Synopsys experience with OpenVX™ for a Face Tracking Application

SYNOPSYS®
Silicon to Software™

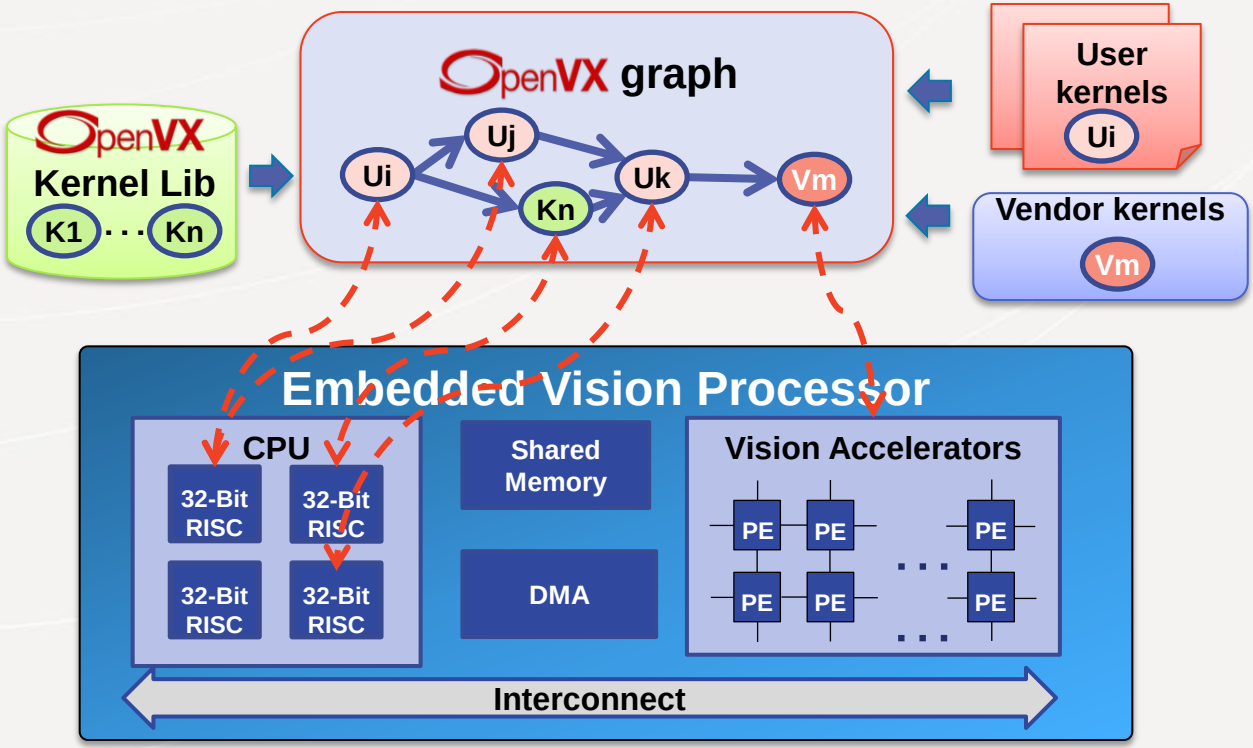
Pierre Paulin

May 4th, 2016

Outline

- Optimizing the OpenVX™ Graph Manager for Embedded Multi-core Architectures
- Face Tracking Example
- Lessons learned

