Comparative Analysis of OpenCL vs. HDL with Image-Processing Kernels on Stratix-V FPGA

Kenneth Hill, Stefan Craciun, Alan George, Herman Lam

NSF Center for High-Performance Reconfigurable Computing
ECE Dept., University of Florida, Gainesville FL, USA
e-mail: {hill, craciun, george, lam}@chrec.org

***Abstract*--Application development with hardware description languages (HDLs) such as VHDL or Verilog involves numerous productivity challenges, limiting the potential impact of reconfigurable computing (RC) with FPGAs in high-performance computing. Major challenges with HDL design include steep learning curves, large and complex codes, long compilation times, and lack of development standards across platforms. A relative newcomer to RC, the Open Computing Language (OpenCL) reduces productivity hurdles by providing a platform-independent, C-based programming language. In this study, we conduct a performance and productivity comparison between three image-processing kernels (Canny edge detector, Sobel filter, and SURF feature-extractor) developed using Altera's SDK for OpenCL and traditional VHDL. Our results show that VHDL designs achieved a more efficient use of resources (59% to 70% less logic), however, both OpenCL and VHDL designs resulted in similar timing constraints (255MHz < f_{max} < 325MHz). Furthermore, we observed a 6× increase in productivity when using OpenCL development tools, as well as the ability to efficiently port the same OpenCL designs without change to three different RC platforms, with similar performance in terms of frequency and resource utilization.**

Keywords—Altera SDK for OpenCL; FPGA; Application Development; High-Level Synthesis

I. INTRODUCTION

RC architectures developed using traditional HDL design methods can efficiently exploit both the intrinsic wide and deep parallelism of many applications [1]. Custom RC architectures have been used to process high frame rates for high-definition (HD) images, and have enabled efficient real-time processing [1], while maintaining a lower power budget in comparison to competing GPU and CPU solutions. However, the adoption of RC platforms for general-purpose and high-performance computing has been limited by numerous productivity challenges faced by developers when using HDL. These challenges include the requirement of specialized training for the design/verification of large and complex RTL code and long compilation times. In addition, there is a lack of development standards across different RC platforms, further complicating the design process. As a result, these development challenges have often caused designers to adopt traditional fixed-architecture devices over RC platforms.

The Open Computing Language (OpenCL) is a parallel programming standard that enables developers to create their parallel applications using a C-based language and target a variety of heterogeneous platforms including CPUs, GPUs, DSPs, and most recently FPGAs. In the case of FPGAs, OpenCL acts as a high-level synthesis tool for HDL development. Using OpenCL, a designer can work with an RC platform while avoiding RTL code as well as platform-specific tools and libraries. The first commercial framework for OpenCL on FPGAs is the Altera SDK for OpenCL. The Altera Offline Compiler (AOC) exploits an application's wide parallelism by using SIMD data types and simple compiler pragmas. The tool further optimizes hardware design by pipelining the datapath to harness deep parallelism available in the application. All of these features help alleviate the specialized and complex training previously needed for hardware design.

In this study, we map three image-processing kernels from the OpenCV library to an Altera Stratix-V FPGA on a variety of RC platforms. The Sobel [2] and Canny [3] kernels are used for edge detection, while the Speeded-Up Robust Features (SURF) [4] kernel is used for feature extraction. These information-extraction kernels represent the initial stage of many image-processing applications. Object-recognition, tracking, and unsupervised-navigation applications all require scale- and rotation-invariant interest points. This suite of image-processing kernels is developed using both traditional HDL design flow and the Altera SDK for OpenCL.

The organization of the paper proceeds by first presenting background information on the selected kernels in Section II. Section III compares the kernel design flows using HDL and OpenCL. We present a productivity and performance comparison between HDL and OpenCL in Section IV, and draw final conclusions from this work in Section V.

II. IMAGE-PROCESSING KERNELS

Edge detection is used to extract the important structural properties in the form of connected curves that represent objects contours. Feature extraction is a technique used to quantize the input image to a relatively small set of interest points. The features extracted help describe the unique

intrinsic characteristics of objects, and have been successfully used to label objects at different scales and from different angles [4]. The most frequently used image-processing kernels for edge detection are the Canny and Sobel filters, while the SURF feature-extractor has been efficiently used to locate interest points that are scale- and rotation-invariant.

