Hardware / software joint design enables high quality imaging technologies

Antony C. Chan

May 13, 2021

This article argues for *concurrent* algorithm-hardware engineering effort, in order to deliver user-driven image quality requirements. Depending on the extent of algorithm to imaging hardware coupling, the R&D project can be classified into four tiers. The author further elaborate the workflow, borrowing examples from the Caltech x Amgen 96-in-1 parallel microscope design project. Through the case studies of Tier 4, 3, and 1 project scopes, it is shown that the algorithm development workflow can adapt accordingly. The article ends with the emphasis that continuous communication with both the end-users and the hardware OEMs are essential for the project success.

Background

Four Tier classification of imaging algorithm design

Imaging algorithm R&D, by definition, is the cross-disciplinary study to understand how optics, image formation, sensing, and information theory interact to purse well-rounded image quality. In scientific work, for reasons unknown, seems to have estranged themselves from the imaging optics design and characterization effort. Perhaps it stems from the fact that it takes decade of experiences just to master software coding and modeling, in order to write good algorithms.

However, it is worth to point out that *concurrent engineering design* of the imaging hardware plays an important role to obtain high-quality measurements. Illustrated below is the over-simplified block diagram of an imaging system; some components (e.g. light source, fluorescence emission wavelength) are a non-negotiable design, while others (e.g. numerical aperture, sensor pixel size) may be adaptable to algorithmic design changes.

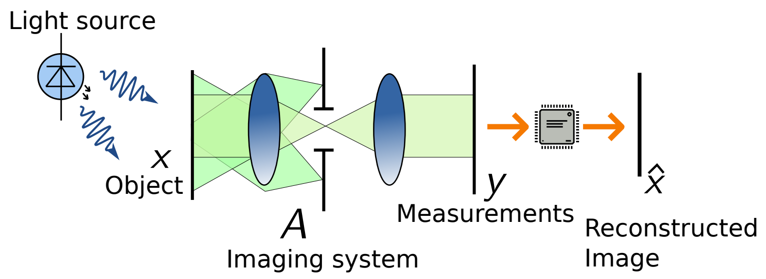

Figure 1: An over-simplified imaging system schematic. A custom light source strikes the object x, which in turn scatters or emits lights. The imaging system A captures the emissions, which is then sensed by the imaging sensors as measurements y.

Depending how much an Algorithm R&D Engineer can influence the system design as a whole, the corresponding product, the algorithm, can be classified into four tiers:

Tier 1: computational optics. (the algorithm) requires full customization of illumination, sample preparation recipe, imaging, and sensing optics, e.g structured illumination microscopy (SIM), acousto-optic fluorescence microscopy;

Tier 2: computational photography requires customization of imaging optics and algorithm, e.g. hyperspectral imaging, light-field photography (Lytro), multi-frame super-resolution (MFSR);

Tier 3: ImageJ / photo-retouching requires custom image restoration algorithm design, and no influence on the imaging system hardware specifications. For example, single-frame patch-based image denoising;

Tier 4: Programming / refactoring. No algorithm or hardware customization required, e.g. Brenner focus score, gamma adjustment, local Laplacian filter.

Algorithm design as part of concurrent HW/SW/Optics engineering

As the saying goes, *form follows function*; ideally the algorithm should co-evolve with the rest of the imaging system (the forms) in order to deliver promising results to the end-users (the functions).

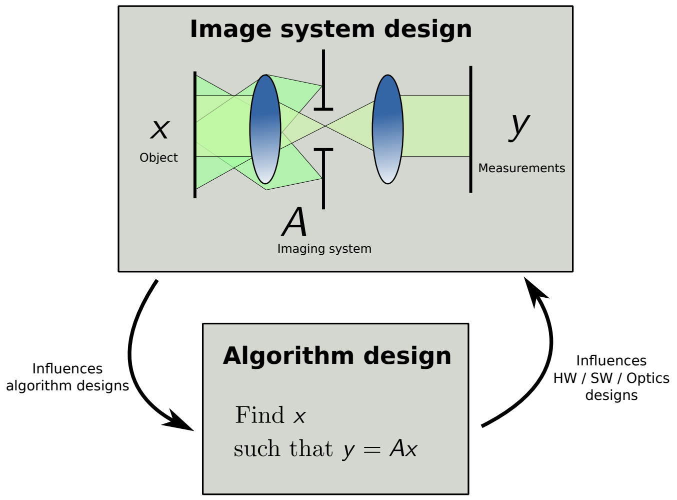

Figure 2: Imaging algorithm design as the integrated step of the imaging optics design. Image courtesy of Prof. Laura Waller at UC Berkeley.