Introduction to OpenVX™ Graph Mapping



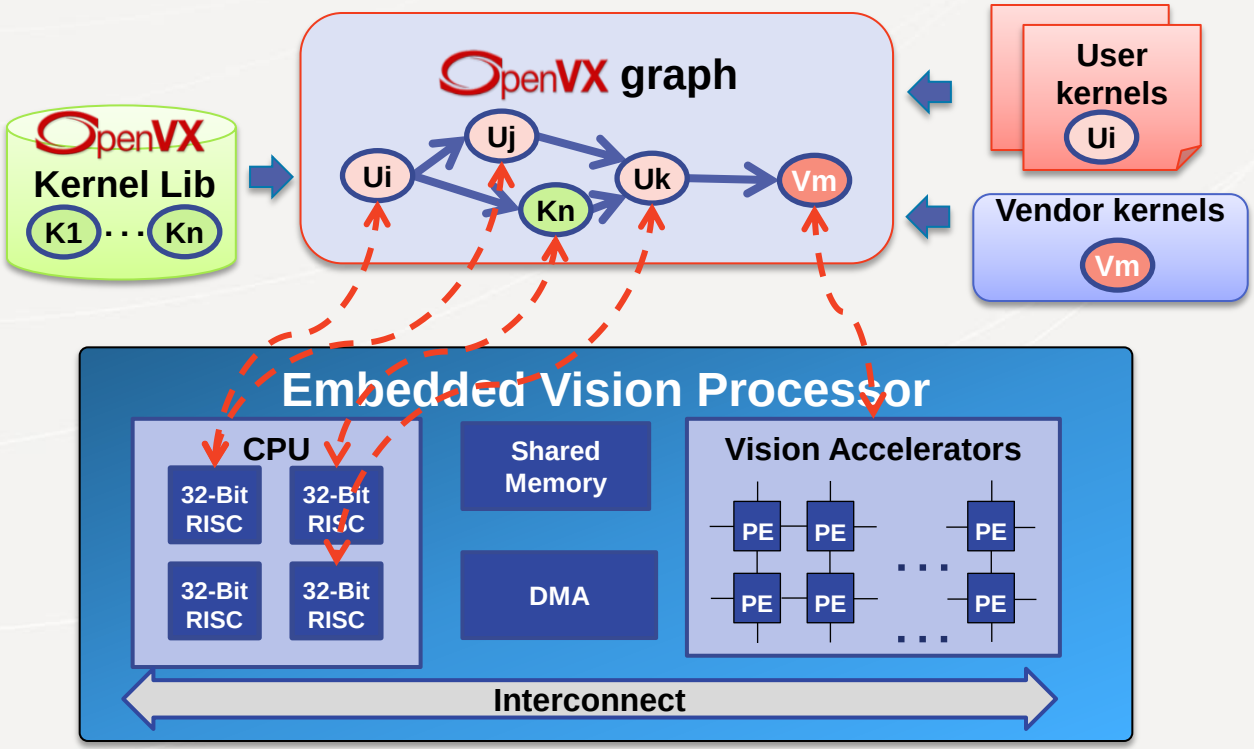
Challenge #1

- Some SoC with embedded vision processors may not have a host processor to run the OpenVX graph creation, verification and deployment steps
- Solution
  - Make the vision multi-core processor self-contained/self-hosting
    - Full OpenVX API available on the vision processor
    - Keep the flexibility to build graphs dynamically
      - As opposed to a solution where the graphs are created and verified with an offline tool
    - Avoid the heavy cost of a host processor running Linux

Challenge #2

- Embedded vision applications have aggressive PPA requirements
- Solution: Optimize OpenVX graph manager to
  - Make smart use of available data memories
    - Caches, tightly coupled memories, shared on-chip memories, external DRAM …
    - Use DMA as much as possible for efficient data buffer movement
  - Allow users to fine-tune the graph mapping using OpenVX hints
    - Node-to-core assignments
    - Data buffer placement
    - Scheduler hints
  - Support pipelined graph execution

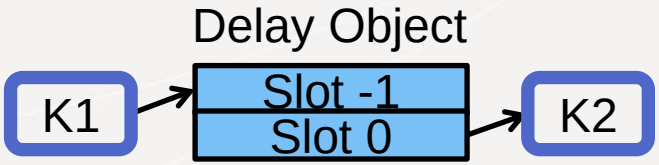
OpenVX Graph Mapping



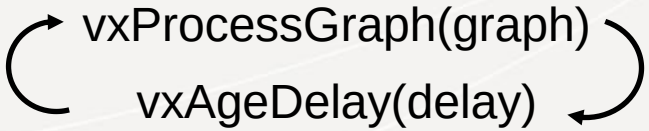
- Graph Mapping
  - Graph manager performs OpenVX node-to-core assignment and load balancing
  - Automatic insertion of communication buffers and memory allocation

Frame-based Pipelined Execution

- Delay objects can be used to allow pipelined graph execution
  - Each kernel runs in parallel at frame-level order



- K1 and K2 in diagram can be executed in parallel, on different cores for example



- Downside: requires more memory to store the temporary objects between kernels, compared to tiling

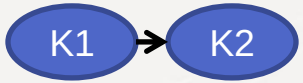
Challenge #3

- Frame-based kernel implementations require access to full input/output images
  - Images may not fit on the small on-chip memory of typical EV processors
- Solutions
  - Use kernel aggregation techniques
  - Use L1 data caches and frames in external memory
    - Only when algorithm does not permit a tiled solution
  - Implement the graph using a tiled data flow