The Canny, Sobel, and SURF kernels are based, at their core, on the fast gradient calculation of pixel intensity values. Local gradients are extracted using a 2D convolution with first- or second-order Gaussian derivative filters. The Sobel filters are used in both the Canny and Sobel kernels to sweep through the input image and extract gradients in the *x* and *y* directions. The SURF kernel uses a larger set of symmetric box filters (Fig.1) that represent second-order Gaussian derivatives along the vertical (*y*), horizontal (*x*) and diagonal (*xy*) directions. For optimal performance, the kernels are mapped to hardware using a sliding window approach.

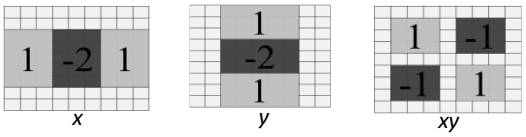

Fig. 1: Box filters used to approximate Gaussian derivatives

A smart buffer (Fig. 2) composed of registers and block RAMs (BRAMs) is used to transfer image rows from external to on-chip FPGA memory. Once full, the smart buffer provides access to all pixel values required to complete the convolution operation. Furthermore, the datapaths designed for each kernel are fully pipelined and enable processing of a continuous input stream of pixels without ever stalling the pipeline.

Fig. 3 displays the output of the Sobel and Canny kernels. The Canny edge detector requires two additional steps from the Sobel operator output. A non-maximum suppression is performed along the gradient direction to thin edges (ideally one pixel thin), and a connectivity analysis step that fills in edge discontinuities. Finally, the last kernel mapped to hardware is the SURF feature extractor.

The SURF architecture is more complex, due to the requirement that interest points have to be rotation- and scale-invariant. The SURF kernel therefore performs the 2D convolution with 24 box filters, spanning from a minimum of 9×9 up to a maximum of 51×51 filter sizes. The same smart buffer used in previous Canny and Sobel kernels is also used as part of the SURF architecture. In addition, after the convolution step, the SURF kernel requires the Hessian determinant calculation [4] for each of the six scales, followed by a non-maximum-suppression that determines the dominant scale. The outputs of the SURF feature extractor are interest-point locations (*x, y* coordinates). Fig. 4 shows the resulting output of the SURF kernel with features marked in red.

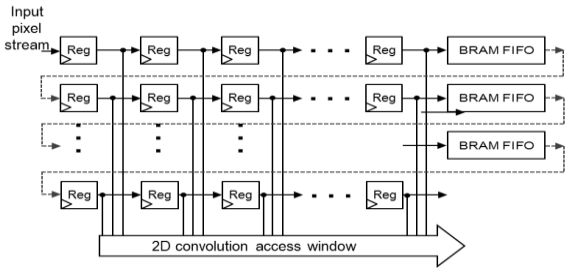

Fig. 2:Block diagram of sliding-window smart buffer

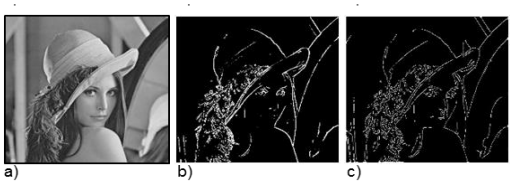

Fig. 3: a) Input image; b) Sobel and c) Canny kernel output

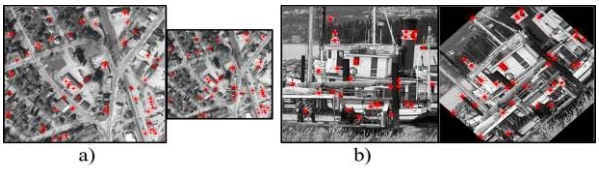

Fig. 4: SURF kernel interest-point locations; a) Scale-invariant interest points; b) Rotation-invariant interest points

III. DESIGN FLOW COMPARISON

The two development methods (OpenCL and HDL) under study follow different design approaches. In this section, we draw a direct side-by-side comparison between stages of the two design flows that must be completed when mapping a kernel to hardware.

