A 1080p H.264/AVC Baseline Residual Encoder for a Fine-Grained Many-Core System

Zhibin Xiao, *Student Member, IEEE,* and Bevan M. Baas, *Member, IEEE*

***Abstract*—This paper presents a baseline residual encoder for H.264/AVC on a programmable fine-grained many-core processing array that utilizes no application-specific hardware. The software encoder contains integer transform, quantization, and context-based adaptive variable length coding functions. By exploiting fine-grained data and task-level parallelism, the residual encoder is partitioned and mapped to an array of 25 small processors. The proposed encoder encodes video sequences with variable frame sizes and can encode 1080p high-definition television at 30 f/s with 293 mW average power consumption by adjusting each processor to workload-based optimal clock frequencies and dual supply voltages—a 38.4% power reduction compared to operation with only one clock frequency and supply voltage. In comparison to published implementations on the TI C642 digital signal processing platform, the design has approximately 2.9–3.7 times higher scaled throughput, 11.2–15.0 times higher throughput per chip area, and 4.5–5.8 times lower energy per pixel. Compared to a heterogeneous single instruction, multiple data architecture customized for H.264, the presented design has 2.8–3.6 times greater throughput, 4.5–5.9 times higher area efficiency, and similar energy efficiency. The proposed fine-grained parallelization methodology provides a new approach to program a large number of simple processors allowing for a higher level of parallelization and energy-efficiency for video encoding than conventional processors while avoiding the cost and design time of implementing an application specific integrated circuit or other application-specific hardware.**

***Index Terms*—AsAP, CAVLC, fine-grained many-core system, H.264/AVC, parallel programming.**

I. Introduction

H.264/AVC IS a video coding standard developed through a collaboration of the ITU-T and ISO [1]. The 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-frame prediction with quarter pixel resolution, intra-prediction, multiple reference pictures, variable block size, context-based adaptive entropy coding, and in-loop de-blocking filter [3]. However, the coding efficiency improvement comes at a huge increase of computational complexity. A combination of all of the new coding features increase the computational complexity by a factor of two for the decoder and larger than one order of magnitude for the encoder compared with previous standards [4].

Most traditional video encoding architectures appear in one of the three forms: dedicated application-specific integrated circuits (ASIC), programmable digital signal processing (DSP) or general-purpose processors with either single-instruction-multiple-data (SIMD) multimedia extension [5] or application-specific instructions processing units [6], or a combination of these two. However, none of these methods achieve both high performance and flexibility for emerging and evolving multimedia standards. Furthermore, H.264/AVC shows less processing regularity and will be difficult for the SIMD approaches which mainly exploit explicit data-level parallelism. The high computational complexity of H.264/AVC makes it difficult to implement a low-power high-definition (HD) video encoder on general-purpose processors and DSPs. Thus, many current H.264/AVC HD encoders are implemented with dedicated ASICs which lacks flexibility and scalability to keep up with the fast development of new video standards [7]–[9]. Some other hybrid architectures use a hardware software co-design approach to speedup only complex tasks in H.264/AVC encoding such as motion estimation and context adaptive binary arithmetic coding [10].

This paper targets energy-efficient H.264 baseline encoding from low resolution to HD video encoding on a fine-grained many-core architecture. Our programmable approach achieves both high performance (up to real-time 1080p) and flexibility. We focus on the parallelization of the H.264/AVC baseline residual encoder which utilizes integer transform (IT), quantization, and context-adaptive variable length coding (CAVLC) to encode residual data from intra and inter prediction procedures. The IT and quantization are well suited for parallel implementation. However, high-performance CAVLC encoders are usually implemented in hardware due to its serial processing property [11], [12]. We choose to implement this software residual encoding accelerator because it is an essential task of H.264 baseline encoding. The configurable and programmable residual encoder can be used as a software co-processor for a full HD encoder.

Manuscript received June 5, 2010; revised October 13, 2010 and January 4, 2011; accepted January 25, 2011. Date of publication March 28, 2011; date of current version July 7, 2011. This work was supported by ST Microelectronics, Intel, UC Micro, NSF, under Grant 0430090 and CAREER Award 0546907, the SRC GRC, under Grant 1598, the CSR, under Grant 1659, Intellasys, S-Machines, and the C2S2 Focus Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity. This paper was recommended by Associate Editor G. Lafruit.