From the perspective of an R&D engineer, it takes the following steps:

1. Book the flight;

2. (Rationale) talk to the end-user to understand the prime project objectives;
3. (Component level specifications) talk to the engineers who actually build the instrument on site;
4. (System level specifications) Model the problem in terms of wet-chemistry, optics, and imaging; quantify the project objectives;
5. (Implementations) Draft an algorithm to solve the model;
6. (Optimization) Do it well; if it fails to meet the user requirement, return to Step 1.

Case studies: High-throughput parallel microscopy with 96 mini-microscopes (96 eyes)

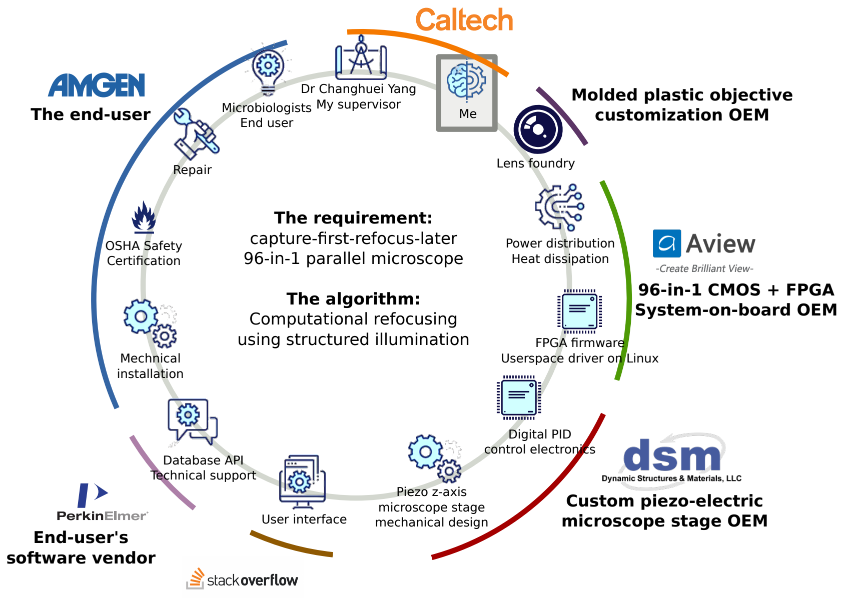
Caltech
AMGEN
The end-user
Dr Changhuei Yang
My supervisor
Microbiologists
End user
Me
Molded plastic objective customization OEM
Lens foundry
Repair
OSHA Safety Certification
The requirement:
capture-first-refocus-later
96-in-1 parallel microscope
The algorithm:
Computational refocusing
using structured illumination
Power distribution
Heat dissipation
Aview
-Create Brilliant View-
96-in-1 CMOS + FPGA System-on-board OEM
Mechnical installation
FPGA firmware
Userspace driver on Linux
Digital PID control electronics
dsm
Dynamic Structures & Materials, LLC
Custom piezo-electric microscope stage OEM
Database API
Technical support
User interface
Piezo z-axis microscope stage mechanical design
PerkinElmer
End-user's software vendor
stackoverflow

Figure 3: Caltech's 96-in-1 parallel microscope is a joint HW/SW project to implement the computation refocusing algorithm using structured illumination, also known as Fourier Ptychography.

Tier 4 algorithm example: refactoring the focus metric algorithm

Talk to the end-users. The end-users, scientists in the Amgen HCS genomics group, have already purchased software license of the PerkinElmer's Columbus image analysis solution. They were able to program the server to compute the specific choice of focus metric, *Power log-log slope*, with the Columbus-specific Java macro. However, the script took more than 20 hours to complete. Amgen scientists would like the author to replicate the algorithm on the 96-eye GPU cluster.

