OpenCV Graph API

Overview and programming by example

Dmitry Matveev
Intel Corporation

December 24, 2018

Outline

G-API: What is, why, what’s for?

OpenCV evolution in one slide

Version 1.x – Library inception

- Just a set of CV functions + helpers around (visualization, IO);

Version 2.x – Library rewrite

- OpenCV meets C++, `cv::Mat` replaces `IplImage*`;

Version 3.0: – Welcome Transparent API (T-API)

- `cv::UMat` is introduced as a *transparent* addition to `cv::Mat`;
- With `cv::UMat`, an OpenCL kernel can be enqeueud instead of immediately running C code;
- `cv::UMat` data is kept on a *device* until explicitly queried.

OpenCV evolution in one slide (cont'd)

Version 4.0: – Welcome Graph API (G-API)

- A new separate module (not a full library rewrite);
- A framework (or even a *meta*-framework);
- Usage model:
  - *Express* an image/vision processing graph and then *execute* it;
  - Fine-tune execution without changes in the graph;
- Similar to Halide – separates logic from platform details.
- More than Halide:
  - Kernels can be written in unconstrained platform-native code;
  - Halide can serve as a backend (one of many).

Why G-API?

Why introduce a new execution model?

- Ultimately it is all about optimizations;
  - or at least about a *possibility* to optimize;
- A CV algorithm is usually not a single function call, but a composition of functions;
- Different models operate at different levels of knowledge on the algorithm (problem) we run.

Why G-API? (cont'd)

Why introduce a new execution model?

- Traditional – every function can be optimized (e.g. vectorized) and parallelized, the rest is up to programmer to care about.
- Queue-based – kernels are enqueued dynamically with no guarantee where the end is or what is called next;
- Graph-based – nearly all information is there, some compiler magic can be done!

What is G-API for?

Bring the value of graph model with OpenCV where it makes sense:

- Memory consumption can be reduced dramatically;
- Memory access can be optimized to maximize cache reuse;
- Parallelism can be applied automatically where it is hard to do it manually;
  - It also becomes more efficient when working with graphs;
- Heterogeneity gets extra values like:
  - Avoiding unnecessary data transfers;
  - Shadowing transfer costs with parallel host co-execution;
  - Increasing system throughput with frame-level pipelining.

Programming with G-API

G-API Concepts

- Graphs are built by applying *operations* to *data objects*;
  - API itself has no "graphs", it is expression-based instead;
- Data objects do not hold actual data, only capture *dependencies*;
- Operations consume and produce data objects.
- A graph is defined by specifying its *boundaries* with data objects:
  - What data objects are *inputs* to the graph?
  - What are its *outputs*?

A code is worth a thousand words

Traditional OpenCV

```
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>

#include <opencv2/highgui.hpp>

int main(int argc, char *argv[]) {
    using namespace cv;
    if (argc != 3) return 1;

    Mat in_mat = imread(argv[1]);
    Mat gx, gy;

    Sobel(in_mat, gx, CV_32F, 1, 0);
    Sobel(in_mat, gy, CV_32F, 0, 1);

    Mat mag, out_mat;
    sqrt(gx.mul(gx) + gy.mul(gy), mag);
    mag.convertTo(out_mat, CV_8U);

    imwrite(argv[2], out_mat);
    return 0;
}
```

OpenCV G-API

```
#include <opencv2/gapi.hpp>
#include <opencv2/gapi/core.hpp>
#include <opencv2/gapi/imgproc.hpp>
#include <opencv2/highgui.hpp>

int main(int argc, char *argv[]) {
    using namespace cv;
    if (argc != 3) return 1;

    GMat in;
    GMat gx  = gapi::Sobel(in, CV_32F, 1, 0);
    GMat gy  = gapi::Sobel(in, CV_32F, 0, 1);
    GMat mag = gapi::sqrt(  gapi::mul(gx, gx)
                          + gapi::mul(gy, gy));
    GMat out = gapi::convertTo(mag, CV_8U);
    GComputation sobel(GIn(in), GOut(out));

    Mat in_mat = imread(argv[1]), out_mat;
    sobel.apply(in_mat, out_mat);
    imwrite(argv[2], out_mat);
    return 0;
}
```

A code is worth a thousand words (cont'd)

What we have just learned?

- G-API functions mimic their traditional OpenCV ancestors;
- No real data is required to construct a graph;
- Graph construction and graph execution are separate steps.

What else?

- Graph is first *expressed* and then *captured* in an object;
- Graph constructor defines *protocol*; user can pass vectors of inputs/outputs like

  `cv::GComputation(cv::GIn(...), cv::GOut(...))`
- Calls to `.apply()` must conform to graph's protocol

On data objects

Graph protocol defines what arguments a computation was defined on (both inputs and outputs), and what are the shapes (or types) of those arguments:

Shape	Argument	Size
`GMat`	`Mat`	Staitic; defined during graph compilation
`GScalar`	`Scalar`	4 x `double`
`GArray<T>`	`std::vector<T>`	Dynamic; defined in runtime

`GScalar` may be value-initialized at construction time to allow expressions like `GMat a = 2*(b + 1)`.

