H.26L Video Server Modeling Using Computational Process Networks

Serene Banerjee
Dept. of ECE, The University of Texas, Austin, TX 78712-1084 USA
{serene}@ece.utexas.edu

Embedded Software Systems, Final Project Report
May 8th, 2002

Abstract

Recent applications such as video conferencing systems, Digital Versatile Disc (DVD) systems, digital cable television, and video streaming, built on years of digital video research. State-of-the-art video coding standards such as H.263 (1995), MPEG-2 (1996), and MPEG-4 (2001) have a common goal of achieving higher compressed video quality for lower bit-rates. The International Video Coding Experts' Group (VCEG) is standardizing the next-generation video-coding standard, H.26L. This will reduce the bit rate by about 50%, at the expense of a roughly four-fold increase in implementation complexity.

Computational Process Networks (CPN) is a scalable framework for real-time data-intensive systems' implementation on commodity multiprocessor workstations. This is an extension of the process networks model, which captures functional parallelism, guarantees determinate execution, and runs in bounded memory if a bounded memory implementation exists. Typically for video encoders bounded memory implementations exist.

As CPN can model the inherent parallelism in the H.26L video encoder and preserve functional precedence, a faster execution of the video encoder is possible. The contributions of this paper are:

- *Model a version of the H.26L video encoder on the CPN framework*
- *Implement the proposed model and test it on* 176×144 *QCIF resolution frames*
- *Explore applications where this implementation would be useful.*

Index terms: *H.26L video encoder, computational process networks, system modeling, functional and data level parallelism, computational complexity, dataflow models*

1 Introduction

The International Telecommunication Union (ITU) is proposing the new video coding standard, H.26L, which is aimed at providing enhanced compression performance at a very low bit rate for real-time, low end-to-end delay applications [1–8]. The model provides bit rate savings of about 50% when compared to an H.263+ codec that has been optimized for visual quality for sequences at low bit-rates. Also, improved network adaptation layers (NALs) are being developed so that the coded video data can be transported in existing and future networks, such as circuit-switched wired networks, Internet Protocol (IP) networks with real–time transport protocol (RTP) packetization, and third-generation wireless systems. But this increase in bit rate vs. quality trade-off comes at the cost of a four–fold increase in implementation complexity when compared to the fully optimized H.263+ codec [8].

In this project the inherent functional parallelism of the H.26L video encoder is exploited and Computational Process Networks (CPN) is used to implement a system level modeling of the encoder, for faster performance. Allen and Evans [9] developed CPN and use their C++ implementation of CPN to implement a real-time beamformer on multiple workstations. He and Zhong [10] use CPN [9] to implement MPEG-4 encoder on a similar model. He, Ahmad and Liou [11, 12] implement a MPEG-4 codec on a group of workstations using hierarchical Petrinets.

On a single processor machine, for encoding three 176×144 QCIF resolution frames of the *Foreman* video sequence the time taken for both the sequential and parallel implementation of the encoders are same. However, this design is scalable to a multi processor workstation, and CPN can then extract more parallelism, for faster encoding. Also, if more number of frames were encoded, parallel processing would reduce encoding time, compared to a sequential implementation.

In this paper, Section 2 presents a formal model of the H.26L video encoder as a homogeneous Synchronous Dataflow model, and discusses how it can be extended to a CPN model. Section 3 provides a description of the software implementation using C/C++/Pthreads, based on the model. The results and a comparison of the original sequential version of the H.26L video encoder with the concurrent implementation are presented in Section 4. Possible applications where concurrent video encoding using POSIX threads would be useful are discussed in Section 5. Finally, Section 6 concludes the paper and provides directions for future work.

2 Modeling

The H.26L video-coding standard separates the video encoder into two separate layers: a *video coding layer* that is responsible for efficiently representing the video content, and a *network adaptation layer* that packs the coded data in an appropriate manner based on the network over which it is being transmitted. The focus of this work is on the video coding layer.

