A framework for optimizing OpenVX Applications on Embedded Many-Core Accelerators

Giuseppe Tagliavini, DEI - University of Bologna
Germain Haugou, IIS - ETHZ
Andrea Marongiu, DEI - University of Bologna & IIS - ETHZ
Luca Benini, DEI - University of Bologna & IIS - ETHZ
Outline

- Introduction
- ADRENALINE: virtual platform
- ADRENALINE: OpenVX run-time
- Conclusion
Many-core accelerators for signal/image processing

1

Throughput Computing

3

General-purpose Computing

CPU

GPGPU

> 100

SW
Mixed
HW

Accelerator Gap

HW IP
Clustered many-core accelerators (CMA)

Multi-core processor

Host

L3

DDR3 memory

Cluster-based design

Cluster memory (optional)

Cluster

Cluster

L2

MPMD Processing Elements

Many-core accelerator

CC

L1

DMA

HWS

HW synchronizer

Cluster controller (optional)

Low latency shared TCDM memory

DMA engine (L1 ↔ L3)

Some examples: STM STHORM, Kalray MPPA, PULP

Clustered many-core accelerators (CMA)
PULP
Parallel Ultra-Low-Power platform

SoC VOLTAGE DOMAIN (0.8V)

L2 MEMORY

CLUSTER BUS

SRAM VOLTAGE DOMAIN (0.5V – 0.8V)

SRAM #0

SRAM #1

SRAM #M-1

SCM #0

SCM #1

SCM #M-1

DMA

LOW LATENCY INTERCONNECT

PERIPHERAL INTERCONNECT

INSTRUCTION BUS

BRIDGE

PERIPHERALS

BRIDGE

BRIDGE

PERIPHERALS

to RMUs

Hybrid memory system

PEAK EFFICIENCY: 400GOPS/W
OpenVX overview

- Foundational API for vision acceleration
  - Focus on *mobile and embedded systems*
- Stand-alone or complementary to other libraries
- Enable **efficient implementations** on different devices
  - CPUs, GPUs, DSPs, *many-core accelerators*
OpenVX programming model

- The OpenVX model is based on a **directed acyclic graph** of nodes (**kernels**), with data (**images**) as linkage

```c
vx_image imgs[] = {
    vxCreateImage(ctx, width, height, VX_DF_IMAGE_RGB),
    vxCreateVirtualImage(graph, 0, VX_DF_IMAGE_U8),
    ...
    vxCreateImage(ctx, width, height, VX_DF_IMAGE_U8),
};
vx_node nodes[] = {
    vxColorConvertNode(graph, imgs[0], imgs[1]),
    vxSobel3x3Node(graph, imgs[1], imgs[2], imgs[3]),
    vxMagnitudeNode(graph, imgs[2], imgs[3], imgs[4]),
    vxThresholdNode(graph, imgs[4], thresh, imgs[5]),
};
vxVerifyGraph(graph);
vxProcessGraph(graph);
```
Virtual images are not required to actually reside in main memory
✓ They define a data dependency between kernels, but they cannot be read/written
✓ They are the main target of our optimization efforts

An OpenVX program must be verified to guarantee some mandatory properties:
✓ Inputs and outputs compliant to the node interface
✓ No cycles in the graph
✓ Only a single writer node to any data object is allowed
✓ Writes have higher priorities than reads.
✓ Virtual image must be resolved into concrete types
ADRENALINE

Application mapping

Virtual Platform

PE0

PEn

Mem

TestApplications

Run-time policies

Run-time support

Platform configuration

http://www-micrel.deis.unibo.it/adrenaline/
Outline

- Introduction
- ADRENALINE: virtual platform
- ADRENALINE: OpenVX run-time
- Conclusion
Virtual platform (1)

- The virtual platform is written in Python and C++
  - Python is used for the architecture configuration
  - C++ is used to provide an efficient implementation of internal model
- A library of basic components is available, but custom blocks can also be implemented and assembled
Virtual platform (2)

- Standard configuration:
  - **OpenRISC core.** An Instruction Set Simulator (ISS) for the OpenRISC ISA, extended with **timing models** to emulate **pipeline stalls**
  - **Memories.** Multi-bank, constant-latency timing mode
  - **L1 interconnect.** One transaction per memory bank serviced at each cycle
  - **DMA.** Single synchronous request to the interconnect for each line to be transferred
  - **Shared instruction cache.** Dedicated interconnect and memory banks.
Outline

- Introduction
- ADRENALINE: virtual platform
- ADRENALINE: OpenVX run-time
- Conclusion
A first solution: using OpenCL to accelerate OpenVX kernels

- OpenCL is a widely used programming model for many-core accelerators

- First solution: OpenVX kernel == OpenCL kernel
  - When a node is selected for execution, the related OpenCL kernel is enqueued on the device

