HOST code written in c++ executes on the host and DEVICE code written in opencl and executes on device. Host code sends commands to the Devices to transfer data between host memory and device memories to execute device code. Serial code executes in a Host (CPU) thread . Parallel code executes in many Device (GPU) threads across multiple processing elements.

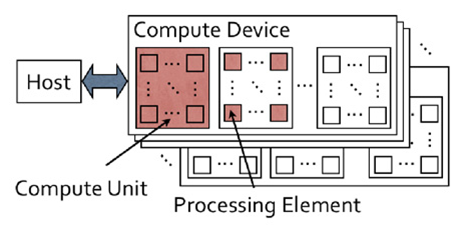


Figure 1.1: **Anatomy of OpenCL**

OPENCL execution model where Serial code executes in a Host (CPU) thread Parallel code executes in many Device (GPU) threads across multiple processing elements. An N-dimension domain of work-items Define the ―best N-dimensioned index space for your algorithm . Kernels are executed across a global domain of work-items . Work-items are grouped into local work groups. Work can be executed in-order or out-of-order.

**1.1 Parallel computing**

Parallel computing is a form of computation in which many calculations are carried out simultaneously .Operating on a principle that large problems can often be divided into smaller ones ,which are solved concurrently .There are several types of parallel computing such as bit level ,instruction, task and data parallelism.Image can also be computed.

When an image is acquired by a camera or other imaging system, often the vision system for which it is intended is unable to use it directly. The image may be corrupted by random variations in intensity, variations in illumination, or poor contrast that must be dealt with in the early stages of vision processing.

OpenCL (Open Computing Language) is the open, royalty-free standard for cross-platform, parallel programming of diverse processors found in personal computers, servers, mobile devices and embedded platforms. OpenCL greatly improves the speed and responsiveness of a wide spectrum of applications in numerous market categories from gaming and entertainment to scientific and medical software.

All frequency filters can also be implemented in the spatial domain and, if there exists a simple kernel for the desired filter effect, it is computationally less expensive to perform the filtering in the spatial domain. Frequency filtering is more appropriate if no straightforward kernel can be found in the spatial domain, and may also be more efficient.

Image filtering is usually done in OpenCV platform, which is used for filtering, processing of an image OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code

Our main goal is to analyse the frequency domain of image filtering in parallel computing unit and serial computing unit. To arrive at the computational speed by comparing the execution time in serial and parallel computing units.

**1.2 Motivation**

With the advent of technology multi-core processors are dominating the sequential ones. This in a way has shifted the focus on increasing parallelism rather clock-rate. Likewise, other programming models like APP came into existence allowing software engineers to cultivate GPU computing applications without ardent knowledge graphical terms. Since, these frameworks had their unique method of applications development, it was quite inconvenient for software developers as they had to rebuilt everytime a new platform was launched. This resulted in the development of OpenCL by the Khronos group allowing parallel programming to be executed across heterogeneous platforms. Nevertheless, this raised the question of performance compromise which is frequently the case with these type of common languages.

**1.3 OBJECTIVES :**

* To familiarize the audience with the setting up of opencL framework in the Microsoft visual studio
* To familiarize the audience with the setting up of openCV framework in the Microsoft visual studio
* Linking the required OpenCV library functions to the visual studio project.
* Linking the required Open CL library functions to the visual studio project.
* Writing up of kernel program in the kernel builder.
* Writing up parallel program in the visual studio for the opencl platform and also serial program in the visual studio for the openCV platform.
* Analyzing the timecomplexity associated with the serial and parallel programming programs incolving filtering of the images.

**1.4 System requirements**

**Software requirements**

Intel kernel builder, Microsoft visual studio 2008 or more, Open CL library function

library like AMDAPPSDKSTREAM,Opencv version 2.4.0 or more

**Operating System**

OS: windows 7 or advanced version

**Hardware requirements**

Processor : Intel Processor

RAM :2GB or more

Hard Disk :500GB

**CHAPTER 2**

**METHODOLOGIES**

# 2.1 PROBLEM DEFINITION

The project described within this document is IMAGE FILTERING USING OPENCL FRAMEWORK designed with the specific goal of executing the images parallely. It is aimed at understanding the concepts of parallel computing using the OpenCL framework and OpenCV framework thereby comparing the time complexities in both the platforms.

**2.2 OpenCL platform**

|  |
| --- |
| OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems and handheld devices using a diverse mix of multi-core CPUs, GPUs, Cell-type architectures and other parallel processors such as DSPs. OpenCL is a framework for parallel programming and includes a language, API, libraries, and runtime system to support software development. Using OpenCL, a programmer can write general purpose programs that execute on GPUs without need to map their algorithms onto a 3D graphics API such as OpenGL.  The OpenCL architecture model use hierarchy of models such as platform Model, Memory Model, Execution Model, and Programming model. The important points are described below.  **Platform Model**   * The Platform model consists of a **host**connected to one or more **OpenCL devices**. * An OpenCL device is divided into one or more compute units (CUs), which are further divided into one or more **processing elements**(PEs). Computations on a device occur within the processing elements. * An OpenCL application runs on a host according to the models, particular to the host platform. * The OpenCL application submits **commands**from the host to execute computations on the processing elements within a device. * The processing elements within a compute unit execute a single stream of instructions as SIMD units or SPMD units.   [[http://www.cdac.in/index.aspx?id=ev_hpc_top1](http://www.cdac.in/index.aspx?id=ev_hpc_hypack13_m02_gpgpu_opencl#top)](http://www.cdac.in/index.aspx?id=ev_hpc_hypack13_m02_gpgpu_opencl#top)  **Execution model**   * Execution of an OpenCL program occurs in two parts: a **host program**that executes on the particular **host**platform and **kernels**that execute on one or more **OpenCL devices.** * The core of the OpenCL execution model is defined by the kernels execute. The concepts of kernel instance are called a **work-item** and these work-items are organised into **Work-groups.** * Execution Model: Context and Command Queues - The host defines a context for the execution of the kernels. The context includes   **Devices :**The collection of OpenCL devices to be used by the host;  **Kernels**The OpenCL functions that run on OpenCL devices);  **Program Objects:**The program source and executable that implement the kernels);  **Memory Objects :**A set of memory objects visible to the host and the OpenCL devices Memory objects contain values that can be operated on by instances of a kernel.   * The OpenCL execution model supports two categories of kernels: **OpenCL Kernels**& **Native Kernels.** |
| [[http://www.cdac.in/index.aspx?id=ev_hpc_top1](http://www.cdac.in/index.aspx?id=ev_hpc_hypack13_m02_gpgpu_opencl#top)](http://www.cdac.in/index.aspx?id=ev_hpc_hypack13_m02_gpgpu_opencl#top) |
| **Memory Model**   * Work-Item(s) executing a kernel have access to four distinct memory regions. **Global Memory**; **Constant Memory**; **Local Memory**; &**Private Memory**. In OpenCL, The kernel or the host can allocate from a memory region, which depends upon the type of allocation and the type of access allowed. * Table describes whether the kernel or the host from a memory region, the type of allocation (static i.e. compile time *versus*dynamic time i.e. runtime) and the type of access allowed i.e. whether the kernel or the host can read and/or write to a memory region. |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | **Global** | **Constant** | **Local** | **Private** | | Host | Dynamic Allocation   Read /Write Memory | Dynamic Allocation   Read /Write Memory | Dynamic Allocation   No | No   Allocation    No | | Kernel | No Allocation   Read /Write Memory | Static Allocation   Read only Memory | Static Allocation   Read / Write Memory | Static  Allocation    No | | |
| Table2.1: **Host and kernel allocation of access specifiers**   * The application running on the host uses the OpenCL API to create memory objects in global memory, and to enqueue memory commands (Refer OpenMP API specification) that operate on those memory objects. * OpenCL uses a relaxed consistency memory model: i.e the state of memory visible to a work item is not guaranteed to be consistent across the collection of work-items at all times. |
|  |

**2.3 Opencv design architecture**

OpenCV has interfaces for C, C++, and Python. The library was originally developed by Intel for C. In Ver. 2.0, its subsequent development resulted in its conversion into a C++ class library. In Ver. 2.2, its functionalities were subdivided and modularized. Below figure shows an outline of design architecture and shows the relationship among image and video libraries, graphical user interface (GUI) libraries, computation and processing libraries, and OpenCV modules.

The functionalities of the main modules are core: Definitions of class and templates, and Core functionalities imgproc: Image processing highgui: I/O of images, videos and imaging devices, and GUI ml: Machine Learning calib3d: Camera calibration and 3D reconstruction features2d: Feature extraction and descriptor extraction objdetect: Object detection video: video analysis flann:

Clustering and search multi-dimensional spaces gpu: General-Purpose computation on GPU legacy: Codes for compatibility In the highgui module, various image formats (e.g. JPEG, PNG, TIFF, JPEG-2000), video codecs and imaging devices (e.g. QTKit, VFW, videoInput, V4L), and GUI frameworks (e.g. Cocoa, Gtk+, Windows API, Qt) were implemented using libraries such as other open source software. Interfaces for images, videos, imaging devices, and GUIs were integrated during the implementation. Therefore, developers can benefit from a combined I/O and GUI without having to consider image formats, video codecs, camera drivers, and different operating systems.

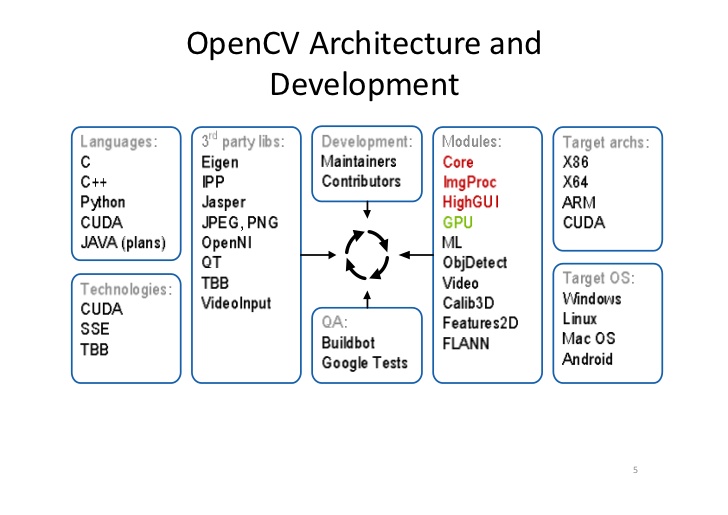


Figure 2.1: **OpenCV Architecture and Development**

**Opencv Structure**

OpenCV is broadly structured into five main components, four of which are shown in Figure 2.1. The CV component contains the basic image processing and higher-level computer vision algorithms; ML is the machine learning library, which includes many statistical classifi ers and clustering tools. HighGUI contains I/O routines and functions for storing and loading video and images, and CXCore contains the basic data structures and content.

.

