Motion Estimation Extension for OpenCL

Authors: Nico Galoppo, Craig Hansen-Sturm
Reviewers: Ben Ashbaugh, David Blythe, Hong Jiang, Stephen Junkins, Raun Krisch, Matt McClellan, Teresa Morrison, Dillon Sharlet

This document presents the motion estimation extension for OpenCL. This extension includes a set of host-callable functions for frame-based motion estimation and introduces motion estimators, or also “motion estimation accelerator objects”. These accelerator objects provide an abstraction of software- and/or hardware-accelerated functions for motion estimation, which can be provided by select OpenCL vendors.

This extension depends on the OpenCL 1.2 built-in kernel infrastructure and on the accelerator extension, which provides an abstraction for domain-specific acceleration in the OpenCL runtime.

1 Motion Estimation Accelerators

1.1 Bugzilla Bugs

TBD

1.2 OpenCL Spec

OpenCL 1.2 and "Accelerator Extension for OpenCL" spec.

2 Motivation

Motion estimation is the process of determining motion vectors that describe the transformation from one 2D image to another; usually from adjacent frames in a video sequence. These functions generally accept one or more full-frame images as input, perform a motion search operation, and return a motion vector field as output. The motion vectors may relate to the whole image (global motion estimation) or specific parts, such as rectangular blocks, arbitrary shaped patches or even per pixel. This extension provides low-level functionality; currently restricted to block matching methods, and therefore these functions return motion vectors of rectangular pixel blocks. A vector field of per-pixel-block distortion values can optionally be returned as well, which provides the sum-of-absolute-differences between best-match source and reference frame pixel blocks that produced the corresponding motion vector.

The motion estimation extension consists of OpenCL built-in kernels[1] which perform motion estimation, as well as motion estimation accelerator objects, also referred to as "motion estimators", which represent the state of the underlying acceleration engine. These kernels are queued for execution from the host using the standard ND-range mechanism. The global ND-range size parameter to **clEnqueueNDRangeKernel** may be used to select a sub-region of the input frames.

A motion estimation accelerator abstracts the functionality of the underlying motion estimation engine, which can be implemented in software, hardware, or a combination of both. These accelerator objects can be created with the proposed **clCreateAccelerator** extension API[2], and serve as kernel argument to the proposed motion estimation built-in kernels. While this is not part of this extension proposal, the use of motion estimation accelerator object in OpenCL C kernel language may be supported in the future as well. The accelerator objects may be reused for the duration of object's lifetime (using normal OpenCL 1.2 reference counting mechanisms).

[1] See section 5.6.1 of the OpenCL 1.2 Core specification
[2] See "Accelerator Extension for OpenCL" specification

3 Proposal

3.1 New Extension

This proposal adds a new vendor extension, named **cl_intel_motion_estimation**, which adds support for motion estimation accelerator objects as well as the proposed motion estimation built-in kernels.

If this extension is supported by an implementation, the above string will be present in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string described in *Table 4.3*, OpenCL v1.2 spec.

3.2 Header File

The proposed interfaces for this extension will be provided in a header file named:

cl_ext.h

3.3 New Host Functions

None

3.4 New Built-in Kernels

```
__kernel void
block_motion_estimate_intel
(
    accelerator_intel_t accelerator,
    __read_only  image2d_t src_image,
    __read_only  image2d_t ref_image,
    __global short2 * prediction_motion_vector_buffer,
    __global short2 * motion_vector_buffer,
    __global ushort * residuals
);
```

3.5 New Tokens

Accepted as a type in the *accelerator_type* parameter of **clCreateAcceleratorINTEL**:

```
CL_ACCELERATOR_TYPE_MOTION_ESTIMATION_INTEL
```

