The Cameron Project: High-Level Programming of Image Processing Applications on Reconfigurable Computing Machines[1]

W. Najjar, B. Draper, A.P.W. Böhm, R. Beveridge

Computer Science Department – Colorado State University

www.cs.colostate.edu/cameron

1 Introduction

1.1 Reconfigurable Computing and Image Processing

Reconfigurable computing maps computation onto flexible and reprogrammable hardware. A typical reconfigurable computing (RC) system consists of a host processor (with a traditional architecture) and one or more reconfigurable co-processors. Proposed hardware architectures for reconfigurable co-processors fall in two broad categories [4]: *netlist computers* with uniform arrays of fine grained logic units based on Field Programmable Gate Arrays (FPGAs), and *functional units-based computers* with (possibly non-uniform) arrays of higher-level functional units (such as multipliers and adders) [2, 10] or full-fledged processors [11]. The former model is one used by the first RC systems and still the most commonly used today.

Applications for reconfigurable computers are typically written as a combination of C or C++ host code and FPGA configuration codes written in a hardware description language such as VHDL or Verilog. Using this approach, several experimental projects have already demonstrated RC systems capable of achieving supercomputer level performance at a cost comparable to a desktop PC or workstation [4].

One application domain where RC systems can have a tremendous impact is image processing (IP). Many IP operators apply local transformations to large image arrays, resulting in thousands of potentially parallel operations. At the same time, the advent of digital cameras, digital video, and multimedia databases (including the web) has made the need for fast and cheap image processors more acute.

1.2 Project Motivation

Although image processing operators provide opportunities for massive parallelism, there have only been a few applications of reconfigurable technology in image processing. We believe that RC *programmability* is the barrier. Most IP experts either specify programs in a high level language (such as C) or else graphically connect standard image processing operators in a visual programming environment such as Khoros [5]. IP programmers are neither experts in digital circuit design, nor are they generally familiar with hardware description languages. Without supportive programming environments for IP application development, reconfigurable systems will go unused by the application domains where they are most relevant.

The goal of the Cameron project is provide a graphical programming environment, high-level language, optimizing compiler, and debugging and performance monitoring tools for image processing on reconfigurable computers. In general, we seek to shift the programming paradigm for reconfigurable computers from hardware-centered to software-centered, thereby making them accessible to image processing experts and portable across RC platforms.

The technology underlying the Cameron project is *dataflow analysis*. New image processing operators in Cameron are written in SA-C, a single-assignment version of the C programming language (with extensions for image processing) that can be directly translated into dataflow graphs. Once the program is transformed into a dataflow graph, dataflow optimizations such as

[1] This work is supported by DARPA under US Air Force Research Laboratory contract F33615-98-C-1319.

loop merging, loop unrolling, and data blocking can be applied (along with standard optimizations to minimize data traffic and maximize operation density on the RC system. Because there is a direct correspondence between dataflow graph components and FPGA hardware components (e.g. each variable corresponds to one wire), optimized dataflow graphs can be mapped onto FPGA configurations.

In addition, the VSIP image processing library defines a set of standard IP operators [VSIP]. By implementing the VSIP library in SA-C and integrating SA-C with the Khoros software package, we provide a graphical programming environment for IP on reconfigurable systems.

2 Related Work

Even though the reconfigurable computing paradigm was first proposed several decades ago [3], it is only now emerging as a high-performance and cost-effective alternative to traditional parallel computing models. The rapid emergence of reconfigurable computing is in large part a result of advancements in field-programmable devices and increased IC density.

As mentioned above, a typical RC program consists of C or C++ host code and FPGA software written in a hardware description language such as VHDL or Verilog. Since hardware description languages (HDLs) are not immediately accessible to conventionally trained programmers, HDLs with C-like syntax (e.g. Handel-C [9]) have been proposed. Unfortunately, this approach still requires the manual or human translation of algorithms into logic circuit structures.

Despite the awkward programming situation, several researchers have demonstrated success with specific image processing applications using reconfigurable systems. Hartenstein, et al used an RC system for real-time jpeg image compression [6], while Benedetti and Perona used FPGAs for extracting and tracking image points in real-time [7]. FPGAs were also used for real-time stereo as part of a face detection/tracking system embedded in an entertainment application [8].

