A COMPLETE DEVELOPMENT ENVIRONMENT FOR IMAGE PROCESSING APPLICATIONS ON ADAPTIVE COMPUTING SYSTEMS

A. Christopher Moorman, and Donald M. Cates, Jr.

Khoral Research Inc.
6200 Uptown Blvd. NE Suite 200
Albuquerque, NM 87110, USA

ABSTRACT

Adaptive computing has always been a topic of great interest, providing a means to automatically map an application to specific hardware. The hardware may be configured to a specific application, thereby providing optimal performance. However, the largest benefit is that this configuration may be performed optimally during program execution. Unfortunately, adaptive computing is a relatively new area of research, therefore imposing serious complications when developing applications for adaptive hardware. This problem limits ACS development to hardware experts, prohibiting application specialists, such as image processing experts, from utilizing these systems. This paper will discuss a development environment that can bring the world of ACS application development to the IP expert with minimal hardware knowledge.

1. INTRODUCTION

While significant research has been done trying to utilize Adaptive Computing Systems (ACS), it has generally been approached as being a strict hardware problem. The number of tools to support application development on adaptive systems is minimal, thereby limiting their acceptance within the technical community. Until recently, the ability to develop applied solutions such as image processing applications on these systems was considered to be impractical, and rarely considered due to the limited resources available to support the development process for these systems.

Adaptive computing systems, commonly referred to as re-configurable systems, are generally made of integrated circuits called Field Programmable Devices. In particular, Field Programmable Gate Arrays (FPGAs) have been used extensively for military and commercial applications. They have been important particularly in research applications, because they offer low start-up costs and allow for design changes to be made easily. Such design changes would not be possible with fixed hardware systems. The standard FPGA consists of a block of logic devices spread out over a large mesh of connections to each element [5]. These logic elements are then surrounded by a series of I/O devices that connect to the chips interface pins. The advantage of these systems is that the user now has a chip that is capable of being programmed to perform the functionality of choice. The user has the ability to specify which connections are to be made and what functions are to be performed by each logic element. One problem is that these FPGAs are programmed via a series of configuration codes, which is currently done through hardware descriptive languages such as VHDL [1]. These languages work well to program control logic for complex logic devices, but, they become extremely complex when developing applied applications such as image processing. This means that an image processing expert would need to become a hardware expert to take advantage of these re-configurable systems without a friendlier development environment.

To bring these systems within reach of applications programmers, the development environment has to handle any hardware issues. In particular, for image processing applications, a image processing development environment would is needed. A major tool for image processing applications today is a visual programming environment. These visual environments allow users to construct complex applications via the connection of basic operations. This level of programming removes any lower abstraction layers, including machine level programming, allowing the IP designer to focus on the specific application. One such development environment is Khoros. While Khoros provides a visual programming environment, it also provides CASE (Computer Aided Software Engineering) tools to support the complete development process.

It should be obvious to the reader that visual programming environments will not entirely bridge the gap between this advanced hardware and true image processing applications. Additional steps must be taken. One such step is the advent of a new language called SA-C (Single Assignment C). While SA-C is actually a subset of C and not really a new language, it should be viewed as a tool to achieve the final goal, that of bringing re-configurable hardware to the IP developer. SA-C will provide a means of linking the FPGA hardware to the visual programming environment provided by Khoros, as will be shown later in this paper.

At this point, it is important to note why Image Processing has been chosen as the specific application area. In order to show this, a short explanation of some image processing issues is in order. Image processing applications generally operate on large images, requiring extremely fast throughput. Sometimes, even real-time throughput is required. Adaptive systems are capable of achieving this type of performance. In addition, their architecture is such that a user can extract parallelisms from the specific application. This allows a user to exploit the inherent parallelisms present in these windowing-type operations. Adaptive hardware systems also allow users to optimize performance by rearranging specific image operators to minimize data flow. The visual programming environment and performance monitoring could be used to achieve a reduction in

data flow and achieve enhanced performance from the application level, thereby, hiding any hardware details from the IP expert.

This paper discusses the details of Khoros, the development adaptive computing environment, SA-C, the language required to map an application to the FPGA hardware, and the overall design process to support adaptive computing systems.

2. Khoros Development Environment

Khoros is a complete software integration and development environment, providing development tools, a visual programming environment to support execution, as well as extensive libraries to support image and signal processing. These development tools support multi-language applications in an object-oriented fashion, allowing users to organize their applications into toolboxes. The cross-platform support, including numerous workstations, embedded platforms, and the HPC (High Performance Computing) arena, allow users to migrate their existing applications with minimal effort.