Accepted as types to the fields of cl_motion_estimator_desc_intel:

```
CL_ME_MB_TYPE_16x16_INTEL                   0x0
CL_ME_MB_TYPE_8x8_INTEL                     0x1
CL_ME_MB_TYPE_4x4_INTEL                     0x2

CL_ME_SUBPIXEL_MODE_INTEGER_INTEL           0x0
CL_ME_SUBPIXEL_MODE_HPEL_INTEL              0x1
CL_ME_SUBPIXEL_MODE_QPEL_INTEL              0x2

CL_ME_SAD_ADJUST_MODE_NONE_INTEL            0x0
CL_ME_SAD_ADJUST_MODE_HAAR_INTEL            0x1

CL_ME_SEARCH_PATH_RADIUS_2_2_INTEL          0x0
CL_ME_SEARCH_PATH_RADIUS_4_4_INTEL          0x1
CL_ME_SEARCH_PATH_RADIUS_16_12_INTEL        0x5
```

3.6 New Types

Parameter type of *accelerator_desc* for **clCreateAcceleratorINTEL**, and parameter type of *param_value* for **clGetAcceleratorInfoINTEL**:

```
typedef struct _cl_motion_estimation_desc_intel {
    cl_uint     mb_block_type;
    cl_uint     subpixel_mode;
    cl_uint     sad_adjust_mode;
    cl_uint     search_path_type;
} cl_motion_estimation_desc_intel;
```

3.7 Additions and Changes to the OpenCL Platform Layer

None

3.8 Additions and Changes to the OpenCL Runtime Layer

3.8.1. Modify section 5.14.1 "Creating Accelerators"

Modify the description of function **clCreateAcceleratorINTEL**.

Table 2.1 must be extended to include the following enumeration constants:

Accelerator Type	Description
CL_ACCELERATOR_TYPE_MOTION_ESTIMATION_INTEL	Create a full-frame motion estimation accelerator

Table 2.2 must be extended to include the following descriptor types:

Descriptor Type	Description
cl_motion_estimator_desc_intel	Used to represent the configuration state of basic full-frame motion estimation accelerators

3.8.2. Modify section 5.14.3 "Accelerator Descriptors"

Add a new entry for the descriptor cl_motion_estimator_desc_intel.

The cl_motion_estimator_desc_intel descriptor structure is describes the configuration of the motion estimation algorithm. This motion estimation descriptor is defined as:

```
typedef struct _cl_motion_estimation_desc_intel {
    cl_uint     mb_block_type;
    cl_uint     subpixel_mode;
    cl_uint     sad_adjust_mode;
    cl_uint     search_path_type;
} cl_motion_estimation_desc_intel;
```

mb_block_type describes the size of the blocks described by the motion estimator. This field influences the size of the output image(s) and buffer(s) of the motion estimation algorithms defined in this extension. Valid values are described in the table below.

Type	Description
CL_ME_MB_TYPE_16x16_INTEL	Each block is of size 16x16 pixels
CL_ME_MB_TYPE_8x8_INTEL	Each block is of size 8x8 pixels
CL_ME_MB_TYPE_4x4_INTEL	Each block is of size 4x4 pixels

subpixel_mode defines the search precision (and hence, the precision of the returned motion vectors). Valid values are described in the table below.

Type	Description
CL_ME_SUBPIXEL_MODE_INTEGER_INTEL	Integer pixel mode searching
CL_ME_SUBPIXEL_MODE_HPEL_INTEL	Half-pixel mode searching
CL_ME_SUBPIXEL_MODE_QPEL_INTEL	Quarter-pixel mode searching

sad_adjust_mode specifies distortion measure adjustment used for the motion search SAD comparison. Valid values are described in the table below.

Type	Description
CL_ME_SAD_ADJUST_MODE_NONE_INTEL	Non-adjusted SAD
CL_ME_SAD_ADJUST_MODE_HAAR_INTEL	Haar transformed SATD (frequency space)

search_path_type specifies the search path and search radius when matching blocks in the neighborhood of each pixel block (optionally offset by the predicted motion vector). Currently, all search algorithms match the source block with pixel blocks in the reference area exhaustively within a [Rx, Ry] radius from the current source pixel block location (optionally offset by the predicted motion vector).

