In Proceedings of the Conference on Design and Architectures for Signal and Image Processing, Edinburgh, Scotland, October 2010.

AUTOMATED GENERATION OF AN EFFICIENT MPEG-4 RECONFIGURABLE VIDEO CODING DECODER IMPLEMENTATION

Ruirui Gu, Jonathan Piat, Mickael Raulet, Jorn W. Janneck, Shuvra S. Bhattacharyya

Department of Electrical and Computer Engineering, University of Maryland
College Park, MD, 20742, USA, Email: rgu, ssb@umd.edu
IETR Laboratory, UMR CNRS 6164, Image and Remote Sensing Group,
35043 RENNES Cedex, FRANCE, Email: jonathan.piat, mraulet@insa-rennes.fr
United Technologies Research Center, Berkeley, CA, USA, Email: jannecjw@utrc.utc.com

ABSTRACT

This paper proposes an automatic design flow from user-friendly design to efficient implementation of video processing systems. This design flow starts with the use of coarse-grain dataflow representations based on the CAL language, which is a complete language for dataflow programming of embedded systems. Our approach integrates previously developed techniques for detecting synchronous dataflow (SDF) regions within larger CAL networks, and exploiting the static structure of such regions using analysis tools in The Dataflow interchange format Package (TDP). Using a new XML format that we have developed to exchange dataflow information between different dataflow tools, we explore systematic implementation of signal processing systems using CAL, SDF-like region detection, TDP-based static scheduling, and CAL-to-C (CAL2C) translation. Our approach, which is a novel integration of three complementary dataflow tools — the CAL parser, TDP, and CAL2C — is demonstrated on an MPEG Reconfigurable Video Coding (RVC) decoder.

1. INTRODUCTION

Upcoming MPEG video coding standards are intended to increase the quality and flexibility of complex and versatile future video coding applications. Since 1988, several MPEG standards have been developed successfully based on available hardware technologies and software support. Early MPEG standards (MPEG-1 and MPEG-2) were specified by textual natural-language descriptions. Starting with MPEG-4, reference software written in C/C++ became the formal specification of the standard. Written in a sequential programming language, this reference software describes a sequential algorithm, effectively hiding the considerable inherent concurrency of a video decoder. Furthermore, the reliance on global memory and state makes the reference description difficult to modularize, resulting in a very monolithic specification format.

At the same time, multi-core devices, which incorporate two or more processors on the same integrated circuits, are becoming increasingly relevant to the design and implementation of DSP systems (e.g., see [1]). Efficient deployment of video processing applications on multi-core systems requires effective parallel exploitation of task level concurrency in order to improve system performance. The drawbacks of existing video standard specification formats and the increasing importance of multi-core platform technologies motivated the development of the Reconfigurable Video Coding (RVC) standard [2]. The key concept of RVC is to enable design and specification of decoders at a higher level of abstraction than that provided by generic, monolithic C-based specifications, and improve high level application analysis and optimization, including exploitation of parallel processing resources.

Dataflow-based programming, with its intrinsic concurrency, is employed in a wide variety of commercial and research-oriented tools related to digital signal processing (DSP) system design. Dataflow modeling techniques underlie many popular graphical tools for DSP system design (e.g., see [3]). In DSP-oriented dataflow graphs, vertices (*actors*) represent computations of arbitrary complexity, and an edge represents the flow of data as values are passed from the output of one computation to the input of another. A variety of dataflow-based languages and tools have been developed for design and implementation of embedded DSP systems. Although all of these languages share the property of data-driven communication between actors, distinct languages generally differ in terms of specialized dataflow modeling features and associated support for analysis and optimization techniques.

Synchronous dataflow (SDF) is a specialized form of dataflow that is streamlined for efficient representation of DSP systems [4]. SDF is a restricted model that handles a limited sub-class of DSP applications, but in exchange for this limited expressive power, SDF provides increased potential for static (compile-time) optimization of DSP hardware and software (e.g., see [5]).

