A High-Performance Parallel CAVLC Encoder on a Fine-Grained Many-Core System

Zhibin Xiao and Bevan Baas
ECE Department, University of California, Davis
{zxiao, bbaas}@ucdavis.edu

Abstract—**This paper presents a high-performance parallel context-based adaptive length coding (CAVLC) encoder implemented on a fine-grained many-core system. The software encoder is designed for a H.264/AVC baseline profile encoder. By utilizing arithmetic table elimination and compression techniques, the data-flow of the CAVLC encoder has been partitioned and mapped to an array of 15 small processors. The parallel workload of each processor is characterized and balanced for further throughput optimization. The proposed parallel CAVLC encoder achieves the real-time processing requirement of 30 frames per second for 720p HDTV. Our experiments show that the presented CAVLC encoder has 4.86 to 6.83 times higher throughput and requires far smaller chip area than the identical encoder implemented on state-of-art general-purpose processors. In comparison to published implementations on common DSP processors, the design has approximately 1.0 to 6.15 times higher throughput while requiring less than 6 times smaller area.**

I. Introduction

H.264/MPEG-4 AVC is a video coding standard developed through a collaboration of the ITU-T and ISO [1]. The new standard is proven to achieve significant video compression efficiency compared with prior standards. (39%, 49% and 64% bit-rate reduction versus MPEG-4, H.263 and MPEG-2 respectively) [2]. This high coding gain increase comes mainly from a combination of new coding techniques such as inter-prediction with quarter pixel accuracy, intra-prediction, multiple reference pictures, variable block size and context-based adaptive entropy coding. The video visual quality is further increased by an in-loop de-blocking filter to reduce edge effects of block-based video coding [3]. However, all of these new techniques come with a cost of high computation complexity which makes a software approach of a real-time HDTV encoder almost impossible in current general-purpose processors and DSPs.

In the baseline profile of H.264/AVC, the context-based adaptive length coding (CAVLC) is used for coding quantized transform coefficients of the residual images [4]. In CAVLC, the reverse zigzag scanned run-length coding, and adaptive VLC tables are used to encode 4x4 or 2x2 block residual data. Many hardware CAVLC encoding architectures have been proposed for real-time video encoding [5-7]. The drawback of hardware design is that it is dedicated and not flexible. The emerging multi-core approach has proven to be a possible solution for real-time H.264 video encoding. Many multi-core processors have been proposed for video applications which are composed of general-purpose cores [8-10]. However, the granularity of the cores is still very coarse which does not fully explore the essential property of current video processing algorithms: a transformation-based small block data-flow processing. This paper presents an implemented and highly parallel CAVLC encoder running on a fine-grained many-core system which contains 164 simple and small processors [11].

The rest of this paper is organized as follows. Section II introduces the CAVLC encoding process and the features of the targeted many-core system. In Section III, we describe the proposed CAVLC encoder in terms of partitioning, mapping and optimization. Section IV shows the performance analysis and results. Section V concludes the paper.

II. CAVLC Encoding and The Targeted Many-core System Architecture

A. CAVLC Encoding

The CAVLC encoder is used for encoding transformed and quantized residual coefficients of one video macroblock with the processing order as shown in Fig. 1. A maximum of 27 blocks must be encoded within one macroblock. Block "-1" contains 16 Luma DC coefficients if the current macroblock is encoded in 16x16 intra mode. Blocks 16 and 17 are formed by the DC coefficients of two Chroma components.

The CAVLC encoder can be partitioned into two phases, scanning phase and encoding phase. In the scanning phase all of the blocks are scanned in zigzag order. In the encoding phase, five different types of statistic symbols are encoded sequentially using look-up tables as Table 1 shows. The complexity of CAVLC mainly comes from the context-adaptive encoding of the 1st and 3rd elements, *coeff_token* and *levels*. The *coeff_token* is encoded for the total number of nonzero coefficients and trailing ones. Five different VLC tables are available for *coeff_token* encoding and the choice of table depends on the number of nonzero coefficients in the neighboring left and top blocks. This data dependency requires a large memory to store the number of nonzero coefficients for high quality video encoding. The *levels* are the nonzero coefficients (excluding trailing ones) encoded in reverse zigzag order. The *levels* code is made up of an all 0 prefix followed by a symbol 1 and suffix. The length of the suffix is initialized to 0 unless there are more than 10 nonzero coefficients and less than 3 trailing ones, in

