OpenCL Device-Side Video Motion Estimation (VME)

What is VME?

- Compute intensive component of video encoding algorithms used to find an efficient encoding of a MB in a source frame
- Computes the best combination of:
  - **motion vectors (MVs)** to exploit temporal redundancy across frames
  - **intra prediction directions** to exploit spatial redundancy within a frame
  - **block partitions** to manage trade-offs in meeting bit-rate requirements
- Additionally has applications in frame rate conversion (FRC), asynchronous space warping (ASW) for virtual reality, and visual analytics

- (streamin/streamout)

What will we cover in this tutorial?

- Quick Architectural Overview
- Software Interface Overview
- Basic Concepts
- Walk-through Sample Applications For Encode Covering Basic Concepts
- Pointers To Advanced Topics

Progression Of Sample Applications

Basic Search

Progression Of Sample Applications

Cost Heuristics Search

Progression Of Sample Applications

Larger Search

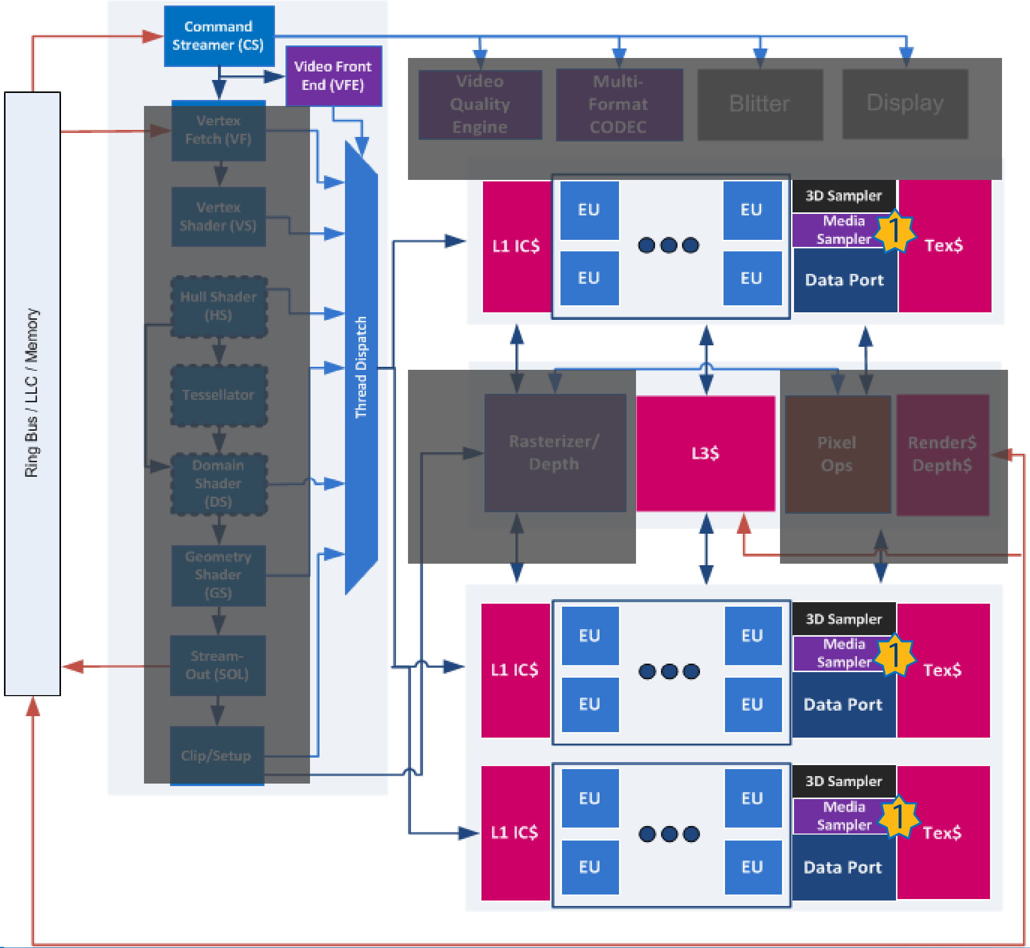
Progression Of Sample Applications

HME

Progression Of Sample Applications

Intra

What does the HW provide?



VME is part of the *Media Sampler*.