A number of dataflow-based formalisms have been developed to describe applications that involve dynamic dataflow behavior. For example, CAL [6] is a language for specifying dataflow actors in a way that is fully general (in terms of expressive power), while clearly exposing functional structures that are useful in detecting important special cases of actor behaviors (e.g., SDF or SDF-like actor behaviors). The CAL language, in terms of its high level of abstraction, is similar to the Stream-Based Functions (SBF) model of computation [7]. Both models share common features relating to modeling of dynamic dataflow behaviors. However, SBF combines the semantics of both dataflow models and process network models, while CAL extends the dataflow model by enriching the properties of individual actors. Furthermore, CAL is a fully-featured programming language, providing both an abstract, dataflow model of computation as well as a comprehensive set of operators and other semantic features for completely specifying the internal behavior of dataflow components.

The DIF language (TDL) provides a standard approach for specifying DSP-oriented dataflow graphs at a high level of abstraction that is suitable for both programming and interchange (across different dataflow-based languages or tools) [8]. TDL provides a unique set of semantic features for specifying graph topologies, hierarchical design structure, dataflow-related design properties, and actor-specific information. TDP (The DIF Package) accompanies TDL, and provides a variety of intermediate representations, analysis techniques, and graph transformations that are useful for working with dataflow graphs that have been captured by TDL.

In order facilitate integration of TDL and TDP into design flows, we present in this paper a common XML-based format called *DIFML*. DIFML is designed for structured exchange of design information between different dataflow-based specification formats, such as TDL and CAL.

In previous work, we have formulated systematic statically schedulable region (SSR) detection and implemented SSR region detection using TDP (The DIF Package) [9]. Code generation from CAL to C (CAL2C) has also been developed in previous work [10], and we have explored integrated application of CAL, TDP, and CAL2C using manual techniques [9]. Simulation results from such manual integration demonstrated that the integrated application leads to improved exploitation of parallelism [11].

This paper builds on these previous efforts, and presents an automated approach for integrating SSR derivation into implementations that are synthesized from CAL specifications. To facilitate such an automated and integrated design flow, we also present in this paper a new XML-based format, called DIFML, which we have developed to exchange information between different dataflow tools. We present experimental results on a reconfigurable video coding application to demonstrate the effectiveness of our automated toolset.

2. RELATED WORK

2.1. Reconfigurable Video Coding

The desire for a more compositional approach to building existing and future video standards, and for a shorter path to parallel implementation has led to the development of the reconfigurable video coding (RVC) standard [2]. The MPEG RVC framework is a new standard under development by MPEG that aims at providing a unified high-level specification of current and future MPEG video coding standards. Rather than building a monolithic piece of reference software, RVC standardizes an "Abstract Decoder Model" (ADM) composed of a network that interconnects a set of video coding tools with uniform interfaces extracted from a library. Decoder descriptions are composed from that library, which permits a wide range of decoding algorithms.

The MPEG RVC framework is currently under development in MPEG as part of the MPEG-B part 4 [12] and MPEG-C part 4 [13] standards. An abstract decoder is built as a block diagram or "network" in which blocks define processing entities called functional units (FUs), and connections represent data paths between the FUs. Such a network is described in MPEG-B part 4 as an XML dialect called the FU Network Language (FNL). RVC also provides in MPEG-C part 4 a normative standard library of FUs, called the "Video Tool Library (VTL)", and a set of decoder descriptions expressed as networks of FUs.

CAL is currently chosen as the language to express the behavior for the coding tools of the library (VTL). Such a representation is modular and helps in formulating the potential configuration of decoders in terms of modifications of network topologies. The ADM is a CAL dataflow program that constitutes the conformance point between the normative RVC specification and all possible proprietary implementations that have to be generated to decode the incoming bitstreams. Thus the MPEG RVC standard leaves open the platforms and the implementation methodologies that can be used to generate proprietary RVC implementations. This provides great flexibility in the development of future RVC technologies and implementations.

2.2. Design Flow

Embedded system design and implementation can be a time-consuming process requiring intensive effort, resources, and time. Hardware description languages (HDLs), such as Verilog HDL [14], are widely used in the design of embedded systems. In an attempt to reduce the complexity of designing in HDLs, which have been compared to the equivalent of assembly languages, a variety of efforts have emerged to raise the abstraction level of associated design processes. For example, companies such as Cadence, Synopsys and Agility Design Solutions are promoting SystemC as a way to combine high level languages with concurrency models to allow faster design cycles for FPGAs than is possible using traditional HDLs.

