Video Processing

Continuous Online Video Processing

Introduction

The current generation of sCMOS cameras can capture full-frame resolution video at either 30 fps or 100 fps, depending on the data interface between camera and computer (USB3.0 or CameraLink). At 16-bits per pixel and 2048x2048 pixels, the maximum data rate for the USB3.0 camera is 240 MB/s. Imaging sessions typically last 30-minutes or less. However, pixels are typically binned down 2x2, and frame rate often reduced; processing speed and storage constraints are the primary motivation for doing so. The effect of doubling resolution on processing time when using the graphics card is nearly negligible, however. By identifying ROIs online and extracting the traces of neural activity allows us to discard acquired images and instead store the traces only, or feed them into an encoder for online analysis.

Graphics Processing Units were traditionally developed for the consumer gaming market. They are optimized for the process which involves translating a continuous stream of information into a two-dimensional image format for transfer to a computer monitor. In the context of gaming, the stream of information received by a GPU describes the state of objects in a dynamic virtual environment, and is typically produced by a video game engine. These processors are highly optimized for this task. However, they are equally efficient at performing the same type of procedure in reverse – reducing a stream of images to structured streams of information about dynamic objects in the image – and thus are popular for video processing and computer vision applications.

Any GPU architecture will consist of a hierarchy of parallel processing elements. NVIDIA's CUDA architecture refers to the lowest level processing element as "CUDA Cores" and the highest level as "Symmetric Multiprocessors." Typically data is distributed across cores and multiprocessors by specifying a layout in C-code using different terminology, "threads" and "blocks." Blocks are then said to be organized in a "grid." Adapting traditional image processing or computer vision algorithms to run quickly on a GPU involves finding a way to distribute threads efficiently, ideally minimizing communication between blocks.

MATLAB makes processing data using the GPU seemingly trivial by overloading a large number of built in functions. Performance varies, however, and often the fastest way to implement a routine is by writing a kernel-type subfunction – written as if it operates on single (scalar) elements only – that can be called on all pixels at once, or all pixel-subscripts, which the function can then use to retrieve the pixel value at the given subscript. The kernel-type function is compiled into a CUDA kernel the first time it's called, then repeated calls call the kernel directly, having minimal overhead. Calls go through the arrayfun()

function.

Data transfers between system memory and graphics memory is often the major bottle-neck. Therefore, this operation is best performed only once. However, once data is on the GPU, many complex operations can be performed to extract information from the image, all while staying under the processing-time limit imposed by the frame-rate of the camera sending the images.

Method and Approach

The entire procedure for processing images and extracting cell signals can be performed in substantially less time than most commonly available tools using the approach described in Aim 1, particularly the methods for restricting the spatial extent of pixel-association operations, and distributing operations across parallel processing cores using a Single Program Multiple Data (SPMD) archetype. However, the total time still exceeds that of the acquisition session. Inefficiency arises from the overhead involved with distributing data and passing information between separate parallel processes. Graphics cards, however execute in what's called Single Instruction Multiple Data (SIMD) fashion, to distribute computation across the thousands of processing cores.

The processing components are implemented using the MATLAB System-Object framework, which allows for slightly faster performance through internal optimizations having to do with memory allocation. Most system objects, each representing one step in the serial processing and signal-extraction procedure, also have companion functions that implement the computation-heavy components of each algorithm using a pre-compiled CUDA kernel.

Benchmarking & General Performance

Built-in MATLAB functions that execute on the GPU can be profiled with benchmarking functions like *gputimeit()*, or with the *tic/toc* functions. When execution isn't fast enough, they need to be replaced with custom functions. The custom functions typically achieve the speed up necessary by enabling the operation to carried out on several frames at once. This reduces the over-head costs inposed for each function call by spreading it over several frames. This solution is not ideal, as it increases the latency of solutions, however does not preclude implementation in real-time system if the procedures are adapted to run on a real-time hybrid system-on-module like NVIDIA's Tegra X1, which should involve minimal effort once a standard set of successful procedures is realized. The current implementation tests the processing time of each stage of the process to ensure that the sum is less than the acquisition time for each frame dictated by the inverse of the frame-rate (30-50 milliseconds).