Khoros has been used for a wide variety of applications, ranging from Automatic Target recognition (ATR) and hyperspectral/multispectral imaging to medical and biological imaging. Khoros provides a significant number of utilities to the programmer, removing several abstraction layers and supporting many layers of programming. The developer has the option of programming at whichever level they prefer. An example is the ability to develop parallel algorithms within Advanced Khoros with minimal knowledge of MPI (Message Interface passing). Khoros provides data model support, allowing a user to work with input as objects, removing all file handling and memory management from the programming task. It also provides streaming models that may be used if performance is of a concern. The significant number of utilities within Advanced Khoros is only part of the complete development environment. The execution and development tools provide complete integration, maintenance and development support. The next sections discuss what each of the tools brings to the development process[4].

2.1 CASE Development Tools

The development tools provided within Khoros give the user the ability to develop, manage, and maintain complex software applications. The complete development suite consists of tools called Craftsman, Composer, and Guise.

Craftsman brings maintenance and management functionality to the software application, providing functions such as object creation, deletion, re-naming, and attribute modification, to name a few. Composer provides advanced functions to a specific software object, allowing users to edit source, generate code, compile the existing application and modify the GUI (Graphical User Interface). Guise, which allows the user to create and modify the applications GUI, provides a graphical representation of the UIS (User Interface Specification) under development. These tools are capable of supporting numerous languages and object types within Advanced Khoros, allowing a user to develop applications for even new languages, such as SA-C, or any user-created languages. The developer need only add the source necessary to provide the needed functionality. The tools will handle the remaining tasks.

With this environment, the user now has the ability to develop, maintain, and manage these SA-C algorithms for the FPGA hardware. The only remaining concern is the method of execution provided by the visual programming environment.

2.2 Visual Programming Environment

Cantata provides a visual programming environment to the developer's applications. This visual environment provides a user with the ability to interconnect a series of basic routines, or primitives, which could be used to develop larger and more complex applications. By providing the visual interface, the developer has the ability to rearrange an application based on performance or functionality or even hardware limitations. An example of Cantata can be found in figure 1.

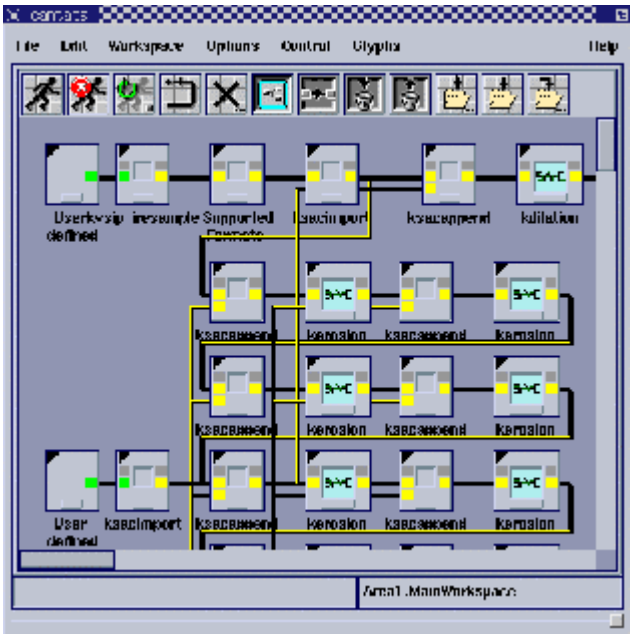

Figure 1. An sample of a workspace in the visual programming environment, Cantata, provided within Advanced Khoros.

In addition to providing a means of visual optimization for applications, visual programming also abstracts lower programming levels from a user, allowing applied experts to develop context specific applications with little or no programming knowledge. Image processing experts can develop complex image enhancement and recognition applications without writing a single line of code.

Additional Benefits to a visual programming environment:

- Abstraction of low-level programming.
- Abstraction of hardware programming
- A means for visual modification and optimization.

Cantata also hides any hardware and execution concerns from the developer while providing multi-architecture support and distributed computing capabilities.

An ATR workspace using SA-C routines is displayed in figure 1. The execution of these routines is currently on the local-host. The ability to execute SA-C objects on the FPGA hardware is still under development. The user will ultimately have the option of executing these routines locally as well as on the embedded systems. This capability allows the image processing expert to utilize existing SA-C library routines within Cantata, develop the IP application, and execute these routines on the FPGA without knowledge of the hardware.