Approaches based on standard C or C++ (with libraries or other extensions allowing parallel programming) are found in the Catapult C tools from Mentor Graphics, and in the Impulse C tools from Impulse Accelerated Technologies. Languages such as SystemVerilog [15] seek to accomplish the same goal, but are aimed at making hardware engineers more productive versus making FPGAs more accessible to software engineers.

There are also a number of high level languages targeting embedded systems. For example, StreamIt [16] is a programming language for high-performance streaming applications. Annapolis Micro Systems, Inc.'s CoreFire Design Suite and National Instruments LabVIEW FPGA provide a graphical dataflow approach to high-level design entry. Our work in this paper is related to such efforts in being tightly coupled with CAL, which is a language oriented towards design and implementation of embedded systems from a high level of abstraction. In addition to the coupling with CAL, another distinguishing aspect of our work is its focus on the domain of video processing, and in particular, reconfigurable video coding.

2.3. XML format

The extensible markup language, widely known as XML, is a markup language that was created by the World Wide Web Consortium (W3C) to overcome limitations of HTML. Like HTML, XML is based on SGML — the Standard Generalized Markup Language. Although SGML has been used in the publishing industry for decades, its perceived complexity intimidated many people that otherwise might have used it. XML was designed with the Web in mind.

A major advantage of XML is that one can encode document information more precisely compared to HTML. This means that programs processing these documents can "understand" them much better and therefore process the information in ways that are not possible for ordinary text processors.

One major application of XML is to make web pages with decent layout that are universally accessible, regardless of browser type. XML also lets one check whether or not optional features are present, and allows for invocation of alternative code to take care of cases where such features are missing.

XML is a promising candidate for carrying data associated with high level text based languages for subsequent use. XML itself is designed to be self-descriptive, which ensures that all of the information from the original file can be understood by other applications. XML tags are not predefined by users. It can be convenient for users to design appropriate tags to describe the context of the information being exchanged.

Representing different languages using a common XML format allows for integrated use of heterogeneous languages within a design flow, thereby allowing designers to combine the unique strengths and features associated with different languages. In our work, as shown in Figure 1, we use CAL to design the targeted system, DIF to optimize the system, and Cal2C as a back-end implementation process. The interfaces in our design flow between CAL and DIF, and between DIF and Cal2C, are based on CALML (an XML-based format associated with CAL), and DIFML, respectively.

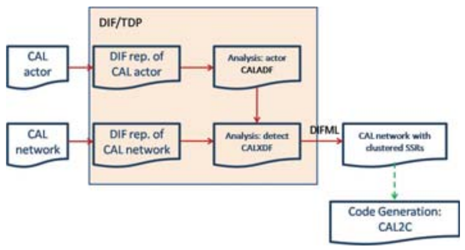

Fig. 1. Automation of efficient video processing system generation.

3. AUTOMATED APPROACH

Our proposed design-to-implementation process is illustrated in Figure 2. Here, CAL is used to describe and model the functionality of the targeted system. DIF and TDP are then applied for analysis and exploration of optimization alternatives. Different optimization techniques target different performance measures, such as real time constraints, power consumption, or buffer size. In this paper, we focus on optimizing the execution speed of the targeted RVC systems. The CAL actors are manually generated in a designer friendly manner. The procedure to transform the system of CAL actor into C code is designed to be automated and we are currently working on the automation. CAL actors can be re-used to build other video processing systems, which is one motivation for the RVC library.

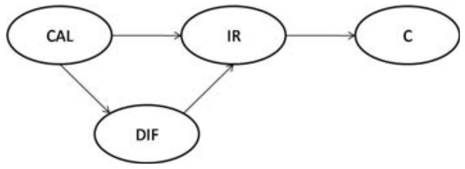

Fig. 2. Automated design-to-implementation flow.

