2019 IEEE 30th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)

# OpenVX Graph Optimization for Visual Processor Units

Madushan Abeysinghe*, Jesse Villarreal†, Lucas Weaver‡, Jason D. Bakos§
*§University of South Carolina, †‡Texas Instruments
E-mail: *madushan@email.sc.edu, †jesse.villarreal@ti.com, ‡l-weaver@ti.com, §jbakos@cse.sc.edu

***Abstract*—OpenVX is a standardized, cross-platform software framework to aid in development of accelerated computer vision, machine learning, and other signal processing applications. Designed for performance optimization, OpenVX allows the programmer to define an application using a graph-based programming model, where the nodes are selected from a repertoire of pre-defined kernels and the edges represent the flow of successive images between pairs of kernels. The graph-based representation exposes spatial and temporal concurrency and provides tuning opportunities to the managing runtime library. In this paper, we present a performance model-based approach for optimizing the execution of OpenVX graphs on the Texas Instruments C66x Digital Signal Processor (DSP), which has similar characteristics to other widespread DSPs such as the Qualcomm Hexagon, Nvidia Programmable Vision Accelerator, and Google Visual Pixel Core. Our approach involves training performance models to predict the impact of tile size and node merging on performance and DRAM utilization. We evaluate our models against randomly-generated, valid, and executable OpenVX graphs.**

***Index Terms*—OpenVX, DSP, domain-specific, computer vision, tuning, optimization, embedded systems, DSL, DSA**

## I. Introduction

Image processing accelerators are now commonly integrated as part of embedded System-on-Chip (SoC) architectures. These types of accelerators go by different names, such as "Visual Processing Units (VPUs)" and "Image Processing Units (IPUs)," but are generally structured as Digital Signal Processor (DSPs)-type architectures, which themselves differ from general purpose processors and Graphics Processor Units (GPUs) in that they rely on compilation techniques to statically schedule all instructions, they have a complex instruction set, and use software-defined scratchpad memory and software-defined asynchronous pre-fetching and buffering of data blocks in the scratchpad using a direct memory access (DMA) controller as opposed to – or in addition to – traditional caches. Examples of such processors include the Google Pixel Visual Core [1], Qualcomm Hexagon DSP [2], Nvidia Programmable Vision Accelerator [3], and the Texas Instruments C66x DSP [4]. In addition, Field Programmable Gate Arrays (FPGAs) loosely fit this category when used with High Level Synthesis compilers [5], [6].

OpenVX is a code-portable and performance-portable open programming interface to aid in the design of accelerated signal processing subroutines, and has widespread support on a variety of coprocessor architectures [7]. It was originally conceived as a domain specific language (DSL) for image processing, but it is extensible to other domains. OpenVX relies on a graph-based model that allows the programmer to compose processing pipelines by assembling a collection of cooperative kernel primitives. This way, each graph node represents a kernel and each edge represents the flow of one image or video frame between kernels or as top-level inputs or outputs. This representation potentially allows the runtime environment to identify opportunities for optimization. Examples of OpenVX vendor implementations are Intel OpenVINO [8], Nvidia VisionWorks [9], AMD OpenVX (AMDOVX) [10], and Texas Instruments Vision SDK [11]. In this paper we target the Texas Instrument's TDA2x System-on-Chip and our baseline results are given by the release version of the Texas Instruments Vision SDK with OpenVX framework [12].

Kernels in an OpenVX graph exchange images, and a kernel cannot begin execution until it receives an image for each of its inputs. Executing an OpenVX graph on a DSP-type architecture requires that images be conveyed as a sequence of smaller fixed-sized tiles for the purpose of buffering in on-chip scratchpad memory. The size of the tiles impacts both the memory system performance and instruction throughput. Additionally, tiles associated with internal edges may optionally be kept in scratchpad only or exchanged with DRAM between kernel invocations.

This paper's contribution is the development of performance models to predict both DMA bandwidth and instruction throughput given features of input and output tiles. Using these models, we select weakly-connected sub-graphs that exchange tiles instead of whole images. This causes the runtime environment to schedule successive kernels after processing each tile as opposed to processing each image. We refer to this as "kernel merging". Our approach performs tile size selection for all individual kernels and groups of merged kernels.

To evaluate the potential performance impact of our tuning methodology, we use large sets of randomly-generated graphs and compare their predicted performance improvement over that of the baseline software. Using this methodology, we achieve a speedup of 1.53 on average, with average error rates of 13% and 8% for DMA and compute respectively.