Figure 1 shows a generic block diagram of a H.26L video compression codec, where the encoded bitstream reduces the spatial and temporal statistical redundancy in video sequences. Modified versions of this block diagram also applies to other video codecs such as H.263, MPEG-2, MPEG-4 simple profile video encoders. Like the present video coding standards, H.26L also has inherent functional and data level parallelism, and hence can be modeled using formal system-level computational models.

In order to model the computation and communication in the video encoder, the statically schedulable Synchronous Dataflow (SDF) model, or dynamically schedulable Boolean Dataflow (BDF) or Dynamic Dataflow (DDF) may be used to guarantee determinacy and correctness [13].

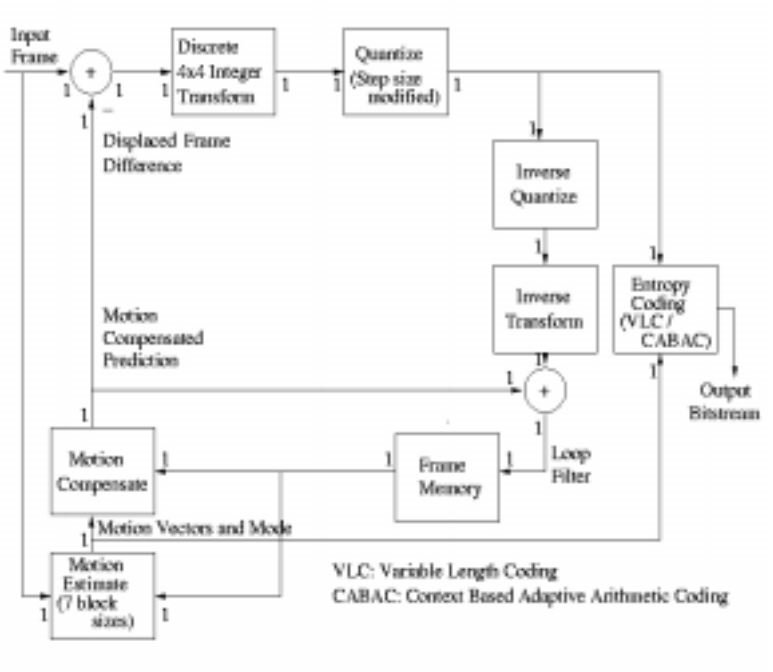

Figure 1. Generalized block diagram of a H.26L video encoder having block-based motion compensation and transform coding. This can be mapped to an homogeneous Synchronous Dataflow (SDF) model with one token on each arc.

Kahn's process network is a superset of dataflow models and is a computational model where concurrent processes are connected by unidirectional first-in-first-out (FIFO) queues. When a block reads values on an input port, the block's execution will halt until enough data is available on that input port. If enough data is available, then the block consumes (dequeues) the data on the port. The results obtained from a process network are determinate, irrespective of the execution order in the model. However, determining whether the process network can be scheduled or not in bounded memory cannot be predicted in a finite amount of time.

Parks [13] extended Kahn's process networks by setting a finite capacity on each arc (queue) and developed a scheduling policy that will yield bounded execution if one exists. When a block tries to write to a full queue, *artificial deadlock* is introduced, which can be corrected by increasing the size of the full queue. Allen and Evans [9] added firing thresholds to the queues to

form computational process networks (CPN). Their model combining process networks, bounded scheduling for realization in finite memory, and firing thresholds from computation graphs [14], avoids running into artificial deadlocks. Their implementation of CPN using C++ and portable operating system interface (POSIX) threads is intended for computationally intensive algorithms on large symmetric multiple workstations. However, with context switching between the nodes and amount of load on each node, tradeoffs between overhead, latency and parallelism will exist. Allen and Evans' [9] computational process networks model is a scalable framework that could be mapped onto a multiprocessor workstation.