The Open RVC-CAL compiler (Orcc) [17] is a tool set under the BSD license to realize an automated design-to-implementation flow for the RVC-CAL dataflow programming language. It has been developed with a back-end that performs CAL-to-C transformation. Transformation to other lower level languages, such as Java, is under development. Source code and test cases can be found on [17]. We have developed XML-based interfaces between different languages and the SSR detection algorithm to provide paths for integration.

The input to the Orcc is an application that is in terms of CAL actors and a CAL network. CAL actors are represented in the form of .cal files, and the CAL network is specified as .xdf file. The output is an automatically generated implementation, which is targeted to a lower level language, such as C, C++ or Java.

The compiler is divided into two phases — a front end and a back end. The front end is responsible for parsing actors and networks, flattening the hierarchical network, and generating actors in the JASON format. The back end is responsible for generating an implementation in a user-specified lower level language.

3.1. CAL

CAL is a dataflow- and actor-oriented language that represents algorithms in terms of networks of communicating dataflow-actor components [6]. A CAL actor is a modular component that encapsulates its own state. The state of an actor cannot be shared with other actors, and thus, an actor cannot modify the state of another actor.

The behavior of an actor is defined in terms of a set of *actions*. The operations an action can perform are consumption (reading) of input tokens, modification of internal state, and production (writing) of output tokens. The topology of the connections among actor input and output ports constitutes what is called a *CAL network*. Compared to the complexity of actors, edges — connections between pairs of actors — are rather simple. The only interaction an actor can have with another actor is through input and output ports that connect the actors. Such connections are represented as edges in a CAL network.

Each action of an actor defines the kinds of transitions that internal states can undergo, and the specific conditions under which the action can be executed (*fired*). The conditions for firing actions in general involve (1) the availability of input tokens, (2) values of input tokens, (3) state of the actor, and (4) priority of the action. In an actor, actions are executed sequentially — i.e., at most one action can be executing at any given time.

CAL is supported by a portable interpreter infrastructure that can simulate a hierarchical network of actors. In addition to the strong encapsulation afforded by the actor description, the dataflow model also makes much more algorithmic parallelism explicit. This allows application of the wide range of dataflow graph transformations to the realization of signal processing systems on a variety of platforms. In particular, platforms will differ in their degree of parallelism, which gives rise to the challenging problem of matching the concurrency of the application representation with the parallelism of the computing machine that is executing it. The newly developed MPEG video coding standard, Reconfigurable Video Coding (RVC) [2], uses the CAL actor language [6] for specifying functional components, and dataflow as the composition formalism [18]. Building a library of CAL actors for RVC systems can reduce the design time, since designers can take or lightly modify available actors to construct a new system instead of starting from scratch.

3.2. DIF

The dataflow interchange format (DIF) is proposed as a standard approach for specifying and integrating arbitrary dataflow-oriented semantics for DSP system design [8]. The DIF language (TDL) is an accompanying textual design language for high-level specification of signal-processing-oriented dataflow graphs. The TDL syntax for dataflow graph specification is designed based on dataflow theory and is independent of any design tool. For a DSP application, the dataflow semantic specification is unique in TDL regardless of the design tool used to originally enter the specification. The TDL grammar and the associated parser framework are developed using a Java-based compiler-compiler called SableCC [19].

TDL is designed as a standard approach for specifying DSP-oriented dataflow graphs. TDL provides a unique set of semantic features to specify graph topologies, hierarchical design structures, dataflow-related design properties, and actor-specific information. Because dataflow-oriented design tools in the signal processing domain are fundamentally based on actor-oriented design, TDL provides a syntax to specify tool-specific actor information, which ensures that all relevant information can be extracted from a given design tool. The DIF Package (TDP) is a software tool that accompanies TDL, and provides a variety of intermediate representations, analysis techniques, and graph transformations that are useful for working with dataflow graphs that have been captured by TDL.