The authors are with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA 95616 USA (e-mail: zxiao@ucdavis.edu; bbaas@ucdavis.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2011.2133290

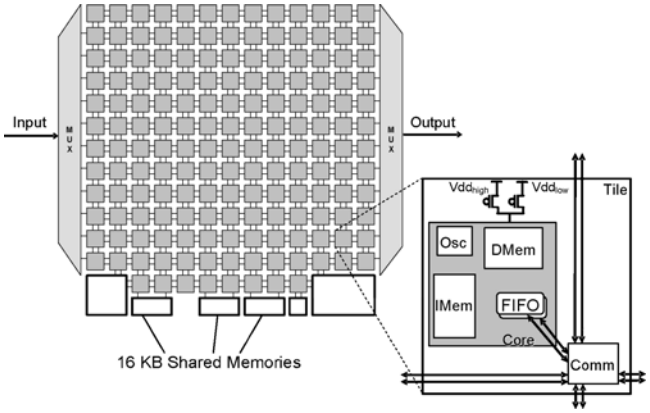


Fig. 1. Architecture of targeted many-core system.

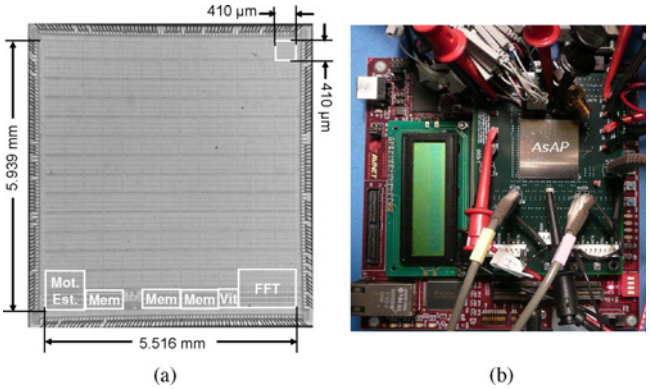


Fig. 2. Fully functional AsAP chip in 65 nm CMOS which runs at a maximum of 1.2 GHz and 1.3 V. (a) Die microphotograph. (b) Testing board.

A. Related Work

Many coarse-grained parallel multicore approaches have been proposed for H.264/AVC encoding. Most of them exploit thread-level or frame-level parallelism in video encoding algorithms. Chen *et al.* [13] proposed a parallel H.264/AVC encoder utilizing multilevel threading. Their results show good speedups ranging from 3.74x to 4.53x over well-optimized sequential code on a quad-core system. Roitzsch [14] proposed a slice-balancing technique for H.264 video decoding by modifying only the encoding stage and reports a performance speedup of up to 4.7. Rodriguez *et al.* [15] use message passing parallelization at group of pictures (GoP) and frame level to speed up H.264/AVC encoding. Zhao *et al.* [16] presented a wavefront parallelization method for H.264/AVC encoding. Their parallelization method is conducted at both frame and macroblock (MB) level. Sun *et al.* [17] proposed a similar parallel algorithm based on a wavefront technique. They partition one frame into different MB regions which are processed independently. The MBs within the MB region are then parallelized with the wavefront technique.

Stream processing has been proposed for multimedia applications that have computational intensity, data parallelism, and producer–consumer localities. The stream model was first proposed by Hoare [18] in communicating sequential processes. With the rapid development of integrated circuit technology, many architectures and processors supporting stream models have emerged, such as Imagine [19] and RAW [20]. Khailany *et al.* [21] use concurrencies between stream commands, data parallelism, instruction-level parallelism, and subword SIMD parallelism to speedup H.264/AVC motion estimation and deblocking filter kernels to achieve realtime 1080p HDTV encoding.

There is also a trend to use graphics processing units (GPUs) to accelerate video applications. Cheung *et al.* [22] presented an overview of video encoding and decoding using multicore GPUs. Chen *et al.* [23] implement H.264/AVC motion estimation on a GPU and report a 12 times speedup versus general-purpose CPUs. However, GPUs are more suitable for applications with abundant explicit thread-level and data-level parallelism and are less efficient for some serial video encoding algorithms in the H.264/AVC standard.