- Programmable through EUs
- Operates on 16x16 macroblocks
- 1 per sub-slice
  - 2 sub-units (co-issuable)

- Implements key motion estimation operations
  - Inter Motion Estimation
  - Sub-pixel refinement
  - Intra Prediction
  - Many more…
- Highly programmable general purpose operations

- Optimized for memory bandwidth
- Provides configurable raw compute
- Smarts in the hands of the programmer

What is OpenCL device-side VME?

- Exposes programmable (AVC) VME functionality in GPU
- Set of built-in functions **callable from user written OpenCL kernels** – maps closely with exposed HW interface
    - Essentially provides a very low-level motion estimation library with a underlying HW implementation – think of it as Inter Performance Primitives (IPP).
- Intel vendor (GPU only) extension to OpenCL 1.2
- Subsumes previous host-side Intel VME extensions in
    - Functionality
    - Flexibility
    - Performance

What new capabilities does it provide to users?

- Exposes GPU VME acceleration capabilities at the level of granularity previously available only to Intel developers.
- Develop codecs with OCL using custom algorithms leveraging media sampler
  - accelerate compute intensive video motion estimation operations
  - quick way to implement motion estimation algorithms
  - quick way to build custom higher-level motion estimation libraries
- Enables quicker and performant development of hybrid CPU+GPU, or GPU only codecs.
- Enables quicker and performant development of FRC, ASW and visual analytics algorithms.

Programming model

- Built-in subgroup functions operating on a 16x16 source macroblock (MB)
- Forces a subgroup size of 16
- Think of it as
  - programming at a SIMD16 thread-level rather than at the work-item level
  - VME operations as subgroups media block reads with media sampler motion estimation operations
- Functions organized as ordered phases of operations to manage complexity
- Opaque payload and results with set and extractor functions

Programming model

```
// Global NDRange size is (176, 1). Workgroup size is (16,1) & subgroup size is 16.
int gid_0 = get_group_id(0);
int gid_1 = 0;

// Each SIMD16 kernel thread processes a column of MBs. The kernel argument ‘height’ is set to 9.
for( int i = 0; i < height; i++ ) {
    gid_1 += 1;
    ushort2 src_coord = gid_0 * 16;
    short2 ref_coord = gid_1 * 16;
    intel_sub_group_avc_ime_payload_t payload =
      intel_sub_group_avc_ime_initialize(
        src_coord,
        CLK_AVC_ME_PARTITION_MASK_16x16_INTEL,
        CLK_AVC_ME_SAD_ADJUST_MODE_NONE_INTEL);
    payload =
      intel_sub_group_avc_ime_set_single_reference(
        ref_coord,
        CLK_AVC_ME_SEARCH_WINDOW_EXHAUSTIVE_INTEL,
        payload );
   intel_sub_group_avc_ime_result_t result =
     intel_sub_group_avc_ime_evaluate_with_single_reference(
       src_img,
       ref_img,
       accelerator,
       payload );
   long mvs =
      intel_avc_ime_get_motion_vectors( result );
   long mvs =
      intel_sub_group_avc_ime_get_inter_distortions( result );
```

Payload initialization phase

Operation search window configuration phase

Evaluation phase

Result processing phase (MVs)

Result processing phase (distortions)

Ordered phases to evaluate a VME operation.

Opaque result types.

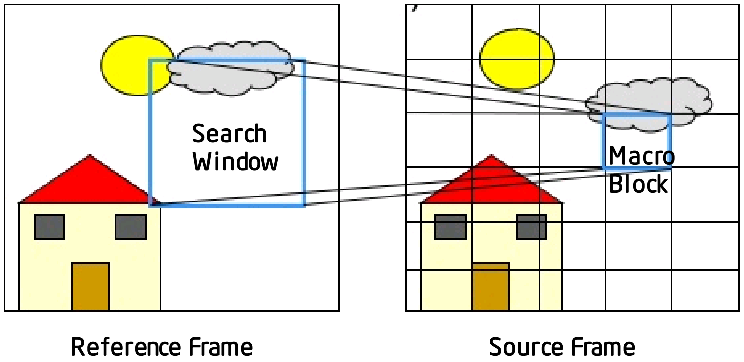
Use extraction functions to get component results.

Remember to return payload. No pass-by-reference.