3.3. Intermediate Representation

The Intermediate Representation (IR) used in Orcc is managed in the form of .jason files. The top-level structure in the Intermediate Representation is an actor. An actor contains parameters, input/output ports, state variables, a list of functions/procedures, a list of actions and an action scheduler

A variable is represented by the Variable class. A Variable has a location, which is the place in the source file where it was declared, a type, a name, and the list of its uses. The list of uses (called 'def-use') is automatically computed and maintained by Orcc. A Variable also has two attributes that may be used depending on the context: a Variable may have an initial expression, with the exception of local variables, and a Variable may have a value, which is its runtime value. The value of a Variable is only used when an actor is interpreted. A GlobalVariable is a Variable whose initial expression may be evaluated as a constant, and accessed with the getConstantValue method. A StateVariable is a GlobalVariable that has an additional "assignable" attribute. This attribute records the information about whether a variable can be assigned or not. A LocalVariable is a Variable that has an "assignable" attribute (like a StateVariable), an SSA (static single assignment) index, and an "instruction" attribute. The "instruction" attribute references the assign instruction where the variable is assigned for the first and only time.

A procedure has parameters and local variables. It has a body made of a list of CFG nodes. A CFG node corresponds to a node in the Control Flow Graph, and is defined by the interface. There are three types of nodes: a BlockNode, an IfNode, and a WhileNode.

Scheduling information (priorities and FSM) are present in the action scheduler. Actions are sorted by descending priority, so the action with the highest priority comes first.

3.4. Integrating Results of DIF Analysis into the C Back End

In our targeted design flow, the analysis of CAL networks and CAL actors is conducted in the DIF environment, as shown in Figure 1. In our current implementation, we detect statically schedulable regions (SSRs) from the DIF-based analysis to optimize scheduling structures for efficient implementation. Currently the input to this form of DIF analysis is a CAL network along with its constituent CAL actors. The output is a set of SSRs, and static schedules corresponding to those SSRs. This SSR and schedule information is generated for efficient system implementation.

The back end of the code generator adopts a round-robin scheduling approach. Round-robin (RR) is a simple scheduling algorithm for executing multiple tasks in an operating system. In the form of RR scheduling that we apply, time slices are assigned to each task in equal portions and in circular order, and no priority ordering is considered across the tasks. Round-robin scheduling is simple, easy to implement, and starvation-free. In the generated system, there is a main scheduler that takes care of all actor schedulers. The main scheduler passes the right of execution to the actor schedulers one by one. When an actor scheduler is selected for execution, and dataflow requirements for one or more actions within the actor are satisfied, the actor scheduler will execute an appropriate action. Then the right of execution is passed to the next actor scheduler.

Static scheduling can be integrated into the RR scheduler in the following way. If some actors can be statically scheduled, that is, the execution of some actors is determined to be continuous and fixed in compile time, then we can combine the schedulers of these actors into one scheduler. For example, suppose row and $transpose$ are actors and $row\_scheduler$ and $transpose\_scheduler$ are corresponding schedulers. Based on SSR detection in TDP analysis, we can determine the scheduling of these two instances as always following the pattern shown in Figure 3. Thus, we can reduce the number of schedulers into one.

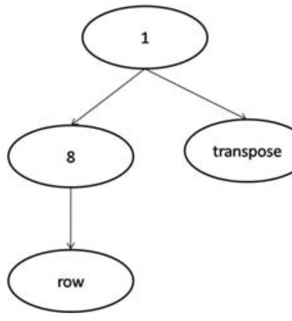

Fig. 3. Static scheduling: actors row and $transpose$

A number of related efforts are underway to develop efficient scheduling techniques for CAL networks. The approach of Platen and Eker [20] sketches a method to classify CAL actors into different dataflow classes for efficient scheduling. Boutellier et al. [21] propose an approach to quasi-static multiprocessor scheduling of CAL-based RVC applications. The approach involves the dynamic selection and execution of "piecewise static schedules" based on novel extensions of flow shop scheduling techniques.

Many previous research efforts have focused on task mapping for multiprocessor systems from other kinds of specification models or languages (e.g., see [3]). For example, Li et al. [22] provide a method for allocating and scheduling tasks using a hybrid combination of a genetic algorithm and ant colony optimization. The approach involves consideration of both global and local memory spaces across the targeted multiprocessor system. Ennals et al. [23] develop a method for partitioning tasks on multi-core network processors.

Compared to prior work on dataflow techniques and multiprocessor system design, major unique aspects of our approach for scheduling are the capability to decompose CAL actors based on their formal action- and port-based semantics, and to construct and subsequently transform SSRs and SSR actors from these decomposed representations.

