General Information

A subset of functions and algorithms in OpenCV library is accelerated on OpenCL(TM) compatible devices. OpenCL (Open Computing Language) is a Khronos(R) standard for software API, with goal to accelerate data processing by a variety of devices (GPUs, CPUs, FPGAs, DSPs, etc), abstracting the exact hardware details.

Refer to OpenCL official page for OpenCL conception and details: https://www.khronos.org/opencl/

OpenCV can utilize acceleration on devices with OpenCL 1.2 (and OpenCL 1.1 with limited functionality) "FULL PROFILE" capability (with online compiler from OpenCL C language).

Accelerated implementations are added via "Transparent API" design. See more details about it here: T-API.

In general, "accelerated" results of algorithms should be similar, but there is no guarantee of bit-exact results from OpenCL backend due different algorithms implementations.

OpenCV OpenCL configuration options

OpenCV is able to detect, load and utilize OpenCL devices automatically. By default, it enables the first GPU-based OpenCL device.

There are several runtime options to configure OpenCL optimizations:

- `OPENCV_OPENCL_RUNTIME`

  Override path to OpenCL runtime or disable OpenCL completelly (`=disabled`)

- `OPENCV_OPENCL_DEVICE`

  Allow the user to select OpenCL device in this format:

  ```
  <Platform>:<CPU|GPU|ACCELERATOR|nothing=GPU/CPU>:<DeviceName or ID>
  ```

  Note: Device ID range is: 0..9 (only one digit, 10 - it is a part of name) Some examples:

  ```
  '' = ':' = '::' = ':GPU:'
  'AMD:GPU|CPU:'
  'Intel:CPU:'
  'AMD::Tahiti'
  ':GPU:1'
  ```

- OpenCL binary cache settings:
  - `OPENCV_OPENCL_CACHE_ENABLE=<bool>` (default value is `true`)
  - `OPENCV_OPENCL_CACHE_WRITE=<bool>` (default value is `true` , use `false` to forbid writing of new kernels into binary cache)
  - `OPENCV_OPENCL_CACHE_LOCK_ENABLE=<bool>` (default value is `true` , necessary for cache integrity in multiprocess application setups)
  - `OPENCV_OPENCL_CACHE_CLEANUP=<bool>` (default value is `true` , use `false` to prevent cache removal for other versions of OpenCL devices)

Developer Notes

To store data and operate with OpenCL OpenCV uses `cv::UMat` instead of CPU-based `cv::Mat` .

It is necessary because in a heterogeneous device environment:

- there is no direct access to accelerator memory from CPU host program, so `cv::Mat::data` pointer is not available
- and there may be cost associated with data transfer, so this operations should not be implemented implicitly.

For best performance, in either case, it is recommended that you do not introduce unnecessary data transfers between CPU and the discrete GPU. OpenCV design guidelines prefer not to invoke OpenCL kernels for non-UMat parameters.

If the OpenCL device doesn't support something or not able to process requested operation, then OpenCV doesn't generate an error. Instead OpenCV switches to other available implementation branches (generic CPU-based branch is always available) according to "Transparent API" design.

The primary OpenCL storage is OpenCL buffers. The most part of OpenCL kernels is designed to work with OpenCL buffers. OpenCL images are almost unused.

OpenCV has experimental OpenCL SVM support (disabled by default via build flag).

Custom OpenCL programs support

OpenCV provides API to load, build and execute OpenCL kernels.

OpenCL kernels are part of OpenCL programs. OpenCV API can handle these types of programs sources:

- (**source**) "OpenCL C" source code (without "#includes" support)
- (**binary**) Binary programs compiled for specific device. They are usually not portable between devices (and/or OpenCL vendors)
- (**SPIR**) OpenCL SPIR programs (required 'cl_khr_spir' extension support from OpenCL runtime). These programs are cross-platform and able to run on different OpenCL-compatible devices. But OpenCL vendor-specific extensions (like "cl_intel_subgroups") are not available.
- (**SPIR-V**) OpenCL SPIR-V programs (requires OpenCL 2.1+, support is not implemented in OpenCV yet).

To define OpenCL program source use OpenCV's cv::ocl::ProgramSource API. To build OpenCL program for target device use OpenCV's cv::ocl::Program API.