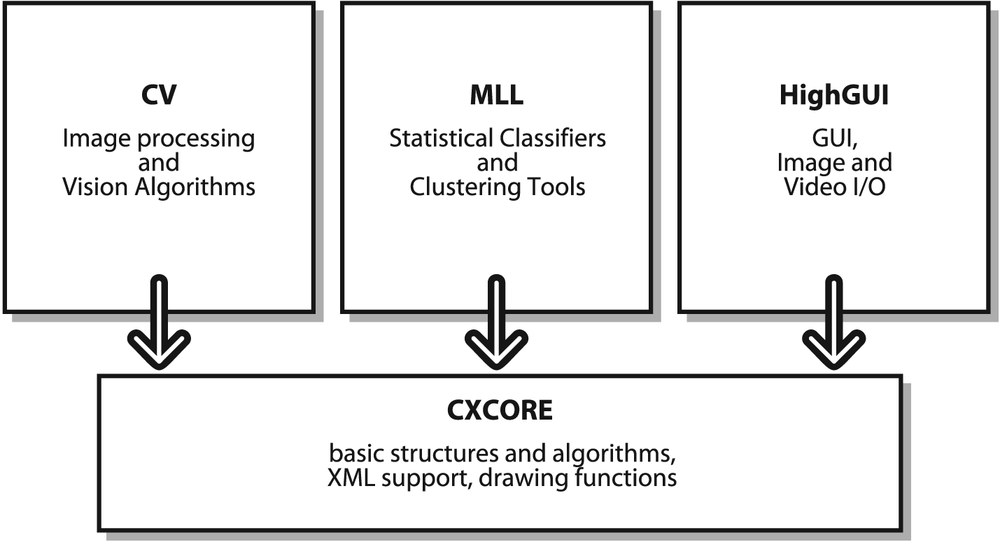


Figure 2.2**: OpenCV Structure**

**CHAPTER 3**

**IMAGE FILTERING USING FFT**

**3.1 Image Filter**

A software routine that changes the appearance of an image or part of an image by altering the shades and colours of the pixels in some manner. Filters are used to increase the brightness and contrast as well as to add a wide variety of textures, tones,and special effects to a picture.

There are so many filters are available .Here Frequency domain is considered since the number of calculation involved in frequency domain considered is less compared to other types.In frequency domain filter a signal can be converted from time domain into frequency domain using mathematical operators called Transformers.

**Different Transformations**

Fourier Series

Fourier Transformations

Laplace Transform

Z Transforms

Since here the main problem is dealing with the band filters of an image ,fourier transform is considered.

**3.2 Fourier Transform**

Fourier Transform is a process that takes in samples of data, and outputs its frequency content

The mathematical formula for fourier transform looks like

https://www.fixstars.com/images/openclbook/dtp_462724_USER_CONTENT_0_html_m6138caeb.gif

The "i" is an imaginary number, and ω is the frequency in radians.

The Fourier Transform operates on continuous data. However, continuous data means it contains infinite number of points with infinite precision. For this processing to be practical, it must be able to process a data set that contains a finite number of elements. Therefore, a process known as the Discrete Fourier Transform (DFT) was developed to estimate the Fourier Transform, which operates on a finite data set. The mathematical formula is shown below.

https://www.fixstars.com/images/openclbook/dtp_462724_USER_CONTENT_0_html_4dd61be3.gif

This formula now allows processing of digital data with a finite number of samples. The problem with this method, however, is that its O(N^2). As the number of points is increased, the processing time grows a power of 2.

**3.3 FAST FOURIER TRANSFORM**

An optimized implementation of DFT is Fast Fourier Transform .FFT algorithm computes the discrete fourier transform of a sequence or its inverse.An FFt rapidly computes such transformation by factorizing the DFT matrix into a product of sparse.In such a way it manages to reduce the time complexity from O(n^2) to O(n log n).

There are different algorithms used for the implementation of FFT.Some of them are Cooley-Tukey algorithm, prime Factor algorithm, Rader’s algorithm and Bluestein’s algorithm.

The core computation of FFT algorithm is Butterfly Operation. The operation is performed on a pair of data samples at a time, whose signal flow graph is shown in figure 3.1

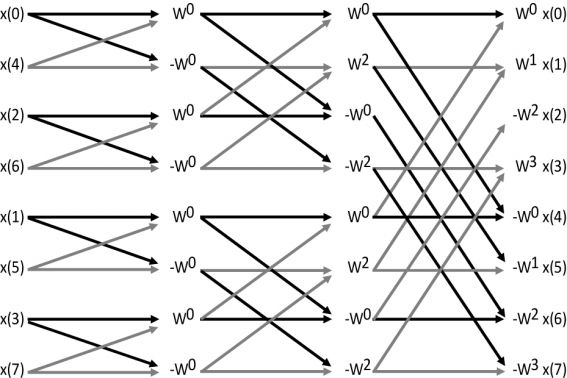


Figure 3.1: **Signal flow graph of butterfly operation**

Considering the butterfly of size-2 it takes the 2 inputs (x0,x1) and it gives two outputs (y0,y1)

Y0=x0+x1

Y1=x0-x1

If we take radix-2 decimation-in-time FFT algorithm on *n* = 2*p* inputs with respect to a primitive *n*-th root of unity \omega^k_n = e^{-\frac{2\pi i k}{n}} r elies on O(*n* log *n*) butterflies of the form:

y_0 = x_0 + x_1 \omega^k_n \, 

y_1 = x_0 - x_1 \omega^k_n, \, 

where *k* is an integer depending on the part of the transform being computed. Whereas the corresponding inverse transform can mathematically be performed by replacing *ω* with *ω*−1 (and possibly multiplying by an overall scale factor, depending on the normalization convention), one may also directly invert the butterflies

x_0 = \frac{1}{2} (y_0 + y_1) \, 

x_1 = \frac{\omega^{-k}_n}{2} (y_0 - y_1), \, 

**Flow diagram of FFT**

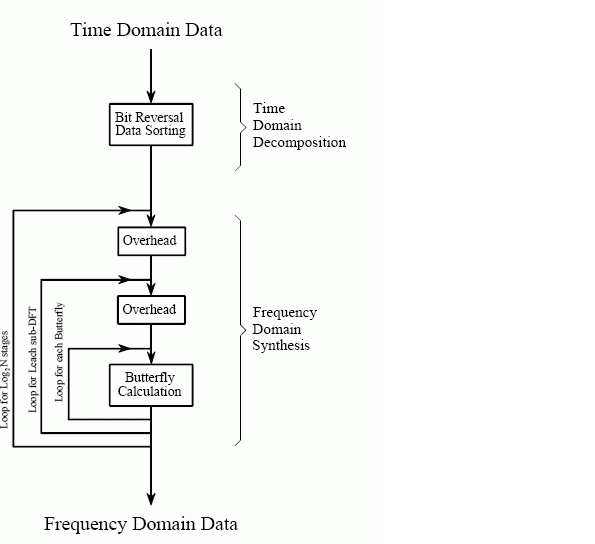
****

Figure 3.2**:Flow diagram of FFT**

The time domain decomposition is accomplished with a bit reversal sorting algorithm. Transforming the composed data into the frequency domain involves nothing and therefore does not appears in the figure 3.2

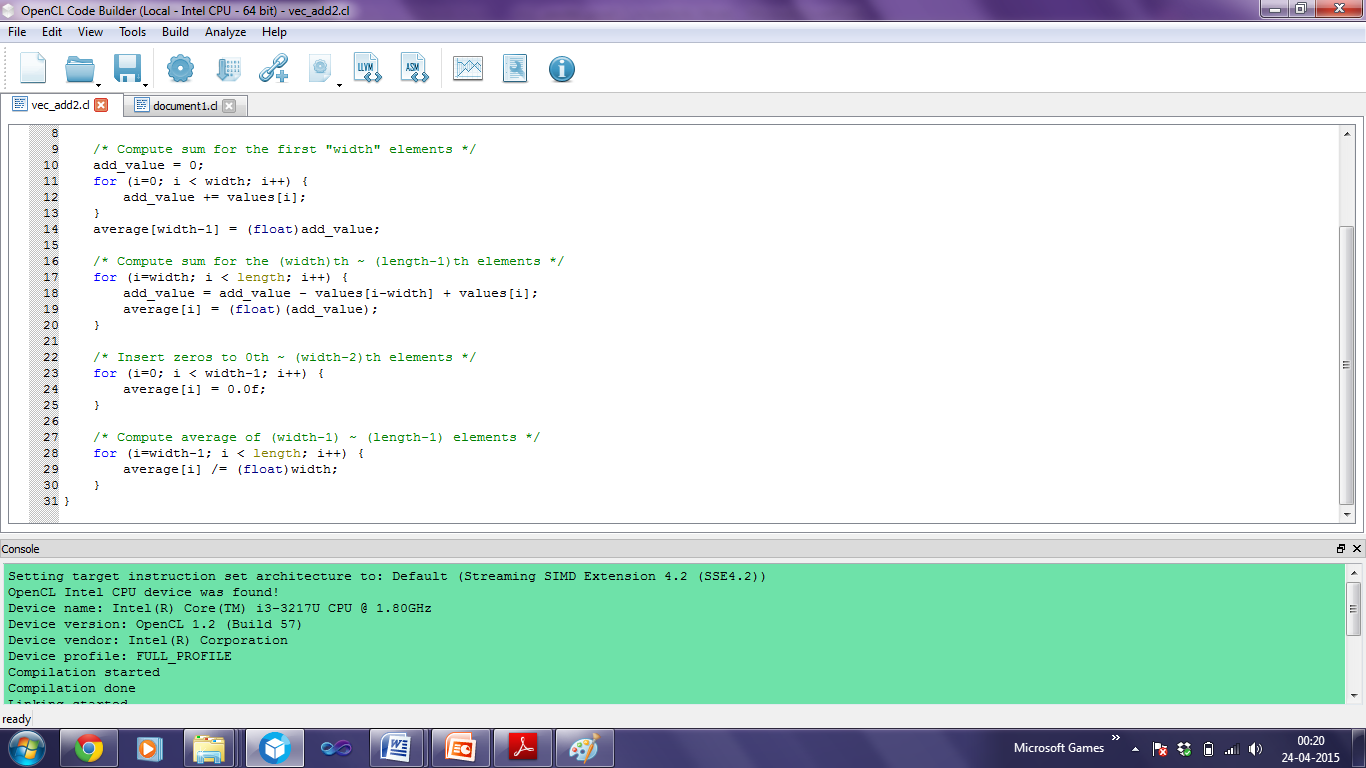
The frequency domain synthesis requires three loops.The outer loop runs through the log n to base 2 stages.The middle loops moves through each of the individual frequency spectra in the stage being worked on.The inner most loop uses the butterfly to calculate the points in each frequency spectra.The overhead box in figure determine the beginning and ending indesex for the loops as well as calculating the sinusoids needed in the butterflies.

**CHAPTER 4**

**CONFIGURATION**

Here the platform consist of coding part of two phases. one is kernel code builder where we write kernel code and we link that kernel code with the main program in visual studio.

**4.1 OpenCL Kernel builder platform**



The OpenCL kernel builder gives the information about the Processor. The console of the kernel builder must appear in green since it indicates the compilation of the builder is successful. The kernel builder also gives information about the Device name ,Device version,Device vendor.

After compiling the program in the kernel builder link the program with the binary code as there will be no linkage between compiled code and the linked code.