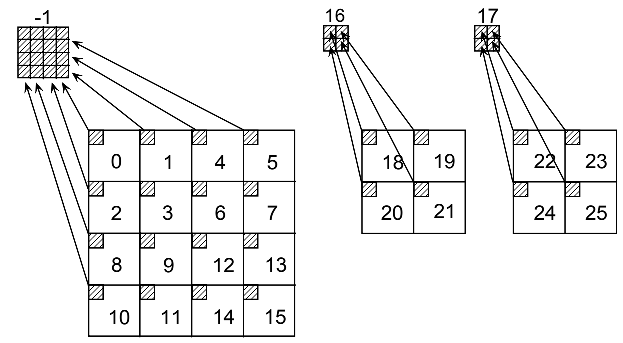


Fig. 1. Scanning order of residual blocks within a macroblock

Table 1. Elements of CAVLC Encoding per Block

Elements	Description
coeff_token	Encodes the number of nonzero coefficient and number of signed trailing ones – one per block
Sign_trail	Encodes the sign of trailing ones –one per trailing ones maximum 3 per block
levels	Encodes the remaining nonzero coefficients – one per level excluding trailing ones
Total_zeros	Encodes the total number of zeros before the last coefficient – one per block
Run_before	Encodes the number of run zeros preceding each nonzero levels in reverse zigzag order

which case it is initialized to 1. The length of the suffix can be adaptively incremented if the current level magnitude is larger than a certain threshold. A maximum of 6-bits are used for suffix encoding [4].

B. *Highlights of Many-Core Processor Array Architecture*

A high-level diagram of the targeted many-core chip architecture is shown in Fig. 2. The system is composed of 164 simple 16-bit DSP processors, three hardware accelerators, and three 16-KB integrated big memories, all connected by a reconfigurable mesh network [11]. Processors can directly communicate with their four nearest-neighbor processors, or distant processors using long-distance-capable configurable links.

Every processor core is a pipelined single issue DSP processor, supporting instructions that are commonly found in commercial DSPs. The fixed-point datapath contains one ALU and one MAC unit. Each processor contains a 128 word instruction memory and a 128-word by 16-bit data memory. Specialized address generation hardware is added to ease the implementation of algorithms requiring sequential, stride or bit-reversed access to data memory. Each processor occupies an area of 0.17 mm^2 and operates at a maximum frequency of 1.1 GHz in 65 nm CMOS technology.

III. THE PROPOSED PARALLEL CAVLC ENCODER

Fig. 3 shows the data flow of the proposed CAVLC encoder. The input residual coefficients are sent to the zigzag and CAVLC scanning block for the 1st phase processing.

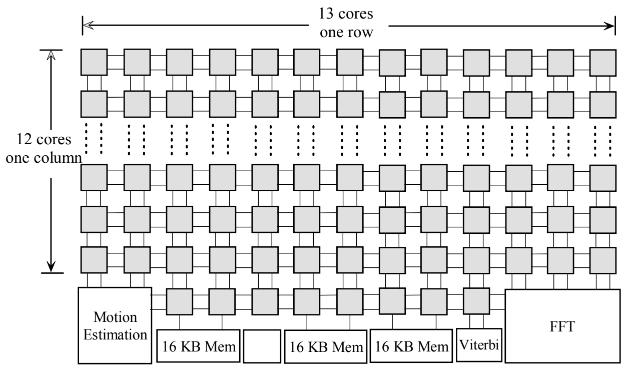


Fig. 2. Architecture of targeted many-core system

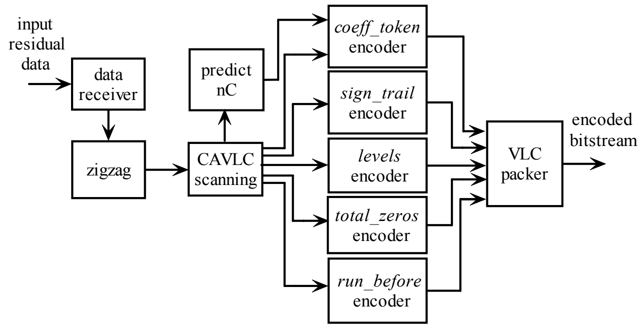


Fig. 3. Data flow diagram of the CAVLC encoder

Then corresponding data elements are distributed to five different encoding units in parallel. The final codes are assembled and packed by the packing unit. An efficient parallel mapping of this architecture on a fine-grained architecture requires overcoming a lot of challenges in terms of memory usage algorithm mapping and throughput optimization. This section describes our approach to these problems.