In this project computational process networks is used to model the data flow and control flow of the H.26L video encoder, to provide real-time encoding, determinacy, correctness, completed and bounded execution. Each functional block is modeled as a CPN node and implemented as a Pthread class. Each arc is represented by a FIFO queue, and fork nodes are implemented so that inputs are copied to multiple outputs. In a similar work He and Zhong [12] model a MPEG-4 simple visual profile video encoder using CPN.

3 Implementation

This work parallelizes a sequential version of the H.26L video encoder available at:

ftp://standards.pictel.com/video-site/h26L/

The source code is in ANSI C and supports CIF (352×288) and QCIF (176×144)resolution image sequences. The source code is targeted for demonstration purposes, and encodes three frames of the QCIF resolution *Foreman* input sequence in 21.6 seconds.

This version of the H.26L video encoder was implemented using CPN, based on the Pthread library and Allen and Evans' [9] C++ implementation of CPN framework available at

http://www.ece.utexas.edu/~allen/PNSourceCode/

This implementation of CPN is scaled by the operating system according to the available number of processors up to the amount of functional parallelism exposed by the PN graph. The classes, *CPNNode* and *CPNQueue* were used to implement the nodes and queues respectively.

The implementation presented here followed the works of He and Zhong's [12] implementation of an MPEG-4 video encoder on CPN. The basic design steps involved are:

- **System modeling:** The sequential code was modeled as self-contained blocks as the Figure 1. Each of these blocks were specified to be one CPN node, that inputs one token, processes it, and outputs one token, after processing on it. Each of these nodes were inherited from the Pthread class *CPNNode*. Adjacent nodes communication via first-in-first-out (FIFO) queues. The queue class is inherited from the Pthread class *CPNQueue*

- **Token specification:** Each frame of the input video sequence is described as one token. Within the frame the block-based computations are done sequentially in this implementation. However, more parallelism could have been extracted, if block based computations could be done parallely, as well.

- **Precedence preservation:** The designed queues with the firing thresholds and the nodes preserve functional precedence, as the nodes do not fire, until there is an output token from the previous node.

(a) (b)

Figure 2. Original (a) and H.26L encoded (b) frame 2 for the 176 × 144 QCIF resolution *Foreman* input video sequence, with 25 : 1 compression

4 Results

The CPN implementation of the H.26L video encoder functions correctly, and the compressed bitstream is successfully decoded by the H.26L video decoder. The input sequence contains three frames of the QCIF resolution (176 × 144) *Foreman* video sequence. The first frame is coded as an *intra* or I-frame, where each block in the frame is coded as it is, and hence the spatial and temporal redundancies are not exploited. The second one is coded as a *bidirectionally predicted* or B-frame, where each block in the frame is forward and backward predicted from the previous and next frame respectively, and the difference is encoded. The last frame is coded as a *predicted* or P-frame, where each block in the frame is forward predicted from the two previous frames.

Figure 2 presents the original and H.26L encoded second frame. The compression achieved for the three frames is 25:1, and the encoding time is 21.6 seconds, which is comparable to the sequential implementation. Though the parallel implementation does not reduce encoding time, for three frames on a single processor, it is expected to perform better on a multiprocessor

platform, where independent tasks can be distributed to different processors. Also, increasing the number of frames to be encoded, will also enable CPN to take advantage of more parallelism.

5 Applications

Video encoding and processing on POSIX threads, exploiting parallelism, will benefit innumerable on-line real-time constraint applications, such as, video transcoding over the Internet, video conferencing, and video streaming applications. Zhou, Vellaikal, Shen and Kuo [15] suggest an on-line scene change detection algorithm in multicast video, which is realizable using POSIX threads. The buffering and processing of frames are modeled on separate processors for scene change detection in their work. For video conference and streaming, after segmenting the video sequence, each segment can be processed on a different processor, to meet real-time deadlines.

6 Conclusions and Future Work

Allen and Evans' [9] CPN framework provides a scalable platform for building computationally intensive signal processing applications on commodity workstations. This work uses their libraries to build a parallel implementation of a H.26L video encoder. For encoding 3 video frames, the concurrent execution of CPN nodes, and the original sequential program takes the same time on a single processor workstation, due to inter dependencies of the nodes. More speedup would be obtained by experimenting on a multiprocessor platform, or by exploiting the parallelism existing in each node. Although the current implementation is far from meeting real-time deadlines, the key insight of this work is that the CPN implementation of the H.26L video encoder can be scaled to a multiprocessor platform for faster execution.