3 Image Processing on Reconfigurable Systems

3.1 Image Processing Requirements

As a rule, image processing applications involve local operations applied across large images; convolution and cross-correlation, for example, are canonical IP operators. To take advantage of this implicit parallelism, a variety of vector and/or pipeline processors have been developed for image processing. Unfortunately, image processing also presents a number of problems that have limited the effectiveness of these SIMD-style systems:

1. *Large images*: Images that are 10,000 x 10,000 pixels are common in aerial and satellite surveillance, and even larger amounts of data are coming in the future (see the DARPA/ITO Surveillance Challenge). For such large images, the communication bandwidth between the host and RC processors becomes the limiting factor, implying that data traffic must be minimized.
2. *Complex applications, simple operators:* ARAGTAP (a typical military IP application) contains over 50 individual IP operators, most of which involve local data calculations If images are transmitted from the host to the coprocessor and back to the host for each operation, data transmission times will overwhelm any benefit from parallel calculation.
3. *Multiple data access patterns:* Although many IP operators consist of concurrent local window-based operations (e.g. convolution), others have different data access patterns. The Fourier transform is one example of an IP operator with a complex data access pattern; cumulative sum (used in computing cumulative histograms) is another. To exploit the parallelism in IP, systems must be flexible enough to adapt to different data access patterns.
4. *Varying pixel precision:* Some images have pixels with twelve bits (or more) of information, while others have only eight or four bits or precision. Moreover, many operators (e.g. thresholding) create binary

images for later processing. Systems that can compute with different word sizes therefore have a significant advantage.

5. *Multiple run-time environments:* By their very nature, IP applications are run on a variety of platforms, from embedded systems to supercomputers. It is essential therefore that IP applications be platform independent and based on standardized modules.

3.2 Reconfigurable Computing Systems and IP Applications

The advantages of RC systems have been extensively discussed in the literature. Their specific potentials for IP applications are:

1. The exploitation of large-scale yet flexible fine grained parallelism.
2. Operations with variable sized numbers, including twelve, eight, four and one bit integers.
3. The integration of storage and computation within RC cells to reduce data traffic.

Reconfigurable computing systems however do face several challenges that must be addressed before they can become a viable option for IP applications. These are:

1. *Programmability*: Currently, the gap between high-level application programming and low-level logic design is very large, and prevents IP experts from programming RC systems.
2. *Placement and Routing*. The time required to place and route a logical circuit design onto a reconfigurable processor is excessive for large applications.
3. *Performance*: On-chip storage resources are crucial in limiting the impact of data traffic on performance through the exploitation, by the compiler, of fine-grained data locality.
4. *Reconfigurable Computing Models*: The two models of RC machines, fine-grained "netlist" computers and coarse-grained functional unit based ones, define two extremes of a wide spectrum. Each model has its advantages and drawbacks. The challenge is to determine which features are best suited for a given application domain.

4 Overview of Cameron

The main objective of this project is to develop software tools that will simplify the programming of reconfigurable systems and make them accessible to a wider population of users. This is accomplished by developing new RC programming environments that are usable by application programmers familiar with high level languages such as C and C++ and unfamiliar with logic circuit design and HDLs.

Since the main application domain of Cameron is image processing, we have chosen Khoros as its graphical user interface. Khoros is a widely popular program development environment designed for IP applications. In addition to software development, program services and visual programming, it supports simulation, data analysis and visualization [Rasure 94]

Program modules that run on the RC machine are written in a single-assignment subset of C called SA-C. The SA-C subset has been extended to support IP constructs as well as compilation for a hardware implementation (Section 5). SA-C programs are compiled into a Data Dependence and Control Flow Graph (DDCFG) which is an excellent platform for:

1. Separating code that will execute on the RC hardware from code for the host processor.
2. Extracting parallelism and analyzing fine-grained data locality.
3. Performing extensive compiler optimizations.
4. Generating VHDL code for the target reconfigurable machine.

SA-C programs are tightly integrated into the Khoros environment. An important aspect of Cameron is the set of compiler optimizations that are offered to the user to fine-tune the mapping of IP algorithms on reconfigurable platforms. To enhance portability across platforms, Cameron relies extensively on the image processing libraries in VSIP[2].

5 The SA-C Language

The rationale behind adopting a subset of C is that C is not well suited for expressing IP algorithms on reconfigurable machines:

- The sequential semantics of C and its liberal use of pointers and aliases makes it hard to analyze and to extract parallelism and locality information. In all cases the compiler must assume a worst case and would

[2] VSIP is a standard API library for Vector, Signal and Image Processing applications (www.vsip.org).

therefore unduly introduce artificial dependencies.

