System Modeling and Software-based Implementation of MPEG-4 Video Encoder

Final Report

For

EE382C Embedded Software Systems

Prof. B. L. Evans

by

Chen He and **Shi Zhong**

May 10, 2000

Abstract

The MPEG-4 standard provides for content-based interactivity, high compression, and/or universal accessibility and portability of audio and video contents. Due to its representation of an audiovisual system in terms of distinct objects (except the simple profile used for wireless video communication) and flexible configuration structure, any MPEG-4 hardware implementation is likely to be application specific. Therefore, software-based implementation that allows flexibility and portability seems to be a natural and viable option.

In this report, we model the MPEG-4 video encoder using Computational Process Networks (CPN), which is a deterministic and concurrent computation model, and implement a scalable software-based encoder in C++ and Portable Operating System Interface (POSIX) threads under the framework proposed by Allen and Evans for constructing high-throughput real-time signal processing applications. The performance of the original sequential implementation and our concurrent implementation of the MPEG-4 simple-profile video encoder are compared on a single-processor workstation. We find that the concurrent implementation is faster since the benefits from concurrent execution of CPN nodes outweigh the overheads created by C++/Pthread programming. Our approach is scalable to multiprocessor environment, which makes the software-based real-time MPEG-4 video encoder on multiple/multiprocessor workstations feasible.

1. Introduction

MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group) and adopted in 1998. The mandate for MPEG-4 was to standardize algorithms for audiovisual coding in multimedia applications so as to support content-based interactivity, high compression, and/or universal accessibility and portability of audio and video contents [1,2].

Due to MPEG-4's content-based representation nature and flexible configuration structure, software-based implementation of an MPEG-4 encoder seems to be a natural and viable option. The main obstacle in such an approach is that it requires a large amount of computing power to support real-time encoding operations. The latest developments in parallel and distributed multi-processor systems, however, promise possible real-time performance for computation-intensive signal processing applications at an affordable cost (e.g. a cluster of workstations). A software-based MPEG-4 encoder implemented by He, Ahmad and Liou [3] and a real-time beamformer implemented on multiple workstations by Allen and Evans [4] are two successful examples.

In particular, Allen and Evans implemented the real-time beamformer on UNIX workstations using Computational Process Networks (CPN) and POSIX threads (Pthreads). CPN is a computation model well suited for modeling computation-intensive real-time DSP applications. POSIX is a recent standard with the goal to provide source-code portability across different platforms. One of its extensions, Pthread, is a standardized model for dividing a program into "lightweight" subtasks, which can be interleaved or run in parallel and thus allows high performance implementation of any CPN model.

In this paper, we first present an overview of the MPEG-4 video encoder, followed by a brief review of the framework proposed by Allen and Evans, on which our software implementation is based. Then we construct a hierarchical CPN model of the core encoder in MPEG-4 simple visual profile. Based on the model, we provide a software implementation using C++/Pthreads and analyze the simulation results. Finally, we present our conclusions.

2. Overview of MPEG-4 Video Encoder

To enable the content-based interactivity, MPEG-4 Video Verification Model [2] introduces the concept of VOP (video object plane). Each frame of an input video sequence is segmented into a number of arbitrary image regions (VOPs), with each of them possibly describing a particular image or video object of interest within scenes. So the video encoder is composed of a number of VOP encoders.

The basic coding structure of a VOP encoder consists of shape coding (for arbitrarily shaped video objects), motion estimation/compensation, and DCT-based texture coding [5]. The latter two compose the core encoder of the MPEG-4 simple visual profile for wireless communication applications. Block-based motion estimation and compensation techniques are employed in MPEG-4 encoder to effectively remove temporal redundancy of the video objects. The intra VOPs as well as the residual errors after motion-compensated prediction are coded using DCT coding on 8 × 8 blocks. Scanning of the DCT coefficients followed by quantization and run-length coding is performed using the techniques and VLC (variable length code) tables defined in the MPEG-1/2 and H.263 standards, as well as the provision for quantization matrices [5]. Alternative techniques such as shape adaptive DCT and wavelet transform may be applied for texture coding.

Other main functionalities supported by MPEG-4 video encoder are spatial and temporal scalability and error robustness at VOP or higher level. Scalability is an important feature when the same video objects are to be made available through channels of different bandwidth or receivers of different processing capability, or to respond to different user requests.

3. Computational Process Networks and POSIX Threads

Formal system-level modeling provides portability and scalability over heterogeneous software environments and guarantees determinacy and correctness.