B. Our Approach and Contribution

This paper demonstrates our fine-grained many-core architecture can achieve high performance and energy efficiency for both video encoding algorithms with high data-level parallelism like IT and quantization and serial algorithms with fine-grained task-level parallelism like CAVLC. We propose a distributed processing approach to parallelize the H.264/AVC residual encoding at 4×4 block level. The proposed fine-grained parallelization exploits the existing locality and streaming nature of H.264/AVC residual encoding algorithms. Our work differs from previous research in that we apply a fine-grained approach to exploit task-level parallelism in H.264/AVC encoding.

The fine-grained parallelization brings challenges for programmers in terms of memory, mapping, throughput, and power optimizations. Our programming methodology yields an H.264/AVC residual encoder capable of realtime 1080p (1920×1080) HDTV encoding with both higher energy efficiency and area efficiency compared with other software approaches in common DSPs and customized hybrid multicore architectures.

The rest of this paper is organized as follows. Section II introduces the features of the targeted many-core system and the corresponding parallel programming methodology. In Section III, the H.264/AVC residual encoding algorithms including transform, quantization, and CAVLC encoding are described and analyzed. Section IV presents the approach to parallelize the residual encoding kernel in terms of partitioning, mapping and optimization. Section V shows the performance analysis and results. Section VI concludes this paper.

II. AsAP Architecture and Programming Methodology

A. Many-Core Array Architecture

The target asynchronous array of simple processors (AsAP) architecture is a fine-grained many-core system which is composed of simple cores that operate at independent clock frequencies and contain small memories for high energy efficiency [24].

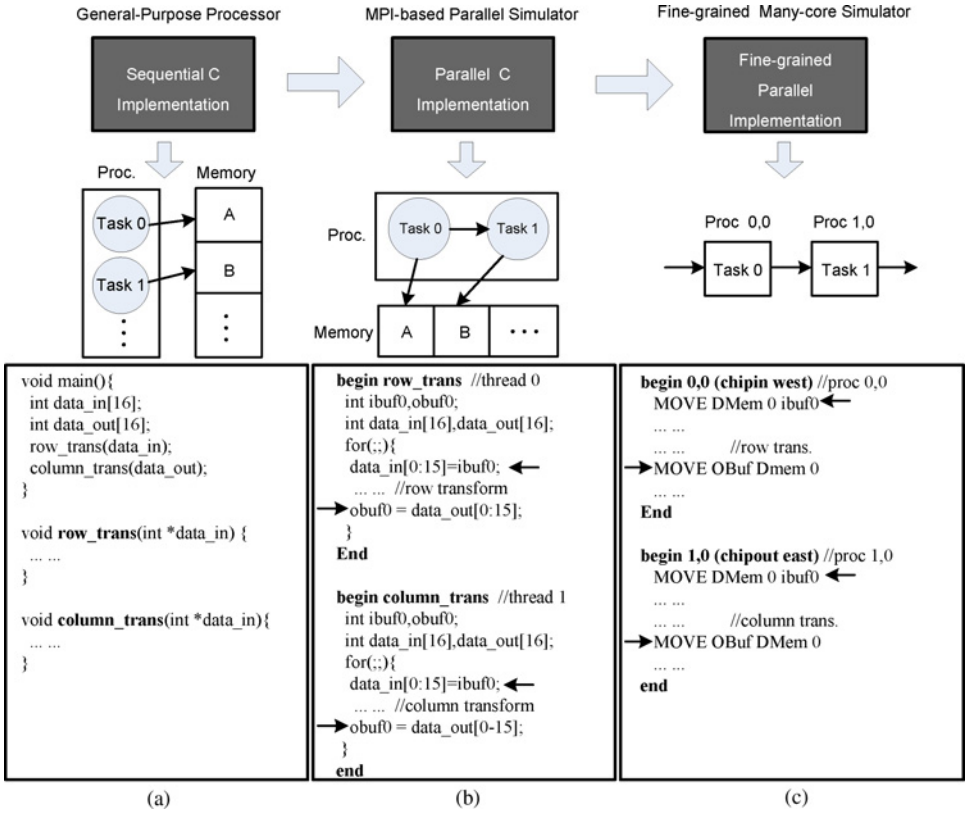


Fig. 3. Fine-grained parallel programming methodology with corresponding multitask application execution models and IT code examples. (a) Sequential C program. (b) Parallel C program. (c) Fine-grained AsAP program.