Talk to the OEMs. PerkinElmer explained that the 20 hour delay is caused by two issues:

Data too big our 96-eye instrument produces 10x more images (=1 TB per plate) than what PerkinElmer can handle;

Too slow The Java-like macro, unlike the "stock" algorithms, runs on single-thread CPU; and

Beyond their pay grade Amgen's subscription is not sufficient enough for PerkinElmer to upgrade the macro syntax to run in parallel.

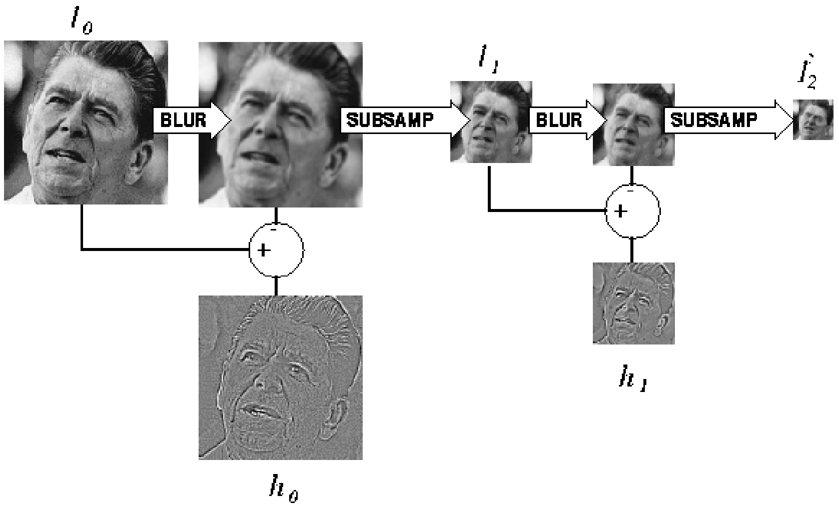

Figure 4: The Laplacian image pyramid is computed by subtracting the downsampled image with the blurred version. The focus metric, *power log-log slope*, computes the ratio of the signal intensity in logarithmic scale.

Model the problem. There isn't a lot of room to model the problem, as the algorithm specification is already firm. It is defined as:

$$h_i(z) = I_i(z) - I_{i-1}(z) \tag{1}$$

$$\mathrm{PLLS}(z) = \frac{\log \|h_3(z)\| - \log \|h_1(z)\|}{\log(2^3) - \log(2^1)}, \tag{2}$$

where $I_I(z)$ represents the image plane at position z that is downsampled by 3 times.

The algorithm development objective is to ensure the algorithm performance

$$\frac{T_{\text{Halide}}}{T_{\text{PerkinElmer}}} = \frac{T_{\text{GPU}}}{T_{\text{CPU}}}. \tag{3}$$

is fully exploited on the 96-eye compute engine.

Do it well. Here, the author could have directly refactored the PerkimElmer Java macro into the Nvidia CUDA language. In fact, the industry accepted approach is to follow the model-based algorithm development workflow. The model-based workflow ensures the *separation of concerns* between the coding effort in the individual hardware layers.