Process Network (PN) is a concurrent model of computation that is a superset of data flow models. As a directed graph, each arc represents a FIFO queue for communication and each node an independent, concurrent process. CPN is proposed by Allen and Evans [4] as a bounded PN model extended with firing thresholds from Computational Graphs. They presented a real-time sonar beamformer implementation on UNIX workstations using CPN model and POSIX threads. Key ideas include:

1) Conventional implementation of UNIX operating system is not capable of deterministic real-time performance while POSIX extensions provide support for real-time applications on UNIX workstations.
2) Computational Process network serves as the reliable formal design methodology for organizing and developing real-time multiprocessor software. It provides necessary scalability and guarantees determinate and complete execution, and bounded scheduling.
3) By carefully designing/dividing the processing node, the parallelism is exploited, the run-time overhead reduced, and the real-time capability achieved on multiple workstations.

It is natural to apply their framework to the MPEG-4 video encoder implementation that needs both the formal system design and extensive computation power. The key to achieving real-time is to deliberately design the processing nodes.

4. Formal Modeling of the MPEG-4 Video Encoder

Kim and Evans [6] described a generic dataflow of video codec system modeled using homogeneous synchronous dataflow (HSDF), in which each functional block is implemented as a star in Ptolemy environment. Hamosfakidis and Paker[7] discussed the concurrency feature in an MPEG-4 video encoding task, which suggests to us that CPN should be a better choice than HSDF in terms of flexibility and scalability. In He’s MPEG-4 encoder implementation [3], real-time capability depends mainly on highly effective scheduling and partitioning schemes, not on

the modeling part. In contrast, in Allen and Evans' work [4], load balancing and partitioning is integrated into the process network model and scheduling among multiple processors is performed automatically, which provides a unified approach and better scalability.

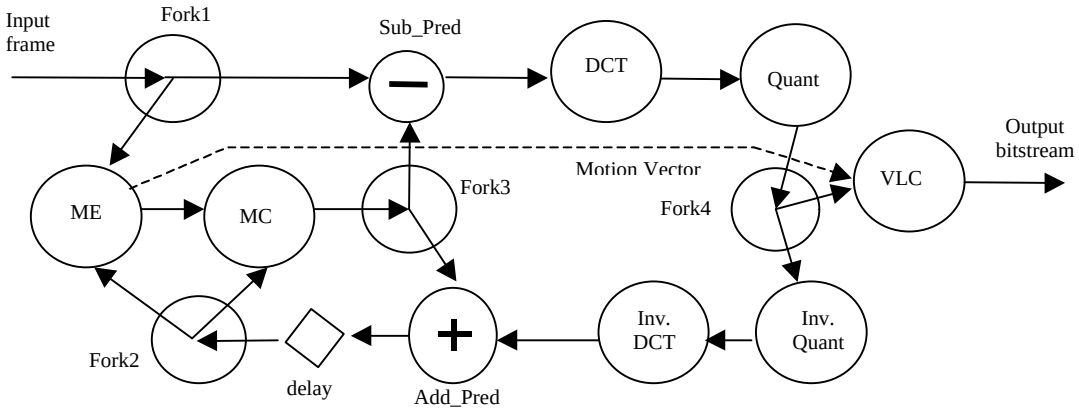

Fig. 1 CPN Model of An MPEG-4 Simple-Profile Video Encoder

Following Allen and Evans' framework, we designed CPN model of the core encoder of the MPEG-4 simple visual profile, as shown in Fig. 1. Each functional block in the encoder is modeled as a CPN node and implemented as a Pthread class. Each arc represents the communication channel between two nodes and is implemented as a FIFO queue with firing threshold. A fork node copies its input to its multiple outputs. Using fork node simplifies all other functional nodes to have only one output. The delay element provides previous reconstructed frame required by motion estimation (ME) and motion compensation (MC) nodes. With an (arbitrary) initial token on the delay arc, the initial execution of ME and MC nodes is enabled and deadlock avoided. Motion vector information is contained in the token and can be used for all nodes. Rate control is embedded in the variable length coding (VLC) node.

Motion estimation is the most computation-intensive part of an MPEG encoder. To balance the load among CPN nodes and achieve real-time performance, we propose a hierarchical model for the ME node as shown in Fig. 2. Obviously, this hierarchical model may apply to any other

nodes in the model if the implementation of the top-level CPN model (Fig. 1) cannot satisfy the real-time requirement.

