Supporting Localized OpenVX Kernel Execution for Efficient Computer Vision Application Development on STHORM Many-Core Platform

Giuseppe Tagliavini
Univ. of Bologna

Germain Haugou
STMicroelectronics

Luca Benini
Univ. of Bologna / ETH Zurich

ABSTRACT

Nowadays Embedded Computer Vision (ECV) is considered a technology enabler for next generation killer apps, and scientific and industrial communities are showing a growing interest in developing applications on high-end embedded systems. Modern many-core accelerators are a promising target for running common ECV algorithms, since their architectural features are particularly suitable in terms of data access patterns and program control flow. In this work we propose a set of software optimization techniques, mainly based on data tiling and local buffering policies, which are specifically targeted to accelerate the execution of OpenVX-based ECV applications by exploiting the memory hierarchy of STHORM many-core accelerator.

Categories and Subject Descriptors

D.1 [**Software**]: Programming Techniques; I.4 [**Computing methodologies**]: Image processing and computer vision; C.1.4 [**Processor architectures**]: Parallel architectures—*mobile processing*

Keywords

OpenVX, many-core systems, embedded computer vision, parallel algorithms

1. HARDWARE ARCHITECTURE

Our target platform is the STHORM SoC by STMicroelectronics, previously known as P2012 [1]. STHORM is a many-core computing accelerator based on processor tiles interconnected by an asynchronous network-on-chip. Each tile (called *cluster*) features 16 dual-issue STxP70 cores, supporting independent instruction streams. Each cluster also contains a multi-banked one-cycle access L1 scratchpad memory, and a dual-channel DMA engine. A L2 scratchpad memory is shared by all clusters, and it is used by software run-time to store accelerator binaries; to minimize SoC size and power consumption, this architecture has no data cache.

The programming environment for STHORM platform is based on OpenCL 1.1 [2]. Similarly to GP-GPUs programming, data must be transferred into shared local memory prior to computation in order to take advantage of low-latency accesses; this is done explicitly in STHORM programming using OpenCL built-in functions for asynchronous work-group copy. In this work we have extended the standard run-time to have direct and explicit control on memory allocation and scheduling of both kernels and DMA transfers.

2. OPTIMIZATION TECHNIQUES

OpenVX [3] is a cross-platform API standard which aims at enabling hardware vendors to provide an interface for accelerated CV low-level primitives. The execution model of OpenVX is based on a directed acyclic graph of vision kernels, with data as linkage. Listing 1 shows an OpenVX sample code that implements a Sobel filter.

```
1 vx_context ctx = vxCreateContext();
2
3 vx_image imgs[] = {
4   vxCreateImage(ctx, width, height, FOURCC_RGB),
5   [...]
6   vxCreateImage(ctx, width, height, FOURCC_U8),
7 };
8 vx_node nodes[] = {
9   vxColorConvertNode(graph, imgs[0], imgs[1]),
10  vxSobel3x3Node(graph, imgs[1], imgs[2], imgs[3]),
11  vxMagnitudeNode(graph, imgs[2], imgs[3], imgs[4]),
12  vxThresholdNode(graph, imgs[4], thresh, imgs[5]),
13 };
14
15 vxVerifyGraph(graph);
16
17 vxProcessGraph(graph);
18 }
```

Listing 1: OpenVX application snippet (C code)

Before being executed, an OpenVX program must be verified to guarantee some mandatory properties of the vision graph. At the end of the canonical verification stage, we introduce the following algorithms:

- *Node scheduling.* A scheduling is determined through a breadth-first visit of the graph, and nodes that release more data references are prioritized. At each step a single kernel node is selected for execution, and all the accelerator cores are always allocated to the running kernel.
- *Buffer allocation.* The allocation policy specifies the maximum number of buffers that are allocated in L1 memory, and their association to input/output graph