A. *Coeff_token Table Selection*

The *coeff_token* symbol is encoded with a table look-up based on the number of nonzero coefficients (TotalCoeff) and trailing +/-1 values (TrailingOnes). In the H.264 standard, five different look-up tables are used for this purpose and the choice of table depends on a parameter nC which is the average of the number of nonzero coefficients of the neighboring left and upper blocks named *nA* and *nB* respectively. Fig. 4 shows the organization of macroblocks within a QCIF frame. The 4x4 blocks within a macroblock are numbered as shown in Fig. 1. The gray and dark gray blocks are data-dependent blocks between neighboring macroblocks. As macroblocks are processed in raster scan order, a large memory is needed to store the number of the nonzero coefficients of those data-dependent blocks. However, as each macroblock needs only the *nA* and *nB* from neighboring 4x4 blocks, the memory requirement can be reduced by maintaining a global memory of *upper_nA* and

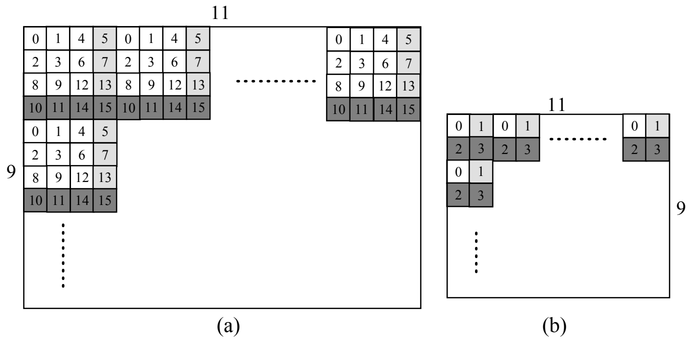


Fig. 4. Macroblocks in a QCIF frame (a) Luma (b) Chroma Cb/Cr

left_nB in one 16-KB on-chip memory of AsAP architecture. For 720p HDTV Luma frames, the *upper_nA* contains 320 parameters and *left_nB* contains 4 parameters. As each parameter uses no more than 5 bits, the *upper_nA* and *left_nB* can be further compressed to save half of the memory. The process to generate *nC* of one macroblock can be described as follows.

First, at the beginning of processing one macroblock, the program checks the availability of the neighboring top and upper macroblocks. If any of them are available, the program performs an update of *upper_nA_local* or *left_nB_local* both of which contain 4 parameters. The *upper_nA_local* and *left_nB_local* are stored in local processor data memory.

Second, the 4x4 blocks are processed in the order as illustrated in Fig. 1. The *upper_nA_local* and *left_nB_local* is updated locally for each 4x4 block so that the neighboring 4x4 blocks within one macroblock can use the up-to-date nA and nB values to generate corresponding *nC* values.

Third, after the final 4x4 bock of the current macroblock is processed, the current *upper_nA_local* and *left_nB_local* are written back to the global shared memoryso that subsequent macroblocks can use them as a starting point for *nC* generation.

The above approach of generating *nC* minimizes memory usage and proves to be an efficient method.

B. Arithmetic Table Elimination and Compression

1) Level Encoding: Instead of using seven large VLC tables, the arithmetic table elimination (ATE) technique is used to encode level information. The level encoding starts from the last nonzero coefficient (excludes trailing ones). Two parameters, *levels* and *vlcnum*, are sent to the encoding unit in each iteration. *Vlcnum* is initialized to 0 or 1 and will be updated for the next level encoding depending on the current level magnitude. The encoding unit encodes VLC0 and VLC1-6 separately with simple shift and addition operations. Due to the limit of the instruction memory, level encoding has been implemented on two processors as shown in Fig. 5. The P1 processor receives *level* information, sends *level* and *vlcnum* to P2 and updates the *vlcnum* each time.