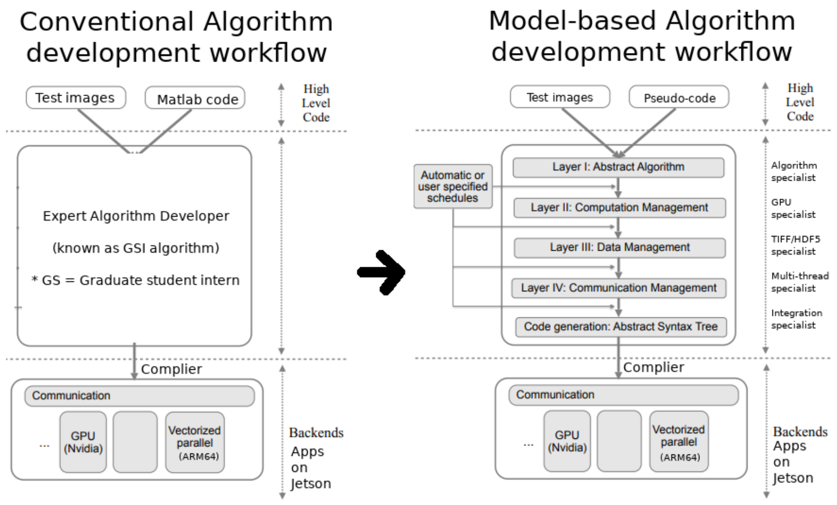
Conventional Algorithm development workflow
Test images
Matlab code
High Level Code
Expert Algorithm Developer
(known as GSI algorithm)
* GS = Graduate student intern
Complier
Communication
GPU (Nvidia)
Vectorized parallel (ARM64)
Backends Apps on Jetson
Model-based Algorithm development workflow
Test images
Pseudo-code
Automatic or user specified schedules
Layer I: Abstract Algorithm
Layer II: Computation Management
Layer III: Data Management
Layer IV: Communication Management
Code generation: Abstract Syntax Tree
Algorithm specialist
GPU specialist
TIFF/HDF5 specialist
Multi-thread specialist
Integration specialist

Figure 5: (Left) The conventional Matlab-to-CUDA algorithm development workflow, in comparison with (Right) model-based algorithm development workflow. Photo courtesy of the Google's Triamisu complier.

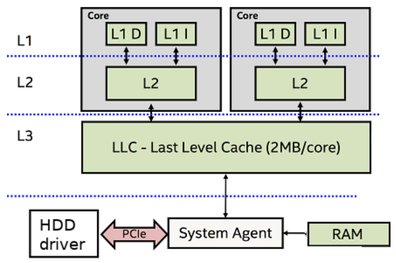

Figure 6: Intel skylake CPU family architecture. The model-based algorithm developement workflow enables the separation of concerns between compute instructions, data cache on CPU, as well as data layout on the RAM and hard disk drive (HDD).

Layer 0: Mathematics Derive the Mathematics of the algorithm, in terms of linear algebra (known as 2-stage convolution and reduction);

Layer 1: Algorithm Implement the algorithm with the *Halide algorithm* language in C++; simulate and validate the reconstructed images using low-end CPUs;

Layer 2: Computation Describe how we wish to accelerate the algorithm on one single GPU, with the *Halide schedule* language in C++;

layer 3: Data layout Describe how the image segments should be read in chunks from the physical hard drives; similarly, how to divide up the reconstructed images in chunk to speed up writing.

Layer 4: Communication Describe how we read the raw image segments in sequence; when to restore them on GPU, and when to write back to database.

Refer to the Appendix on the last page of the article for the Halide/C++ code snippets of the Layer 1 and Layer 2 implementations.

Tier 3 algorithm workflow: customized GPU cluster for high-throughput image reconstruction

Talk to the end-users. Amgen scientists evaluates an algorithm with a stepwise utility metric:

Score 3 if the sample-to-visualization time is within email checking time, i.e. around 5 min;

Score 2 if the time is within coffee/snake break, i.e. between 10 and 30 min;

Score 1 if the time aligns with employee's sign off hours, i.e. no less than 8 hours, no more than 12 hours;

Score 0 otherwise.

Talk to the OEMs. Moore's law automatically improves algorithm as technology advances. By waiting patiently for hardware customization, the author ended up more efficient program without any Mathematical change to the algorithm itself.

Nvidia GPU was willing to sell 4x GPU graphic cards (Tesla K80) at a discount;

Liquid cooling OEM can customize the GPU liquid cooling loop, if the author can send them the PCB footprint of the GPU boards;

Western digital hard drive OEM recommended the hybrid storage solution to store 16 Terabyte worth of data; 6x spinning hard drives on RAID5, read/write cache on nvme SSD;

Halide language research group delivered online free tutorial on how to refactor any image processing code using the model-based programming language, known as Halide-C++.