Profiling and benchmarking capability will also be added to Cantata, providing visual performance feedback. This will allow users to modify the hardware mapping process or to identify any bottlenecks within the application. Additional information on this topic can be found in Section 4.

3. SA-C Single Assignment C

Single Assignment C (SA-C), was developed to minimize the problems associated with the hardware mapping procedure. While it is true that SA-C is a subset of C, it is important to note that it has been enhanced to truly support IP applications. The functionality removed was due to hardware mapping issues. The next sections discuss why SA-C was created and how it has been enhanced to support IP applications.

3.1 Language Overview

To help understand why SA-C was created, a list of objectives is shown below [2]:

- Efficient Expression of IP Constructs.
- Tailored to ACS Platforms.
- Ease of use.

These three objectives, taken together, express the point that SA-C was developed to bring the IP and Adaptive Computing communities together.

The ability to support multidimensional arrays is crucial in IP applications. The ability to access sub-structures within large images is also important for window-based and template-based procedures. SA-C addresses both of these issues while providing a significant number of statistical operations over multiple regions of the images.

As mentioned earlier, there are a significant number of problems that occur when mapping software to FPGA hardware. To eliminate some of these problems, certain features of the C language were removed from SA-C. For example in SA-C, only single assignment is allowed, and no pointer or recursion is supported. In addition, SA-C was developed to support the notion of Data Flow Graphs (DFG's), which are used for hardware mapping.

Finally, SA-C had to be easy to use before any real acceptance could be realized. Therefore SA-C follows the C syntax. In addition, it can be called from C and C++, and with the integration of SA-C into Khoros, a complete development environment has been created.

It is important to stress that SA-C is not an inferior language. In-fact, quite the opposite is true. Reassignment, pointers, and recursion are lost, but true IP support is gained. C provides minimal support of multiple dimensional arrays by allowing arrays of arrays requiring a series of mallocs, with no real data access methods. Users are forced to control data access, boundary conditions and overlapping. SA-C hides all this from the user, allowing the user to access images as windows at any focus within the complete image. Since SA-C controls the data access methods, it is capable of exploiting parallelisms that may be present for a specific application.

Finally, C does not provide multiple-bit precision, which SA-C does. Multiple-bit precision is crucial for optimized FPGA applications. Multiple-bit precision supports proper resource allocation by utilizing the necessary hardware to support minimal bit precision.

3.2 Image Processing Features

As mentioned earlier, image processing applications are generally window-oriented or template-oriented. Many of the operations such as convolution and basic filtering are kernel based. This type of manipulation is data intensive, requiring a significant number of data retrievals. SA-C handles all of the data access and allocation, allowing users to manipulate images using stencils, windows, and slices within an image. This capability allows for complex routines like convolution to be implemented with only a few lines of code. Looping constructs have been enhanced to support specific data acquisition procedures, significantly reducing the amount of code needed for data access.

Another important feature required within image processing is the generation of image statistics. A significant number of statistical methods have been incorporated within SA-C. Some of these include scalar reductions like min, max, median, and mean, as well as advanced measures like histogram statistics. SA-C supports statistical operations on a complete image, or on portions or regions within the image. This functionality actually gives SA-C the appearance of a higher level language, allowing users to specify operations with mere key words rather than lengthy function calls or code segments.

4. Complete Development Procedure

The details of some items from the development environment have been presented, and to some, it may be apparent that the notion of hardware mapping has not been addressed at this point. The integration of SA-C and Khoros provide the complete environment to the standard applications developer. These tools provide a means of developing complex image processing applications with SA-C, and executing them locally from within Cantata. In addition, if pre-compiled versions exist for the FPGA hardware, the user can execute these rather than the local versions within Cantata.

The intention is to provide a user with a complete development system, which means the IP applications should be executable locally, executable on the FPGAs using pre-compiled versions of the software, and executable on FPGAs using no pre-compiled versions. The first two methods of development and execution are specifically designed for the image processing expert. They require little or no hardware knowledge. The final step in the development process is the development of these pre-compiled

libraries. While hardware knowledge is required for this step, it is merely from a optimization standpoint rather than a development standpoint. The complete development process is be shown in figure 2 shown below[2].