2) Table Compression: The other symbols: *coeff_token*, *total_zeros* and *run_before* can be encoded by adopting ATE. However, if the tables can fit into one processor, table

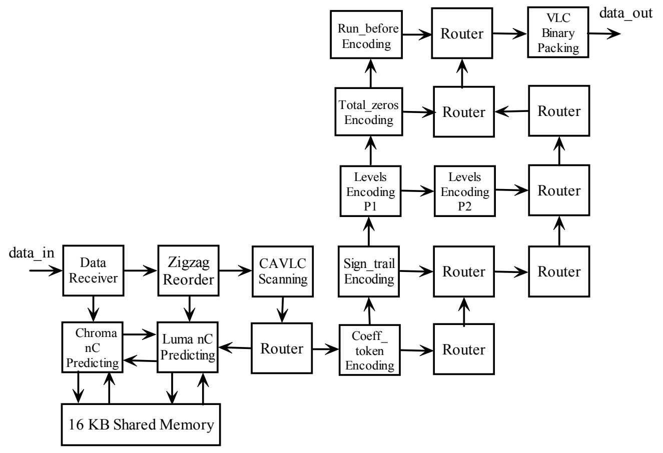


Fig. 5. CAVLC mapping without long-distance interconnections

look-up operations will be much faster than computations. We use a compression technique based on the structure of the VLC tables. The width of processor data memory is 16 bits and most of the data of the VLC tables are less than 4 bits except for some data inside the *coeff_token* when the number of total nonzero coefficients is larger than 12. Moreover, the VLC table used to encode *total_zeros* has a triangular structure, where most of data are zeros. Based on the above observation, we can divide the tables into smaller compressed tables and then determine which table to use at run-time with little extra computation. Table 2 compares the size of the compressed VLC tables and the uncompressed ones. A compression ratio of more than 4 has been achieved so that the encoding tasks of relative symbols can take place in one processor.

Table 2. Size of compressed VLC look-up table

VLC table	Original memory size (16-bit word)	Compressed memory size (16-bit word)
Coeff_token	448	111
Total_zeros	504	123
Run_before	224	55

C. Partitioning and Dataflow Mapping

As Fig. 3 shows, the CAVLC encoder can be easily partitioned into a number of independent serial and parallel tasks. When implementing the encoder on the array processor, each task is first mapped to a single processor to allow parallel execution. If either more memory or high performance is required than can be provided by a single processor, the task is mapped to multiple processors. Code for each processor is implemented independently, considering only its inputs and outputs. For many applications, including the CAVLC encoder presented here, writing the individual programs for the application's tasks is significantly easier than writing a single large sequential program, largely due to the encapsulation of complexity and the automatic handling

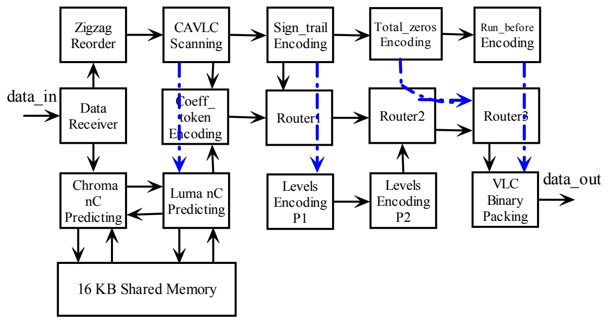


Fig. 6. CAVLC mapping with long-distance interconnections

of inter-processor communication.

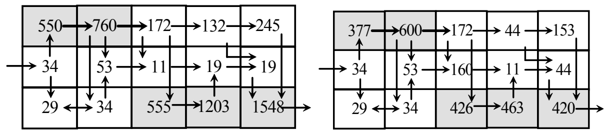
The major advantage of the fine-grain multiple-instruction multiple-data (MIMD) array over conventional programmable platforms is its capability of efficient stream processing. An application level pipeline can be created that allows the serial part of the CAVLC encoder (the scanning phase) to run in parallel. As data flow through the system, they are operated upon as soon as they are available. Each processor must store only a small amount of data (up to a 4x4 block data) for local computation.