Launching kernels

To launch OpenCL kernels developer needs to specify:

- Kernel name from OpenCL program
- Kernel arguments (see below)
- Task dimension (OpenCV supports up to 3 dimenstion tasks)
- Global work size
- Local work size (OpenCL workgroup size)

It is developer responsibility to define OpenCL kernel ABI and pass **compatible** arguments to these custom kernel. OpenCV doesn't not verify passed arguments (some check still be done by OpenCL runtime itself).

OpenCV's entity for OpenCL kernel is cv::ocl::Kernel. Kernel can be instantiated from `cv::ocl::Program` object.

Kernel arguments API is handled by cv::ocl::KernelArg class. These kind of arguments can be passed to OpenCL kernel:

- Constants: integers, float/doubles scalars, including vectorized variants (`float4`)

  Fills single argument of OpenCL Kernel: `<TYPE> parameter1`

- UMat data information:
    - ReadOnly/WriteOnly/ReadWrite:

      `__global <TYPE>* ptr, int step, int offset, int rows, int cols`

      (complete set of parametets for 2D UMat)
    - ReadOnlyNoSize/WriteOnlyNoSize/ReadWriteNoSize:

      `__global <TYPE>* ptr, int step, int offset`

      (reduced set of parametets for 2D UMat)
    - PtrReadOnly/PtrWriteOnly/PtrReadWrite:

      `__global <TYPE>*`

      (other parameters can be passed as constants separatelly)

How to example

1. Check that OpenCL device is available:

   ```
   cv::ocl::Context ctx = cv::ocl::Context::getDefault();
   if (!ctx.ptr())
   {
       cerr << "OpenCL is not available" << endl;
       return 1;
   }
   ```

   To compile kernels from source code we need OpenCL online compiler:

   ```
   cv::ocl::Device device = cv::ocl::Device::getDefault();
   if (!device.compilerAvailable())
   {
       cerr << "OpenCL compiler is not available" << endl;
       return 1;
   }
   ```

   For SPIR kernels we should check "cl_khr_spir" extension:

   ```
   if (!device.isExtensionSupported("cl_khr_spir"))
   {
       cerr << "'cl_khr_spir' extension is not supported by OpenCL device" <<
   endl;
       return 1;
   }
   ```

2. Define OpenCL program source, where `opencl_kernel_src` is null-terminated string with kernel sources (for example, read from file via `std::fstream`).

   ```
   cv::ocl::ProgramSource source("", "custom_program_sample", opencl_kernel_src,
   "");
   ```

For SPIR kernels we need to pass address and length of program binaries in SPIR format. Example for `std::vector<char> program_binary_code`:

```
cv::ocl::ProgramSource source = cv::ocl::ProgramSource::fromSPIR(
      "", "custom_program_sample_spir",
      (uchar*)&program_binary_code[0], program_binary_code.size(), "");
```

3. Compile/build OpenCL program for current OpenCL device:

```
cv::String errmsg;
cv::ocl::Program program(source, "", errmsg);
if (program.ptr() == NULL)
{
    cerr << "Can't compile OpenCL program:" << endl << errmsg << endl;
    return 1;
}
```

4. Get OpenCL kernel by name

```
cv::ocl::Kernel k("magnutude_filter_8u", program);
if (k.empty())
{
    cerr << "Can't get OpenCL kernel" << endl;
    return 1;
}
```

5. Pass kernel arguments and launch kernel via `run()` method:

```
size_t globalSize[2] = {(size_t)src.cols, (size_t)src.rows};
size_t localSize[2] = {8, 8};
bool executionResult = k
    .args(
        cv::ocl::KernelArg::ReadOnlyNoSize(src), // size is not used (similar
to 'dst' size)
        cv::ocl::KernelArg::WriteOnly(result),
        (float)2.0
    )
    .run(2, globalSize, localSize, true);
if (!executionResult)
{
    cerr << "OpenCL kernel launch failed" << endl;
    return 1;
}
```

Note: OpenCL kernel doesn't perform any memory allocations - all UMat buffers must be pre-allocated before lauch.

Complete sources of example is available here