When integrating SSRs into real implementations, we distinguish between two kinds of SSRs, as shown in Figure 4. In the first type, all CAL actors inside the SSR are preserved from their original structures in the corresponding CAL network, such as $SSR1$ in Figure 4. In the second kind of SSR, there is at least one partial CAL actor, of which some ports do not belong to the SSR, such as $SSR2$. When implementing SSRs of the second type, we divide each partial actor into two separate actors, as shown in Figure 5. In Figure 5, actor C is split into two new actors: C_1 and C_2. C_1 is statically scheduled in SSR_2, and C_2 has its own dynamic scheduler. Currently, implementation of the first kind of SSR is complete, and integration of the second kind of SSR is under development.

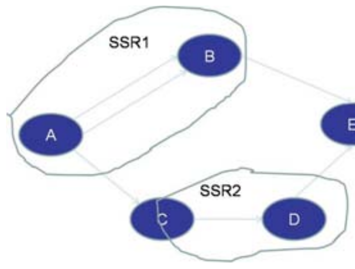

Fig. 4. Two kinds of statically schedulable regions.

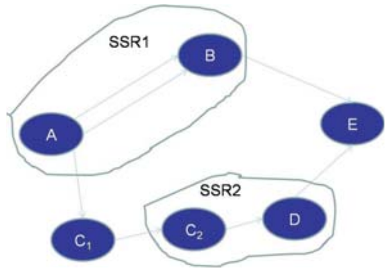

Fig. 5. SSR: splitting one CAL actor into two actors.

Figure 6 shows three kinds of options to integrate SSRs into Orcc. Option 3 is to modify the generated C code by integrating SSRs. Option 2 is to introduce SSRs into intermediate representations, that is, into generated intermediate code that is based on .jason files and .difml files. Option 1 is to introduce SSRs in the front end where the CAL network is parsed into JASON files. Option 3 is generally the simplest to implement, while option 1 has the potential to produce more efficient implementations since the structure of SSRs can be exploited more rigorously in scheduling and related dataflow transformations.

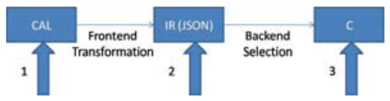

Fig. 6. Code generation procedure.

We have implemented option 3 as an initial prototype of SSR integration. In our ongoing and future work, we are exploring implementations of options 2 and 3.

4. THE DIFML FORMAT

As described previously, the dataflow interchange format (DIF) is proposed as a standard approach for specifying and integrating arbitrary dataflow-based semantics for DSP system design [8], and The DIF language (TDL) is an accompanying textual design language for high-level specification of signal-processing-oriented dataflow graphs.

In order to describe DIFML, we introduce a number of concepts associated with the general XML format: node, element, attribute and tag. A node is a part of the hierarchical structure that makes up an XML document. "Node" is a generic term that applies to any type of XML document object, including elements, attributes, comments, processing instructions, and plain text. A tag is a markup construct that begins with < and ends with >. An element is a logical component of a document. The element's content may contain markup, including other elements, which are called "child elements". An attribute is a markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag.

DIFML is designed as an XML-based format for exchanging information between TDL and other tools and languages, and more generally, between arbitrary pairs of dataflow environments. There are different elements in DIFML and these elements are listed in a hierarchical way. The element at the highest level is *graph*, while *topology* and *interface* are lower level element. Under *topology*, there are three elements at the same level: *nodes*, *edges* and *interface*. For each element, there are three kinds of attributes: *implicitAttributes*, *builtInAttributes* and *userDefinedAttributes*. ImplicitAttributes are those attributes necessary and inherent to the element, such as the id of a node. BuiltInAttributes are attributes that are recognized as part of the DIF language, typically through corresponding reserved words or other kinds of language constructs. For example, for an edge element in and SDF model within a DIF graph (i.e., within a graph that is defined with the sdf keyword), there are three kinds of builtInAttributes: the production rate, consumption rate, and delay. UserDefinedAttributes are attributes that users add to selected elements at their own discretion. The following is a simple example of an SDF model in the DIFML format. For conciseness, we just show part of the associated DIFML file.

```
<?xml version='1.0' encoding='UTF-8'?>
<difml xmlns='http://www.ece.umd.edu/DIFML'>
    <graph>
        <implicitAttributes>
            <name val='dat2cd'/>
            <type val='SDFGraph'/>
        </implicitAttributes>
        <topology>
            <nodes>
                <node>
                    <implicitAttributes>
                        <id val='A'/>
                    </implicitAttributes>
                    <builtInAttributes>
                        <nodeWeight type='DIFNodeWeight'↩
                            />
                    </builtInAttributes>
                </node>
            </nodes>
        </topology>
        <interface>
            <port>
                <implicitAttributes>
                    <direction id='InA' nodeId='A' val='↩
                        IN'/>
                </implicitAttributes>
            </port>
        </interface>
    </graph>
    <!—Automatically generated from DIF file—>
</difml>
```