A. VHDL design flow

Application development using the *VHDL* language generally follows the design flowchart in Fig. 5. The designer begins by specifying the hardware architecture at the register-transfer level (RTL) using the hardware description language. A test bench is developed and used to perform functional verification by checking each stage of the design using a cycle-accurate simulation. This *functional simulation* step verifies code syntax and ensures functional and behavioral correctness of the design. This part of the design process can easily span an extended amount of time due to the fine granularity (i.e., bit-level) of the simulation.

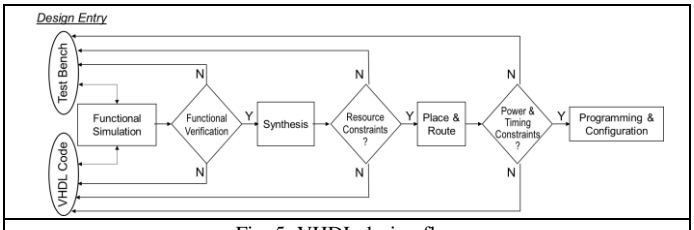

Fig. 5: VHDL design flow

If a problem occurs, the VHDL developer is required to inspect the simulation at the level of individual bits. After verification, the *synthesis* stage analyzes the design and translates it to logic gates. This stage optimizes the resource utilization of the design. If the application requires more resources than are available, the designer has the option of selecting a different, larger device, or optimizing the VHDL code and the structure of the design to lower resource utilization. This design-tuning phase can vary in complexity, especially in the case when the required resources are unavailable and the design needs to be scaled back. Finally, when the VHDL developer has ensured the design will fit on the device, the design proceeds to *place and route*.

In the *place-and-route* stage, the fitter maps the physical design to the device resources by matching logic functions to the optimal logic cell location. This operation is one of the most taxing parts of the VHDL compilation process, taking multiple hours and in some cases days to complete. While performing *place and route*, the fitter tries to meet the power and timing constraints specified by the designer. If these constraints cannot be met, the fitter issues a compilation error messages. Again, the VHDL designer has the option of changing the device or returning to the initial step and restructuring the design (i.e., edit VHDL code).

It is evident that the typical VHDL design flow can encounter several feedback loops that require the developer to go back to the initial stage of the design process. The steep learning curves associated with platform-specific tools delay VHDL developers from migrating their design to larger, faster, or more power-efficient FPGAs. These productivity challenges that characterize the VHDL design process have deterred application developers from targeting FPGAs.

B. *OpenCL design flow*

The Altera OpenCL SDK for FPGAs introduces the design flow shown in Fig. 6. This design flow alleviates many of the productivity bottlenecks of current VHDL tools by delaying the compilation to the last stage of the design process.

During the initial development stage, the application is described using a C-like syntax for generating one or more OpenCL kernels. The Altera Offline Compiler (AOC) translates the OpenCL code directly to a RTL design that is portable across most RC platforms featuring supported Altera FPGAs. The tool further simplifies the design flow by automatically handling interactions between different memory objects (external memory, BRAMS and register arrays) and the pipelined datapath. These interactions include the control logic to stall the pipeline, and the ability to schedule new work-items. The designer is no longer required to work at the RTL level, enabling programmers to create custom hardware architectures without specialized training for the design and verification of large and complex RTL code.

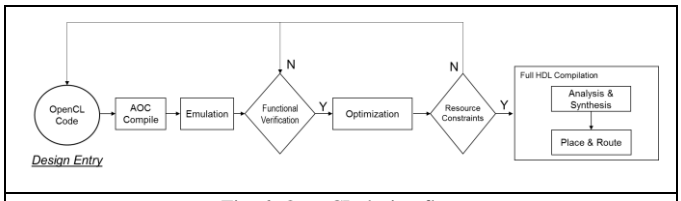

Fig. 6: OpenCL design flow

In addition to generating an RTL design, the AOC builds a special executable file to use with the full-system emulator. The emulator included with the AOC verifies the kernel code based on the targeted FPGA platform. Unlike VHDL test benches where a designer verifies individual bits, the AOC emulator allows designers to debug the OpenCL code in a software environment with GNU debugger support. The designer is able to set software breakpoints and inspect data elements while emulating their hardware prototype on an x86-based system. The familiarity of using tools like the GNU debugger reduces the learning curve required to use RC platforms. Another key advantage of the AOC emulator is that the host C/C++ code, which runs on the host CPU, is designed and verified along with the kernel code (in the FPGA) in an integrated manner. Traditionally, the host code for a VHDL project is designed and tested independently, after the kernel architecture is mapped to the FPGA, leaving timing, power, and resource utilization analysis to the post place-and-route design phase.