As Fig. 3 shows, the CAVLC scanning and VCL packing unit need to distribute symbols and collect codes in parallel respectively. Parallel distribution is inefficient on a single processor because it needs to configure the output port at runtime and thus wastes many clock cycles. Fig. 5 shows a 20-processor parallel mapping using only nearest-neighbor connections. The CAVLC scanning unit sends statistical information only to the *coeff_token* encoding unit and the *coeff_token* encoding unit passes the information immediately to the next *sign_trail* encoding unit. This takes place for every encoding unit before it begins to operate on its own portion of data. This approach simplifies the mapping and will not degrade the throughput since the code produced by each unit needs to be collected in sequential order by the VLC packing unit anyway. In Fig. 5, the nC prediction unit is implemented on two processors for Luma and Chroma separately. The large memory supports two independent interfaces, which is ideal for this case.

The mapping in Fig. 5 is not as efficient as possible due to the constraints of a maximum of two input ports per processor and only nearest-neighbor processor communication. Eight routing processors are required to pass data around the graph. Fig. 6 shows a compact 15-processor mapping using long-distance interconnections. The four dashed lines represent long-distance links. The length of all the links are less than two processors. A savings of five routing processors shows the efficiency of the low overhead long-distance interconnection architecture.

D. Throughput Optimization

The throughput of the 15-processor mapping can be further optimized by characterizing the workload of each processor and remapping the workload for a better balance. The throughput of the design depends on the 4x4 block processing time of each processor which can be calculated by counting the time interval between the reception of the first input data and the transmission of the last output data. Fig. 7(a) shows an approximate clock cycle counts for each processor to process one 4x4 block of an intra-coding frame with the non-optimized codes. The blocks in gray are in the critical path of the encoding flow, which includes *zigzag reorder*, *CAVLC scanning*, *level encoding P1&P2*, and *VLC binary packing*, respectively. We adopted two approaches to optimize the mapping.



(a) non-optimized coding (b) optimized coding

Fig. 7. Processor workload non-optimized versus optimized coding

First, all programs are optimized using explicit pipeline data forwarding and hardware memory address generators in the programs. Pipeline data forwarding can reduce non-necessary NOPs used to deal with read-after-write (RAW) data hazards. Hardware memory address generation coupled with block repeat instructions also eliminate unnecessary cycles.

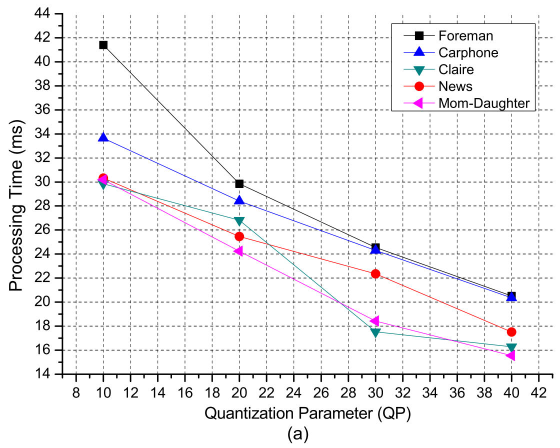
Second, the workload of VLC packing is remapped onto two routing processors. The codes can be packed as soon as they are produced by each encoding unit. Processors are labeled with an (*x, y*) row-column address beginning from the leftmost top processor labeled as (0,0). Notice the routing processor (3,1) is not used for VLC packing of level codes from processor (3,2) because it would block the bypassing of other coded data to processor (4,1). Fig. 7(b) shows the optimized workload on the same array of processors. Since the throughput depends mainly on the slowest block, the optimized encoder throughput is 2.57 times higher than the throughput of the non-optimized one.

IV. Results and Performance Analysis

The JM H.264 reference software [12] supplied the starting point for our CAVLC encoder implementation. Then we partition and rewrite the encoder in C on a parallel simulator based on MPI library. The encoder was then translated into assembly code running directly on the RTL model of the many-core chip. All processors including the 16 KB memory operate at a clock frequency of 1.07 GHz. The following performance analysis is based on the optimized 15-processor mapping shown in Fig. 6.