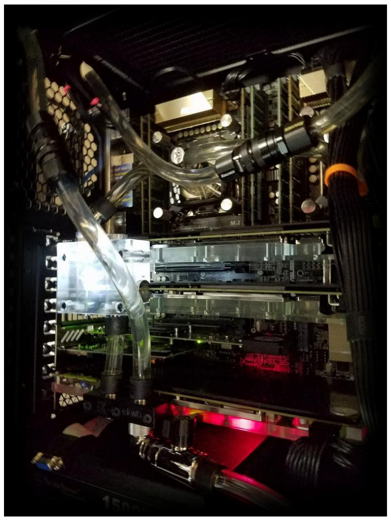
Figure 7: (Top) Customized GPU liquid cooling module; (Bottom) the Caltech's GPU farm, later delivered to Amgen.

Model the problem. The Caltech's image reconstruction algorithm, aims at minimizing the scalar value of the following cost function:

$$f(x, P) = \sum_i^N \| |F^{-1}[PQ_i F(x)]| - y\|_2^2 + \lambda \|x - 1\|_2^2, \quad \text{subject to } x \in \mathbb{C} \tag{4}$$

$$\hat{x} = \arg\min_{x,P} f(x, P). \tag{5}$$

where N is the number of brightfield and darkfield images captured during acquisition. The algorithm development objective is quantified as:

$$\text{minimize } \frac{T_{\text{Halide}}}{T_{\text{Matlab}}} = \frac{\rho_{\text{liquid cooling}}\rho_{\text{disk array}}}{N_{\text{GPU}}} \times \frac{T_{\text{GPU}}}{T_{\text{CPU}}} \times N \tag{6}$$

$$\text{maximize Score}[T_{\text{Halide}}] = \begin{cases} 3 & \text{if } T_{\text{Halide}} \leq 5\,\text{min} \\ 2 & \text{if } 10\,\text{min} \leq T_{\text{Halide}} \leq 30\,\text{min} \\ 1 & \text{if } 8\,\text{hour} \leq T_{\text{Halide}} \leq 12\,\text{hour} \\ 0 & \text{otherwise.} \end{cases} . \tag{7}$$

As shown above in the algorithm performance objective, the author can simply wait for a fully customized liquid cooling GPU cluster to get a free boost of the algorithm, all without one single line of code changes. The author made the engineering decision to spend 30% of the time to draft the GPU cluster specifications, 20% time to negotiate the customization costs, and 50% time to refactor existing code into the Halide language.

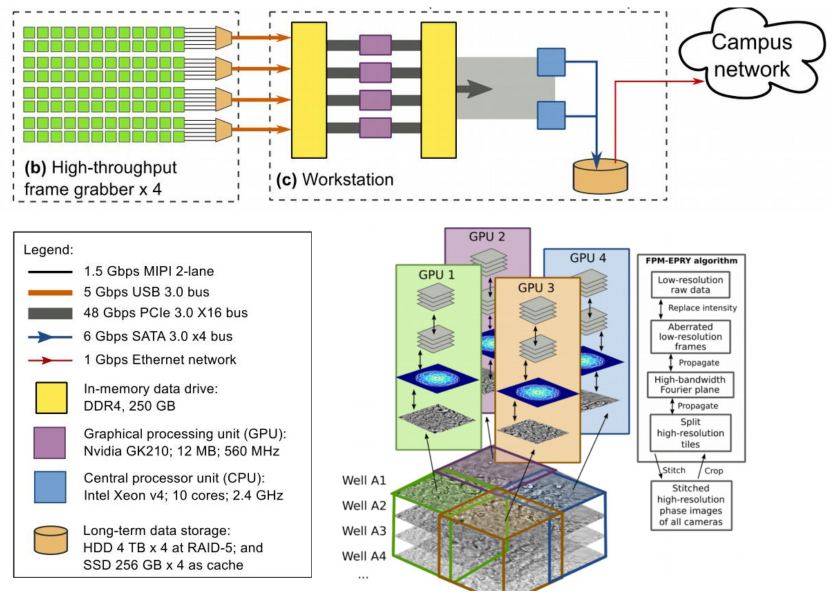
Campus network
(b) High-throughput frame grabber x 4
(c) Workstation
Legend:
1.5 Gbps MIPI 2-lane
5 Gbps USB 3.0 bus
48 Gbps PCIe 3.0 X16 bus
6 Gbps SATA 3.0 x4 bus
1 Gbps Ethernet network
In-memory data drive: DDR4, 250 GB
Graphical processing unit (GPU): Nvidia GK210; 12 MB; 560 MHz
Central processor unit (CPU): Intel Xeon v4; 10 cores; 2.4 GHz
Long-term data storage: HDD 4 TB x 4 at RAID-5; and SSD 256 GB x 4 as cache
GPU 1
GPU 2
GPU 3
GPU 4
Well A1
Well A2
Well A3
Well A4
...
FPM-EPRY algorithm
Low-resolution raw data
Replace intensity
Aberrated low-resolution frames
Propagate
High-bandwidth Fourier plane
Propagate
Split high-resolution tiles
Stitch
Crop
Stitched high-resolution phase images of all cameras