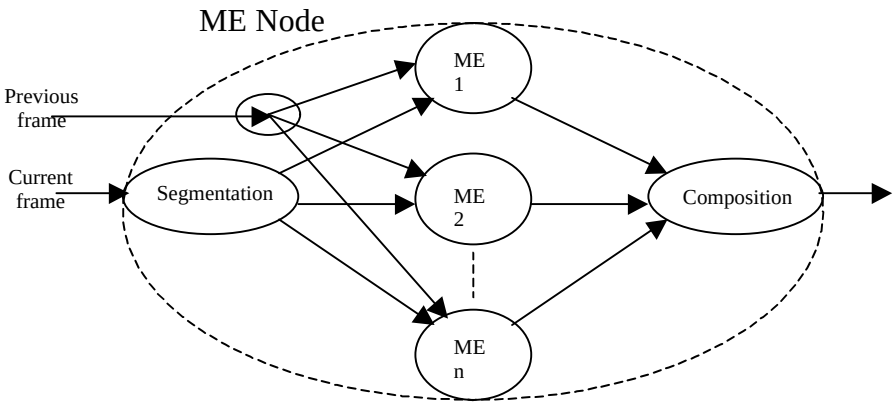

Fig. 2 Hierarchical Model of the Motion Estimation Node: Each ME_i (i=1,…,n) subnode processes part of the current frame and is designed to be able to operate in real time.

5 Software Implementation

Our software implementation is based on Allen's CPN software package [8] (C++ and Pthread) and the free MPEG-2 codec source code (ANSI C) from MPEG Software Simulation Group (MSSG) [9]. Allen's CPN software package provides user an infrastructure for implementing real-time applications that can be modeled by CPN. Two basic classes, *CPNNode* and *CPNQueue*, are provided to implement the nodes and queues, respectively. There are two distinct advantages using his software package: a) determinate concurrent execution guaranteed by the semantics of formal PN model; b) portability and scalability provided by Pthread implementation.

Given the limited time, we only implemented the MPEG-4 simple-profile video encoder, in which the inputs are all rectangular frames. We have not done with refining each CPN node and balancing the computation load among them. Our main accomplishment is a C++/Pthread implementation of the top-level CPN model shown in Fig. 1. Major efforts involved in converting the existing MPEG source code into C++/Pthread implementation lie in:

1) *Data and control flow segmentation.* The original sequential code is partitioned into self-contained blocks following the top-level CPN model. Each block in the block diagram is implemented as one CPN node, which is a Pthread class inherited from *CPNNode*. Neighboring nodes communicate through a FIFO queue, which is a class inherited from *CPNQueue*.
2) *Token design.* Two types of tokens, frame-based and block-based, are used to represent different dataflow among the CPN nodes. Block-based data type is used to facilitate the computation of DCT/IDCT and Quantization/Inverse Quantization nodes. Some macroblock information (e.g. motion vectors) is needed by most of the nodes and thus embedded in both types of tokens.

6. Simulation Results

Our simulation is done on a single Intel Pentium III Xeon (733MHz) processor running Linux. We test our encoder on a test video sequence, record the running times (average over 10 trials) and analyze the performance difference between our concurrent implementation and the original sequential implementation of the encoder. To ensure fair comparison, *jmake* (http://slug.arlut.utexas.edu/~jmake/) is used to compile both versions and there is no other process load on the workstation when we conduct our experiments (the workstation is standalone).

First, we have obtained successful encoding results, which is decodable and playable using existing MPEG-2 player. This verifies the correct concurrent execution guaranteed by CPN model.

Secondly, we observed that our concurrent implementation is on average 30% faster than the original sequential implementation as shown in Fig. 3. This result is amazing because we run both versions on a single processor and the C++/Pthread implementation should create certain

amount of run-time overhead. We think the reason is that the benefits from concurrent execution of different parts of the encoder and possible overlap between CPU operations and memory I/O operations outweigh the run-time overhead created. Speedup variations with respect to number of frames reflect measurement error, initial setup time, etc.