Flag	Description
CL_ME_SEARCH_PATH_RADIUS_2_2_INTEL	Exhaustive search within [±2, ±2] radius
CL_ME_SEARCH_PATH_RADIUS_4_4_INTEL	Exhaustive search within [±4, ±4] radius
CL_ME_SEARCH_PATH_RADIUS_16_12_INTEL	Exhaustive search within [±16, ±12] radius

3.8.3. Modify section 5.14.4 “Using Accelerators”

Motion estimation accelerators encapsulate the internal state of the motion estimation engine and serve as kernel argument to a set of proposed motion estimation built-in kernels. An application can run the accelerated motion estimation functions on an OpenCL device by enqueuing one of the proposed built-in kernels. Effectively, these built-in kernels can be viewed as ‘black-box’ motion estimation functions to the application. The motion estimation accelerator object serves as one of the kernel arguments to the proposed built-in kernels. The kernels can be enqueued for execution by the OpenCL runtime using **clEnqueueNDRangeKernel**.

This extension includes a set of new built-in kernels for motion estimation. These kernels can be created by first creating a program with built-in kernels and then creating a kernel from the resulting program object. The following paragraphs contain a list of proposed built-in kernels with function signature; a program object containing an implementation of the kernel can be created by providing the name of the kernel as listed below to **clCreateProgramWithBuiltInKernels.**

The kernel

```
__kernel void
block_motion_estimate_intel
(
    accelerator_intel_t accelerator,
    __read_only  image2d_t src_image,
    __read_only  image2d_t ref_image,
    __global short2 * prediction_motion_vector_buffer,
    __global short2 * motion_vector_buffer,
    __global ushort * residuals
);
```

computes motion vectors by comparing a 2d image source with a 2d reference image, producing a vector field motion vectors. The algorithm searches the best match of each pixel block in the source image by searching an image region in the reference image, centered on the coordinates of that pixel block in the source image (optionally offset by the prediction motion vectors).

This kernel optionally takes a vector field of motion vector predictors via the *prediction_motion_vector_buffer* kernel argument. The kernel also optionally returns a vector field of per-pixel-block information records. Each record contains the best-match distortion (SAD) value and additional search result information.

When enqueuing this kernel, *global_work_size* and *global_work_offset* determine the region of interest of the input frames. The dimension of the output motion vector image is dependent on the size of the region of interest and partitioning mode specified by the accelerator.

accelerator is a valid accelerator object created by **clCreateAcceleratorINTEL**, where the type of the accelerator must be CL_ACCELERATOR_TYPE_MOTION_ESTIMATION_INTEL.

src_image is the input source image, typically representing 8-bit luminance information. *image_channel_order* and the *image_data_type* of *src_image* are restricted as follows:

Channel Order	Src Channel Data Type
CL_R	CL_UNORM_INT8

ref_image is the input reference image, typically representing 8-bit luminance information. *image_channel_order* and the *image_data_type* must match *src_image*, as follows:

Channel Order	Src Channel Data Type
CL_R	CL_UNORM_INT8

motion_vector_buffer is the output motion vector buffer, representing a vector field of pixel block motion vectors, stored linearly in row-major order. The elements (pixels) of this image contain a motion vector for the corresponding pixel block, with its x/y components packed as two 16-bit integer values. Each component is encoded as a S13.2 fixed point value (two's complement). The precision is determined by the *subpixel_mode* defined in the accelerator descriptor when created with **clCreateAcceleratorINTEL**. The size of the returned data is determined by the *mb_block_type* defined at accelerator creation time. This buffer needs to be sized appropriately such that it fits the results of all

pixel blocks of the source image, i.e. the number of 16x16 source pixel blocks times the number of returned motion vectors per source block (1, 4, or 16).

prediction_motion_vector_buffer is an optional input buffer representing a vector field of prediction motion vectors, stored linearly in row-major order. When provided, the motion vectors in this image are used to offset the search center for each 16x16 pixel block search window – the application should provide one offset motion vector per 16x16 source pixel block. In feedback algorithms, where the output *motion_vector_buffer* of one frame is used to provide *prediction_motion_buffer* as input to the next frame, the application is responsible for downsampling the output buffer to the prediction buffer. The size of this buffer is the number of 16x16 source pixel blocks in each dimension. The application can choose **not** to provide prediction motion vectors by providing NULL as the *arg_value* argument to **clSetKernelArg()**, in which case the prediction motion vectors are implied to be (0,0).

residuals is a buffer representing a field of residuals or "distortion values" (one for each returned motion vector). These "residuals of the compensated image" represent the sum-of-absolute-differences (SAD) between the source frame pixel block and the best-match reference frame pixel block that produced the returned motion vector. The *sad_adjust_mode* defined in the accelerator determines whether plain SAD or SATD (Haar-adjusted) values are returned. The application can choose not to provide prediction motion vectors by providing NULL as the *arg_value* argument to **clSetKernelArg()**, in which case this information is not returned.