Figure 8: Compute hardware & software joint design. The algorithm implementation is tailor-made to run efficiently on the customized GPU cluster.

Do it well. This is how the author refactored Caltech's FPM algorithm, following the model-based algorithm implementation workflow, as described previously in the section *Tier 4 algorithm example*.

Tier 1 algorithm workflow: Caltech's FPM algorithm

Talk to the end-users The end-user, Amgen high-content genomic/phenotypic screening group, wished to capture microscope images at:

- 20x magnifications (i.e. 0.25 NA);

- 10+ plates per day, 96 wells per plate, 8,000 cells per well;
- visualize the images within the duration of "coffee break".

Existing commercial solutions, e.g. PerkinElmer, Nikon, were unable to meet the requirements. The author's supervisor at Caltech believed that his technology, *Fourier Ptychographic microscopy* (FPM), will enable high-throughput imaging via the capture-first-refocus-later feature.

Talk to the engineers who actually make the instruments I talked to various OEMs, here are the responses that I received:

Lens foundry 10-page CodeV optical simulations of the custom lens design reveals that the theoretical axial resolution is around 10 micrometer.

96-well plate manufacturer Molded-plastic is known to have +/- 150 micrometer height variation of each wells. Suggested Amgen to order those made with glass bottom and thermo-setting plastic sidewalls.

Piezo-electic microscope stage OEM Piezo-flexure stage has open-loop response time of 10 milliseconds, but close-loop

Inventor of Caltech's FPM algorithm The algorithm, properly calibrated against existing structured illumination system, can extend the optical axial resolution by 3x.

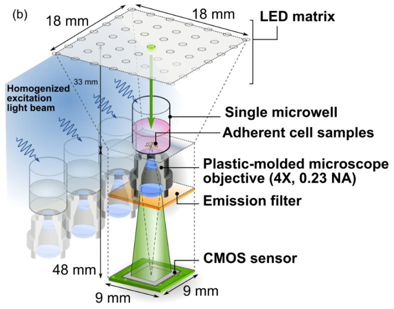

Figure 9: Schematic of the FPM microscope, repeated by 96 times.

Model the problem. The user-driven focusing requirement can be described in terms of statistical variation:

$$\sigma = \sigma_{\text{tip-tilt}} + \sigma_{\text{well plate curvature}} + \sigma_{\text{lens offset}}, \quad (8)$$

and the design goal is

$$\sigma \leq \Delta z_{\text{lens}} \times G_{\text{FPM}} = 10\,\mu m \times 3 = 30\,\mu m. \quad (9)$$

Upon the follow-up material sourcing meetings with various 96-well manufacturers, it turns out even the best plate material will always have curvature variation of $\approx \pm 35\,\mu m$, two times larger than our specifications.

Do it well. Amgen, upon recognizing the issue, delegated a staff HCS scientist to further expand the sourcing effort of the well plate curvature variation.