References

[1] G. Bjontegaard, "H.26L Test Model Long Term No. 1 (TML-1) Draft 2," Tech. Rep. 1, ITU-T Video Coding Experts Group, Aug. 1999. *ftp://standard.pictel.com/video-site/9908_Ber/q15h36d2.doc.*

[2] A. P. Joch, "H.26L: Analysis and Real-Time Encoding Algorithms," Master's thesis, Dept. of Electrical and Comp. Engg., The University of British Columbia, Jan. 2002.

[3] K. Dovstam, "Video Coding in H.26L," Master's thesis, Dept. of Microelectronics and Information Tech., Royal Institute of Technology, Apr. 2000. *http://www.it.kth.se/docs/Reports/DEGREE-PROJECT-REPORTS/000601-Kristofer-Dovstam.pdf.*

[4] M. Flierl, T. Wiegand, and B. Girod, "Multihypothesis Pictures for H.26L," in *Proc. IEEE Int. Conf. on Image Proc.*, vol. 3, pp. 526–529, Oct. 2001.

[5] A. Hallapuro, V. Lappalainen, and T. D. Hamalainen, "Performance Analysis of Low Bit Rate H.26L Video Encoder," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Proc.*, vol. 2, pp. 1129–1132, May 2001.

[6] V. Lappalainen, A. Hallapuro, and T. D. Hamalainen, "Optimization of Emerging H.26L Video Encoder," in *Proc. IEEE Workshop on Signal Proc. Systems*, vol. 1, pp. 406–415, Sept. 2001.

[7] G. J. Sullivan, T. Wiegand, and T. Stockhammer, "Using the Draft H.26L Video Coding Standard for Mobile Applications," in *Proc. IEEE Int. Conf. on Image Proc.*, vol. 3, pp. 573–576, Oct. 2001.

[8] S. Wenger, "A High Level Syntax for H.26L: First Results," in *Proc. IEEE/SPIE Int. Conf. on Visual Comm. and Image Proc.*, vol. 4067, pp. 1307–1316, June 2000.

[9] G. E. Allen and B. L. Evans, "Real-Time Sonar Beamforming on Workstations Using Process Networks and POSIX Threads," *IEEE Trans. on Signal Proc.*, vol. 48, pp. 921–926, Mar. 2000.

[10] Y. He, I. Ahmad, and M. L. Liou, "Real-time Interactive MPEG-4 System Encoder Using a Cluster of Workstations," *IEEE Trans. on Multimedia*, vol. 1, pp. 217–233, June 1999.

[11] Y. He, I. Ahmad, and M. L. Liou, "A Software-Based MPEG-4 Video Encoder Using Parallel Processing," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 8, pp. 909–920, Nov. 1998.

[12] C. He and S. Zhong, "System Modeling and Software-based Implementation of MPEG-2 Video Encoder," in *Proc. IEEE Asilomar Conf. on Signals, Systems and Comp.*, vol. 2, pp. 1058–1062, Oct. 2000.

[13] T. M. Parks, *Bounded Scheduling of Process Networks.* PhD thesis, Dept. of Electrical Engg. and Comp. Sciences, University of California at Berkeley, Dec. 1995. *http://ptolemy.eecs.berkeley.edu/papers/95/parksThesis/.*

[14] R. M. Karp and R. E. Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queuing," *SIAM Journal*, vol. 14, pp. 1390–1411, Nov. 1966.

[15] W. Zhou, A. Vellaikal, Y. Shen, and C.-C. J. Kuo, "On-line Scene Change Detection of Multicast Video," *Academic Press Journal on Visual Comm. and Image Representation*, vol. 12, pp. 1–16, Mar. 2001.