## II. Background

Like other domain-specific processors, Visual Processor Units (VPUs) favor the use of software-controlled on-chip memories instead of traditional caches. A scratchpad offers several advantages over a traditional multi-level cache. A

2160-052X/19/$31.00 ©2019 IEEE
DOI 10.1109/ASAP.2019.00-19

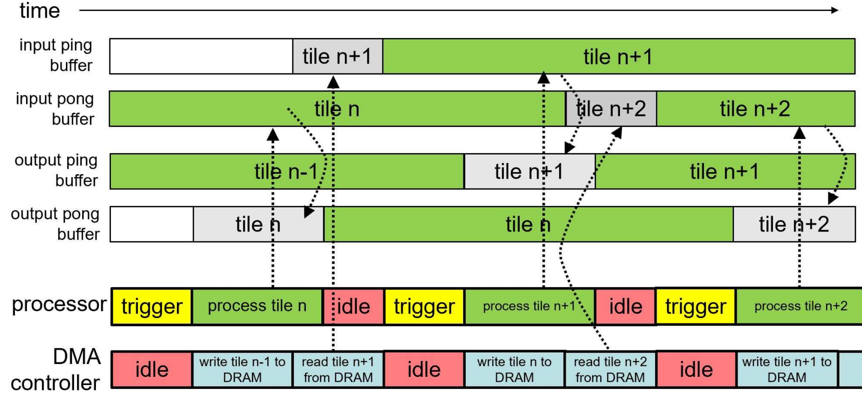


Fig. 1. Example timing diagram for double buffering execution on the TI C66x DSP. In this example, the execution is memory-bounded, since the time required for the DMA transfers exceeds the time required for processing one tile. "trigger time" is the time needed for the processor to initiate concurrent DMA transfers.

scratchpad combined with DMA controller operates asynchronously with the processor, allowing the programmer to overlap DRAM transfers with the corresponding compute workload. For applications where the access pattern is fixed, a scratchpad makes more efficient use of memory, as caches are susceptible to conflict misses when accessing multidimensional data. Scratchpads can also achieve higher throughput to off-chip memory by transferring larger blocks of consecutive words. For example, a scratchpad might transfer tiles of 128 by 96 four byte pixels, generating 512-byte consecutive transfers per row, larger than a typical cache line of 64 bytes.

The introduction of image tiling, DMA, and scratchpad into the OpenVX runtime on the Texas Instruments C66 DSP provided a benefit of a 2.3 speedup as compared to using cache [12]. This baseline implementation decomposes each input, output, and intermediate image into 64x48-pixel tiles, each of which is transferred between the on-chip L2 scratchpad and off-chip DRAM using the integrated DMA controller. The DSP cores in our target platform (the TDA2x) have an L2 memory size of 288 KB, which the program code (usually the bootloader) can configure as a partial hardware-controlled L2 cache and partial software-controlled scratchpad. For our experiments, it is configured as a 128 KB scratchpad SRAM with the rest as cache.

Since the DSP is an in-order architecture, the latency of data exchanges between the cache and DRAM cannot be hidden and, as a result, a cache miss always causes the DSP to idle until the miss is complete. On the other hand, the latency of exchanges between the scratchpad and DRAM may be hidden by exposing concurrency between the DSP core and DMA controller. In this way, the DSP processes tile $n$ concurrently with the outgoing transfer of output tile $n-1$ and incoming transfer of input tile $n+1$. This is referred to as "ping-pong buffering" or "double buffering". The scratchpad memory, while being an on-chip memory, exists at level 2 and is itself cached by a 32 KB 2-way set associative L1 cache.

When executing an OpenVX graph, all tiles comprising each of a kernel's output edges are transferred back from scratchpad to DRAM before the DSP begins executing any other kernel in the graph. For example, if a graph contains a kernel that has two inputs and one output, the software will allocate three scratchpad buffers–one for each edge–until the kernel has completed processing its output image and the DMA engine transfers the last of its tiles to DRAM. At this point, the software will re-allocate the scratchpad memory for the next kernel and start fetching tiles from DRAM.

As shown in Fig. 1, the software allocates two scratchpad buffers, ping and pong, for each input and output image of the kernel. Each buffer holds one tile and the software operates an outer loop that processes one tile. In each iteration, the software instructs the DMA controller to transfer the previously-computed output tile into DRAM and transfer the next input tile into scratchpad. After computing, the processor might need to wait until the DMA transfer completes before processing the next tile.

The "trigger time" refers to the time required for the processor to initiate DMA transfers and the "waiting time" refers to the difference in total DRAM transfer time and compute time. Note that for smaller tiles, the cumulative trigger time across the frame can be significant since there are more blocks to trigger per frame than for larger tiles. The tile size can vary significantly, since the pixel size varies from one to four bytes and some kernels have more inputs and outputs than others. If a kernel's compute time exceeds its DMA transfer time then it is compute bound and its waiting time will be near zero.

Tile size selection has a potentially significant effect on performance, and the vendor implementations currently offers limited support for automatically tuning this parameter. Also, although OpenVX's graph-based model decomposes the application into discrete operations, there are still opportunities for combining kernels within subgraphs to improve memory locality and avoid unnecessary transactions with off-chip memory.

For example, consider two connected kernels that each have one input and one output. During execution, each kernel requires that an input tile and output tile to be stored in scratchpad at any given time. The first kernel will transfer

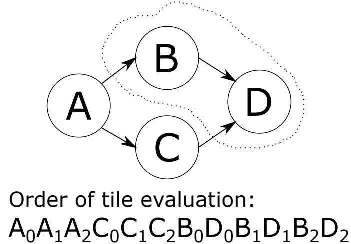


Fig. 2. A graph with nodes A, B, C, and D, nodes B and D are grouped, and the image is comprised of three tiles. $K_n$ denotes the execution of tile $n$ for node $K$. In this case, the DSP must alternate execution of tiles within the set of grouped nodes.

all its output tiles to DRAM before the second kernel begins retrieving each of these tiles for processing. Alternatively, if the first kernel conveys each of its output tiles as input to the second kernel through a scratchpad buffer and without transferring the tile to DRAM, the combination of both kernels will require that three total tiles are stored in scratchpad at any given time, comprising the input tile, intermediate tile, and output tile.