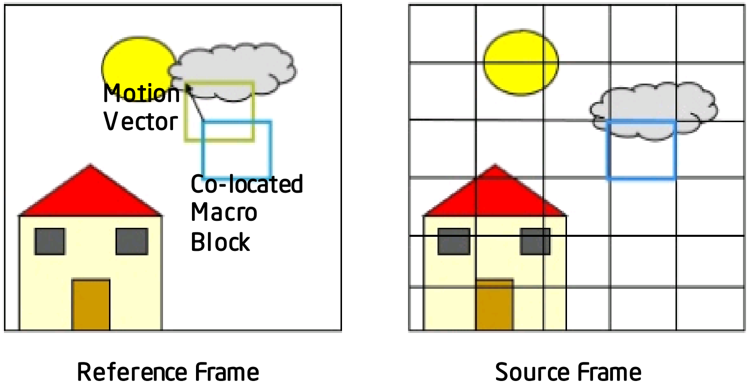
Integer Motion Estimation (IME)

- Fundamental and compute intensive part of VME
- Performs motion estimation at a full pixel resolution
  - on a given source macroblock in a source frame,
  - and a search window in a reference frame
- to determine the
  - best integer motion vectors,
  - associated distortions,
  - and the best macroblock shape decision combination.

[3]

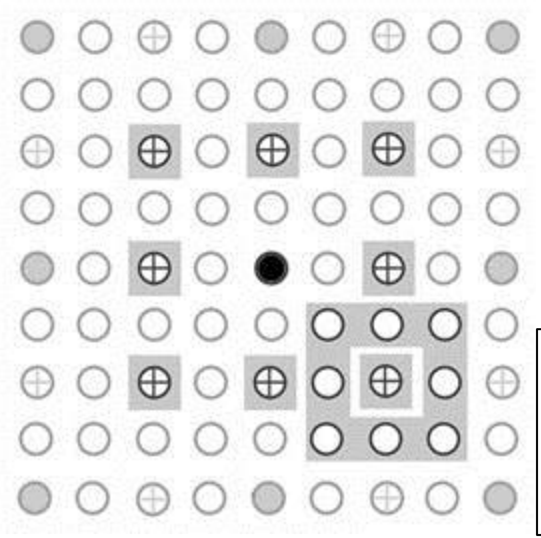


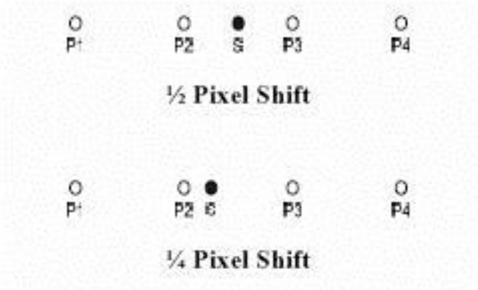
[3]



Motion Estimation Refinement (REF)

- FME
  - Motion often occurs in less than integer resolution.
  - Refines the IME full pixel result to find the best sub-pixel search result – half or quarter pixel resolution.






```
HALF PEL      : (-1, 5, 5, -1)/8     i.e. s = (-P1 + P2 * 5 +P3 * 5 - P4 + 4) / 8
QUARTER PEL   : (-1, 13, 5, -1)/16   i.e. c = (-P1 + P2*13 + P3 * 5 - P4 + 8)/ 16
```

The quarter-pels are actually the averages of its nearest integer and half pixel values.