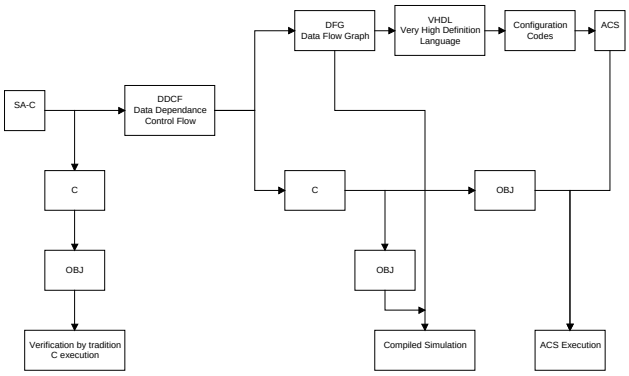

Figure 2. ACS Design Flow Diagram.

The initial branch of the SA-C block is what is achieved by the integration of SA-C with Khoros. This provides the user with the ability to execute on a local host. The SA-C source is compiled to a standard C version, which is compatible with Khoros, thereby providing portability across multiple platforms.

The remainder of the design process provides optimization and hardware mapping to the ACS platform. It is believed that the optimization and mapping can be efficiently done through the use of Data Flow Graphs (DFGs) and Data Dependence and Control Flow Graphs (DDCFs).

The DDCF graph provides an efficient means to make optimizations to the source. Some of these would include loop fusion, unrolling, and reordering. In addition to optimizations, it provides an optimized source to the DFG.

The lower branch immediately after the DDCF in figure 2 provides a means of optimizing the SA-C application. The tools will be modified to support the visual modification of this DDCF. The optimized code can then be compiled down to the local hardware or the ACS. This capability is extremely important in evaluating the performance achieved after optimization. A visual representation of the performance measures and benchmarking will also be added to Cantata, providing additional performance feedback.

The top branch of figure 2 is representative of the complete development process. The previous two methods mentioned were developed to bring the adaptive computing systems within reach and to reduce prototyping and application development time. This last step is representative of the ground up development process.

After the DDCF graph, the process flows into the DFG. This step provides an additional level of optimization from a different perspective. These are actually a flatter version of the DDCF. This is the point where machine-specific transformations occur. It should be apparent that some sort of hardware language would be required to get to the FPGA level. The design process will now have several options at this point, the utilization of pre-compiled VHDL and macros, or the generation of completely new VHDL source. The latter opinion would add significant compilation time to the development process but would allow for efficient VHDL to be generated for the users specialized image processing applications.

5. SUMMARY

The goal of this project is to provide a complete design environment for the image processing expert which could be used to develop applications for adaptive computing systems. The integration of the SA-C language with Khoros provides such an environment. Khoros brings the tools needed to develop, manage and maintain software objects, while providing a means of execution via a visual programming environment. The SA-C language provides advanced functionality specifically designed for image manipulation and image processing applications. When these are combined with pre-compiled SA-C libraries for a specific FPGA hardware, IP experts can develop complex IP applications with no hardware knowledge. The execution of these applications is supported within Cantata locally or on the FPGA hardware. This part of the design process will significantly reduce prototype and development time of IP applications on the FPGA, as well as making it possible for IP designers to utilize this advanced hardware.

The development of these SA-C libraries and routines for the FPGA is accomplished via the DFG and DDCF graphs. Optimized VHDL is generated from these graphs, which is then directly mapped to the hardware by using the FPGA vendor-supplied software. This portion of the development process is somewhat time consuming, and requires additional compilation time. However, the time should be reduced due to the generation of optimized VHDL. Both methods combined, provided a complete IP development solution while reducing development time and improving performance of generated applications.

6. REFERENCES

[1] D. Perry. *VHDL*. McGraw-Hill, 1993.

[2] Najjar W., Bohm W., Draper B., and Beveridge R. “Optimized Compilation of Visual Programs for Image Processing on Adaptive Computing Systems”. *Cameron Project Kickoff Presentation*, Washington D.C., USA, 1998, pages 3-4.

[3] Bohm W. “The Sassy Language - Version 0.6”. Colorado State University web page: http://www.cs.colostate.edu/bohm/, 1998.

[4] Khoral Research’s development research web page: http://www.khoral.com, 1998.

[5] Hammes J., Draper B., and Bohm W. “Sassy: A language and optimizing compiler for image processing of reconfigurable computing systems”. Colorado State University: http://www.cs.colostate.edu/bohm/, 1998.

[6] Microsoft Research’s Speech Technology Group web page: http://www.research.microsoft.com/research/srg/.