For Example **: average of a input values is to be found**

Here for finding the average ,average kernel is initialised

\_\_kernel void average(\_\_global int \*values, \_\_global float \*average, int length, int width)

Width is used to Compute sum for the first "width" elements and than Compute the sum for the (width)th ~ (length-1)th elements

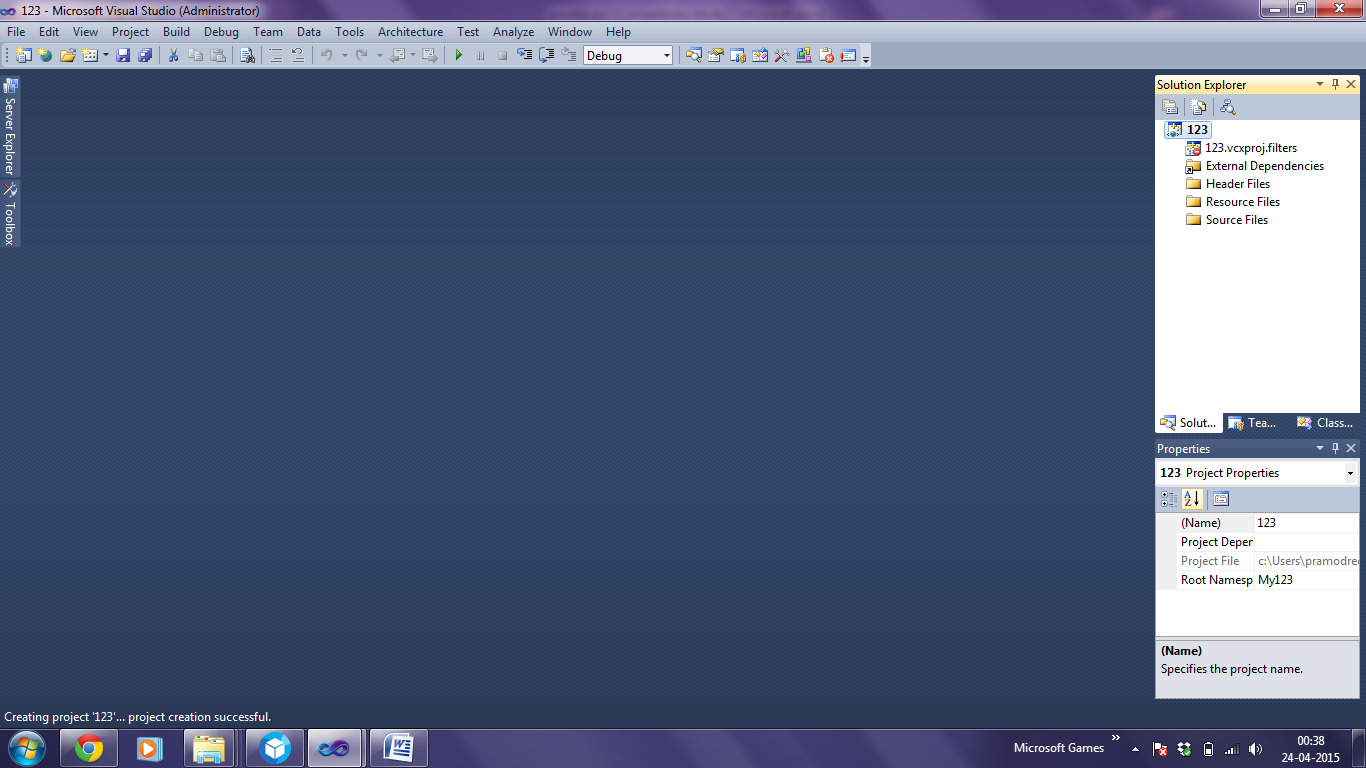
Insert zeros upto (width-1) elements and compute average of (width-1) ~ (length-1) elements and it is stored in average

**4.2 Visual studio setup**

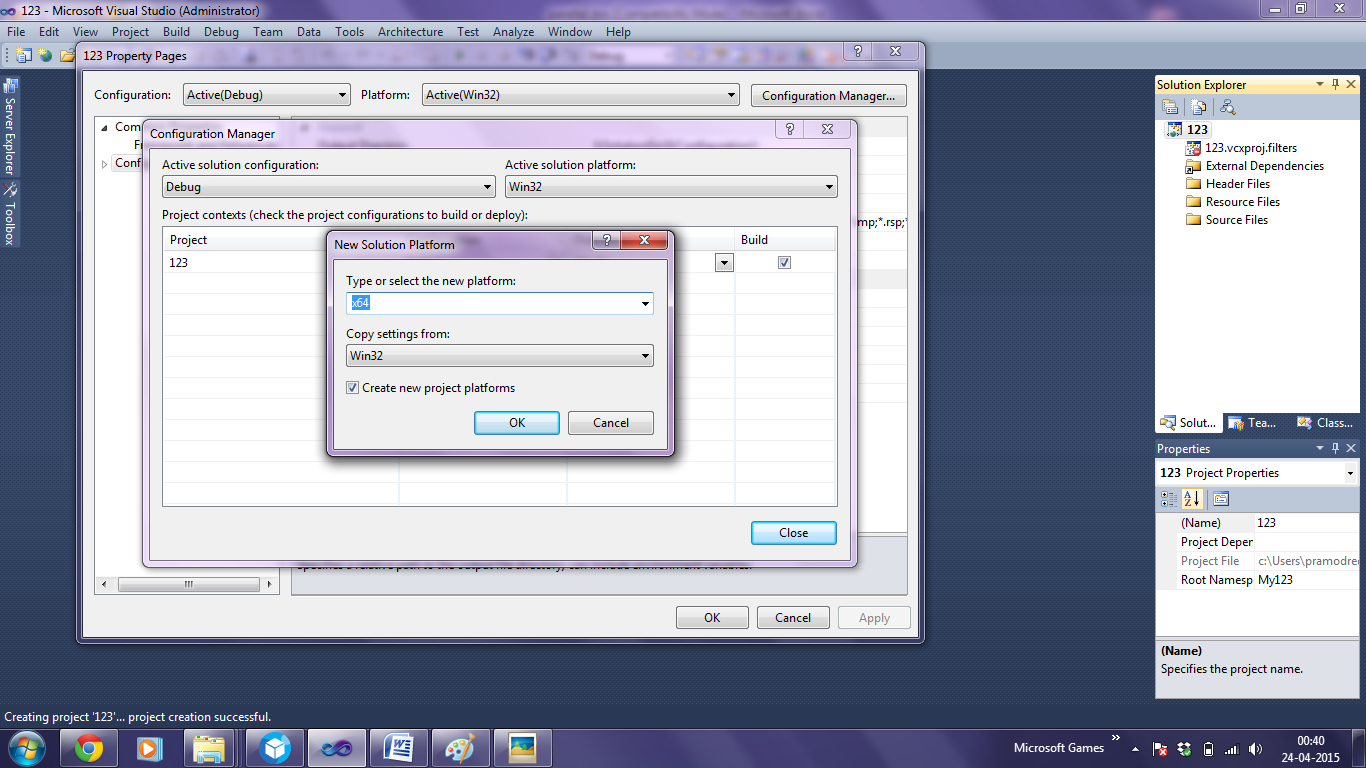
In visual studio we write main program and the is linked with the kernel builder opencl code from kernel builder

The following steps describe the setup of opencl code execution in visual studio.

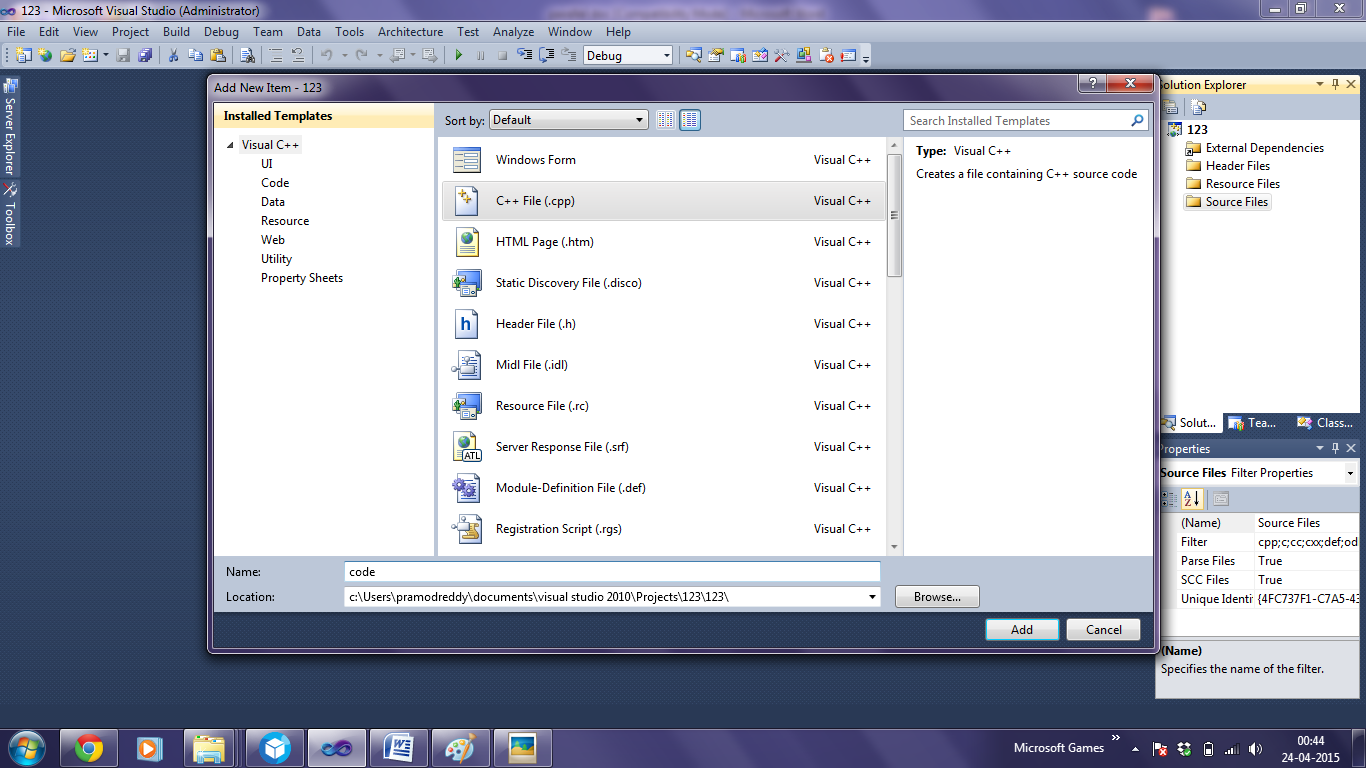
Open ->new project->win 32 console application->give project name->press next->under additional options select empty project->click finish



Under properties->configuration manager ->select configuration properties->active solution platforms click new->under type or select the new platform select x64 press enter