A. Performance Analysis and Comparison

The throughput of the software encoder is highly dependent on specific test video sequences and encoder parameters. In the H.264 standard, the coded block patterns (CBP) are used to determine the all-zero residual blocks which are not necessary to be encoded. Considering the CBP



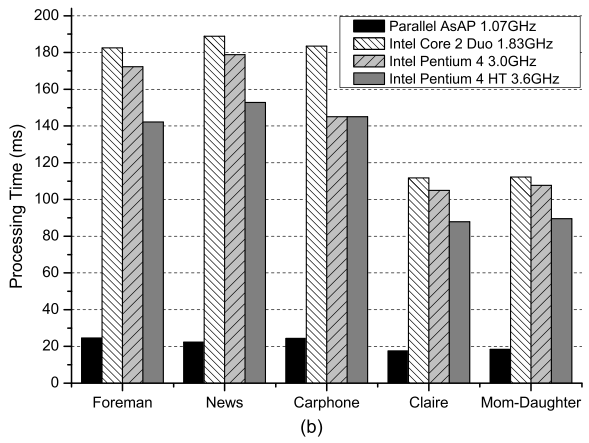


Fig. 8 Processing time (one 720p HDTV frame) of (a) the proposed CAVLC with varying QP (b) the CAVLC on different platforms with QP=30

Table 3. Specs of general-purpose computers for comparison

Machine	CPU	Freq. (GHz)	Cache (Byte)*	Memory (GByte)	Operating System
Lenovo Thinkpad T60	Intel Core 2 Duo T2400	1.83	32K/32 K/2M	2	Windows XP SP2
Dell OptiPlex GX620	Intel Pentium 4	3.0	32K/32 K/2M	1	Windows XP SP2
Dell GX280	Intel Pentium 4 HT	3.6	32K/32 K/2M	1	Windows XP SP2

*L1instruction cache/L1 Data cache/L2 shared cache

effects, we performed simulations using five QCIF sequences including Foreman, News, Carphone, Claire and Mom-Daughter with different quantization parameters. Simulation results are calculated by averaging the processing time of one I type and one P type frame. Fig.8 (a) shows the scaled time for the proposed CAVLC encoder to process one 720p (1280x720) HDTV frame. The results indicate that the encoder meets the real time requirement of processing 720p HDTV at 30 frames per second (processing time of 33.3 ms) when QP is larger than 18 (QP normally is chosen as 28).

In order to compare the performance of this design on the many-core chip [11] with other software platforms, we implemented and measured a version of the exact same encoder written in C from JM software and compiled with Visual C++ 6.0 on different state-of-the-art general purpose computers whose specifications are summarized in Table 3. All implementations are sequential and multi-threading is not utilized. Although these processors support single-instruction multiple-data (SIMD) instructions, they are of little use in speeding up a CAVLC implementation due to the serial nature of the CAVLC algorithm.

Fig.8(b) shows the scaled processing time of one 720p HDTV frame (QP = 30) using the many-core chip AsAP, Intel Core 2 Duo, Intel Pentium 4, and Intel Pentium 4 HT machines. All of the general-purpose machines are far from achieving real time 720 HDTV processing requirements. The throughput of the proposed parallel CAVLC encoder on the 1.07 GHz many-core chip is 4.86 to 6.83 times the throughput of the 3.6 GHz Pentium 4 HT machine which is the fastest among the three general-purpose machines. AsAP processors are very small in silicon area in comparison—the scaled area of the 112 mm^2 90 nm Pentium 4 HT processor is about 20.2 times larger than our CAVLC design including the area of 15 processors and one large memory [13].

The H.264 baseline encoder has also been implemented on different DSP platforms. The CAVLC requires about 18.2% of the H.264 baseline encoder computation for one reference picture encoder [14]. Based on this loading fraction, we can estimate the CAVLC encoder performance of published software H.264 baseline encoders on various DSP platforms. To make a fair comparison, the frequency of DSPs has been scaled to the same technology as AsAP. Table 4 summarizes the estimated performance of CAVLC encoders on common DSP platforms in terms of frame rate of 720p HDTV. The throughput of the presented CAVLC design on AsAP is about 1.0 to 6.15 times higher than the scaled published software encoders on common DSP platforms despite the fact our design uses a much more demanding test sequence. A smaller value QP=20 and two IP sequences will generate more non-zero residual data than other test sequences with larger QP value and more P frames. In addition, the scaled areas of the other processors are about 6.2 times larger than the area of our complete CAVLC encoder [15].