The motion estimation built-in kernels are queued for execution by the host application using **clEnqueueNDRangeKernel()**. This function will return the usual error codes, augmented with the following specific error codes for this kernel:

CL_INVALID_WORK_DIMENSION if work_dim is **not** 2. This built-in kernel requires a 2D ND-range.

CL_INVALID_WORK_GROUP_SIZE if local_work_size is **not** NULL. This built-in kernel requires the work-group size to be set by the runtime.

CL_INVALID_IMAGE_SIZE if an image object is specified as an argument value and the image dimensions (image width, height, specified or compute row and/or slice pitch) are not supported by this built-in kernel, as listed above.

CL_INVALID_IMAGE_FORMAT if an image object is specified as an argument value and the image format is not one of the image formats supported by this built-in kernel, as listed above.

3.9 Additions and Changes to the OpenCL Language

None

4 Interactions with Other Extensions

The motion estimation extension is based on the accelerator extension, and is defined in terms of additions to the base accelerator extension document.

5 Sample Code

5.1 Full Frame Block Mode

The following code snippet demonstrates how to set up and queue a simple full-frame motion estimation pass for 16x16 pixel blocks.

```
// Compute output size
int mbSize = 16;
size_t widthInMB = (width + mbSize - 1) / mbSize;
size_t heightInMB = (height + mbSize - 1) / mbSize;

// Create a motion estimator "accelerator" object
cl_motion_estimation_desc_intel desc = {
    CL_ME_MB_TYPE_16x16_INTEL,
    CL_ME_SUBPIXEL_MODE_INTEGER_INTEL,
    CL_ME_SAD_ADJUST_MODE_NONE_INTEL,
    CL_ME_SEARCH_PATH_RADIUS_4_4_INTEL
};

cl_accelerator_intel accelerator =
    clCreateAcceleratorINTEL(context,
    CL_ACCELERATOR_TYPE_MOTION_ESTIMATION_INTEL,
    sizeof(cl_motion_estimation_desc_intel), &desc, 0);

// Input images
cl_image_format format = { CL_R, CL_UNORM_INT8 }; // illuminance plane
cl_mem srcImage = clCreateImage2D(context, CL_MEM_READ_ONLY, &format,
    width, height, 0, pSrcBuf, &err);
cl_mem refImage = clCreateImage2D(context, CL_MEM_READ_ONLY, &format,
    width, height, 0, pRefBuf, &err);

// Output image for MB motion vectors
cl_mem outMVBuffer = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
   widthInMB * heightInMB * sizeof(cl_short2), 0, 0, &err);

clSetKernelArg(kernel, 0, sizeof(cl_accelerator_intel), &accelerator);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &srcImage);
clSetKernelArg(kernel, 2, sizeof(cl_mem), &refImage);
clSetKernelArg(kernel, 3, sizeof(cl_mem), NULL); // disable predictor motion vectors
clSetKernelArg(kernel, 4, sizeof(cl_mem), &outMVBuffer);
clSetKernelArg(kernel, 5, sizeof(cl_mem), NULL); // disable extra MB info output

// Run the kernel
// Notes:
//  1) This built-in *requires* to let runtime determine the WG size, and require 2D ND-range
//  2) We can specify ROI by using NDRange global offset (not shown here)
const size_t originROI[3] = { 0, 0, 0 };
const size_t sizeROI[3] = { width, height, 1 };
clEnqueueNDRangeKernel(queue, kernel, 2, originROI, sizeROI, NULL, 0, 0, 0);

// Read resulting motion vectors
clEnqueueReadBuffer(queue, outMVBuffer, CL_TRUE, 0,
   widthInMB * heightInMB * sizeof(cl_short2), pMVOut, 0, 0, 0);

clReleaseAcceleratorINTEL(accelerator);
```