This approach reduces the total number of DMA transfers from three to two, which can improve performance if the sum of DMA transfer time of both kernels exceeds the sum of their execution time. However, storage of additional tiles will potentially limit the maximum tile size.

Also, as shown in Fig. 2, grouping kernels in this way increases the frequency at which the DSP must alternate its execution state between kernels, putting increased pressure on the L1 instruction cache. Likewise, processing multiple tiles at once increases the size of the working set, increasing pressure on the L1 data cache.

## III. Data-Driven Performance Models

As described in the previous section, the OpenVX programming model grants the runtime environment two degrees of freedom for optimization. The first is how to decompose the input, output, and intermediate images into tiles. DMA performance depends on characteristics of the DMA transfers, such as tile size and geometry. Also, since such architectures also have L1 data caches, tile size can also affect L1 data cache performance. The second is how to schedule the kernels onto the processor. The processor can execute the kernels in any order so long as it obeys the graph's data dependencies. The data dependencies are defined using the graph edges, but their granularity can be in terms of whole images or individual tiles. This way, the runtime environment can execute each kernel for a whole image or only a single tile. In the latter case, the processor must switch between kernels at a higher rate than in the former case, which potentially affects L1 instruction cache performance. We propose the use of DMA and compute performance models to facilitate achieving automated exploration of optimization decisions.

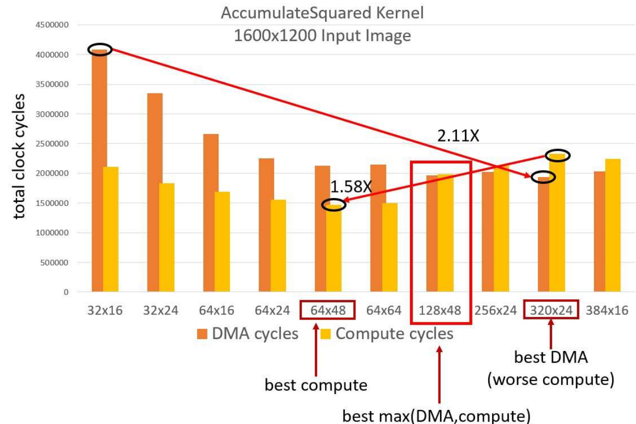


Fig. 3. DMA and compute cycles for AccumulateSquared Kernel for a Variety of Tile Sizes.

### A. Performance Bounds

Fig. 3 shows a comparison of the total number of DMA and compute cycles required to process a 1600x1200 pixel image for the AccumulateSquared OpenVX kernel for tile sizes ranging from 32x16 to 384x16. The best DMA performance is achieved at tile size 320x24, which is a 2.11 speedup as compared to the worst-performing tile size of 32x16. The best compute performance is achieved at 64x48, which has a speedup of 1.58 as compared to the worst-performing tile size of 320x24. Note that 320x24 is the worst-performing tile size for compute but the best for DMA. However, since the execution time is $max(time_{DMA}, time_{compute})$, the best performing tile size is 128x48.

The current release of the OpenVX implementation for the TDA2x sets all tile sizes to 64x48. Our results show that for individual kernels, the 64x48 tile sizes achieves within 21% of the best performing tile size for DMA time, and within 8% of the best performing tile size for compute time, and within 12% of the best when assuming the execution time is $max(time_{DMA}, time_{compute})$.

### B. DMA Model

As shown in Fig. 1, the DMA engine concurrently transfers the previously-computed output tile(s) from scratchpad to DRAM, and then transfers the next input tile(s) from DRAM to scratchpad. At the same time, the DSP core processes the current tile(s), reading and writing to scratchpad. DMA performance depends on characteristics of the transfer, such as the size of the transfer, the number of consecutive bytes accessed, and the stride length and frequency. Our approach is to develop a DMA performance model that associates features of the transfer to an achieved DMA bandwidth.

To build a training set for the model, we used performance counters on the DSP to measure effective DMA bandwidth over a variety of tile sizes and pixel depths, as well as characteristics specific to each individual OpenVX kernel such

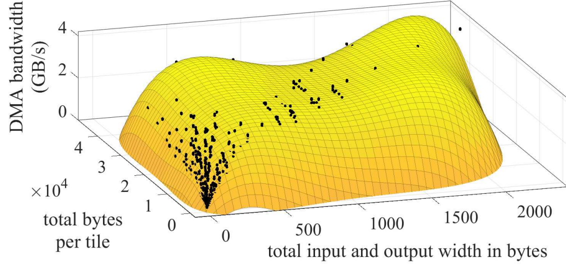


Fig. 4. Order-4 Polynomial Fit of DMA Bandwidh vs. Total Tile Size and Total Tile Width.

as its number of inputs and outputs, and the differences in input and output tile size caused by the halo region of stencil-based kernels. The data set covered 3240 kernel invocations over 36 tilable OpenVX 1.1 [13] kernels for all of their respective pixel depths and over 25 different tile sizes for a 1600x1200 image size. From these tests we found that DMA throughput varies from 189 MB/s to 4.33 GB/s.