C. *OpenCL optimizations*

The goal of optimizing an AOC OpenCL kernel is to create a deep and wide pipelined architecture that does not stall during execution. A variety of strategies were used to optimize the selected image-processing kernels in this study. We describe one such example to demonstrate the effectiveness of Altera's OpenCL tools for generating pipelined datapaths. Each of the three target kernels requires fast access to a window of neighboring pixels; therefore, a sliding-window architecture provides the optimal solution. Fig. 7 illustrates the simple syntax required to generate a sliding-window architecture in OpenCL. Fig. 7a describes a shift register using the *#pragma unroll* statement. The *#pragma unroll* instructs the compiler to perform each loop iteration in parallel if data dependencies permit it.

The second stage of the kernel, illustrated in Fig. 7b, uses *#pragma unroll* to form a wide pipeline design. Fig. 7b is used to unroll the 2D convolution between two 3×3 filters with coefficients G_x, G_y and a 3×3 window of pixel values *rows(i,j)*. Each loop iteration is parallelized and, as a result, all 18 multiplications for the two convolutions are executed in parallel. Using both code snippets in Fig. 7, the OpenCL tool constructs a fully pipelined, 2D-convolution operator that produces a new result every clock cycle.

```
#pragma unroll
 for(int i=MAX_VAL;i>0; --i)
 {
      rows[i] =rows[i - 1];
 }
 rows[0] = frame_in;
```

a)

```
int x = 0;
int y = 0;
#pragma unroll
for(int i=0; i<3; ++i)
{
  #pragma unroll
  for (int j=0; j<3; ++j)
  {
    x+=rows(i,j)*Gx[i][j];
    y+=rows(i,j)*Gy[i][j];
    }
}
```

b)

Fig.7: a) OpenCL example code that generates a register chain; b) OpenCL example code that generates a sliding window for 2D convolution

IV. COMPARATIVE ANALYSIS: VHDL. vs. OPENCL

This comparative study between OpenCL and VHDL has been performed using three image-processing kernels (Sobel, Canny, and SURF). We first describe the experimental setup, followed by performance and productivity results.

A. RC platform and development tools

The selected image-processing kernels were implemented on 28nm Altera Stratix-V FPGAs. The Gidel ProceV was one of three selected platforms for this study. Development tools provided by Gidel included the PROCDeveloper's Kit for VHDL design and the OpenCL Board Support Package. In addition to the ProceV, the kernels were ported to Nallatech's PCIe385-D5 and Bittware's S5PH-Q boards using the same OpenCL code developed for the ProceV platform. The Nallatech PCIe385-D5 features a Stratix-V D5 FPGA, while the Bittware S5PH-Q uses the same Stratix-V D8 FPGA as the ProceV board.

B. RC architecture for selected kernels

The OpenCL architecture for the three different kernels is based on the same approach used for the VHDL design. Each kernel forms a fully pipelined datapath capable of processing one pixel every clock cycle. The OpenCL *__global* memory type manages data movement between the FPGA and external memory (*SDRAM or SODIMMs*). Inside the FPGA, each image-processing kernel is partitioned into smaller modules that perform key operations on each pixel. Each of these modules is described as an independent OpenCL *__kernel* that is connected using *cl_altera_channels* OpenCL vendor extension. The *cl_altera_channels* extension acts as a simple FIFO between modules, extending the depth of the datapath. Inside each module is a window buffer in addition to the primary logic performed on the input stream. The last module in the pipeline is connected to the *__global* output stream.

C. Performance comparison

The resource utilization and performance for the three selected kernels are shown in Table 1. The top three sections of the table present the Altera OpenCL results on three different platforms (Nallatech PCIe385, Gidel ProceV, and Bittware S5PH-Q). The fourth section shows the results of the VHDL design on the ProceV platform. The general trend apparent in Table 1 is that VHDL kernel designs use fewer resources while maintaining a similar clock frequency to OpenCL. In the best case, the VHDL design uses 30% of the logic needed by OpenCL design (HD720 Sobel filter on ProceV). On average, VHDL designs require 35% of the logic and 66% of the memory bits used by the OpenCL kernels (ProceV). The average percent difference in clock frequency across all kernels at every resolution is 4.09% (ProceV). An important performance measure in image processing is frames per second (FPS). The percentage difference in FPS is between 2% and 10%. In, summary, it can be observed that the OpenCL kernels in this study achieve similar performance to their VHDL counterparts at the cost of additional resources.