B. *Processor Activity Analysis*

A more detailed analysis of processor execution reveals some interesting insights into the bottleneck of our design. Fig. 9 illustrates processor activity and stall frequency for a macroblock encoding. The activity of each processor (the amount of time spent executing, instead of stalling), is

Table 4. Estimated performance of CAVLC encoder on common DSP platforms

Platform	Target App.	Processor Type	Technology	Area (mm^2)	Freq. (MHz)	Scaled Area to 65nm (mm^2)	Scaled Freq. to 65nm (MHz)	Test Sequence	CAVLC Performance (fps 720p)
TI C642 [16]	QCIF 30-40 fps	8-way VLIW	130nm CMOS	72	600	18	1200	60 frames IPPP…P QP=24	9-12
TI C642 [17]	CIF 24fps	8-way VLIW	130nm CMOS	72	600	18	1200	50 frames IPPP...P QP=25	28
ADSP BF561 [18]	CIF 30fps	Dual-core DSP	130nm CMOS	N/A	600	N/A	1200	N/A	36
TI C641 [19]	QCIF 24.5fps	8-way VLIW	130nm CMOS	72	600	18	1200	100 frames IPPP… P QP=28	7.4
This work AsAP	720p HDTV 30fps	Array (15 cores)	65nm CMOS	2.89*	1070	2.89*	1070	2 frames IP QP=20	36.0-41.3

*This value includes the area of 15 AsAP cores and one 16 KB shared memory

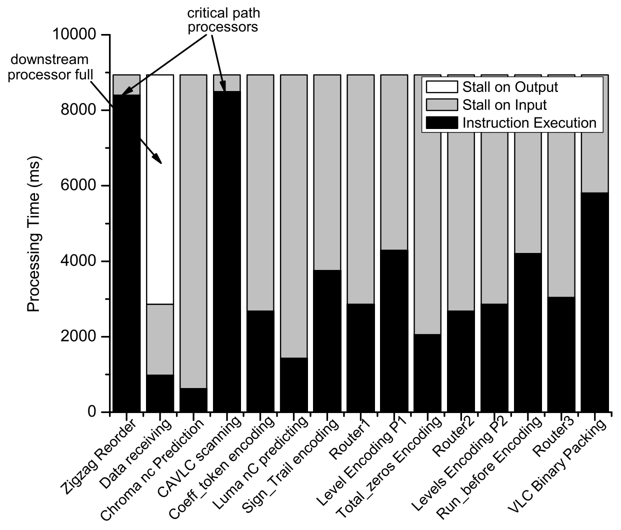


Fig. 9. Processor Activity of the CAVLC encoding one macroblock

indicated by the black bar in the figure. The white bar indicates the time stalled on output, while the gray bars indicate the time spent waiting for input to arrive. Fig. 9 shows that the *Zigzag Reorder* and *CAVLC scanning* processors are running most of the time and they are both bottlenecks of our design because the two algorithms are block-oriented and need to serially scan all the incoming data. The *data receiving* processor stalls most of time on output which indicates the downstream processor is full because the destination processor is not fast enough. All other processors stall most of the time on input which indicates the source processors are not providing data at an adequate rate. The large amount of stall cycles in Fig. 9 indicate a large slack for most of the processors, which provides a potential to run other tasks if multithreading is supported, or to reduce the clock rate and supply voltage to increase energy efficiency.

V. CONCLUSION

In this paper, a high-performance parallel CAVLC encoder for H.264/AVC is implemented on a fine-grained many-core system. The proposed software encoder employs arithmetic table elimination and compression techniques to map the data-flow of the CAVLC encoder onto an array of 15 small processors plus a large shared memory. The parallel workload of each processor is characterized and balanced for further throughput optimization. A utilization of long-distance interconnection reduces the number of required processors by 25%. This CAVLC design is the first software implementation on a fine-grained many-core system that can support real time 720p HDTV encoding to our best knowledge. The design achieves much higher throughput and much lower silicon area requirements compared with other software implementations on state-of-art general-purpose processors and DSPs. Our experiments show that the throughput of the proposed CAVLC encoder is 4.86 to 6.83 times higher than the throughput of the same design implemented on the state-of-art general-purpose processors and requires far less circuit area. The throughput is approximately 1.0 to 6.15 times higher than the throughput of published software encoders on common DSP platforms, and it requires approximately 6.2 times less area.