Each of the observations in the dataset is associated with a specific kernel, tile size, and pixel depth. We associated each of these with two values: (1) the sum of tile widths in bytes for each of the inputs and outputs, and (2) the total tile size in bytes of all the inputs and outputs. We then associated these two values with the resulting DMA bandwidth. In general, tile width plays an important part of DMA bandwidth because wider tiles have more consecutively accessed pixels, which reduces the frequency of row activations in the DRAM controller, while total tile size determines the total payload transferred, reducing the impact of time needed to start the DMA transfer.

Fig. 4 shows a polynomial fit of DMA bandwidth to these two values. Each data point is shown as a black dot, while the surface shows the fitted curve. This model suffers from over-fitting, achieving a root-mean-square (RMS) training accuracy of 32 MB/s but a testing accuracy of 248 MB/s over an observed range of 189 MB/s to 4.33 GB/s. However, the model illustrates the nonlinear nature of the DMA behavior, the steep slope in bandwidth for smaller tiles, and the point of diminishing returns for larger tiles. In order to improve model accuracy, we developed methods to increase the number of features that comprise the independent variables for which we hope to predict bandwidth.

In the original dataset, we labeled each observation as a feature vector using a combination of elements chosen from: (1) input tile width in pixels (**TIW**), (2) input tile height in pixels (**TIH**), (3) output tile width in pixels (**TOW**), (4) output tile height in pixels (**TOH**), (5) total input width (tile width x pixel size x number of inputs) (**TTIW**), (6) total output width (tile width x pixel size x number of outputs) (**TTOW**), (7) total input size, (TTIW x TIH) (**TIS**), (8) total output size, (TOW x TOH) (**TOS**), (9) number of inputs (**NI**), (10) number of outputs (**NO**), (11) pixel depth (**PD**), (12) stencil neighborhood width (**SNW**), and (13) stencil neighborhood height (**SNH**). Each combination of feature vectors caused multiple observations to have the same feature vector value but with different DMA bandwidths. We refer to each of these sets of observations as "classes".

Table I shows three different feature sets and the resulting average class size and mean error of each classes' members to the average DMA bandwidth of the class. Our best mean error came from the TTIW, TIH, TTOW, TOH (total tile input width, tile height, total tile output width, total output width) feature set. Note that input tile size and output tile size differ in the presence of neighborhood-based kernels such as filters, as well as kernels having a different number of input vs. output images.

TABLE I
TRAINING RESULTS FOR VARIOUS DMA MODEL FEATURE SETS

| Features | Ave. class size | Mean class error |
|---|---|---|
| TTIW, TTOW, TIH, TOW, NI, NO, TIS, TOS | 2.9 | 114 MB/s |
| TIW, TIH, NI, NO, TIS, TOS, SNW, SNH | 2.9 | 219 MB/s |
| TTIW, TIH, TTOW, TOH | 3.3 | 100 MB/s |

After decomposing our original training set into classes, the model behaves as a lookup table for inputs that match observed training points. Otherwise the model uses interpolation to predict the bandwidth.

The model performs interpolation as shown in Equation 1. For each class $class_i$, the model calculates the distance between the input features, $F_{in}$ and the features of each of the classes $class_i^F$. The model sorts the classes according to their distances to the input features, then the model calculates a weighted average with respect to the inverse normalized distances. This model achieved an RMS testing error of 62 MB/s.

$$\begin{gathered}\forall i : class_i^d = |F_{in} - class_i^F| \\ sort(class, class^d) \\ bw_{predicted} = \sum_i^N \frac{(class_i^d)^{-1}}{\sum_j^N \left(\frac{class_j^d}{\sum_k^N class_k^d}\right)^{-1}} \times class_i^{bw}\end{gathered} \tag{1}$$

The model is illustrated in Fig. 5, where the center node represents an unknown feature vector to be predicted and the five outer nodes represent the five closest observed feature vectors and their associated bandwidths.

### C. Compute Model

The C66x DSP relies on the compiler to statically schedule its instructions. All stalls resulting from data, control, and structural hazards are explicitly defined in the object code

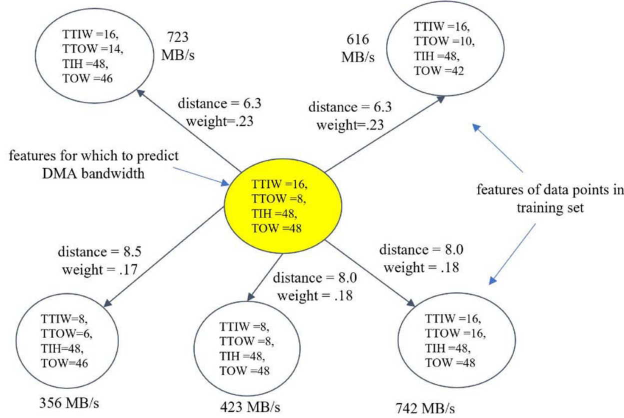


Fig. 5. Example of Model Interpolation.