Our microscope stage OEM, on the other hand, is willing to accept the *Engineering change proposal* (ECP) to incorporate the adjustable

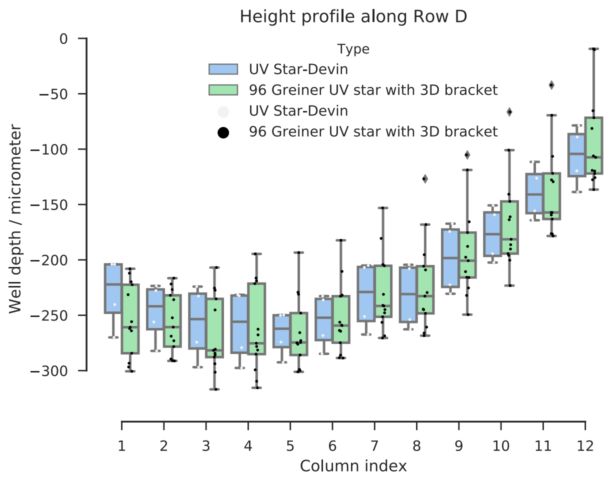

Figure 10: Vertical offsets of the 96-well plates (N=13, Groups=2).

kinematic mount on the underside of their stage, in order to eliminate the plate tip-tilt variation issue.

The author also up-adjusted the extended focus range from 3x to 4.5x, by reducing the number of darkfield images that are used in the FPM algorithm.

Finally, Amgen hired a system integration scientist, to manually calibrate the lens offsets to match the average well-plate curvature.

Concluding remarks

The author emphasized the benefit of algorithm-hardware *concurrent* design, so that the end-user's requirements can be met in the most efficient manner. The 96-in-1 parallel microscope project, a joint engineering feat of Caltech and Amgen, was selected the elaborate how algorithm can flourish, if it is given an opportunity to co-evolve with the imaging system optics, sensor, and the compute hardware. An algorithm R&D engineer, if needed be, can engage in pure code porting / refactoring effort (Tier 4) to accelerate existing (Matlab) program on specific compute architecture, via the *model-based* development workflow.

Appendix: Code examples

Layer 1: Downsample algorithm, described in Halide/C++

```
Func
autofocus::downsample(Func f) {
    Func downx{"downx"}, downy{"downy"};

    downx(x, y, z) = (f(2 * x - 2, y, z) + 4.0f * (f(2 * x - 1, y, z) + f(2 * x + 1, y, z)) +
                      6.0f * f(2 * x, y, z) + f(2 * x + 2, y, z)) /
                     16.0f;
    downy(x, y, z) =
        (downx(x, 2 * y - 2, z) + 4.0f * (downx(x, 2 * y - 1, z) + downx(x, 2 * y + 1, z), z) +
         6.0f * downx(x, 2 * y, z) + downx(x, 2 * y + 2, z)) /
        16.0f;
    return downy;
}
```

Layer 2: computation implementation with Halide/C++

```
if (target() == Target::x86_64) {

                // Use AVX SIMD instructions along x-axis
        downx.vectorize(x, 8,
                // If 8 does not evenly divide image width, recompute the blur, don't bother to cache partial results
                TailStrategy::FoldInwards)
                // Multithread along y-axis
                .parallel(y)
                // Compute in the inner loop of downy
                .compute_at(downy, x)
                // Cache the value in the outer loop of downy
                .store_at(downy, y);

                // Use AVX SIMD instructions along x-axis
        downy.vectorize(x, 8, TailStrategy::FoldInwards)
                // Multithread along y-axis
                .parallel(y);
}
else if(target() == Target::CUDA) {
        // GPU-specific schedules here
}
```

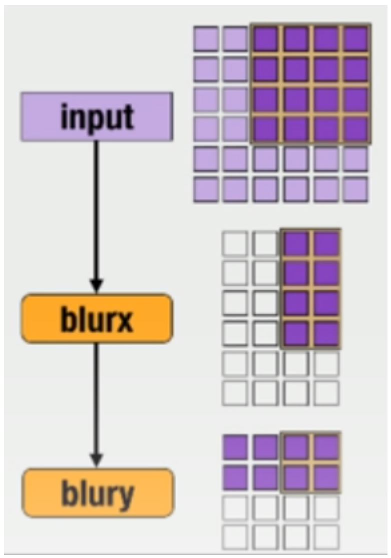

Figure 11: Graphical illustration of the two-stage blur algorithm. Downsampling operation is omitted for clarity. Photo courtesy of Andrew Adams from MIT.