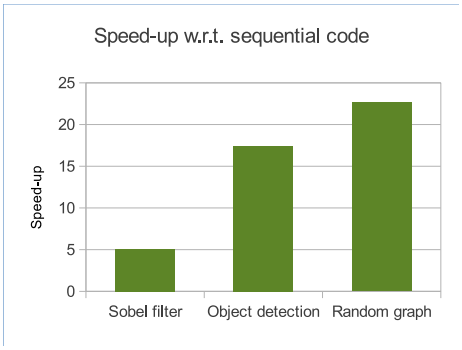

Figure 1: Speed-up w.r.t. sequential C code

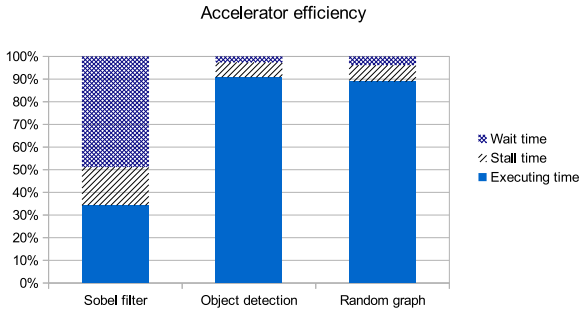

Figure 2: Analysis of accelerator efficiency

image parameters. The visit order of the scheduling algorithm guarantees that allocated buffers are used and then released as soon as possible.

- *Buffer sizing*. This algorithm computes the maximum size for allocated buffers in L1 memory. Graph input/output data reside in L3 memory, and to fit L1 buffers the images are partitioned into smaller blocks (*tiles*). The allowed size for tiles strictly depends on access patterns used by kernels. A tiling descriptor specifies the minimum image area used by each kernel to compute a single output value.

Our main goal is the maximization of *execution efficiency*, defined as the amount of time that accelerator cores spend to execute kernel code over the total graph execution time spent on accelerator side. This implies to minimize waiting time due to memory transfers to and from L3 memory.

3. EXPERIMENTAL RESULTS

We have executed a set of experiments on a STHORM evaluation board featuring a Cortex-A9 host processor (667 MHz) and a STHORM chip (430 MHz). The L3 memory bridge is clocked very conservative at 40MHz, so the bandwidth is limited to 320MB/s for the read channel and 160MB/s for the write channel. Compared with ARM processor, which uses all the DDR3 bandwidth (1.25GBps per channel), STHORM chip is severely penalized. This is a worst case w.r.t. to a fully integrated SoC, where the accelerator has a greater L3 bandwidth.

The applications used as benchmarks are:

- *Sobel filter*: an edge detector based on Sobel filter [5];
- *Object detection*: an abandoned/removed object detection algorithm based on NCC background subtraction, as described in [4];
- *Random graph*: a synthetic benchmark including 10 OpenVX nodes, with a significant branching structure.

3.1 Comparison with sequential code

Figure 1 shows the speed-up of OpenVX accelerated versions w.r.t. the sequential C version executed on the host processor (ARM Cortex A9). The speed-up for *Sobel filter* is limited, as it is composed by kernels with a low computational workload, and so the benefits of using any acceleration technique are very limited.

3.2 Execution efficiency

Figure 2 depicts the accelerator efficiency fo the selected benchmarks, computed as the percentage of total graph execution time over 100 graph instances. At *execution time* the cores are actually executing kernel instructions. The *wait time* is the time for node scheduling in the OpenCL run-time, that is the waiting time to complete DMA transfers before node execution. Finally, the *stall time* includes the stall cycles due to floating-point unit and memory contention on the cluster local interconnect (instruction cache, L1 memory banks).

The wait time for *Sobel filter* is about 40% of the total, due to the fact that DMA transfers are longer than kernel computations.

4. CONCLUSIONS AND FUTURE WORKS

In this work we have proposed a set of techniques to improve execution efficiency of CV algorithms on the STHORM platform, and we have implemented a framework with an OpenVX standard front-end that implements these features. Experimental results shows that our approach to localized execution of vision kernels provides huge benefits in terms of speed-up.

Our future work will be focused on extending this approach, with the aim to apply it to other many-core accelerators.

5. REFERENCES

[1] L. Benini, E. Flamand, D. Fuin, and D. Melpignano. P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 983–987, 2012.

[2] Kronos Group. The OpenCL 1.1 Specifications. `http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf`, 2010.

[3] Kronos Group. The OpenVX API for hardware acceleration. `http://www.khronos.org/openvx`, 2013.

[4] M. Magno, F. Tombari, D. Brunelli, L. Di Stefano, and L. Benini. Multimodal abandoned/removed object detection for low power video surveillance systems. In *Advanced Video and Signal Based Surveillance, 2009. AVSS'09. Sixth IEEE International Conference on*, pages 188–193. IEEE, 2009.

[5] I. Pitas. *Digital image processing algorithms and applications*. Wiley.com, 2000.