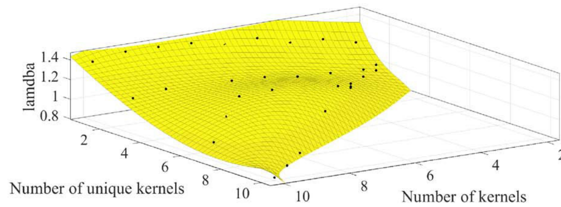


Fig. 6. Order-4 Polynomial Fit of Lambda vs. Number of Kernels and Number of Unique Kernels.

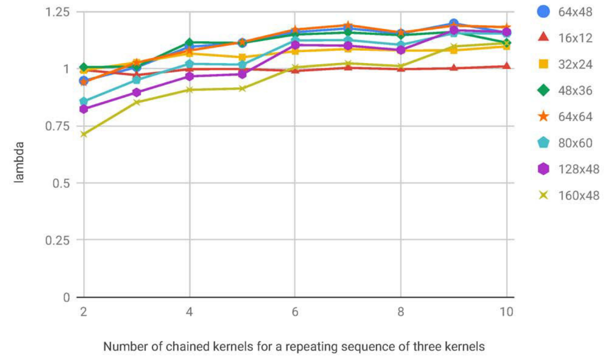


Fig. 7. Lambda vs. number of merged kernels for a pattern of three repeated kernels for various tile sizes.

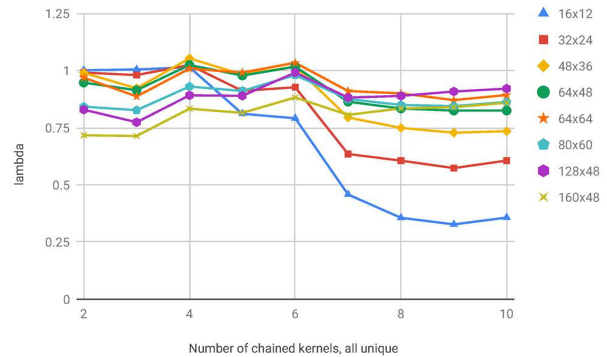


Fig. 8. Lambda vs. number of merged kernels for a sequence of unique kernels for various tile sizes.

and the number of cycles per loop iteration is reported by the compiler. Since instructions are statically scheduled, the only source of performance uncertainty (aside from DMA bandwidth) are the number of stalls caused by L1 cache misses.

As shown in Fig. 3, the number of raw compute cycles needed by an OpenVX kernel (without including the effects of the DMA controller) is determined by the kernel and its associated tile size. The kernel's L1 access pattern and its tile size determine the number of stalls from L1 data cache misses. L1 data cache performance is maximized at different tile sizes for different kernels.

Our compute model includes a table that associates the kernel name and total tile size to an observed compute time. We average the compute time for multiple observations having the same kernel name and total tile size, and report the closest match for unobserved inputs.

The L1 instruction cache impacts performance when the processor alternates between multiple OpenVX kernels at tile granularity, in which each kernel is scheduled onto the processor to process each tile instead of each image. In this case, the processor switches operating states between different kernels at a higher rate than it would if each kernel processed a image. The increased rate of state changes reduces the locality of the instruction cache. We refer to this effect as the "lambda scaling," where a lambda term defines the speedup/slowdown experienced by the processor as additional kernels are merged together at tile granularity.

Fig. 7 shows the lambda value as the number of merged kernels is increased from 2 to 10. For this test, kernels are added from a repeating sequence of three unique kernels (vxOr, vxNot, vxAdd). In this case, the lambda value trends upward with increased kernel count.

Fig. 8 shows the lambda value for a sequence of unique kernels. In this case, the lambda value trends downward with increased kernel count, due to the loss of instruction cache locality that results from having a larger number of unique kernels.

We model the lambda value by associating the lamba value with number of merged kernels and number of unique marged kernels. Fig. 6 shows a polynomial fit for this data for tile size 64x48, which has an RMS error of 0.0554. In order to improve accuracy, we constructed a piece-wise interpolated model, as described above for our DMA performance model. This improved the RMS error to 0.0338.

Consider the example merged OpenVX graph shown in Fig.

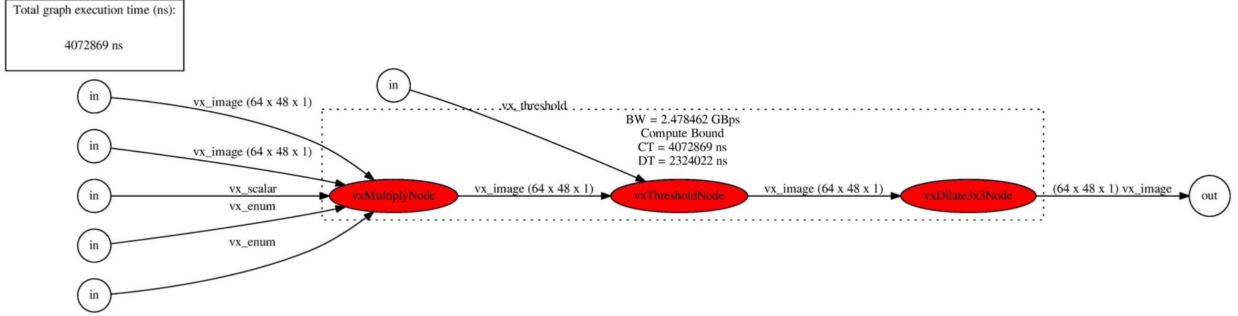


Fig. 9. Predicted performance of a merged Multiply-Threshold-Dilate graph generated by our random graph generator.