IME Programming Interface

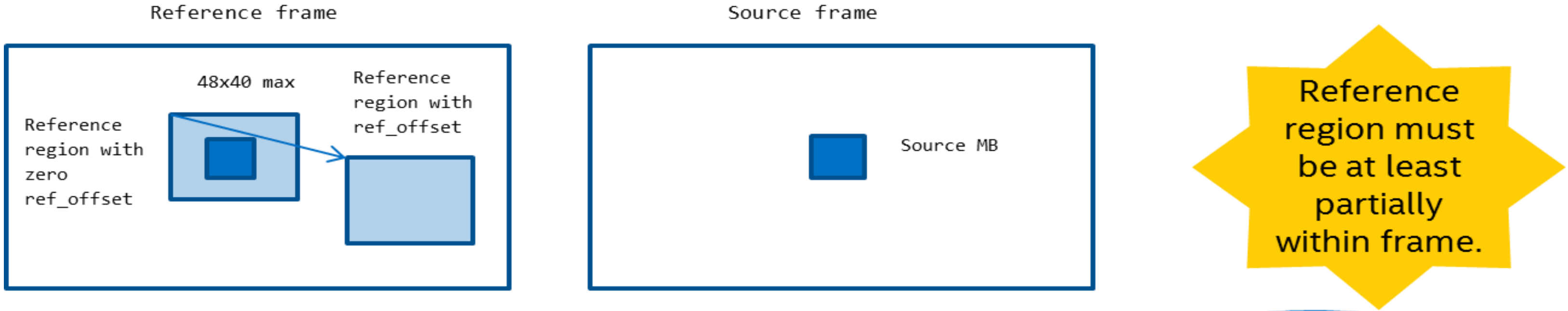
- IME initialization phase
  - Create an initialized payload for an IME operation.

```
ushort2  src_coord;
...
uchar  partition_mask =
  CLK_AVC_ME_PARTITION_MASK_16x16_INTEL  &
  CLK_AVC_ME_PARTITION_MASK_16x8_INTEL & CLK_AVC_ME_PARTITION_MASK_8x16_INTEL  &
  CLK_AVC_ME_PARTITION_MASK_8x8_INTEL;

intel_sub_group_avc_ime_payload_t payload =
  intel_sub_group_avc_ime_initialize(
    src_coord,                           // source MB offset (top-left corner) in pixel units
    partition_mask,                      // enable 16x16, 16x8, 8x16, 8x8 shapes
    CLK_AVC_ME_SAD_ADJUST_MODE_NONE_INTEL  // SAD distortions with no transform
  );
```

IME Programming Interface

- IME search configuration phase
  - Configures an initialized payload for a single reference search.



EXHAUSTIVE	48x40 search region with exhaustive single reference search
SMALL	28x28 search region with exhaustive search
TINY	24x24 search region with exhaustive search
EXTRA TINY	20x20 search region with exhaustive search
DIAMOND	48x40 search region with diamond single reference search
LARGE DIAMOND	48x40 search region with large diamond single reference search

Best quality

~1.25x-2x fast as EXH

Trade-off between EXH & DIA

IME Programming Interface

- IME search configuration phase
  - Configures an initialized payload for a single reference search.

See programmer's guide for configuration setting for dual reference search.

```
...

// Configure the initialized IME payload for a single reference search.
// The search window is 48x40 with exhaustive search, with its location
// offset specified by the ref_offset.

intel_sub_group_avc_ime_payload_t payload =
  intel_sub_group_avc_ime_set_single_reference(
    ref_offset,                                // reference window offset
    CLK_AVC_ME_SEARCH_WINDOW_EXHAUSTIVE_INTEL,// 48x40 reference window size
    payload                                   // previous initialized payload
  );
```

IME Programming Interface

- Adjust reference regions
  - Adjust reference offset so that the reference search region is on the appropriate frame boundary for the given source coordinate if specified search region is fully out-of-bounds.

```
// Get the reference search window 2D size for a single reference search.
ushort2 ref_window_size =
  intel_sub_group_ime_ref_window_size(CLK_AVC_ME_SEARCH_WINDOW_EXHAUSTIVE_INTEL, false );

// Adjust IME reference window offsets so that the reference windows stay
// within the reference frame
ref_offset =
  intel_sub_group_avc_ime_adjust_ref_offset(
    ref_offset, src_coord, ref_window_size, convert_ushort2( get_image_dim( ref_image ) ) );

intel_sub_group_avc_ime_payload_t payload =
  intel_sub_group_avc_ime_set_single_reference(
    ref_offset,                              // reference window offset
    CLK_AVC_ME_SEARCH_WINDOW_EXHAUSTIVE_INTEL, // 48x40 reference window size
    payload                                  // previous initialized payload
  );
```

IME Programming Interface

- IME evaluation phase
  - Perform the actual IME operation in the VME unit based on the configured payload.
  - Operation latency is incurred here.
  - All result components are packed into an opaque return object.

```
// Evaluate the IME operation with its configured payload.
// Note: src_img & fwd_img must appear consecutively in kernel parameter list.
intel_sub_group_avc_ime_result_t result =
    intel_sub_group_avc_ime_evaluate_with_single_reference(
        src_img, fwd_ref_img, vme_sampler, payload );
```