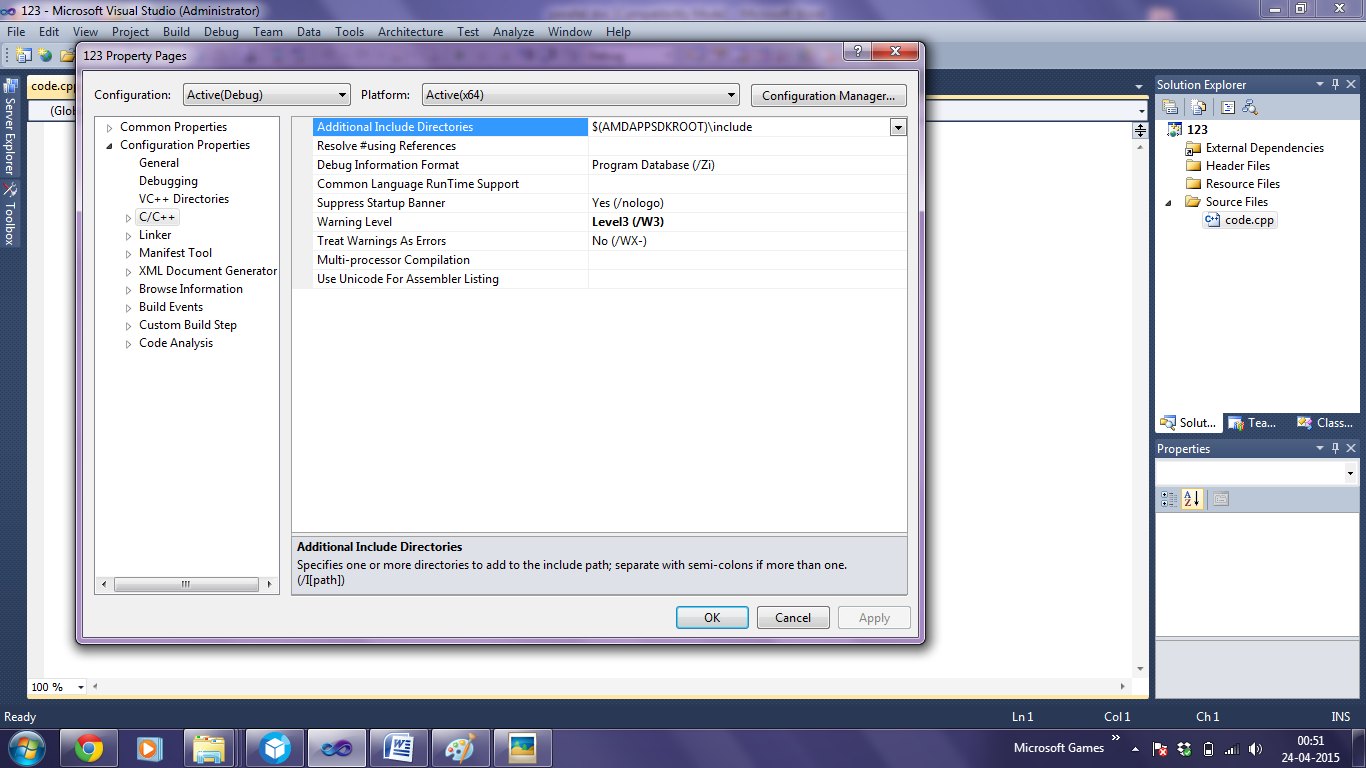


Goto solution explorer->select source files->add items->click add new item->select c++ file and give name for main code ->click add

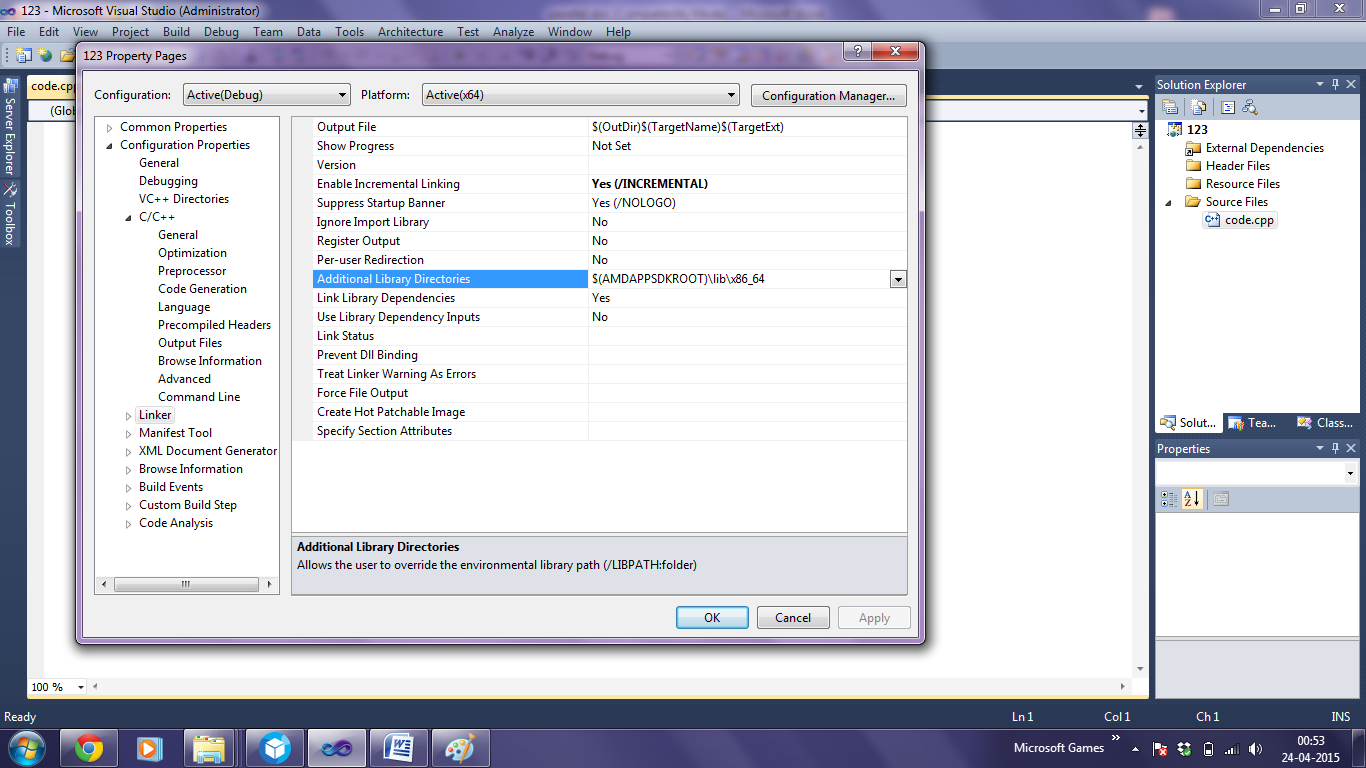


Under properties->select general->under additional include directoties->type $(AMDAPPSDKROOT)\include

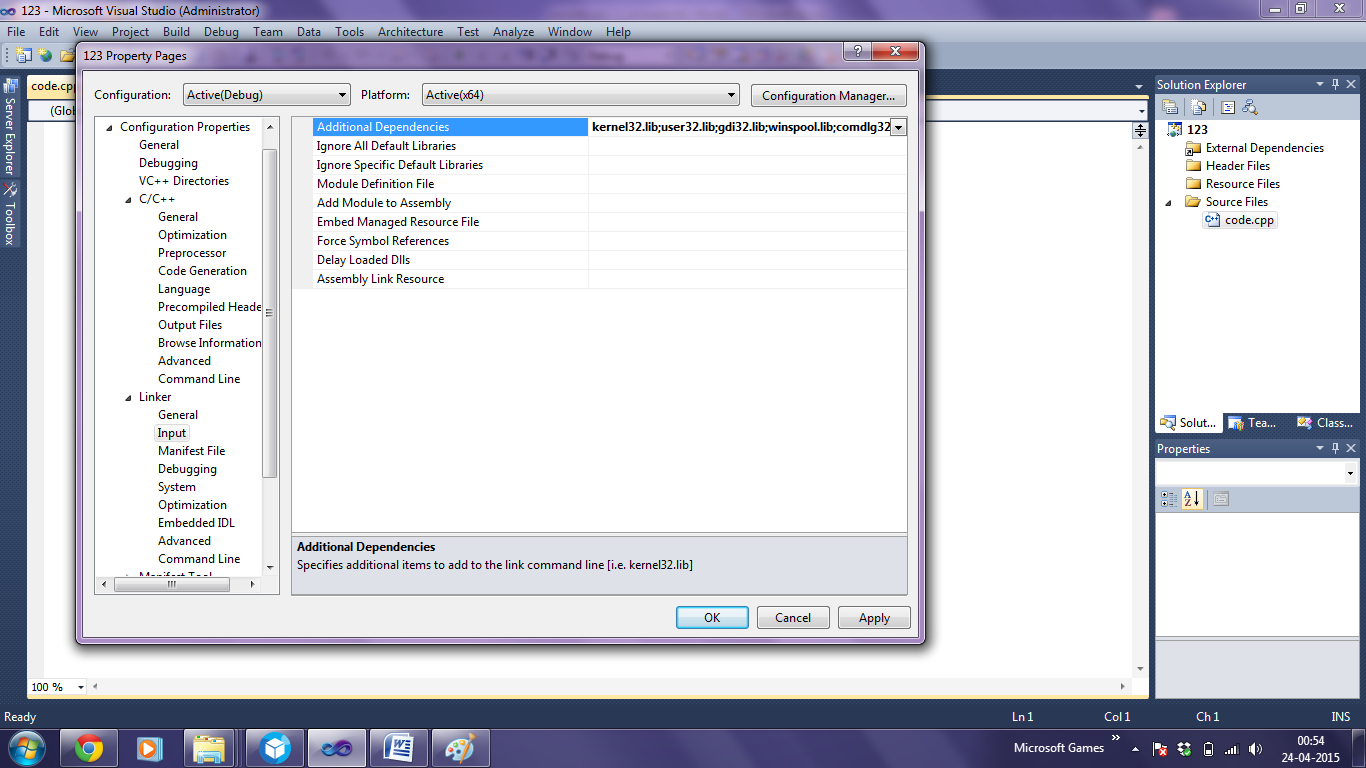
This is to add open cl directory to the visual studio



Under properties->select linker->unde general ->in the blank additional dependencies type $(AMDAPPSDKROOT)\lib\x86\_64



Under properties->select linker->input->add extension to the additional dependencies OpenCl.lib



Thus the environment is set up in the visual studio .now it is ready to execute open CL programs in the visual studio

**4.3 Setup of OpenCV (2.4.9) platform**

**To set opencv path**

Go to computer>system properties>advanced system settings>environment variables>path>edit>add

E:\project\softwares\opencv\build\x64\vc11\bin

Note: Since extracted opencv version in that location

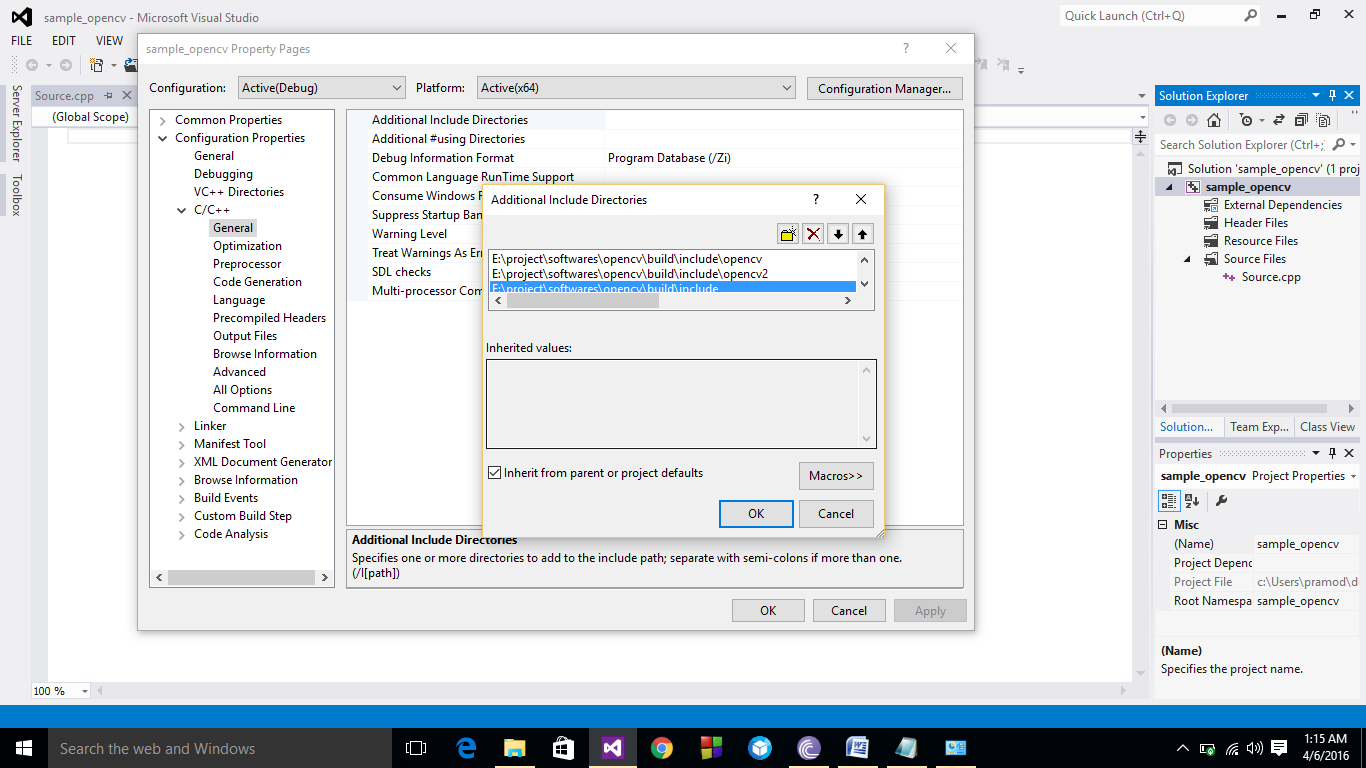
**To add library functions to visual studio**