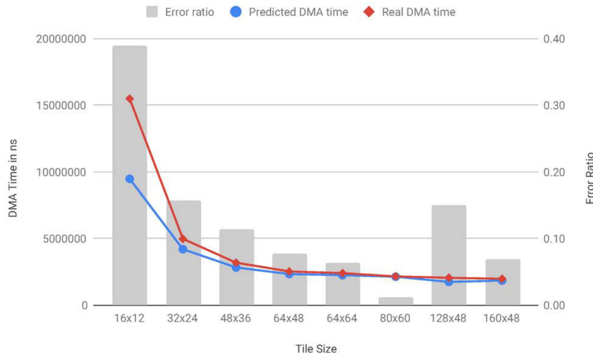


Fig. 10. DMA Model Accuracy for Multiply-Threshold-Dilate Graph.

9. Fig. 10 shows the predicted and actual DMA bandwidth for eight tile sizes as well as the error relative to the actual DMA bandwidth. In this case, the error is highest for the smallest tile size tested, 16x12.

Fig. 11 shows the predicted and actual compute time, based on both the tile size and predicted lambda value. In this case, the error is highest at the point of highest change, where the tile size is 32x24. Other tile sizes exhibit an error of less than 11%.

## IV. Automated Node Merging and Tile Size Selection

As best of our knowledge there are no benchmarks defined for OpenVX. In the literature, previous work on optimizing OpenVX graphs has relied on using a small set of relatively small graphs to evaluate the proposed techniques [14] [15]. This approach makes it difficult to generalize results. In order to generalize our modeling results we developed a tool that can randomly generate synthetic graphs having a given number of kernels. The only graphs considered are those that are unique, valid, and schedulable OpenVX graphs (as determined by the vendor runtime). For each of these, we enumerate all possible subgroupings and tile sizes for each group.

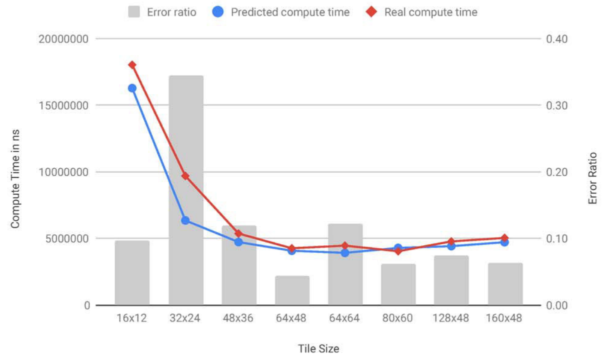


Fig. 11. Compute Module Accuracy for Multiply-Threshold-Dilate Graph.

$$B_n = \sum_{k=0}^{n-1} \binom{n-1}{k} \times B_k \qquad (2)$$
$$B_0 = 1$$

### A. Model Accuracy

A graph of $n$ nodes may be decomposed into $B_n$ groups, where $B_n$ is the recursively-defined "Bell number", as defined in Equation 2. The Bell number scales exponentially, e.g. $B_6 = 203$, $B_7 = 877$, ..., $B_{12} = 4,213,597$. For an OpenVX graph, the possible groupings must be weakly connected with respect to the edges that carry image data (the OpenVX vx_image type). This means there must be a path between any pair of nodes in a group if all edges are treated as bidirectional (otherwise there would be no reason to group the nodes).

For each graph we generate, we enumerate all valid node groupings. For each grouping, we enumerate all possible tile size to group assignments, chosen from 16x12, 32x24, 48x36, 64x48, 64x64, 80x60, 128x48, 160x48. Assignments

that exceed the scratchpad capacity of 128 KB are disregarded. For each grouping and assigned tile size, we use the DMA and compute models to predict its execution time, which is $max(DMAtime, computetime)$. The graph execution time is computed as the sum of group execution times.

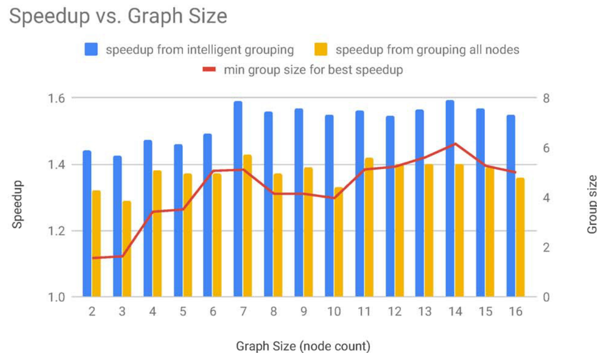


Fig. 12. Comparison of the speedup given by optimized node merging vs naively merging all graph nodes. Both speedups are relative to the baseline case of no merging (executing each node independently). The line represents the average minimum group size found for the optimized merging method. Each data point shown for graph sizes of up to 12 nodes are computed by analyzing 200 unique graphs. Random graphs of sizes 13 to 16 are rare, and in these cases less than 200 unique graphs are available and only the graphs found are included in the analysis.