Table 1: Logic resources and performance results for image-processing kernels on all RC platforms

Method	Board	Resolution	Kernel	Logic	Util	DSP	Mem bits	Freq (MHz)	FPS
Altera OpenCL	Nallatech PCIe385_D5	VGA	Sobel	42378	25%	18	2152400	309.40	935
			Canny	49055	28%	63	2458064	309.40	821
			SURF	60765	35%	27	6366608	290.44	776
		HD720	Sobel	42389	25%	18	2201552	317.96	322
			Canny	49103	28%	63	2818512	304.32	308
			SURF	60864	35%	27	9610640	277.23	273
	Gidel ProceV_D8	VGA	Sobel	44310	17%	18	2109472	291.46	814
			Canny	50762	19%	63	2414880	293.34	769
			SURF	63271	24%	27	6324832	259.00	720
		HD720	Sobel	44329	17%	18	2158624	285.23	295
			Canny	50796	19%	63	2775328	279.88	285
			SURF	63369	24%	27	9568864	265.25	258
	Bittware S5PH-Q_D8	VGA	Sobel	41602	15%	18	2087008	305.52	896
			Canny	48332	18%	63	2392672	311.13	861
			SURF	59937	22%	27	6301088	272.18	765
		HD720	Sobel	41592	15%	18	2136160	296.82	312
			Canny	48305	18%	63	2753120	322.78	319
			SURF	60018	22%	27	9545120	277.85	274
VHDL	Gidel ProceV_D8	VGA	Sobel	13523	5%	12	1753264	283.13	795
			Canny	16268	6%	13	1815728	296.30	807
			SURF	26164	10%	32	2745520	279.80	778
		HD720	Sobel	13526	5%	12	1769648	278.87	302
			Canny	16473	6%	13	2052272	293.34	293
			SURF	26320	10%	32	3754160	281.53	284

One of the most important advantages of OpenCL designs is code portability across different platforms. The results in Table 1 illustrate how portable the OpenCL kernels were across all three platforms, featuring similar resource utilization and timing performance. The OpenCL kernel frequency ranged from 259-323 MHz with an average difference of 3.55% between the PCIe385 and S5PH-Q platforms.

D. Productivity comparison

Fig. 8 shows the tasks that were completed to finalize the RC design in VHDL and OpenCL. The percentage of overall time dedicated to each individual task is also shown in the pie charts of Fig. 8. The development time for the three image-processing kernels, using VHDL tools, was completed in six months (Fig. 8a). The majority of that time was dedicated to ensuring correct functional simulation (40%) of the three kernels. Each individual VHDL block was tested in simulation

using test benches. The meticulous VHDL approach gives developers the ability to build and verify the design down to individual bits. However, the same fine-grained approach hinders a less detailed and quicker functional verification.

During the synthesis phase, the synthesis tools provided by FPGA board vendors (e.g., Gidel's ProcWizard) interface VHDL code with board-specific resources such as memory and control signals. The place-and-route stage of the design is another VHDL design hurdle. In order to meet timing and power constraints, the VHDL design must be physically mapped and tested on the FPGA fabric, which is a well-documented productivity bottleneck [5]. Finally, when all design specifications are met, the bit file resulting from the compilation is used to program the FPGA. The host code that provides an interface between the FPGA board and the host processor represents another productivity obstacle because it lacks a common standard between different FPGA boards.