Go to Properties>c/c++>general>additional include directories>edit>add

E:\project\softwares\opencv\build\include

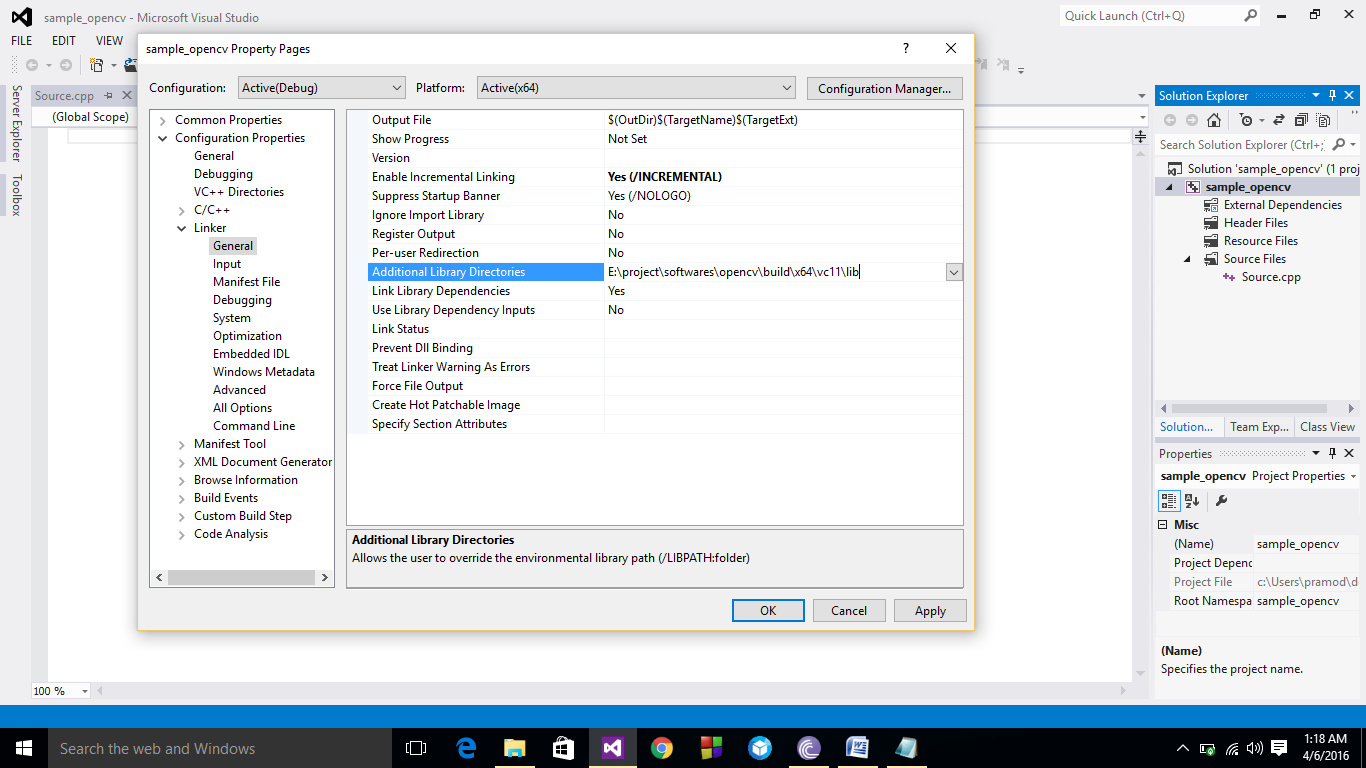
E:\project\softwares\opencv\build\include\opencv