Fig. 12 summarizes the performance impact of our proposed optimization method for randomly-generated graphs. The left bar of the plot shows the average speedup of the best-performing grouping and tile size assignment for each graph size. As shown this value ranges from 1.43 to 1.59 and is generally invariant across graph sizes. The trace shows the average group size that achieved the best speedup for each graph size. This value trends upward for larger graphs and ranges from 1.56 for a graph size of 2 to 6.15 for a graph size of 14. The right bar shows the predicted speedup of the best performing tile size when all the graph nodes are associated with a single group. This value range is generally 0.2 less than the best speedup number for intelligent grouping.

Fig. 13 shows the distribution of speedups over the whole enumeration of all groupings and tile sizes for a randomly-chosen 5-node graph, used here as an example. The best predicted speedup achieved with this graph is 2.00, but less than 1% of the enumerations achieve a predicted speedup of greater than 1.5, 30% achieve a predicted speedup of 1.0 to 1.5, and 69% achieve a predicted speedup less than 1. This result illustrates the complexity of finding a parameter set that yields positive results.

## V. Related Work

The widespread adoption of OpenVX has motivated on-going work in OpenVX performance optimization as well as development of new customized architectures to serve as execution targets. Yang et al extended Nvidia's VisionWorks

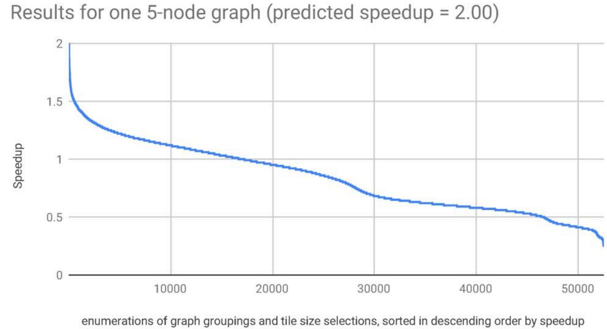


Fig. 13. Distribution of predicted speedup over enumerated parameterizations of a single 5-node graph.

libray [9] to create a real-time OpenVX implementation [16]. ADRENALINE is a virtual many core SoC platform with a corresponding OpenVX runtime based on OpenCL [17]. It achieves a speedup of 2 to 5 for OpenVX graphs as compared to equivalent OpenCL implementations executed on the same platform.

### A. Dataflow/OpenVX Graph Optimization via Node Merging/Clustering

The general approach of identifying nodes to merge in a dataflow graph (or OpenVX graph in particular) offers various benefits depending on what type of platform is targeted. These efforts may be broadly categorized into the granularity of the hardware resources targeted, such as a pool of processor cores or individual function units. In the case of mapping processor cores, node merging can provide a mechanism to constrain crossbar switch sizes in the case of FPGA-based multiprocessor system-on-chip platforms [18], improve cache performance on traditional shared memory multicores [15], and minimize data movement for FPGA-based soft-core DSPs [14] [19]. In the case of individual functional units such as the case when using FPGA-based high-level synthesis, node merging allows for constraining total hardware cost by optimizing the dataflow graph before lowering the graph to a C-language description immediately prior to its synthesis to gates [20].

### B. Performance Modeling

Machine learning is emerging as an alternative to analytical-based [21] and simulation-based performance modeling [22] [23]. Analytical models are generally not effective at finding the optimal tile sizes for various processor architectures [24]. Auto Tuning is perhaps the most widely-used approach for tile size selection, which involves exhaustively trying a variety of tile sizes and selecting the one with the lowest execution time [24].

### C. DSLs (Domain Specific Languages)

OpenVX's graph-based method for describing dataflows is similar to other efforts to develop performance-portable

domain-specific languages. Quio et al extended Hipacc, an open source source-to-source compiler to automatically fuse kernels by analyzing the AST (Abstract Syntax Tree) of a compiled image processing application written in Hipacc DSL [25]. PolyMage [26] and Halide [27] are two popular domain-specific languages, both embedded in C++, for auto-tuning image processing applications. Both use a "position independent" representation, in which the programmer provides an element-level description of a computation with only relative indexing and without the surrounding loop constructs or any notion of the order in which the elements are processed. PolyMage relies on polyhedral analysis to generate tiled loop nests. Halide requires its programs to include both a functional representation as well as a meta-program that specifies a method that Halide should follow to optimize the program.

## VI. Conclusions and Future Work

In this paper we describe our approach for modeling the impact of subgraph merging and tile size selection on OpenVX graph performance. Our performance models predict the impact of tile size and geometry on DMA bandwidth and compute time. We apply machine learning techniques to train the models from data collected from observed OpenVX kernel performance with various tile characteristics. Our models achieve average error rates of 13% and 8% for DMA and compute respectively. Using the models to select node merging and tile size achieves average predicted speedups in the range of 1.43 to 1.59 for 2-to 16-node graphs.