As shown in the above example, DIFML follows the same format as XML files, and defines a series of elements and attributes. Note that there is an element named *node*. This name is in correspondence with the related definition in the DIF language, and has different a meaning from the node concept in XML terminology, which is a generic concept that applies to any type of XML document object.

Currently, the DIFML parser supports several major dataflow models that are recognized in the DIF language, including SDF [4], cyclo-static dataflow (CFDF) [24], parameterized synchronous dataflow dataflow (PSDF) [25], CAL dataflow (CALDF) [6], and multidimensional synchronous dataflow (MDSDF) [26].

5. EXPERIMENTAL RESULTS

We apply our automated design-to-implementation flow to an RVC MPEG4 SP decoder. We generate three kinds of code using Orcc tools:

1. C code with "C" as back end;
2. C code with "C+SSR" (C code generation integrated with derived SSRs) as back end;
3. C code with "C+modified_SSR" — based on the derived SSRs, we manually compute token production rate and consumption rate information to enhance static scheduling.

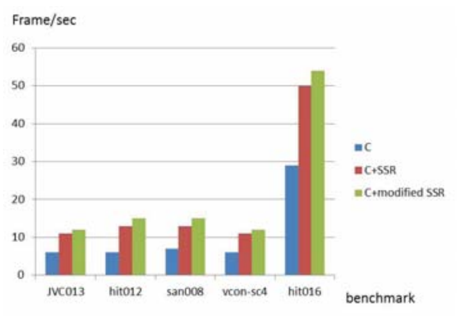

Fig. 7. Experimental results for MPEG4 RVC SP decoder.

We generate three kinds of C projects using CMake. The projects are compiled and built using Microsoft Visual C++ 2008. The generated executables are executed on a Sony VAIO laptop with an Intel Pentium 1.2GHz processor. The experimental results are shown in Figure 7.

The results show an improvement of approximately a factor of two in performance after we integrate SSR derivation and scheduling compared with direct C-based implementation from the CAL system. If we modify the CAL actors based on results of SSR derivation, the performance is even better. Currently, the modification based on SSRs is performed by hand. Automating this part of the optimization process is an interesting direction for future work.

The results show a significantly higher frame rate on benchmark 5 of hit016. This is because for the first four benchmarks, the display sequence is set to 352x288, while for the fifth benchmark the display sequence is 176x144. The smaller display size consumes less resources and runs faster.

6. CONCLUSION

This paper proposes an automated design flow from user-friendly design to efficient implementation of reconfigurable video coding systems. We have developed tools and techniques to support both designer productivity and implementation efficiency by strategically combining complementary dataflow languages and tools.

Our approach integrates previously developed techniques for detecting SDF-like regions within larger CAL networks, and exploiting the static structure of such regions using analysis tools in The Dataflow interchange format Package (TDP). This integration is achieved using a new XML format, called DIFML, which we have developed to transfer information across different dataflow environments. Experimental results demonstrate significant performance improvements on an MPEG Reconfigurable Video Coding (RVC) decoder.

7. REFERENCES

[1] T. Chen and Y. K. Chen, "Challenges and opportunities of obtaining performance from multi-core CPUs and many-core GPUs," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 2009.