IME Programming Interface

- IME result extraction phase
  - Needed for extracting the various component results from the opaque result object..
  - Spec describes how the results components are mapped to the individual work-items.
  - Essentially distributes the IME results returned by the VME HW into a format that can be conveniently accessed using OpenCL work-item based subgroup functions.

```
// Extract IME results.
ushort sads         = intel_sub_group_avc_ime_get_inter_distortions( result );
uchar major_shape   = intel_sub_group_avc_ime_get_inter_major_shape( result );
uchar minor_shapes  = intel_sub_group_avc_ime_get_inter_minor_shapes( result );
uchar directions    = intel_sub_group_avc_ime_get_inter_directions( result );
long bi_mvs         = intel_sub_group_avc_ime_get_motion_vectors( result );
int2 bi_mvs_int     = as_int2( bi_mvs );
short2 fwd_mvs      = as_short2( bi_mvs.s0 );
short2 bwd_mvs      = as_short2( bi_mvs.s1 );
```

Motion Vector Costing

- VME is not only about minimizing distortion
  - Need to consider the bits to encode MVs as well – rate distortion optimization.
- MVs for a block are differentially encoded w.r.t to its “predicted” MV which is based on it neighbors MVs
  - Need to bias the MVs closer to its predicted MV.
  - Uniformity of produced MVs reduced bits to encode them.
- When bit rate requirements are stringent, need to be more aggressive in trading-off quality for compression.
- In difficult to encode frames, there is no point in having more or longer MVs.
- In FRC and ASW, uniformity of MVs is more important than minimizing distortion.
- MV costing is a scheme to do exactly this using a user specified cost center and a cost table

Shape Costing

- VME is not only about minimizing distortion
  - Need to consider the bits to encode the MB partitioning scheme (shapes) as well – rate distortion optimization.
- We aim to achieve better compression with B-slices (or frames), so might want to bias larger shapes for B-slices than P-slices
- When bit rate requirements are stringent, need to be more aggressive in trading-off quality for compression.
- In difficult to encode frames, there is no point in having more partitions.
- Shape costing is a scheme to do exactly this

Motion Vector Costing

- IME configuration phase
  - Configures initialized & search window configured payload with costing scheme.

```
intel_sub_group_avc_ime_payload_t payload =
   intel_sub_group_avc_ime_set_single_reference(
       ref_offset, CLK_AVC_ME_SEARCH_WINDOW_EXHAUSTIVE_INTEL, payload);

// Configure cost heuristics.
uchar slice_type = CLK_AVC_ME_SLICE_TYPE_PRED_INTEL;
uchar qp = 45;
uint2 packed_cost_table =
    intel_sub_group_avc_mce_get_default_inter_motion_vector_cost_table(slice_type, qp);
uchar cost_precision = CLK_AVC_ME_COST_PRECISION_QPEL_INTEL;
ulong cost_center = 0;

// Update the payload with the cost function.
payload =
   intel_sub_group_avc_ime_set_motion_vector_cost_function(
       cost_center, packed_cost_table, cost_precision, payload);
```

Shape Costing

- IME configuration phase
  - Configures initialized & search window configured payload with costing scheme.