Buffered Operations

Combining frames for each operation can result in near linear speedup. For example, for the phase-correlation step required for motion correction, the FFT and IFFT are called on 16 image-frames at once, and the time take to accomplish is approximately the same as if the operation were called on 1 frame. This essentially leads to a 16x speedup, though the latency is also increased slightly. The best size to use is difficult to pre-determine, and typically must be measured for varying size 'chunks' using the benchmarking functions indicated above. The system objects manage the details necessary to allow buffered chunks of video to be passed to each stage without introducing artifacts at the temporal edges between chunks.

Image Pre-Processing & Motion Correction

Pre-processing is implemented as with the offline procedure, with a few changes. Images are aligned in chunks, and they are aligned sequentially to two templates. One template is the most recent stable frame from the preceding chunk. The other is a recursively temporal-low-pass filtered image that mitigates slow drifts. Aligning to the first template is usually more stable as the brightness of cells in the recent image will be more similar to those in the current chunk than will be the brightness of cells in the slow-moving average.

The displacement of each frame is found to sub-pixel precision, then used with a custom bicubic resampling kernel that replaces any pixels at the edges with images from the moving average.

Sequential Statistics

A number of statistics for each pixel are updated online and can be used for normalization and segmentation procedures later in the process. These include the minimum and maximum pixel intensity, and the first four central moments, which are easily converted to the mean, variance, skewness, and kurtosis. The formulas for making these calculations are given below, and are performed in a highly efficient manner as data are kept local to each processing core, and repeat computations are minimized.

```
n = n + 1;
f = F(rowIdx,colIdx,k);
d = single(f) - m1;
dk = d/n;
dk2 = dk\^2;
s = d\*dk\*(n-1);

% UPDATE CENTRAL MOMENTS
```

```
m1 = m1 + dk;
m4 = m4 + s\*dk2\*(n.\^2-3\*n+3) + 6\*dk2\*m2 - 4\*dk\*m3;
m3 = m3 + s\*dk\*(n-2) - 3\*dk\*m2;
m2 = m2 + s;
% UPDATE MIN & MAX
fmin = min(fmin, f);
fmax = max(fmax, f);
```

Furthermore, the value used to update each central moment at each point in time can be used as a measure of change in the distribution of each pixel caused by the current pixel intensity, as explained next.

Non-Stationarity & Differential Moments

Stationary refers to the property of a signal such that its statistics do not vary over time, i.e. its distribution is stable. Neural signals tend to specifically *not* have this property, in contrast to other measurable components such as those contributed by physiologic noise (heart-rate, respirations, etc.). Thus, by analyzing the evolution of statistical measures calculated for each pixel as frames are added in sequence gives a highly sensitive indicator of neural activity. This is done using a routine analogous to that for updating central moments given above, except the values returned are not only the updated moment, but also the updating component – essentially the partial derivative with respect to time. This is illustrated below, including the normalization functions which convert the partial-moment values to their variance, skewness, and kurtosis analogues:

```
% COMPUTE DIFFERENTIAL UPDATE TO CENTRAL MOMENTS

dm1 = dk;

m1 = m1 + dm1;

dm4 = s*dk2*(n^2-3*n+3) + 6*dk2*m2 - 4*dk*m3;

dm3 = s*dk*(n-2) - 3*dk*m2;

dm2 = s;

m2 = m2 + dm2;

% NORMALIZE BY VARIANCE & SAMPLE NUMBER -> CONVERSION TO dVar, dSkew, dKurt

dm2 = dm2/max(1,n-1);

dm3 = dm3*sqrt(max(1,n))/(m2^1.5);

dm4 = dm4*n/(m2^2);
```

These functions run on images representing the image intensity, and also on images taken from sequential differences indicating the temporal derivative of image intensity. The combination of outputs from these operations indicate both when image intensities are significantly high relative to past distribution, and also when intensities are changing significantly faster than learned from their past distribution.

Surface Classification: Peaks, Edges, Curvature

Edge-finding methods are employed for establishing boundaries between cells, and first and second-order gradients are used to compute local measures of curvature from an eigenvalue decomposition of the local Hessian matrix. I won't go into detail, as the utility of these procedure in the most recent implementation has been lost, but nevertheless, the operation is optimized and ready to be plugged back in when further development calls for better accuracy informing cell-segmentation, or when a faster or more accurate motion-correction algorithm is called for.