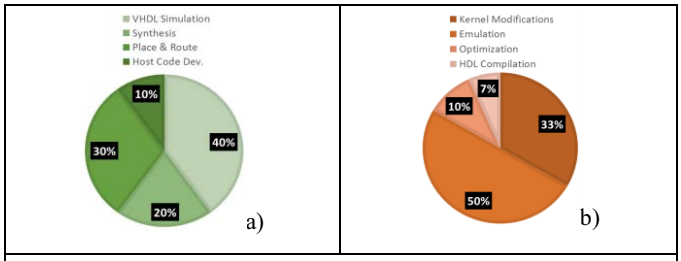

Fig. 8: Project development time and completed tasks; a) VHDL: 6 months; b) Altera OpenCL: 1 month

OpenCL tackles many of the VHDL productivity bottlenecks mentioned above. The allocation of time spent on different tasks for the OpenCL design flow is shown in Fig. 8b. The overall OpenCL project was developed over a span of 30 days. The largest portion of development time involved identifying errors in the code using the full system emulator. This debugging session was performed on an x86 host, with compile times closer to software (seconds). The emulator includes support for the GNU debugger and was launched in the same debug session as the host executable. Software breakpoints and memory inspections can be performed between the host and FPGA kernel, allowing developers to easily track data interactions between the RC platform and host CPU. Code modifications needed to correct the errors identified by the emulator comprised the second-most, time-consuming task during OpenCL development. The time in this section includes both modifications performed on the kernel and host software. OpenCL concurrently verifies both kernel and host code, saving development time.

The majority of the optimizations required by OpenCL kernels used *#pragma unroll* directives as discussed in Section III, and the *cl_altera_channels* OpenCL vendor extension as discussed in Subsection B above. Performing the hardware compilations for the different kernels is the final task in the OpenCL design flow. Compared to the VHDL design flow, another important gain in productivity is the time saved on VHDL compilations. Although Altera OpenCL still needs to perform a full hardware compilation to generate the AOC executable, this process can be delayed until after the design has been fully verified. Overall, we observed a 6× increase in productivity when using the OpenCL design methods.

V. CONCLUSIONS

In this study we compared traditional VHDL methods for application-development with the OpenCL design flow for three image-processing kernels (Sobel and Canny edge detectors, and SURF feature-extractor). On average the kernel design time using OpenCL was six times faster than traditional VHDL methods. OpenCL provides a higher level of abstraction to the programmer and includes development tools that postpone costly hardware compilations until the end of the design process. Our results showed that the productivity advantage of OpenCL comes at the cost of increased resource usage. VHDL designs achieved a more efficient use of resources (59% to 70% less logic), while maintaining similar timing constraints (255 MHz < f_{max} < 325 MHz). The percent difference in performance (FPS) for the OpenCL and VHDL kernels on a Stratix-V D8 FPGA is between 2% and 10%. Furthermore, OpenCL development tools enabled seamless portability of code between different FPGA boards. We successfully executed all three OpenCL kernels on three RC platforms with no code modifications. The resulting designs on all FPGA platforms achieved similar performances in terms of operational frequency and resource utilization.

ACKNOWLEDGEMENTS

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant Nos. EEC-0642422 and IIP-1161022. The authors gratefully acknowledge equipment and tools provided by Altera, Gidel, Nallatech, and Bittware.

REFERENCES

[1] B. A. Draper, "Accelerated Image Processing on FPGAs," in *IEEE Transactions on Image Processing*, Vol. 12, No. 12, Dec. 2003, pp. 1543 – 1551.

[2] L. Mintzer, "Digital Filtering in FPGAs," in *Proc. of the 28th Asilomar Conference on Signals, Systems and Computers*, 31 Oct. – 2 Nov., 1994, pp. 1373 – 1377.

[3] C. Gentsos, C. L. Soiropoulou, S. Nikolaidis, N. Vassiliadis, "Real-Time Canny Edge Detection Parallel Implementation for FPGAs," in *Proc. of the 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 12 – 15 Dec., 2010, pp. 499 – 502.

[4] D. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proc. of 7th International Conference on Computer Vision (ICCV)*, 20 – 27 Sept., 1999, pp. 1150 – 1157.

[5] S. Merchant, A. D. George, H. Lam, G. Stitt, "Strategic Challenges for Application Development Productivity in Reconfigurable Computing," in *IEEE Nat. Aerospace and Electronics Conference*, 16 – 18 Jul., 2008, pp. 209 – 218.

[6] T. S. Czajkowski, "From OpenCL to High-Performance Hardware on FPGAs," in *Proc. of the 22nd International Conference on Field Programmable Logic and Applications*, 29 – 31 Aug., 2012, pp.531-534.