ACKNOWLEDGMENT

The authors gratefully acknowledge support from Intellasys, SRC GRC Grant 1598 and CSR Grant 1659, ST Microelectronics, UC Micro, NSF Grant 0430090 and CAREER Award 0546907, Intel and S Machines; assistance from D. Truong and A. Tran; and thank Jean-Pierre Schoellkopf and Patrick Cogez.

REFERENCES

[1] "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)," Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT-G050r1, Fairfax, VA, 2003.

[2] A. Joch, et al, “Performance comparison of video coding standards using Lagrangian coder control,” in *Proc. IEEE Int. Conf. on Image Processing*, 2002, pp. 501-504.

[3] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC Video Coding Standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no.7, 2003, pp. 560-576.

[4] G. Bjøntegaard and K. Lillevold, “Context-Adaptive VLC(CVLC) Coding of Coefficients,” Doc. JVT C028r1.doc, May 2002.

[5] Y. K. Lai, C. C. Chou, and Y. C. Chung, “A simple and cost effective video encoder with memory reducing CAVLC,” in *IEEE Int. Sym. on Circuits and Systems (ISCAS)*, May 2005, vol. 1, pp. 432–435.

[6] C. D. Chien, et al, “A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications,” in *IEEE Int. Sym. on Circuits and Systems (ISCAS)*, May 2006, pp. 3838–3841.

[7] Rahman, C.A. Badawy, W.B., “CAVLC Encoder Design for Real-Time Mobile Video Applications”, in *IEEE Transactions on Circuits and Systems II: Express Briefs*, Oct. 2007, vol.64 pp. 873-877.

[8] M. Taylor et al., “A 16-issue multiple-program-counter microprocessor with point-to-point scalar operand network,” in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2003, pp. 170-171.

[9] D. Pham et al., “The design and implementation of a first-generation CELL processor,” in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2005, pp. 184-185.

[10] S. Keckler et al., “A wire-delay scalable microprocessor architecture for high performance systems,” in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2003, pp. 168-169.

[11] Dean Truong, Wayne Cheng, Tinoosh Mohsenin, Zhiyi Yu, Toney Jacobson, Gouri Landge, Michael Meeuwsen, Christine Watnik, Paul Mejia, Anh Tran, Jeremy Webb, Eric Work, Zhibin Xiao, Bevan Baas, “A 167-processor 65 nm Computational Platform with Per-Processor Dynamic Supply Voltage and Dynamic Clock Frequency Scaling,” *Symposium on VLSI Circuits,* June 2008.

[12] JVT H.264/AVC Reference Software Version JM 12.4, http://iphome.hhi.de/suehring/tml/

[13] Darrell Boggs, et al, “The Microarchitecture of the Intel Pentium 4 Processor on 90nm Technology,” in *Intel Technology Journal*, Feb. 2004.

[14] W.I.Choi, et al, “Fast motion estimation with modified diamond search for variable motion block sizes,” in *IEEE Trans. on Image Processing*, Sept. 2003, vol.3, pp.371-374.

[15] Sanjive Agarwala,et al, “A 600-MHz VLSI DSP,” in *IEEE Journal of Solid-State Circuits*, vol.37, no.11, pp.1532-1544, Nov. 2002.

[16] Hong-Jun Wang, et al, "H.264/AVC Video Encoder Implementation Based on TI TMS320DM642", *International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, pp. 503-506, Dec., 2006.

[17] Li Zhuo, Qiang Wang, Feng, D.D., Lansun Shen, "Optimization and Implementation of H.264 Encoder on DSP Platform," in *IEEE Int. Conf. on Multimedia and Expo (ICME)*, July 2007, pp. 232-235.

[18] Kant S. et al, “Real time H.264 video encoder implementation on a programmable DSP processor for videophone applications”, in *Int. Conf. on Consumer Electronics (ICCE)*, Jan., 2006, pp. 93-94.

[19] Zhe Wei, Canhui Cai, “Realization and optimization of DSP based H.264 encoder,” in *IEEE Int. Sym. on Circuits and Systems (ISCAS)*, May, 2006.