E:\project\softwares\opencv\build\include\opencv2



Go to>properties>linker>general>additional library directories>add

E:\project\softwares\opencv\build\x64\vc11\lib



Go to>properties>linker>input>additioal dependencies>add

opencv\_calib3d249d.lib

opencv\_contrib249d.lib

opencv\_core249d.lib

opencv\_features2d249d.lib

opencv\_flann249d.lib

opencv\_gpu249d.lib

opencv\_highgui249d.lib

opencv\_imgproc249d.lib

opencv\_legacy249d.lib

opencv\_ml249d.lib

opencv\_nonfree249d.lib

opencv\_objdetect249d.lib

opencv\_ocl249d.lib

opencv\_photo249d.lib

opencv\_stitching249d.lib

opencv\_superres249d.lib

opencv\_ts249d.lib

opencv\_video249d.lib

opencv\_videostab249d.lib

opencv\_calib3d249.lib

opencv\_contrib249.lib

opencv\_core249.lib

opencv\_features2d249.lib

opencv\_flann249.lib

opencv\_gpu249.lib

opencv\_highgui249.lib

opencv\_imgproc249.lib

opencv\_legacy249.lib

opencv\_ml249.lib

opencv\_nonfree249.lib

opencv\_objdetect249.lib

opencv\_ocl249.lib

opencv\_photo249.lib

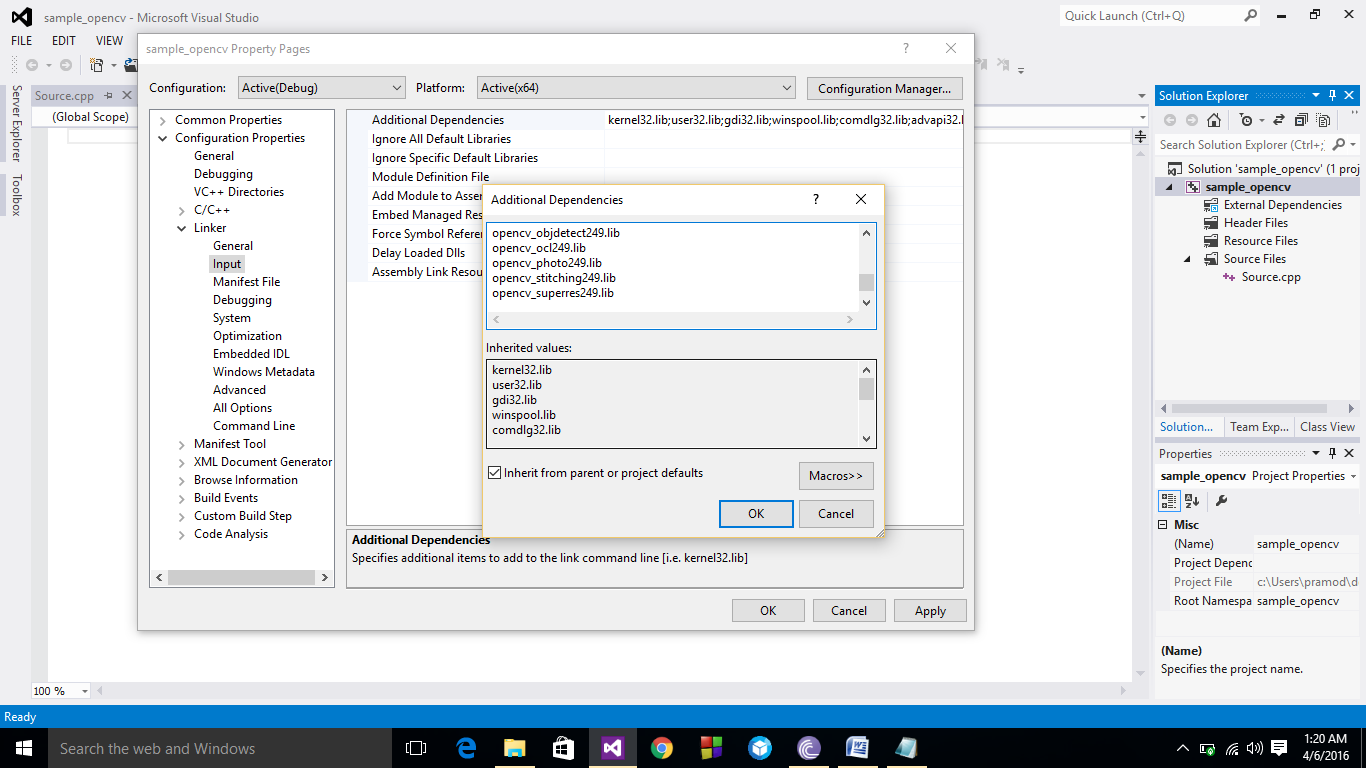
opencv\_stitching249.lib

opencv\_superres249.lib

opencv\_ts249.lib

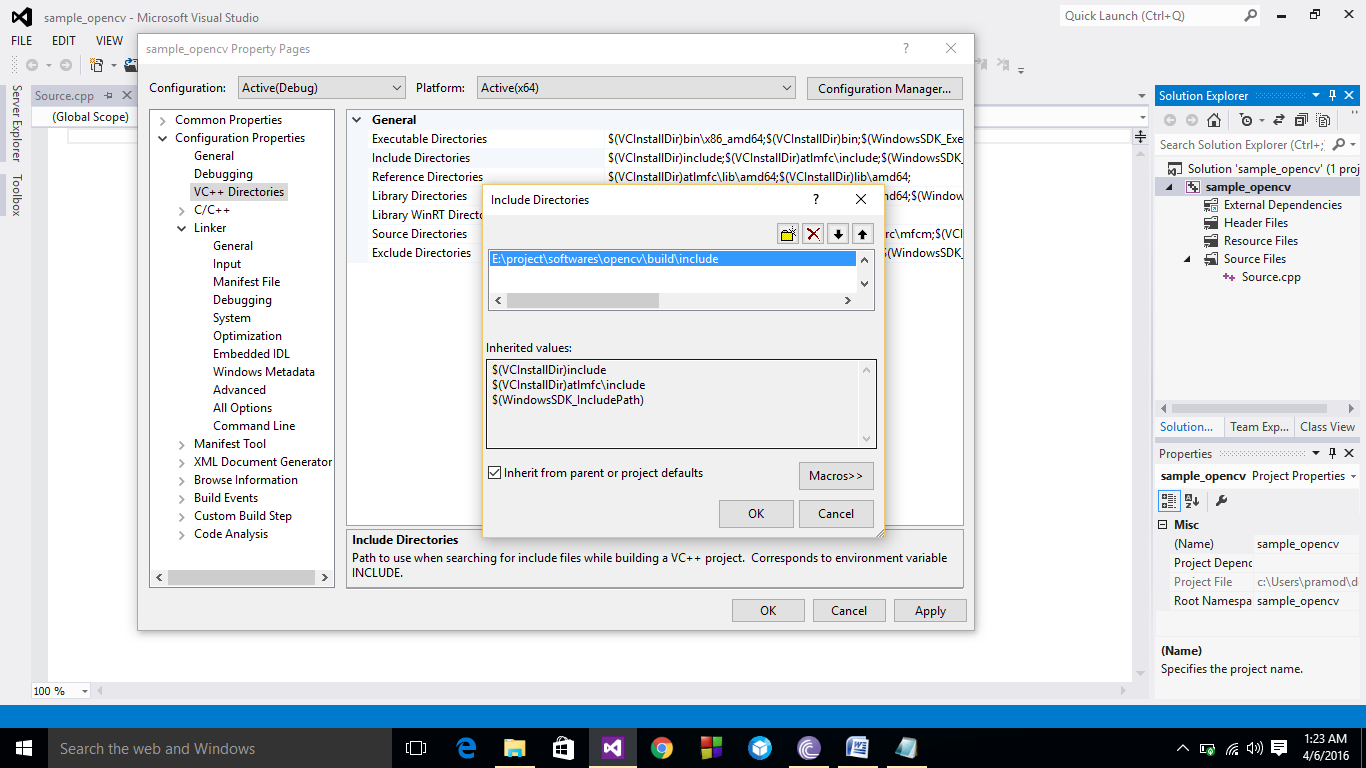
opencv\_video249.lib

opencv\_videostab249.lib



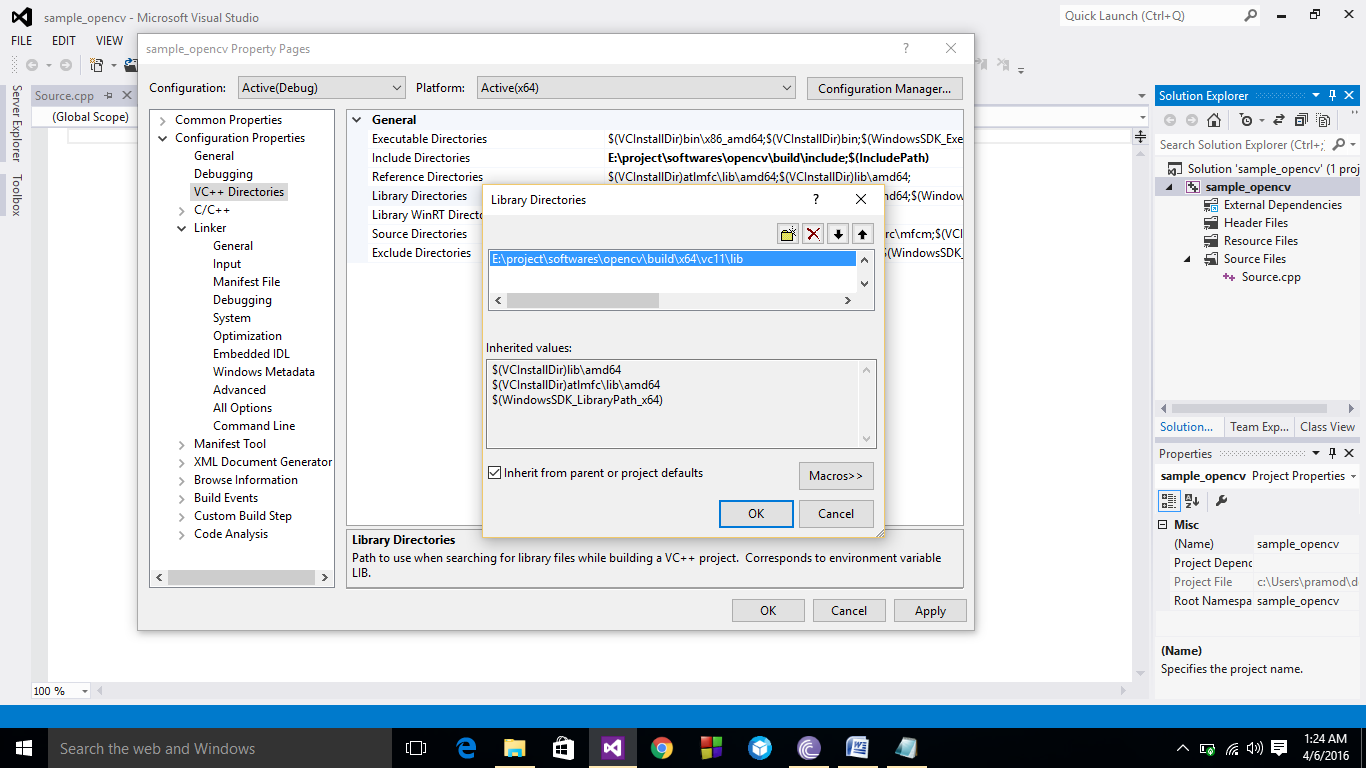
Go to>properties>vc++ Directories>include directories>edit>add

E:\project\softwares\opencv\build\include



Go to>properties>vc++ Directories>library directories>edit>add

E:\project\softwares\opencv\build\x64\vc11\lib



**Note:**All above are configured in x64 bit platform

**Chapter 5**

**IMPLEMENTATION**

**5.1 IMPLEMENTATION OF FFT USING OPENCL**

OpenCL consists of host code and kernel code

FFT consist of mainly 6 parts

1) Spin Factor

2) Bit Reversal

3) Normalization

4) Butterfly Operations

5) Matrix transformation

6) Filter(edge Detection)

We assign each kernel to each part and we link it to host code .

**Spin Factor**

1. \_\_kernel void spinFact(\_\_global float2\* w, int n)
2. {
3. unsigned int i = get\_global\_id(0);
5. float2 angle = (float2)(2\*i\*PI/(float)n,(2\*i\*PI/(float)n)+PI\_2);
6. w[i] = cos(angle);
7. }

The code is used to pre-compute the value of the spin factor "w", which gets repeatedly used in the butterfly operation. The "w" is computed for radian angles that are multiples of (2π/n), which is basically the real and imaginary components on the unit circle using cos() and -sin().

**Bit reversing**

1. \_\_kernel void bitReverse(\_\_global float2 \*dst, \_\_global float2 \*src, int m, int n)
2. {
3. unsigned int gid = get\_global\_id(0);
4. unsigned int nid = get\_global\_id(1);
6. unsigned int j = gid;
7. j = (j & 0x55555555) << 1 | (j & 0xAAAAAAAA) >> 1;
8. j = (j & 0x33333333) << 2 | (j & 0xCCCCCCCC) >> 2;
9. j = (j & 0x0F0F0F0F) << 4 | (j & 0xF0F0F0F0) >> 4;
10. j = (j & 0x00FF00FF) << 8 | (j & 0xFF00FF00) >> 8;
11. j = (j & 0x0000FFFF) << 16 | (j & 0xFFFF0000) >> 16;
13. j >>= (32-m);
15. dst[nid\*n+j] = src[nid\*n+gid];
16. }

shows the kernel code for reordering the input data such that it is in the order of the bit-reversed index Also, note that a separate memory space must be allocated for the output on the global memory. These types of functions are known as an out-of-place function. This is done since the coherence of the data cannot be guaranteed if the input gets overwritten each time after processing.

**Normalisation**

1. \_\_kernel void norm(\_\_global float2 \*x, int n)
2. {
3. unsigned int gid = get\_global\_id(0);
4. unsigned int nid = get\_global\_id(1);
6. x[nid\*n+gid] = x[nid\*n+gid] / (float2)((float)n, (float)n);
7. }

The code in List should be self-explanatory. It basically just dives the input by the value of "n". The operation is performed on a float2 type. Since the value of "n" is limited to a power of 2, division by shifting is only possible for integer types. Shifting of a float value will result in unwanted results.

.

**Butterfly operation**

1. \_\_kernel void butterfly(\_\_global float2 \*x, \_\_global float2\* w, int m, int n, int iter, uint flag)
2. {
3. unsigned int gid = get\_global\_id(0);
4. unsigned int nid = get\_global\_id(1);
6. int butterflySize = 1 << (iter-1);
7. int butterflyGrpDist = 1 << iter;
8. int butterflyGrpNum = n >> iter;
9. int butterflyGrpBase = (gid >> (iter-1))\*(butterflyGrpDist);
10. int butterflyGrpOffset = gid & (butterflySize-1);
12. int a = nid \* n + butterflyGrpBase + butterflyGrpOffset;
13. int b = a + butterflySize;
15. int l = butterflyGrpNum \* butterflyGrpOffset;
17. float2 xa, xb, xbxx, xbyy, wab, wayx, wbyx, resa, resb;
19. xa = x[a];
20. xb = x[b];
21. xbxx = xb.xx;
22. xbyy = xb.yy;
24. wab = as\_float2(as\_uint2(w[l]) ^ (uint2)(0x0, flag));
25. wayx = as\_float2(as\_uint2(wab.yx) ^ (uint2)(0x80000000, 0x0));
26. wbyx = as\_float2(as\_uint2(wab.yx) ^ (uint2)(0x0, 0x80000000));
28. resa = xa + xbxx\*wab + xbyy\*wayx;
29. resb = xa - xbxx\*wab + xbyy\*wbyx;
31. x[a] = resa;
32. x[b] = resb;
33. }

The kernel for the butterfly operation, which performs the core of the FFT algorithm, is shown in above. Each work item performs one butterfly operation for a pair of inputs. Therefore, (n \* n)/2 work items are required.

The variable butterfly Size represents the difference in the indices to the data for the butterfly operation to be performed on. The butterfly Size is 1 for the first iteration, and this value is doubled for each iteration.

Next, we need to know how the butterfly operation is grouped. Looking at the signal flow graph. In the first iteration, the number of groups is the same as the number of butterfly operation to perform, but in the 2nd iteration, it is split up into 2 groups. This value is stored in the variable butterfly Group Num.

The differences of the indices between the groups are required as well. This is stored in the variable butterfly Group Distance.

Next, determine the indices to read from and to write to. The butterfly Grp Base variable contains the index to the first butterfly operation within the group. The butterfly Group Offset is the offset within the group. These are determined using the following formulas.

**MATRIX TRANPOSE**

Here the columns are changed to rows and rows are changed to coloumns

The kernel code looks like

1. \_\_kernel void transpose(\_\_global float2 \*dst, \_\_global float2\* src, int n)
2. {
3. unsigned int xgid = get\_global\_id(0);
4. unsigned int ygid = get\_global\_id(1);
6. unsigned int iid = ygid \* n + xgid;
7. unsigned int oid = xgid \* n + ygid;
9. dst[oid] = src[iid];
10. }

**Filtering**

1. \_\_kernel void highPassFilter(\_\_global float2\* image, int n, int radius)
2. {
3. unsigned int xgid = get\_global\_id(0);
4. unsigned int ygid = get\_global\_id(1);
6. int2 n\_2 = (int2)(n>>1, n>>1);
7. int2 mask = (int2)(n-1, n-1);
9. int2 gid = ((int2)(xgid, ygid) + n\_2) & mask;
11. int2 diff = n\_2 - gid;
12. int2 diff2 = diff \* diff;
13. int dist2 = diff2.x + diff2.y;
15. int2 window;
17. if (dist2 < radius\*radius) {
18. window = (int2)(0L, 0L);
19. } else {
20. window = (int2)(-1L, -1L);
21. }
22. 900 7:00:00 AM
23. image[ygid\*n+xgid] = as\_float2(as\_int2(image[ygid\*n+xgid]) & window);
24. }

The spatial frequency obtained from the 2-D FFT shows the DC (direct current) component on the 4 edges of the XY coordinate system. A high pass filter can be created by cutting the frequency within a specified radius that includes these DC components. The opposite can be performed to create a low pass filter. In general, a high pass filter extracts the edges, and a low pass filter blurs the image.