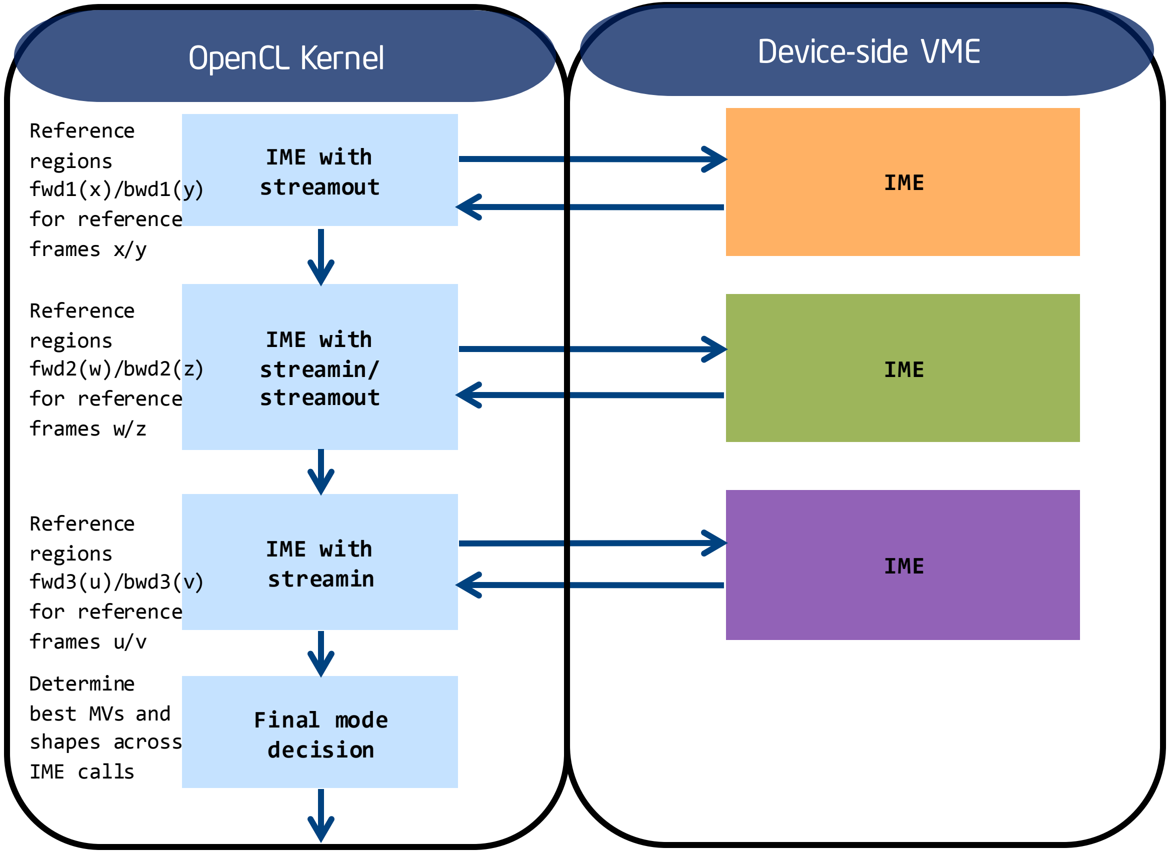
```
// Get the default packed shape cost penalties.
  ulong packed_shape_cost_penalties =
      intel_sub_group_avc_mce_get_default_inter_shape_cost_penalty(
          slice_type, qp );

  // Update the payload with the inter shape cost penalties.
  payload =
      intel_sub_group_avc_mce_set_inter_shape_cost_penalty (
       packed_shape_cost_penalties,
       payload );
```

Unbounded IME Search

- Chaining IME operations
  - To search beyond the VME search region limits (48x40 for single reference), the user must call VME evaluation functions multiple times.
  - Search regions may be even across multiple reference frames.
  - VME HW supports this using the streamin/streamout feature
  - Can also be used to get the best motion vectors for all major shapes using a single VME call.

Unbounded IME Search



Unbounded IME Search

Search region 1

```
short2 refCoord = refCoord0; ulong cost_center = cost_center0;
intel_sub_group_avc_ime_payload_t payload = intel_sub_group_avc_ime_initialize( srcCoord, partition_mask, sad_adjustment );
payload = intel_sub_group_avc_ime_set_single_reference( refCoord, CLK_AVC_ME_SEARCH_WINDOW_EXHAUSTIVE_INTEL, payload );
payload = intel_sub_group_avc_ime_set_motion_vector_cost_function( cost_center, packed_cost_table, search_cost_precision, payload );
intel_sub_group_avc_ime_result_single_reference_streamout_t resultsout;
resultsout = intel_sub_group_avc_ime_evaluate_with_single_reference_streamout( srcImg, refImg, accelerator, payload );
```

Search region 2

```
refCoord = refCoord1; cost_center = cost_center1;
intel_sub_group_avc_ime_single_reference_streamin_t resultsin = intel_sub_group_avc_ime_get_single_reference_streamin( resultsout );
payload = intel_sub_group_avc_ime_initialize( srcCoord, partition_mask, sad_adjustment );
payload = intel_sub_group_avc_ime_set_single_reference( refCoord, CLK_AVC_ME_SEARCH_WINDOW_EXHAUSTIVE_INTEL, payload );
payload = intel_sub_group_avc_ime_set_motion_vector_cost_function( cost_center, packed_cost_table, search_cost_precision, payload );
resultsout = intel_sub_group_avc_ime_evaluate_with_single_reference_streaminout( srcImg, refImg, accelerator, payload, resultsin );
resultsin = intel_sub_group_avc_ime_get_single_reference_streamin( resultsout );
```