[2] S. S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet, "Overview of the MPEG reconfigurable video coding framework," *Journal of Signal Processing Systems*, June 2009. [Online]. Available: http://dx.doi.org/10.1007/s11265-009-0399-3

[3] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 2nd ed. CRC Press, 2009.

[4] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.

[5] S. S. Bhattacharyya, R. Leupers, and P. Marwedel, "Software synthesis and code generation for DSP," *IEEE Transactions on Circuits and Systems — II: Analog and Digital Signal Processing*, vol. 47, no. 9, pp. 849–875, September 2000.

[6] J. Eker and J. W. Janneck, "CAL language report, language version 1.0 — document edition 1," Electronics Research Laboratory, University of California at Berkeley, Tech. Rep. UCB/ERL M03/48, December 2003.

[7] B. Kienhuis and E. F. Deprettere, "Modeling stream-based applications using the SBF model of computation," *Journal of Signal Processing Systems*, vol. 34, no. 3, 2003.

[8] C. Hsu, M. Ko, and S. S. Bhattacharyya, "Software synthesis from the dataflow interchange format," in *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*, Dallas, Texas, September 2005, pp. 37–49.

[9] R. Gu, J. Janneck, M. Raulet, and S. S. Bhattacharyya, "Exploiting statically schedulable regions in dataflow programs," *Journal of Signal Processing Systems*, January 2010. [Online]. Available: http://www.springerlink.com/content/7828n20m31186635/

[10] M. Wipliez, G. Roquier, M. Raulet, J. Nezan, and O. Deforges, "Code generation for the MPEG reconfigurable video coding framework: From CAL actions to C functions," in *Proceedings Multimedia and Expo, IEEE International Conference*, June 2008, pp. 1049–1052.

[11] R. Gu, J. W. Janneck, S. S. Bhattacharyya, M. Raulet, M. Wipliez, and W. Plishker, "Exploring the concurrency of an MPEG RVC decoder based on dataflow program analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, 2009.

[12] *MPEG video technologies – Part 4: Video tool library*, ISO/IEC FDIS 23002-4, 2009.

[13] *MPEG systems technologies – Part 4: Codec Configuration Representation*, ISO/IEC FDIS 23001-4, 2009.

[14] Z. Navabi, *Verilog Digital System Design: Register Transfer Level Synthesis, Testbench, and Verification*. McGraw Hill, 2006.

[15] S. Sutherland, S. Davidmann, and P. Flake, *SystemVerilog for Design: A Guide to Using SystemVerilog for Hardware Design and Modelingn*, 2nd ed. Springer, 2006.

[16] M. I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," in *Proceedings of the International Workshop on Rapid System Prototyping*, San Jose, California, USA, 2006, pp. 151–162.

[17] "Open RVC CAL compiler." [Online]. Available: http://sourceforge.net/apps/trac/orcc/

[18] M. Mattavelli, I. Amer, and M. Raulet, "The reconfigurable video coding standard," *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 159–167, May 2010.

[19] E. M. Gagnon and L. J. Hendren, "SableCC, an object-oriented compiler framework," in *Proceedings of TOOLS (26)*, 1998, pp. 140–154.

[20] C. v. Platen and J. Eker, "Efficient realization of a CAL video decoder on a mobile terminal," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, October 2008.

[21] J. Boutellier, V. Sadhanala, C. Lucarz, P. Brisk, and M. Mattavelli, "Scheduling of dataflow models within the reconfigurable video coding framework," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, October 2008.

[22] M. Li, H. Wang, and P. Li, "Tasks mapping in multi-core based system: hybrid ACO&GA approach," in *Proceedings of the International Conference on ASIC*, October 2003.

[23] R. Ennals, R. Sharp, and A. Mycroft, "Task partitioning for multi-core network processors," in *Proceedings of the International Conference on Compiler Construction*, April 2005.

[24] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, February 1996.

[25] B. Bhattacharya and S. S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2408–2421, October 2001.

[26] P. K. Murthy and E. A. Lee, "Multidimensional synchronous dataflow," *IEEE Transactions on Signal Processing*, vol. 50, no. 8, pp. 2064–2079, August 2002.