- Limiting factor:
  - too much code!
  - memory bandwidth
OpenCL bandwidth

- Experiments performed with OpenCL runtime on a STHORM evaluation board → same results using the virtual platform

![Diagram showing OpenCL bandwidth for various tasks with and without the available bandwidth.](image-url)
OpenVX for CMA

- We realized an OpenVX framework for many-core accelerators coupling a **tiling approach** with algorithms for **graph partition and scheduling**

- Main goals:
  - Reducing the memory bandwidth
  - Maximize the accelerator efficiency

- Several steps are required:
  - Tile size propagation
  - Graph partitioning
  - Node scheduling
  - Buffer allocation
  - Buffer sizing
Localized execution

- **Localized execution** → when a kernel is executed by a many-core accelerator, read/write operations are *always* performed on *local buffers* in the L1 scratchpad memory.

- **Smiley**: Reads/writes on L1 do no stall the PEs
- **Sad**: In real platforms the L1 is often too small to contain a full image
- **Sad**: In addition, multiple kernels requires more L1 buffers
- **Sad**: During DMA transfers cores are waiting
Localized execution with tiling

- Images are partitioned into smaller blocks (tiles)
- **Double buffering** → overlap between data transfers and computation

😊 Single tiles always fit L1 memory
😊 Transfer latency is hidden by computation
😊 Tiling is not so trivial for all algorithms → data access patterns
Common access patterns for image processing kernels

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A) POINT OPERATORS</td>
<td>Compute the value of each output point from the corresponding input point</td>
<td>Basic tiling</td>
</tr>
<tr>
<td>(B) LOCAL NEIGHBOR OPERATORS</td>
<td>Compute the value of a point in the output image that corresponds to the input window</td>
<td>Tile overlapping</td>
</tr>
<tr>
<td>(C) RECURSIVE NEIGHBOR OPERATORS</td>
<td>Like the previous ones, but also consider the previously computed values in the output window</td>
<td>Persistent buffer</td>
</tr>
<tr>
<td>(D) GLOBAL OPERATORS</td>
<td>Compute the value of a point in the output image using the whole input image</td>
<td>Host exec / Graph partitioning</td>
</tr>
<tr>
<td>(E) GEOMETRIC OPERATORS</td>
<td>Compute the value of a point in the output image using a non-rectangular input window</td>
<td>Host exec / Graph partitioning</td>
</tr>
<tr>
<td>(F) STATISTICAL OPERATORS</td>
<td>Compute any statistical functions of the image points</td>
<td>Graph partitioning</td>
</tr>
</tbody>
</table>
Tile size propagation
__kernel void threshold(__global unsigned char *src, int srcStride,
        __global unsigned char *dst, int dstStride,
        short width, short height,
        short bandWidth, char nbCores,
        __global unsigned char *params) {

    int i, j;
    int id = get_id();
    unsigned char threshold = params[4];
    int srcIndex = 0, dstIndex = 0;
    for (j=0; j<height; ++j) {
        for (i=id*bandWidth; i<(id+1)*bandWidth && i<width; ++i) {
            unsigned char value = src[srcIndex+i];
            dst[dstIndex+i] = (value >= threshold? value: 0);
        }
        srcIndex+=srcStride;
        dstIndex+=dstStride;
    }
}
Bandwidth reduction

The graph shows the bandwidth reduction in MB/s for various operations and tasks, comparing OVX and OpenCL. The available bandwidth is indicated by the dotted line.

- Random graph: OVX 34, OpenCL 290
- Edge detector: OVX 36, OpenCL 307
- Object detection: OVX 71, OpenCL 922
- Super resolution: OVX 8, OpenCL 71
- FAST9: OVX 24, OpenCL 44
- Disparity: OVX 31, OpenCL 307
- Pyramid: OVX 8, OpenCL
- Optical: OVX 15, OpenCL 15
- Canny: OVX 18, OpenCL 199
- Retina preproc.: OVX 359, OpenCL 1391
- Disparity 54: OVX 22, OpenCL 779

Available BW is indicated by the dotted line at 10000 MB/s.
Speed-up w.r.t. OpenCL

- Random graph: 6.73
- Edge detection: 3.86
- Object detection: 3.46
- Super resolution: 3.50
- FAST9: 2.81
- Disparity: 5.64
- Pyramid: 2.92
- Optical: 1.00
- Canny: 3.12
- Retina preproc.: 5.04
- Disparity S4: 9.61
Outline

- Introduction
- ADRENALINE: virtual platform
- ADRENALINE: OpenVX run-time
- Conclusion
Next steps

- Virtual platform
  - More accurate models
  - Multi-cluster configuration
- OpenVX runtime
  - Models evolution
- FPGA emulator
THANKS!!!