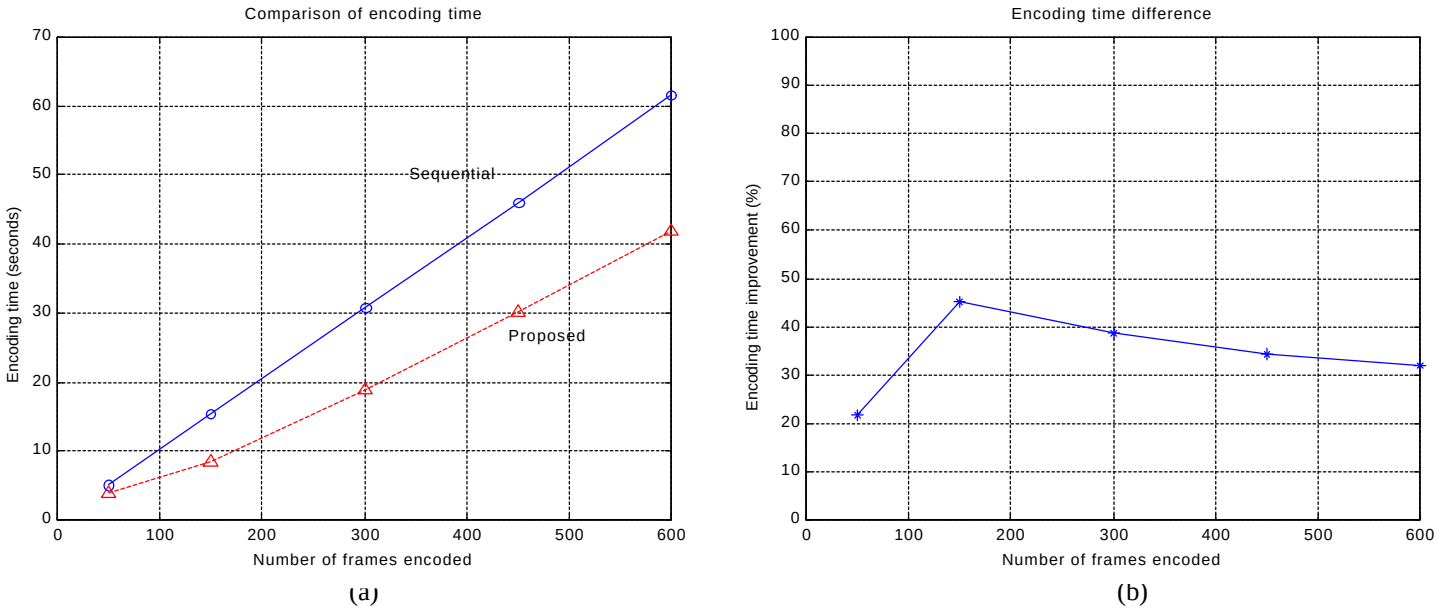

Fig. 3 (a) Encoding time comparison. The sequential encoder is the original C implementation and the proposed is our concurrent implementation. The test video sequence is of frame size 128×128 and color format 4:2:0 (YUV). (b) Encoding time improvement percentage of our implementation over the original implementation.

Finally, we want to emphasize that on a single processor, the speedup resulted from concurrent execution is not related to the size of FIFO queue and depends just on the inherent parallelism of the designed PN model. In our experiments, different queue sizes (1~24) have been tried and shown no effect on the execution time. However, we believe that the queue size will have an impact on multiprocessor platform (it will affect the multiprocessor scheduling) and a deliberately designed CPN model (e.g. with load-balanced nodes) will be the key to achieving real-time performance on multiprocessor workstations.

7. Conclusion and Future Work

The CPN model guarantees the correct concurrent execution and the C++/Pthread implementation provides portability and automatic scalability. Combining these two, Allen's

software package does provide an excellent infrastructure for implementing high-throughput real-time signal processing applications. This is again demonstrated by our experiment with the MPEG-4 simple-profile video encoder.

Following our experimental results on a single processor, we expect our implementation of the MPEG-4 simple-profile video encoder to speed up (approximately) linearly on multiprocessor or multiple workstations. So real-time MPEG-4 encoding using our implementation will be possible on multiprocessor workstations.

Future work needed to complete a real-time MPEG-4 video encoder on workstation include: adding shape coding to support the object-oriented interactive capability, profiling and balancing the computation load of each CPN Node, and testing its performance on multiprocessor platform.

References

[1] ISO/IEC JTC1/SC29/WG11 N2995, *Overview of the MPEG-4 Standard*, http://drogo.cselt.stet.it/mpeg/standards/mpeg-4/mpeg-4.htm, Oct. 1999.

[2] T. Sikora, "The MPEG-4 Video Standard Verification Model," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, no.1, pp. 19-31, Feb. 1997.

[3] Y. He, I. Ahmad and M. L. Liou, "A software-based MPEG-4 video encoder using parallel processing," *IEEE Trans. Circuits and Systems for Video Tech*., vol. 8, no. 7, pp. 909-920, Nov. 1998.

[4] G. E. Allen and B. L. Evans, "Real-Time Sonar Beamforming on Workstations Using Process Networks and POSIX Threads", *IEEE Transactions on Signal Processing*, pp. 921-926, Mar. 2000

[5] G. Cote, B. Erol, M. Gallant, and F. Kossentini, "H.263+: Video Coding at Low Bit Rates," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 849-866, Nov. 1998.

[6] J.-I. Kim and B. L. Evans, "System modeling and implementation of a generic video codec," *Proc. IEEE Workshop on Multimedia Signal Processing*, Los Angeles, CA, Dec. 1998, pp. 311-316.

[7] A. Hamosfakidis and Y. Paker, "Concurrency analysis for real time MPEG-4 video encoding," *IEEE Int. Conf. on Multimedia Computing and Systems*, Jun. 1999, vol. 2, pp. 862-866.

[8] G. Allen, *Computational Process Network Source Code*, http://www.ece.utexas.edu/~allen/CPNSourceCode/.

[9] MPEG Software Simulation Group, *Free MPEG Software*, http://www.mpeg.org/MPEG/MSSG/.