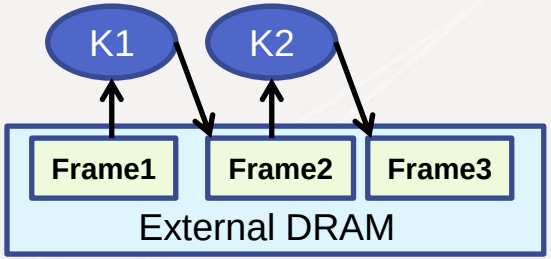
Tile-based Pipelined Execution

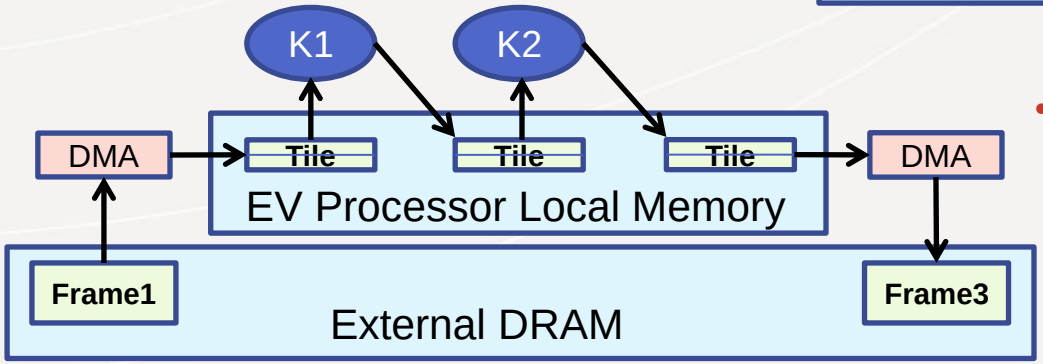
Reducing memory size and power

- Logical Model
  - Data flow between Kernels



- Classical Kernel Implementation
  - Host-Device frame buffer movement
  - Efficiency/memory size/power issues!





- Tiled implementation
  - Enhanced OpenVX runtime
  - Data “tunneled” through small(er) local memory
  - No round-trip to host

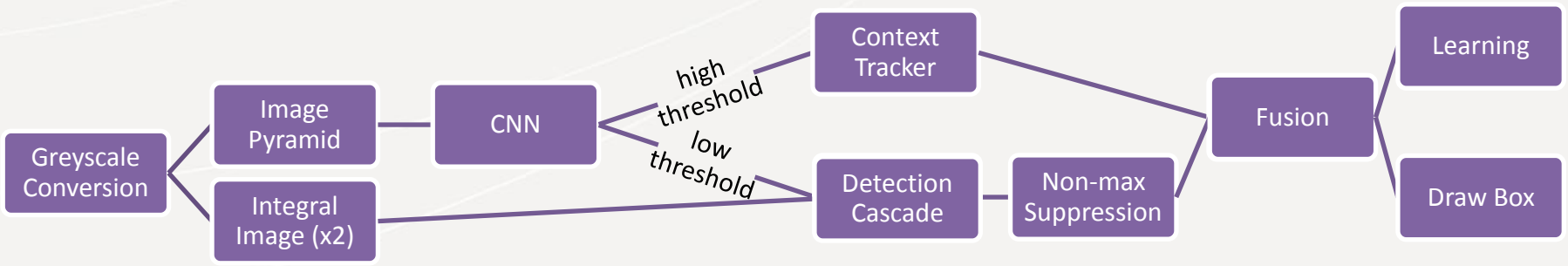
Challenge #4

- OpenVX standard set of vision functions is currently limited
  - Real-life embedded vision applications will require additional functionality
- Solutions
  - Extend standard set with “vendor” kernels optimized for target architecture
  - Provide an efficient way for users to implement their own kernels
    - Including support for user kernel tiling
  - Optimize OpenVX graph manager to efficiently map graphs that combine
    - Standard, vendor-supplied and user-defined nodes
    - Tile-based and frame-based nodes
  - Increased standard kernel library planned in future OpenVX releases