In future work we will extend our performance models to capture the effect of merging OpenVX kernels at the loop level, allowing kernels to pass intermediate results through registers instead of scratchpad buffers. This will allow for a 2-pass optimization, merging kernels at the loop level and also at the scratchpad level to maximize performance.

## References


[1] J. Redgrave, A. Meixner, N. Goulding-Hotta, A. Vasilyev, and O. Shacham. Pixel Visual Core:Google's Fully ProgrammableImage, Vision, and AI Processor For Mobile Devices. Presented at Hot Chips: A Symposium on High Performance Chips, 2018.

[2] L. Codrescu. Architecture of the Hexagon™ 680 DSP for mobile imaging and computer vision. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–26, Aug 2015.

[3] M. Ditty, A. Karandikar, and D. Reed. Nvidia's Xavier SoC. Presented at Hot Chips: A Symposium on High Performance Chips, 2018.

[4] J. Sankaran and N. Zoran. TDA2X, a SoC optimized for advanced driver assistance systems. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2204–2208, May 2014.

[5] Xilinx. Xilinx SDAcel, 2019. https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html.

[6] Intel. Intel FPGA SDK for OpenCL, 2019. https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html.

[7] Erik Rainey, Jesse Villarreal, Goksel Dedeoglu, Kari Pulli, Thierry Lepley, and Frank Brill. Addressing System-Level Optimization with OpenVX Graphs. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, CVPRW '14, pages 658–663, Washington, DC, USA, 2014. IEEE Computer Society.

[8] Intel. OpenVINO Toolkit, 2019. https://software.intel.com/en-us/openvino-toolkit.

[9] F. Brill and E. Albuz. NVIDIA VisionWorks Toolkit. Presented at the GPU Technology Conf, 2014.

[10] AMD. AMD OpenVX, 2019. https://gpuopen.com/compute-product/amd-openvx/.

[11] R. Staszewski K. Chitnis and G. Agarwal. Ti vision sdk, optimized vision libraries for adas systems. *White Paper: SPRY260, Texas Instrument Inc.*, April 2014.

[12] K. Chitnis, J. Villarreal, B. Jadav, M. Mody, L. Weaver, V. Cheng, K. Desappan, A. Jain, and P. Swami. Novel OpenVX implementation for heterogeneous multi-core systems. In *2017 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 77–80, Oct 2017.

[13] Kronos Group. The openvx 1.1 specification, 2017. http://www.khronos.org/registry/OpenVX/specs/1.1/OpenVX_Specification_1_1.pdf.

[14] Giuseppe Tagliavini, Germain Haugou, Andrea Marongiu, and Luca Benini. Optimizing Memory Bandwidth Exploitation for OpenVX Applications on Embedded Many-core Accelerators. *J. Real-Time Image Process.*, 15(1):73–92, June 2018.

[15] D. Dekkiche, B. Vincke, and A. Merigot. Investigation and performance analysis of OpenVX optimizations on computer vision applications. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–6, Nov 2016.

[16] Kecheng Yang, Glenn A. Elliott, and James H. Anderson. Analysis for Supporting Real-time Computer Vision Workloads Using OpenVX on Multicore+GPU Platforms. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 77–86, New York, NY, USA, 2015. ACM.

[17] G. Tagliavini, G. Haugou, A. Marongiu, and L. Benini. ADRENALINE: An OpenVX Environment to Optimize Embedded Vision Applications on Many-core Accelerators. In *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 289–296, Sep. 2015.

[18] A. Cilardo, E. Fusella, L. Gallo, and A. Mazzeo. Joint communication scheduling and interconnect synthesis for FPGA-based many-core systems. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–4, March 2014.

[19] Hossein Omidian and Guy G. F. Lemieux. JANUS: A compilation system for balancing parallelism and performance in OpenVX. *Journal of Physics: Conference Series*, 1004:012011, apr 2018.

[20] H. Omidian and G. G. F. Lemieux. Exploring automated space/time tradeoffs for OpenVX compute graphs. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 152–159, Dec 2017.

[21] R. Saussard, B. Bouzid, M. Vasiliu, and R. Reynaud. Optimal performance prediction of adas algorithms on embedded parallel architectures. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 213–218, Aug 2015.

[22] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso. A survey of performance modeling and simulation techniques for accelerator-based computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):272–281, Jan 2015.

[23] Andrew Kerr, Gregory Diamos, and Sudhakar Yalamanchili. Modeling gpu-cpu workloads and systems. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU-3, pages 31–42, New York, NY, USA, 2010. ACM.

[24] S. Tavarageri, L. Pouchet, J. Ramanujam, A. Rountev, and P. Sadayappan. Dynamic selection of tile sizes. In *2011 18th International Conference on High Performance Computing*, pages 1–10, Dec 2011.

[25] Bo Qiao, Oliver Reiche, Frank Hannig, and Jürgen Teich. Automatic kernel fusion for image processing dsls. In *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems*, SCOPES '18, pages 76–85, New York, NY, USA, 2018. ACM.

[26] Ravi Teja Mullapudi, Vinay Vasista, and Uday Bondhugula. Polymage: Automatic optimization for image processing pipelines. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 429–443, New York, NY, USA, 2015. ACM.

[27] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 519–530, New York, NY, USA, 2013. ACM.