- C is not tailored to IP applications. It does not support true *n-dimensional* (nD) arrays. Instead, multi-dimensional arrays are represented by arrays of arrays which requires the frequent use of clumsy *mallocs*.
- C does not support variable bit resolution, which is necessary for achieving high computational densities on RC machines.

```
/* Memory allocation of two n-dimensional arrays */
A = (int**) malloc (n  * sizeof(int*));
B =  … malloc … ;
for (i = 0; i<n; i++)
   {A[i]= ...malloc…;
     B[i] = … malloc..; }
h = s/2;

/* Convolution with a fixed mask M */
for (i = h; i< n-h-1; i++)
  for (j = h; j<n-h-1; j++) {
     c = 0;
    for (p = 0; p<s; p++)
            for ( q = 0; q<s; q++)
              {c += M[p][q]*A[i-h+p][j-h+q];}
     B[i][j]=c;}
```

Figure 1. Convolution example in C.

```
int8 B[n,n] = for window W[s,s] in A  {
                    int8 ip =  for w in W dot m in M
                               return( sum(w*m) );
                 }  return( array(ip) );
```

Figure 2. Convolution example in SA-C.

Figure 1 shows a convolution example that illustrates the differences between C and SA-C. The main features of SA-C are:

1. *n-Dimensional Arrays,* with support for substructure access methods. Examples include:
   - *i.* The access to a k-dimensional slice out of an n-dimensional array such as a vector within an image.
   - *ii.* Support of stencils which are predefined neighborhoods within an array and are used in morphology operations with a mask.
   - *iii.* Support of windowing operations. A window is an n-dimensional array within another n-dimensional array of larger size. They simplify the expression of convolution and image filtering operations.
2. *Variable Precision.* SA-C supports variable precision integers and fixed-point variables. It implements compile time checking of sizes as well as size coercion.
3. *Loop Returns.* A SA-C loop can return an array, a reduced value or an expression. The shape of the array is determined by the loop generator and array concatenation is supported. Reductions can be scalar (min, max, sum, product, mean), arrays (of min-indices or max-indices) or statistical (histograms).

6 Compiler Optimizations and Mapping to Hardware

SA-C codes that are intended for execution on a RC machine are compiled into a dataflow graph. A large set of compiler optimizations are available to the user to allow the customized tuning of these codes for the particular target machine. These optimization allow tradeoffs between:

- i. *Parallelism*: which determines the number of concurrent loop iterations executing on target machine.
- ii. *Pipelining*: which is determined by the size of loop body executing on the machine.
- iii. *Data bandwidth*: which is the amount of data required by the loop code executing on the RC platform to be moved into and out of the host memory.
- iv. *Logic resources*: which is the number of logic block and other resources, such as FPGA routers, used by the computation.

Compiler optimizations have a wide range of impacts on resource utilization on a RC machine. Loop merging increases the size of the loop body and provides better data locality and the possibility of pipelining within the loop body. It also offers the potential for the reuse of logic blocks on the FPGA when dynamic reconfiguration is supported. Loop merging might require a small increase in data bandwidth between the host and the RC machine. Loop unrolling requires a large increase in data bandwidth to and from the RC machine as well

as an increase in resources on the RC machine. All these and other optimizations would benefit from buffers within each cell or logic block of the RC machine as well as from on-chip dual-ported memory to cache the data being used by the codes in the RC machine. Some recent RC architectures, such as the UCI Morphosys M1 chip [10], do provide a high data bandwidth between the RC system memory and its processing units as well as on chip interleaved dual-ported SRAM memory banks.

Figure 3 shows the SA-C code for a square root function of a 12-bit integer. The dataflow graph of the loop body is shown in Figure 4. The loop iterates six times on this body. The required precision on the result of each loop iteration increases by one bit each time: the result of the first iteration can be expressed in one bit, that of the last in six. Taking advantage of this fact can reduce the size of the loop body implementation on an FPGA-based machine at the cost of customizing a different stage for each of the six iterations. Figures 5, 6 & 7 show, respectively, a simple iterative implementation, an unrolled loop implementation and a pipelined and unrolled loop implementation. For implementation on an FPGA-based RC machine, the simple solution in Figure 5 would probably result in a short clock cycle as well as a small area. The unrolled solution in Figure 6 reduces the amount of memory traffic at the cost of increased cycle and area. The one in Figure 7 reduces the clock cycle time to the delay of a loop body plus a latch at the cost of additional area. This example illustrates many of the tradeoffs in the implementation of sequential and parallel loops on RC machines.