Online Cell Segmentation & Tracking

Cells are segmented by first running sequential statistics on the properties of identifiable regions on a pixel-wise basis. That is, as regions are identified in a method similar to that used offline in Aim 1, the region-properties are calculated (Centroid, Bounding-Box, etc.) and statistics for these properties are updated at each pixel covered by a proposed region. After sufficient evidence has gathered, Seeds are generated by finding the local peak of a seed-probability function that optimizes each pixel's proximity to a region centroid, and distance from any boundary. Regions are grown from these seed regions, and registered in a hierarchy that allows for co-labeling of cellular and sub-cellular components. Newly identified regions occur as new seeds, where as seeds overlapping with old regions are used to identify sub-regions, or to track regions over time.

Signal Extraction from Subcellular Compartments

I also have functions for the extraction of normalized Pointwise-Mutual-Information (nPMI), which can operate on a pixel-to-pixel basis or on a region-to-pixel basis. This operation accumulates mutually informative changes in all pixels in the maximal bounding-box (e.g. 64x64 pixels) surrounding each identified regions centroid. The weights given by this function can take on values between -1 and 1, and can be used to inform any reduction operations to follow. Additionally, spatial moments can indicate the subcellular distribution of activity across the identified region. In this context, the first spatial moment M_{00} indicates the mean signal intensity.

User Interface for Parameter Tuning

Some system-objects also incorporate a user interface to aid in parameter selection for tuning. ## Compression: as a Tool, a Goal, as an Explanation

This section describes general compression algorithms as well as compression algorithms specifically tailored to application in video, accomplished by searching for both temporal and spatial redunancy.

Data Management

Essential to the long-term success of any operation is an effective strategy for the management I've all data coming in and out of the system. Especially important is the insured facility for continued access and translation of data as systems evolve or as data is passed to other infrastructure. Strategies that meet the need of data generators and users often if it's somewhere on a spectrum of strictly defined data structures that minimize errors but are strict and difficult to evolve, versus those that require verbose documentation which hinders performance but are adaptable.

The new structures that scientists typically use are on the flexible and self-describing Spectrum and, these include two files for large image formats, fits files, cifs, Matlab structure definitions, which are an implementation of hdf5 file structures. And similarly standard JavaScript object notation in parentheses Json structures. At the other end exist Raw structures compression codex and other processing procedures, and cannot be accessed reliably without thorough knowledge of the source code for napi, or extensive packing and guessing. In the middle exist and number of serialization formats such as Google protobuf format, Thrift, Avril, Captain T, ND growing number of other formats that try to offer flexibility in evolution over time as well as variable levels of self-description and performance. These formats are often to find in simple files and are used by code generators 2 produce computational e efficient code for serialization and D serialization without losing cross platform compatibility or risking rapid deprecation as the needs for the format evolve with the needs of the users. Types can be added to provide new functionality without damaging backwards compatibility. Additionally this model is off in accompanied by a translator herbal component to JavaScript object notation which makes the format ultimately universal in the case of a desperate need for compatibility when performance no longer matters

Tags: data, directory, format, programming, serialization ## Data Scaling

This section describes the reality of how data scales as more and more sensors are added. Typically we humans think linearly, but the ramifications of increasing something like sensor size is often exponential. Furthermore, many operations we perform have costs that scale exponentially, and in my humble opinion are not even worth attempting if any such procedure must applied continuously on a data=stream. ## Distributed Dataflow and Streaming

Language: Is MATLAB the best tool for this job?

- Standard language in many engineering programs
- Proprietary
- Performance
- Compatibility
- Need for a "SandBox"
- Lacks modularity
-

Alternatives Languages

- Python
- C/C++
- Java/Scala
- Javascript/Node
- GO, Haskell

Databases

Big Data

- not exactly...
- disparate simple queries across

Computational Models

- Dataflow Processing
- Actors model
- Petri Nets
- Graph Processing - i.e. Tensorflow

Information and Informativity

This section presents a background in information theory in the context of behavior sensors and video streams. I further elaborate on what types of information neuroscientists and healthcare providers would hope to extract from these data, focusing on a larger picture of connecting the process of extraction directly the ultimate application.

Whereas previous attempts to process data focus largely on putting it in an intermediate state that is workable with current limitations on computation, visualization, and developer interaction with the data-streams, we now have an opportunity to skip intermediate states and develop with applications in mind.

Transition into compression by showing that previous attempts were essentially compression algorithms developed specifically for the data and preconceived notions about results, but that we can also apply very well developed general compression algorithms, with slight modifications to extract features such as localized bitrate that quantify (in an unbiased way) information content in an selectable region of a video-frame as it changes over time.