This kernel is linked to each function in host code which can be implemented

**5.2 OpenCV Implementation of FFT**

In OpenCV platform Convolution and Deconvolution test is applied for performing the FFT

**Convolution**

Convolution theorem for continuous case: h(t) and g(t) are two functions and H(f) and G(f) are their corresponding Fourier Transform, then convolution is defined as Where g\*h is in time domain and then convolution theorem is given by

Fourier transform of the convolution is just the product of the individual Fourier transform. Convolution of two function is equal to the their convolution in opposite order

s\*r =r\*s .

If s(t) is function represents signal then r(t) is response function and their convolution smears the signal s(t) in time according to response function r(t).Discrete case:If signal s(t) is represented by its sampled values at equal time interval sj , rk is discrete set of numbers corresponds to response function then rk tells what multiple of the input signal is copied into identical output channel.Therefore, the discrete convolution with response function of finite duration M is given by

Capture1.PNG

If response function is non-zero in some range where M is very large even integer, then response function is called as finite impulse response (FIR)

**The discrete convolution theorem**

* If signal sj is periodic with period N, then its discrete convolution with response function of finite duration N is member of the discrete Fourier transform pair

Capture1.PNG

**Deconvolution**

Deconvolution is the process of undoing the smearing in a data set that has occurred under the influence of a known response function, for example, because of the known effect of a less-than-perfect measuring apparatus. The defining equation of deconvolution is the same as that for convolution, except now the left-hand side is taken to be known, is to be considered as a set of N linear equations for the unknown quantities. Solving these simultaneous linear equations in the time domain of is unrealistic in most cases, but the FFT renders the problem almost trivial. Instead of multiplying the transform of the signal and response to get the transform of the convolution, we just divide the transform of the (known) convolution by the transform of the response to get the transform of the deconvolved signal. This procedure can go wrong mathematically if the transform of the response function is exactly zero for some value Rn, so that we can’t divide by it. This indicates that the original convolution has truly lost all information at that one frequency, so that a reconstruction of that frequency component is not possible. You should be aware, however, that apart from mathematical problems, the process of deconvolution has other practical shortcomings. The process is generally quite sensitive to noise in the input data, and to the accuracy to which the response function rk is known. Perfectly reasonable attempts at deconvolution can sometimes produce nonsense for these reasons.

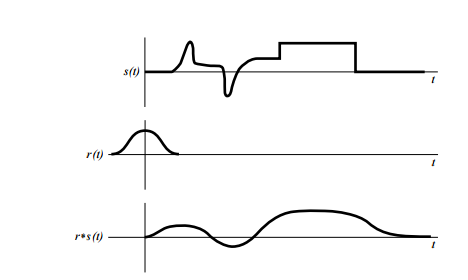


Figure 5.1:**Example of the deconvolution of two functions**

A signal s(t) is convolved with a response function r(t). Since the response function is broader than some features in the original signal, these are “washed out” in the convolution. In the absence of any additional noise, the process can be reversed by deconvolution

**CHAPTER 6**

**RESULTS**

**6.1Image comparison results**

Input image for OpenCL platform is only pgm image.

The sample input image is



Figure 6.1**: Input image OpenCL**

The sample output image is

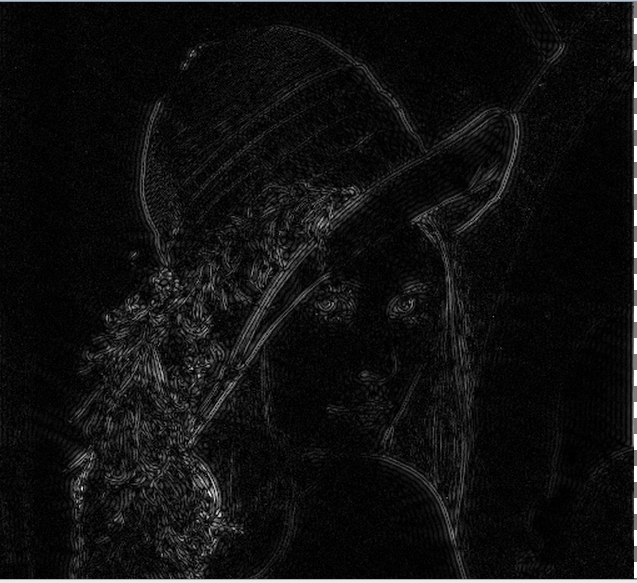


Figure 6.2: **Output image OpenCL**

The sample input image of OpenCV is



Figure 6.3:**Input image for OpenCV**

The sample output image of Opencv is



Figure 6.4:**Output image of OpenCV**

**6.2 Time comparison results**

|  |  |  |  |
| --- | --- | --- | --- |
| Size of image in pixels | OpenCL CPU(max 4 units) | OpenCL GPU(max 16 units) | OpenCV |
| 512\*512 | 1.019 | 0.596 | 1.347 |
| 256\*256 | 0.894 | 0.422 | 0.491 |
| 128\*128 | 0.847 | 0.389 | 0.250 |
| 64\*64 | 0.843 | 0.387 | 0.187 |

Table 6.1: **Results of performance comparison of OpenCL and OpenCV**.

NOTE: Time complexity of opencl can be reduced by changing work group size and also by using no of units.The performance increases significantly by increasing computational units.

NOTE2:The image size of 1024\*1024 pix is not supported in this case as image format accepted by the max GPU unit is only 512\*512 pix.

By comparing the time between different image sizes, the time significantly decreases between CPU and GPU of OpenCL units as the no of compute units is increasing. As the image size decreases the time taken for execution in OpenCV decreases as the size of image becomes lesser, this is due to the fact that execution of kernel function consume some amount of time. so as image size decreases it is consuming significant time.

**CHAPTER 7**

**FUTURE WORK**

**7.1 CONCLUSION**

From the result we can infer that by using OpenCL platform the time complexity of the program has been exponentially decreased with the increase of compute units and also with changing the platform CPU and GPU. Though the code looks clumsy and allocation of functions for kernels is difficult one can learn them with little bit of effort. With the advancement of technologies the no of computing units has been increasing so processing speed of the execution increases. This parallel computing language can be used for execution of program within milliseconds.

**7.2 Future enhancements**

OpenCL is essentially a subset of C (actually, C99) with some additional extensions. As a result, it is a very familiar working environment for developers comfortable with programming in C. It uses a host/device model in which a C program running on a host distributes tasks to [processing](http://signal-processing.mil-embedded.com/topics/processing) elements on one or more devices and manages memory, data movements, and error handling. The calculations to be performed on the target device are written in OpenCL in what is called a kernel, which can be thought of as similar to a C function. At runtime multiple kernel programs are launched (called work-items) on the target device, and each kernel performs its calculations on a subset of data defined by the host program. This set of work-items is called a work-group, and the data can be segmented into 1D, 2D, or 3D grids for distribution to the individual kernels. Work-groups, in turn, operate independently from one another. OpenCL also has a hierarchical memory model consisting of global, local, constant, and private memories.

This model of memory hierarchies and executable kernels is what makes OpenCL a powerful tool for expressing both data- and task-paralellism. It is also what makes OpenCL useful for heterogeneous system programming as different types of targets can be grouped as individual work-groups. This power comes at a cost, however.

One argument made against OpenCL is that it is seen as too verbose – that is, many functions need to be called in setting up a device execution and cleaning up afterward. This is arguably not a serious criticism – parallel computing is, by its nature, a complex task. Different types of applications require different approaches to parallelism, different synchronization, etc. To capture these differences one needs a rich language. Besides, OpenCL code is inherently easier to create, read, and understand (and thus maintain) than any HDL or DSP assembly language code.

OpenCL is not a perfect solution, however. There are significant shortcomings that have the potential to either limit its adoption for FPGAs and DSPs, or completely kill it. The first problem is the host/device nature of OpenCL. Many applications for which DSPs and FPGAs are targeted do not lend themselves to host/device task acceleration development models – they require data flow models which do work on streams of data, continuously processing as the data arrives and pushing the results out for later-stage processing or storage. OpenCL does not handle this model well, although Altera is working with The Khronos Group to address this in a future release.

OpenCL takes a lot of the work of task distribution and data movements off the shoulders of developers. It does not, however, manage everything. As was mentioned previously, OpenCL’s very explicit memory heirarchy is one of its strengths. It allows the developer to specify in very specific terms what memories are to be used for various data storage. Unfortunately, it may not always be obvious which of the four memory types (Global, Local, Constant, or Private) will give the application the best performance. In addition, care must be given to insure that shared memories are properly semaphore controlled, as race conditions are still a very real possibility (and like in legacy development environments, often very hard to detect). What this means is that the developer must have a good understanding of the architecture of the target platform and must code explicitly to that architecture in order to get good performance. This also means that OpenCL is not a write-once-run-anywhere sort of language – porting to different platforms and new architectures could be challenging.

Lastly, OpenCL’s latest release is version 2.0 (following 1.0, 1.1, and 1.2) while TI only supports 1.1 and Altera only 1.0 (albeit with some limited later feature support in both cases). This means that many of the latest features are not available for developers on these platforms. While many of these features may not make sense for DSP or FPGA platform development, there are undoubtedly many important and useful features that do. In addition, it can be frustrating working with a tool with incomplete support. Given the general popularity of OpenCL and the rapid pace of its development, both TI and Altera will probably be playing catch-up for some time.

Probably the biggest factor that could make OpenCL a widely adopted platform for DSP and FPGA system development is, surprisingly, the community excitement around GPUs for DSP programming. As more and more people learn OpenCL and use it for real-world problems, the strengths and weakness of the language will become more apparent and new techniques for solving problems will be developed. In time people will need to turn from GPUs to DSPs and FPGAs to gain performance/watt advantages and will rely on OpenCL to make that possible.