Face Tracking OpenVX™ Case Study

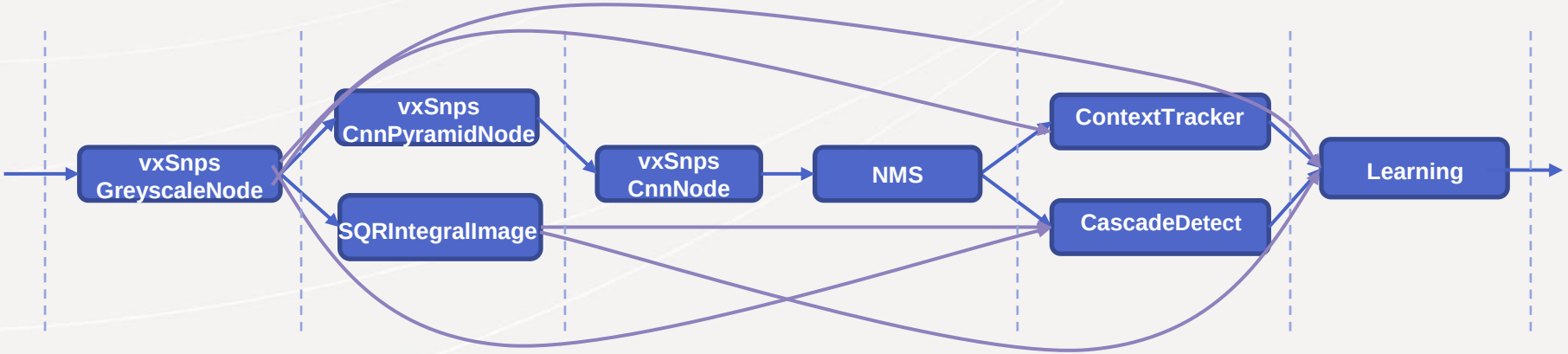
Face Tracking Example

- Detects multiple faces, tracks one identified face
- Derived from the Tracking-Learning-Detection (TLD) algorithm
  - Modified to use CNN-based face detection
- Introduction of a context-aware tracking that complements the CNN detections
  - Improves tracking accuracy, adds distinctiveness



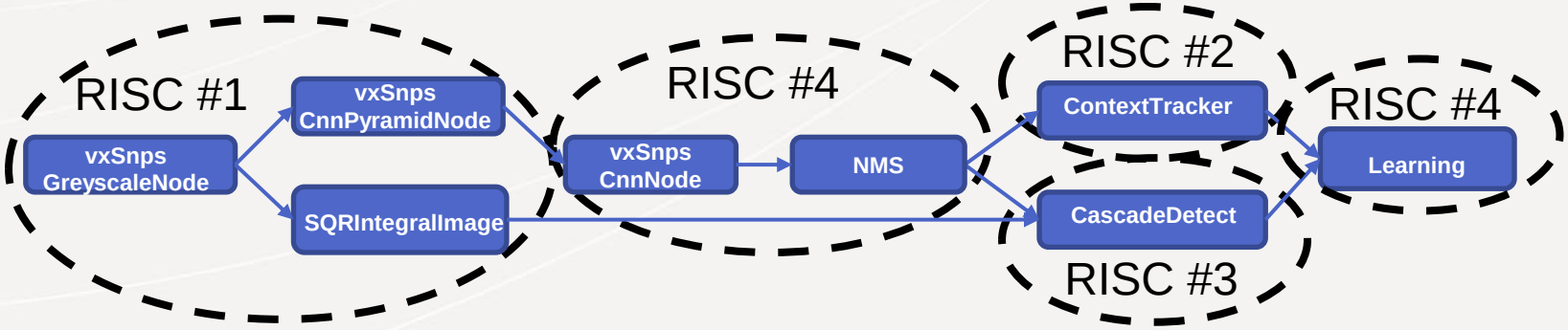
Face Tracker Mapping

- The face tracking application is captured as an OpenVX™ frame-based graph
- Kernels consume/produce OpenVX data objects
  - Images, arrays, matrices, scalars, etc.



Graph Scheduling and Assignment

- Node-to-core assignment is manually done to favor a good load balancing between cores, and to limit the inter-core data bandwidth
  - OpenVX™ hint mechanism for core assignment