7 Conclusion

Reconfigurable computing is novel computing paradigm that offers huge performance potentials in many application domains including image processing. While recent advances in VLSI technology have been the main enabling factor of RC, the programmability of these systems remains one of the major roadblocks to its wider acceptance within the community of scientific and engineering application programmers. The main thrust of the Cameron Project is to develop and implement accessible software solutions that open the potentials of RC to a wide user community.

Several IP algorithms include operations where one of the operands is a compile-time known constant. These can benefit from compiler based partial evaluation that reduces the number and complexity of arithmetic operation and therefore increases the computational density.

References

[1] VSIP Consortium. Vector Signal Image Processing Forum, October 1997, www.vsip.org.

[2] C. Ebeling, D.C. Cronquist, and P. Franklin. RaPiD: reconfigurable pipelined datapath. In Proc. Field Programmable Logic, pages 126--135. Springer-Verlag, 1996.

[3] G. Estrin etal. Parallel processing in a restructurable computer system. IEEE Trans. on Electronic Computers, pages 747 -- 755, Dec.1963.

[4] W.H. Mangione-Smith et al. Seeking solutions in configurable computing. IEEE Computer, 30(12):38 -- 43, December 1997.

[5] J. Rasure and S. Kubica. The Khoros Application Development Environment. Experimental Environments for Computer Vision and Image Processing, H. Christensen and J. Crowley (eds), World Scientific Press, 1994.

[6] R. Hartenstein, et al. A Reconfigurable Machine for Applications in Image and Video Compression. In Proc: Conf. on Compression Technologies and Standards for Image and Video Compression, Amsterdam, 1995.

[7] A. Benedetti and P. Perona. Real-Time 2-D Feature Detection on a Reconfigurable Computer. In Proc: Conference on Computer Vision and Pattern Recognition, Santa Barbara, 1998.

[8] T. Darrell, et al. Integrated person tracking using stereo, color, and pattern detection. In Proc: Conference on Computer Vision and Pattern Recognition, Santa Barbara, 1998.

[9] OXFORD Hardware Compilation Group. The Handel language. Technical Report, Oxford University 1997.

[10] F. Kurdahi and N. Bagherzadeh The Morphosys Project. www.eng.uci.edu/morphosys.

[11] E. Waingold et al. Baring it all to Software: Raw Machines, IEEE Computer, September 1997, pp. 86-93.

```
uint6 v =
  for uint4 i in [6] {
      uint12 nasq = ((asq + a) << 2) | 1;
      uint6 sa = a << 1;
      next tvsq = (tvsq << 2) | ((vsq >> 10) & 3);
      next vsq = vsq << 2;
      next a, next asq =
                        if (nasq <= tvsq) return (sa |1, nasq)
                        else return (sa, asq << 2);
      } return (a);
```

Figure 3. SA-C code of *sqrt* function.

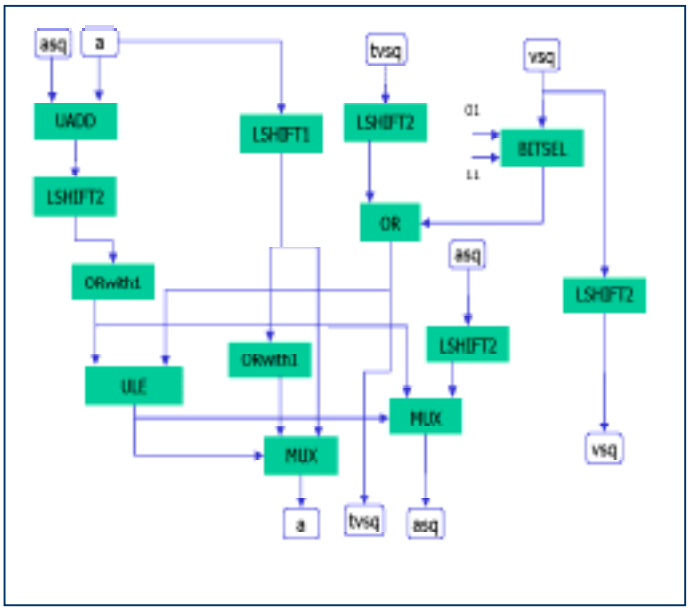

Figure 4. Dataflow graph of *sqrt* function loop body.