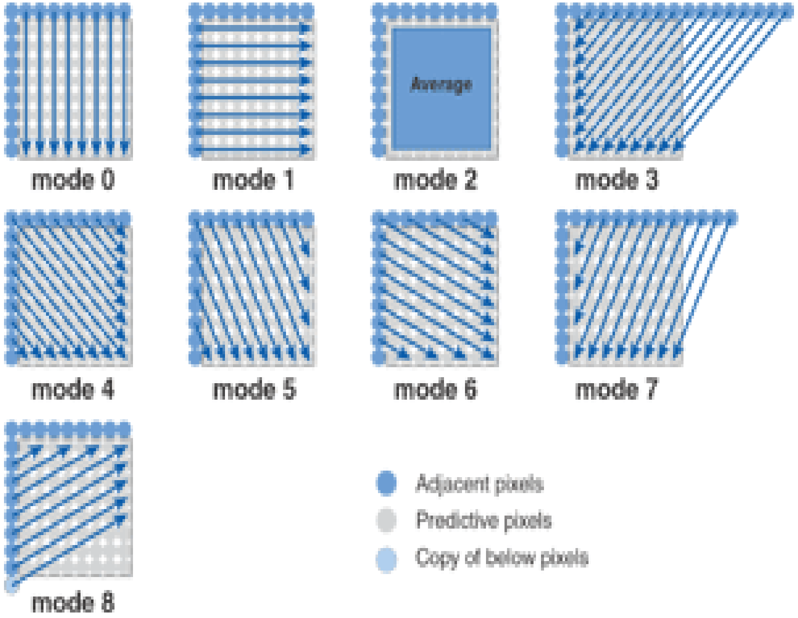
Search region 3

```
refCoord = refCoord3; cost_center = cost_center3;
resultsin = intel_sub_group_avc_ime_get_single_reference_streamin( resultsout );
payload = intel_sub_group_avc_ime_initialize( srcCoord, partition_mask, sad_adjustment );
payload = intel_sub_group_avc_ime_set_motion_vector_cost_function( cost_center, packed_cost_table, search_cost_precision, payload );
payload = intel_sub_group_avc_ime_set_single_reference( refCoord, CLK_AVC_ME_SEARCH_WINDOW_EXHAUSTIVE_INTEL, payload );
intel_sub_group_avc_ime_result_t result =
   intel_sub_group_avc_ime_evaluate_with_single_reference_streamin( srcImg, refImg, accelerator, payload, resultsin );

long mvs = intel_sub_group_avc_ime_get_motion_vectors( result );
```

Intra Prediction Estimation (IPE)

- Determines spatial correlation between MBs within the source frame to predict the source MB from the edge pixels from neighboring MBs.
- Determines
  - the best intra prediction modes,
  - and the best shape partitioning combination.
- Source image must be NV12, if configured for chroma based estimation.



IPE Initialization

- Creating an initialized payload for a VME IPE operation.
  - Source coordinates of the source MB in pixel units relative to the top-left corner of the source frame.

```
ushort2  src_coord;
  ...

intel_sub_group_avc_sic_payload_t payload =
  intel_sub_group_avc_sic_initialize(
    src_coord        // source MB offset in pixel units
  );
```

Intra Estimation Configuration

- Configuring the initialized payload
  - Set the available edges for the MB
    - Depends on neighbor availability and algorithm
  - Load-in and provide VME the edge pixels
    - Use subgroup media block read functions
    - Generally done on original source pixels for performance in fast implementations
  - Cost configuration functions are provided for rate-distortion optimization.