Node-to-core assignment

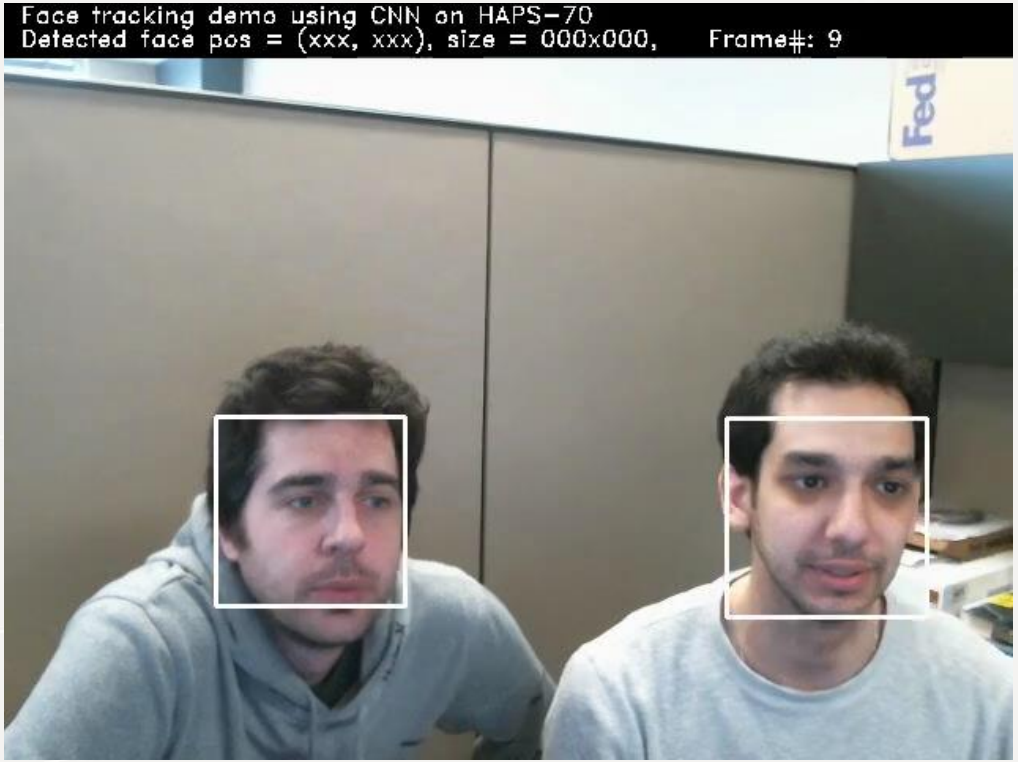
Challenges

- Current OpenVX™ standard kernels are not always flexible enough
  - More efficient RGB-to-Y greyscale conversion kernel developed
    - Replaces standard Color Convert and Channel Extract kernels
  - A more flexible Image Pyramid kernel developed
    - Handles any downscaling factor
  - A more flexible Integral Image user kernel developed
    - Produces square integral images
- Use of non-standard kernels reduces portability across multiple platforms
  - Working with Khronos to generalize these functions in future releases

Open Issues

- Shared global data structures cannot be easily captured by OpenVX™
  - Large structures (such as face models) are read by multiple kernels
  - Capturing face models as OpenVX object and passing it to all nodes that read it would duplicate data and increase memory requirement
  - Updating the face models had to be done outside the scope of OpenVX to avoid race conditions
    - Defined a schedule that kept the worker cores active while the main application code was updating the models

Face Tracker Demo – on FPGA board



- Frames captured from execution on HAPS-70 S12 FPGA prototyping board

Lessons Learned

Lessons Learned

- Implementing OpenVX™ for a multi-core vision processor requires a lot of optimization
  - Smart data placement/movement strategies
  - Tiled-based data flow execution, with support for user kernels
  - Allow users to fine-tune graph mapping decisions using hints
  - Efficient support of mix of kernel types
    - Standard, vendor and user kernels
    - Tiled and frame-based kernels
  - Do not rely on a host processor running Linux for graph creation

Lessons Learned, cont'd.

- Capturing real-life vision applications in OpenVX™ is challenging
  - Some of the standard vision kernels are not flexible enough
  - Not all vision kernels can easily be tiled
  - Not all applications nicely fit into a data flow representation with nodes that can be well balanced on the available cores
- Some performance degradation is possible due to use of a standard programming model
  - Performance vs. productivity vs. portability trade-off
  - Having an optimized graph manager for the target architecture is key to achieving good performance and win that trade-off

Resources

- **Synopsys DesignWare EV Family Of Vision Processors:**
  - https://www.synopsys.com/dw/ipdir.php?ds=ev52-ev54
- **May 2015 Embedded Vision Summit Technical Presentation: "Low-power Embedded Vision: A Face Tracker Case Study," Pierre Paulin, Synopsys**
  - http://www.embedded-vision.com/platinum-members/synopsys/embedded-vision-training/videos/pages/may-2015-embedded-vision-summit-2
- **May 2015 Embedded Vision Summit Technical Presentation: "Tailoring Convolutional Neural Networks for Low-Cost, Low-Power Implementation," Bruno Lavigueur, Synopsys**
  - http://www.embedded-vision.com/platinum-members/synopsys/embedded-vision-training/videos/pages/may-2015-embedded-vision-summit
- Please come by to see our EV Processor demos at the Synopsys booth:
  - Location tbd