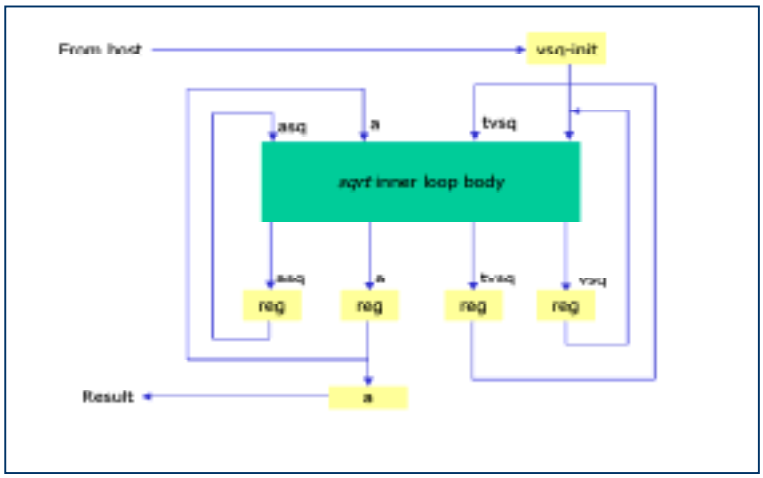

Figure 5. Simple iterative loop implementation.

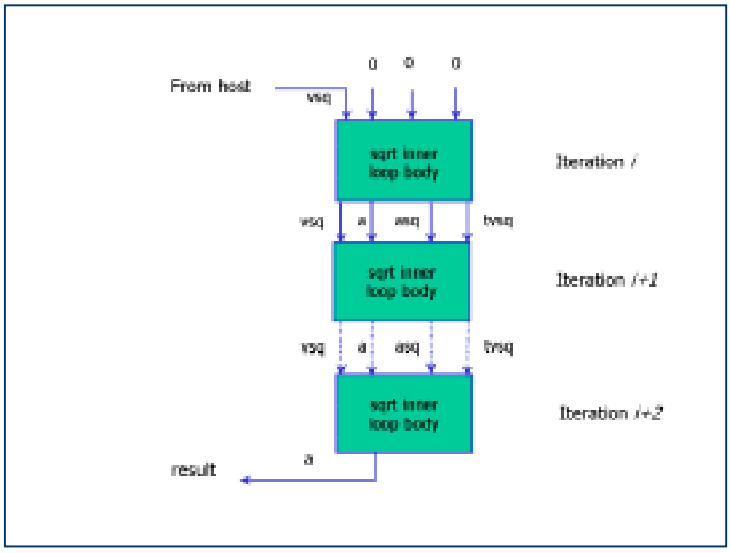

Figure 6. Unrolled loop body.

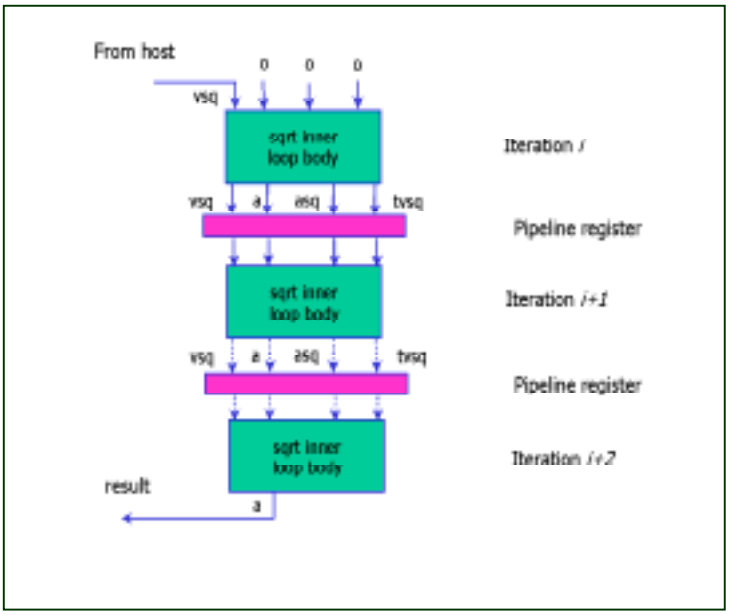

Figure 7. Pipelined and unrolled loop implementation.