Customization example

Tuning the execution

- Graph execution model is defined by kernels which are used;
- Kernels can be specified in graph compilation arguments:

```
#include <opencv2/gapi/fluid/core.hpp>
#include <opencv2/gapi/fluid/imgproc.hpp>
...
auto pkg = gapi::combine(gapi::core::fluid::kernels(),
                         gapi::imgproc::fluid::kernels(),
                         unite_policy::KEEP);
sobel.apply(in_mat, out_mat, compile_args(pkg));
```

- OpenCL backend can be used in the same way.

Operations and Kernels

Specifying a kernel package

- A kernel is an implementation of operation (= interface);
- A kernel package hosts kernels that G-API should use;
- Kernels are written for different backends and using their APIs;
- Two kernel packages can be merged into a single one;
- User can safely supply his own kernels to either *replace* or *augment* the default package.
  - Yes, even the standard kernels can be *overwritten* by user from the outside!
- Heterogeneous kernel package hosts kernels of different backends.

Operations and Kernels (cont'd)

Defining an operation

- A type name (every operation is a C++ type);
- Operation signature (similar to `std::function<>`);
- Operation identifier (a string);
- Metadata callback – desribe what is the output value format(s), given the input and arguments.
- Use `OpType::on(...)` to use a new kernel `OpType` to construct graphs.

```
G_TYPED_KERNEL(GSqrt,<GMat(GMat)>,"org.opencv.core.math.sqrt") {
    static GMatDesc outMeta(GMatDesc in) { return in; }
};
```

Operations and Kernels (cont'd)

Implementing an operation

- Depends on the backend and its API;
- Common part for all backends: refer to operation being implemented using its *type*.

OpenCV backend

- OpenCV backend is the default one: OpenCV kernel is a wrapped OpenCV function:

```
GAPI_OCV_KERNEL(GCPUSqrt, cv::gapi::core::GSqrt) {
    static void run(const cv::Mat& in, cv::Mat &out) {
        cv::sqrt(in, out);
    }
};
```

Operations and Kernels (cont'd)

Fluid backend

- Fluid backend operates with row-by-row kernels and schedules its execution to optimize data locality:

```
GAPI_FLUID_KERNEL(GFluidSqrt, cv::gapi::core::GSqrt, false) {
    static const int Window = 1;
    static void run(const View &in, Buffer &out) {
        hal::sqrt32f(in .InLine <float>(0)
                     out.OutLine<float>(0),
                     out.length());
    }
};
```

- Note `run` changes signature but still is derived from the operation signature.

Understanding the "G-Effect"

Understanding the "G-Effect"

What is "G-Effect"?

- G-API is not only an API, but also an *implementation*;
  - i.e. it does some work already!
- We call "G-Effect" any measurable improvement which G-API demonstrates against traditional methods;
- So far the list is:
  - Memory consumption;
  - Performance;
  - Programmer efforts.

Note: in the following slides, all measurements are taken on Intel® Core™-i5 6600 CPU.

Understanding the "G-Effect"

Memory consumption: Sobel Edge Detector

- G-API/Fluid backend is designed to minimize footprint:

Input	OpenCV MiB	G-API/Fluid MiB	Factor Times
512 x 512	17.33	0.59	28.9x
640 x 480	20.29	0.62	32.8x
1280 x 720	60.73	0.72	83.9x
1920 x 1080	136.53	0.83	164.7x
3840 x 2160	545.88	1.22	447.4x

- The detector itself can be written manually in two `for` loops, but G-API covers cases more complex than that;
- OpenCV code requires changes to shrink footprint.

Performance: Sobel Edge Detector

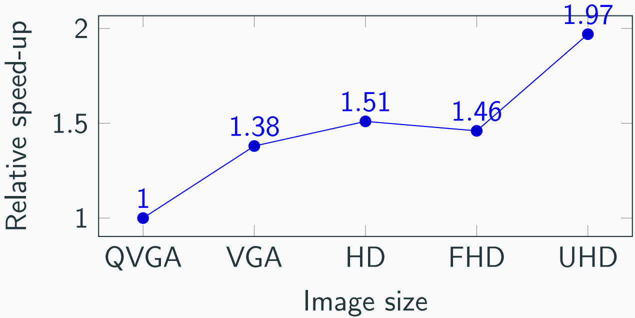
- G-API/Fluid backend also optimizes cache reuse:

Input	OpenCV ms	G-API/Fluid ms	Factor Times
320 x 240	1.16	0.53	2.17x
640 x 480	5.66	1.89	2.99x
1280 x 720	17.24	5.26	3.28x
1920 x 1080	39.04	12.29	3.18x
3840 x 2160	219.57	51.22	4.29x

- The more data is processed, the bigger "G-Effect" is.

Understanding the "G-Effect"

Relative speed-up based on cache efficiency



The higher resolution is, the higher relative speed-up is (with speed-up on QVGA taken as 1.0).

Resources on G-API

Resources on G-API

Repository

- `https://github.com/opencv/opencv` (see `modules/gapi`)
- Integral part of OpenCV starting version 4.0;

Documentation

- `https://docs.opencv.org/master/d0/d1e/gapi.html`
- A tutorial and a class reference are there as well.

Thank you!