**REFERENCES**

[1] Website: <https://www.fixstars.com/en/opencl/book/OpenCLProgrammingBook>

[2] Website:

<https://www.khronos.org/opencl> reference cards and documentation

[3] OpenCL programming by example by Ravishekhar Banger and Koushik Bhattacharya

# [4] Accelerating computer vision algorithms using OpenCL framework on the mobile GPU - A case study by

# G.Wang ,ECE dept Rice university ,Houston,Texas,USA J.Yun and Y.Xiong

# [5] High speed image space parallel processing for computer-generated integral imaging system by

Ki-Chul Kwon, Chan Park, Munkh-Uchral Erdenebat, Ji-Seong Jeong, Jeong-Hun Choi, Nam Kim, Jae-Hyeung Park, Young-Tae Lim, and Kwan-Hee Yoo

## [6] Point to point processing of digital images using parallel computing by

## Eric Olmedo, Jorge de la Calleja, Antonio Benitez, Ma. Auxilio Medina in

International Journal of Computer Science Issues (IJCSI), Vol. 9, Issue 3, No 3, 2012

**APPENDIX**

**OpenCL host code**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#ifdef \_\_APPLE\_\_

#include <OpenCL/opencl.h>

#else

#include <CL/cl.h>

#endif

#include "header1.h"

#define PI 3.14159265358979

#define MAX\_SOURCE\_SIZE (0x100000)

#define AMP(a, b) (sqrt((a)\*(a)+(b)\*(b)))

cl\_device\_id device\_id = NULL;

cl\_context context = NULL;

cl\_command\_queue queue = NULL;

cl\_program program = NULL;

enum Mode {

forward = 0,

inverse = 1

};

int setWorkSize(size\_t\* gws, size\_t\* lws, cl\_int x, cl\_int y)

{

switch(y) {

case 1:

gws[0] = x;

gws[1] = 1;

lws[0] = 1;

lws[1] = 1;

break;

default:

gws[0] = x;

gws[1] = y;

lws[0] = 1;

lws[1] = 1;

break;

}

return 0;

}

int fftCore(cl\_mem dst, cl\_mem src, cl\_mem spin, cl\_int m, enum Mode direction)

{

cl\_int ret;

cl\_int iter;

cl\_uint flag;

cl\_int n = 1<<m;

cl\_event kernelDone;

cl\_kernel brev = NULL;

cl\_kernel bfly = NULL;

cl\_kernel norm = NULL;

brev = clCreateKernel(program, "bitReverse", &ret);

bfly = clCreateKernel(program, "butterfly", &ret);

norm = clCreateKernel(program, "norm", &ret);

size\_t gws[2];

size\_t lws[2];

switch (direction) {

case forward:flag = 0x00000000; break;

case inverse:flag = 0x80000000; break;

}

ret = clSetKernelArg(brev, 0, sizeof(cl\_mem), (void \*)&dst);

ret = clSetKernelArg(brev, 1, sizeof(cl\_mem), (void \*)&src);

ret = clSetKernelArg(brev, 2, sizeof(cl\_int), (void \*)&m);

ret = clSetKernelArg(brev, 3, sizeof(cl\_int), (void \*)&n);

ret = clSetKernelArg(bfly, 0, sizeof(cl\_mem), (void \*)&dst);

ret = clSetKernelArg(bfly, 1, sizeof(cl\_mem), (void \*)&spin);

ret = clSetKernelArg(bfly, 2, sizeof(cl\_int), (void \*)&m);

ret = clSetKernelArg(bfly, 3, sizeof(cl\_int), (void \*)&n);

ret = clSetKernelArg(bfly, 5, sizeof(cl\_uint), (void \*)&flag);

ret = clSetKernelArg(norm, 0, sizeof(cl\_mem), (void \*)&dst);

ret = clSetKernelArg(norm, 1, sizeof(cl\_int), (void \*)&n);

/\* Reverse bit ordering \*/

setWorkSize(gws, lws, n, n);

ret = clEnqueueNDRangeKernel(queue, brev, 2, NULL, gws, lws, 0, NULL, NULL);

/\* Perform Butterfly Operations\*/

setWorkSize(gws, lws, n/2, n);

for (iter=1; iter <= m; iter++){

ret = clSetKernelArg(bfly, 4, sizeof(cl\_int), (void \*)&iter);

ret = clEnqueueNDRangeKernel(queue, bfly, 2, NULL, gws, lws, 0, NULL, &kernelDone);

ret = clWaitForEvents(1, &kernelDone);

}

if (direction == inverse) {

setWorkSize(gws, lws, n, n);

ret = clEnqueueNDRangeKernel(queue, norm, 2, NULL, gws, lws, 0, NULL, &kernelDone);

ret = clWaitForEvents(1, &kernelDone);

}

ret = clReleaseKernel(bfly);

ret = clReleaseKernel(brev);

ret = clReleaseKernel(norm);

return 0;

}

int main()

{

cl\_ulong time\_start, time\_end, total\_time;

clock\_t start, end;

start = clock();

cl\_mem xmobj = NULL;

cl\_mem rmobj = NULL;

cl\_mem wmobj = NULL;

cl\_kernel sfac = NULL;

cl\_kernel trns = NULL;

cl\_kernel hpfl = NULL;

cl\_platform\_id platform\_id = NULL;

cl\_uint ret\_num\_devices;

cl\_uint ret\_num\_platforms;

cl\_uint compute\_unit = 0;

cl\_uint groupsize\_unit=0;

cl\_uint groupitem\_unit=0;

cl\_int ret;

cl\_float2 \*xm;

cl\_float2 \*rm;

cl\_float2 \*wm;

pgm\_t ipgm;

pgm\_t opgm;

FILE \*fp;

const char fileName[] = "E://project//a//all filters.cl";

size\_t source\_size;

char \*source\_str;

cl\_int i, j;

cl\_int n;

cl\_int m;

size\_t gws[2];

size\_t lws[2];

/\* Load kernel source code \*/

fp = fopen(fileName, "r");

if (!fp) {

fprintf(stderr, "Failed to load kernel.\n");

exit(1);

}

source\_str = (char \*)malloc(MAX\_SOURCE\_SIZE);

source\_size = fread(source\_str, 1, MAX\_SOURCE\_SIZE, fp);

fclose( fp );

/\* Read image \*/

readPGM(&ipgm, "E://project//a//lena.pgm");

n = ipgm.width;

m = (cl\_int)(log((double)n)/log(2.0));

xm = (cl\_float2 \*)malloc(n \* n \* sizeof(cl\_float2));

rm = (cl\_float2 \*)malloc(n \* n \* sizeof(cl\_float2));

wm = (cl\_float2 \*)malloc(n / 2 \* sizeof(cl\_float2));

for (i=0; i < n; i++) {

for (j=0; j < n; j++) {

((float\*)xm)[(2\*n\*j)+2\*i+0] = (float)ipgm.buf[n\*j+i];

((float\*)xm)[(2\*n\*j)+2\*i+1] = (float)0;

}

}

/\* Get platform/device \*/

ret = clGetPlatformIDs(1, &platform\_id, &ret\_num\_platforms);

printf("platform id=%d \n",platform\_id);

ret = clGetDeviceIDs( platform\_id, CL\_DEVICE\_TYPE\_GPU, 1, &device\_id, &ret\_num\_devices);

printf("device id=%d \n",device\_id);

ret = clGetDeviceInfo(device\_id, CL\_DEVICE\_MAX\_COMPUTE\_UNITS, sizeof(cl\_uint), &compute\_unit, NULL);

printf(" compute unit: %d \n", compute\_unit);

//ret = clGetDeviceInfo(device\_id, CL\_DEVICE\_GLOBAL\_MEM\_SIZE, sizeof(cl\_uint), &groupsize\_unit, NULL);

//printf(" max work group size: %d \n", groupsize\_unit);

ret = clGetDeviceInfo(device\_id, CL\_DEVICE\_MAX\_WORK\_ITEM\_DIMENSIONS, sizeof(cl\_uint), &groupitem\_unit, NULL);

printf(" max work item size: %d \n", groupitem\_unit);

/\* Create OpenCL context \*/

context = clCreateContext(NULL, 1, &device\_id, NULL, NULL, &ret);

/\* Create Command queue \*/

queue = clCreateCommandQueue(context, device\_id, 0, &ret);

/\* Create Buffer Objects \*/

xmobj = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, n\*n\*sizeof(cl\_float2), NULL, &ret);

rmobj = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, n\*n\*sizeof(cl\_float2), NULL, &ret);

wmobj = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, (n/2)\*sizeof(cl\_float2), NULL, &ret);

/\* Transfer data to memory buffer \*/

ret = clEnqueueWriteBuffer(queue, xmobj, CL\_TRUE, 0, n\*n\*sizeof(cl\_float2), xm, 0, NULL, NULL);

/\* Create kernel program from source \*/

program = clCreateProgramWithSource(context, 1, (const char \*\*)&source\_str, (const size\_t \*)&source\_size, &ret);

/\* Build kernel program \*/

ret = clBuildProgram(program, 1, &device\_id, NULL, NULL, NULL);

/\* Create OpenCL Kernel \*/

sfac = clCreateKernel(program, "spinFact", &ret);

trns = clCreateKernel(program, "transpose", &ret);

hpfl = clCreateKernel(program, "highPassFilter", &ret);

/\* Create spin factor \*/

ret = clSetKernelArg(sfac, 0, sizeof(cl\_mem), (void \*)&wmobj);

ret = clSetKernelArg(sfac, 1, sizeof(cl\_int), (void \*)&n);

setWorkSize(gws, lws, n/2, 1);

ret = clEnqueueNDRangeKernel(queue, sfac, 1, NULL, gws, lws, 0, NULL, NULL);

/\* Butterfly Operation \*/

fftCore(rmobj, xmobj, wmobj, m, forward);

/\* Transpose matrix \*/

ret = clSetKernelArg(trns, 0, sizeof(cl\_mem), (void \*)&xmobj);

ret = clSetKernelArg(trns, 1, sizeof(cl\_mem), (void \*)&rmobj);

ret = clSetKernelArg(trns, 2, sizeof(cl\_int), (void \*)&n);

setWorkSize(gws, lws, n, n);

ret = clEnqueueNDRangeKernel(queue, trns, 2, NULL, gws, lws, 0, NULL, NULL);

/\* Butterfly Operation \*/

fftCore(rmobj, xmobj, wmobj, m, forward);

/\* Apply high-pass filter \*/

cl\_int radius = n/8;

ret = clSetKernelArg(hpfl, 0, sizeof(cl\_mem), (void \*)&rmobj);

ret = clSetKernelArg(hpfl, 1, sizeof(cl\_int), (void \*)&n);

ret = clSetKernelArg(hpfl, 2, sizeof(cl\_int), (void \*)&radius);

setWorkSize(gws, lws, n, n);

ret = clEnqueueNDRangeKernel(queue, hpfl, 2, NULL, gws, lws, 0, NULL, NULL);

/\* Inverse FFT \*/

/\* Butterfly Operation \*/

fftCore(xmobj, rmobj, wmobj, m, inverse);

/\* Transpose matrix \*/

ret = clSetKernelArg(trns, 0, sizeof(cl\_mem), (void \*)&rmobj);

ret = clSetKernelArg(trns, 1, sizeof(cl\_mem), (void \*)&xmobj);

setWorkSize(gws, lws, n, n);

ret = clEnqueueNDRangeKernel(queue, trns, 2, NULL, gws, lws, 0, NULL, NULL);

/\* Butterfly Operation \*/

fftCore(xmobj, rmobj, wmobj, m, inverse);

/\* Read data from memory buffer \*/

ret = clEnqueueReadBuffer(queue, xmobj, CL\_TRUE, 0, n\*n\*sizeof(cl\_float2), xm, 0, NULL, NULL);

/\* \*/

float\* ampd;

ampd = (float\*)malloc(n\*n\*sizeof(float));

for (i=0; i < n; i++) {

for (j=0; j < n; j++) {

ampd[n\*((i))+((j))] = (AMP(((float\*)xm)[(2\*n\*i)+2\*j], ((float\*)xm)[(2\*n\*i)+2\*j+1]));

}

}

opgm.width = n;

opgm.height = n;

normalizeF2PGM(&opgm, ampd);

free(ampd);

/\* Write out image \*/

writePGM(&opgm, "E://project//a//output.pgm");

/\* Finalizations\*/

ret = clFlush(queue);

ret = clFinish(queue);

ret = clReleaseKernel(hpfl);

ret = clReleaseKernel(trns);

ret = clReleaseKernel(sfac);

ret = clReleaseProgram(program);

ret = clReleaseMemObject(xmobj);

ret = clReleaseMemObject(rmobj);

ret = clReleaseMemObject(wmobj);

ret = clReleaseCommandQueue(queue);

ret = clReleaseContext(context);

destroyPGM(&ipgm);

destroyPGM(&opgm);

free(source\_str);

free(wm);

free(rm);

free(xm);

end =clock();

printf( "Number of seconds: %f\n", (clock()-start)/(double)CLOCKS\_PER\_SEC );

return 0;

}