Intra Estimation Configuration

```
void sic_kernel(
    __read_only image2d_t src_vme_image, __read_only image2d_t ref_image,
    __read_only image2d_t src_read_image, ushort2              src_coord
)
{
    ...
    // Initialize the MB neighborhood mask, intraEdges.
    uint intraEdges,_leftEdge, leftUpperPixel, upperEdge;
    intraEdges =
        CLK_AVC_ME_INTRA_NEIGHBOR_LEFT_MASK_ENABLE_INTEL       |
        CLK_AVC_ME_INTRA_NEIGHBOR_UPPER_MASK_ENABLE_INTEL      |
        CLK_AVC_ME_INTRA_NEIGHBOR_UPPER_LEFT_MASK_ENABLE_INTEL |
        CLK_AVC_ME_INTRA_NEIGHBOR_UPPER_RIGHT_MASK_ENABLE_INTEL;

      // If this is a left-edge MB, then disable left edges.
      if( ... ) {
        intraEdges &= ~CLK_AVC_ME_INTRA_NEIGHBOR_LEFT_MASK_ENABLE_INTEL;
        intraEdges &= ~CLK_AVC_ME_INTRA_NEIGHBOR_UPPER_LEFT_MASK_ENABLE_INTEL;
      }
      // If this is a right edge MB then disable right edges.
      if( ... ) {
        intraEdges &= ~CLK_AVC_ME_INTRA_NEIGHBOR_UPPER_RIGHT_MASK_ENABLE_INTEL;
      }
      // If this is a top-edge MB, then disable top edges.
      if( ... ) {
        intraEdges &= ~CLK_AVC_ME_INTRA_NEIGHBOR_UPPER_LEFT_MASK_ENABLE_INTEL;
        intraEdges &= ~CLK_AVC_ME_INTRA_NEIGHBOR_UPPER_RIGHT_MASK_ENABLE_INTEL;
        intraEdges &= ~CLK_AVC_ME_INTRA_NEIGHBOR_UPPER_MASK_ENABLE_INTEL;
      }
```

Specify the available edges for the MB.

Intra Estimation Configuration

```
// Read left edge.
int2 edgeCoord; edgeCoord.x = srcCoord.x - 4; edgeCoord.y = srcCoord.y;
uint leftEdgeDW = intel_sub_group_media_block_read_ui( edgeCoord, 1, 16, src_read_image );
leftEdge = as_uchar4( leftEdgeDW ).s3;

// Read upper left corner.
edgeCoord.x = srcCoord.x - 4; edgeCoord.y = srcCoord.y - 1;
uint leftUpperPixelDW = intel_sub_group_media_block_read_ui( edgeCoord, 1, 16, src_read_image );
leftUpperPixel = as_uchar4( leftUpperPixelDW ).s3;
leftUpperPixel = intel_sub_group_shuffle( leftUpperPixel, 0 );

 // Read upper edge.
edgeCoord.x = srcCoord.x; edgeCoord.y = srcCoord.y - 1;
upperEdge = intel_sub_group_media_block_read_uc( edgeCoord, 16, 1, src_read_image );

// Read upper right edge.
edgeCoord.x = srcCoord.x + 16; edgeCoord.y = srcCoord.y - 1;
upperRightEdge = intel_sub_group_media_block_read_uc( edgeCoord, 16, 1, src_read_image );
```

Read the available edges for the MB using media block reads.

```
// Initialize a SIC operation.
intel_sub_group_avc_sic_payload_t payload = intel_sub_group_avc_sic_initialize( src_coord );
// Configure an optional SKC operation.
payload = intel_sub_group_avc_sic_configure_skc( ... );
 // Configure an IPE operation with the neighboring edges.
payload =
    intel_sub_group_avc_sic_configure_ipe(
        intraPartMask, intraEdges,
        leftEdge, leftUpperPixel, upperEdge, upperRightEdge,
        CLK_AVC_ME_SAD_ADJUST_MODE_HAAR_INTEL, payload );
```

Configure the SIC payload for the IPE with an option SKC.

Bonus Samples

- vme_hme
  - Hierarchal motion estimation – faster technique to cover larger search areas
- vme_wpp
  - Wavefront Parallel Processing – video encoding exhibit a wavefront data dependency pattern for processing MBs
- vme_interlaced
  - Interlaced content – need to process the top and bottom field lines separately

References

1. https://www.khronos.org/registry/OpenCL/extensions/intel/cl_intel_device_side_avc_motion_estimation.txt
2. compute_samples/docs/programmer_guides//cl_intel_device_side_avc_vme_programmers_manual.pdf
3. Images in slide for IME taken from https://www.slideshare.net/samvrudhi96/video-compression-55568195

intel®