The AsAP platform targets applications which can be partitioned into small tasks running separately on small and simple processors [25]. A second generation design allows processors to operate at independent supply voltages and contains 16 kB shared memories [26].

Fig. 1 shows a high-level diagram of the AsAP chip which is fabricated in 65 nm complementary metal-oxide-semiconductor (CMOS) technology. Fig. 2 shows the AsAP chip die microphotograph and test board. The system is composed of 164 16-bit homogenous DSP processors, three dedicated accelerators, and three 16-kB integrated shared memories, all of which have local clock oscillators and are connected by a reconfigurable globally asynchronous locally synchronous (GALS) clocking style mesh network [27]. Compared with synchronous and mesochronous on-chip communication approach [28], the GALS approach simplifies the clock design, provides easy scaling into future deep submicrometer technologies and increases energy efficiency.

Each DSP processor contains a 16-bit datapath with a 40-bit accumulator and 128 word instruction and data memories. Although processors are not tailored specially for video encoding, they handle residual encoding very well since most of the encoding tasks require very small amounts of instruction and data memories. Processor tiles are connected through configurable nearest-neighbor or long-distance links.

In our platform, each processor can run at one of two supply voltages V_{ddHigh} and V_{ddLow} and optimized clock frequencies. This per-core based supply voltage and frequency configuration feature is useful for achieving maximum power efficiency in video applications with dynamic workloads as demonstrated in Section V.

B. Parallel Programming Methodology

Fig. 3 shows the parallel programming methodology for the proposed video encoder. The methodology is divided into three steps, which is further illustrated with corresponding multitask application execution models and examples of IT composed of row and column transform tasks.

We first implement a sequential C program, which uses a traditional shared memory model on a general-purpose processor as shown in Fig. 3(a). The IT tasks are implemented as C functions. The algorithm is fully verified to ensure bit-level correctness compared with H.264/AVC JM software [29].

Then the sequential algorithm is partitioned into multiple parallel tasks which are implemented with simple C programs separately as shown in Fig. 3(b). The IT can be divided into two tasks, row and column transforms. The two tasks can be combined by linking their inputs and outputs using a graphic user interface-based mapping tool. We have developed a Linux-based parallel simulator based on message passing interface library to verify the parallel C implementation. All the partitions in this level are coarse-grained and have no constraints on available resources including data and instruction memory.

Then coarse-grained tasks are repartitioned to fit on the resource-constrained AsAP processors. As shown in Fig. 3(c), the row and column transforms are implemented on individual AsAP processors. The final encoder is simulated on the config-

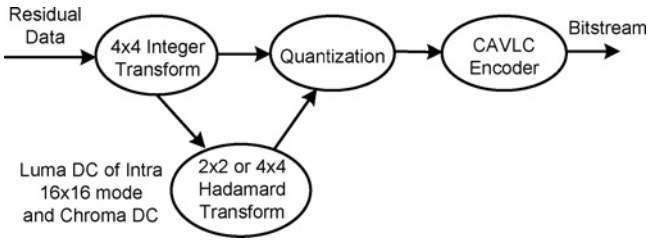


Fig. 4. Flow diagram of residual data encoding procedure in an H.264/AVC encoder.

urable Verilog register transfer language model of our platform using Cadence NCVerilog. By using the activity profile of the processors reported by the simulator, we evaluate its throughput and power consumption. The distributed processing approach is suitable for video and communication applications with streaming features so that large shared memories are avoided and each processor can work on its own piece of data.

III. Residual Encoding in H.264/AVC

Fig. 4 shows the residual data encoding procedure in the H.264/AVC baseline profile. First, a 4×4 IT is applied to the residual data from either intra or inter prediction procedures. For the intra 16×16 prediction mode, an additional 4×4 Hadamard transforms (HTs) is applied to the 16 luma direct current (DC) values within one MB. If the residual data are chroma DC coefficients, a 2×2 HT is applied. The CAVLC encoder encodes the zig-zagged 4×4 or 2×2 quantized transform coefficients and sends the bitstream out. All the functional blocks depicted in Fig. 4 are described in detail in the following sections.

A. Integer Transform

The H.264/AVC encoder uses three different transforms in the baseline encoder. They are 4×4 IT, 4×4 HT, and 2×2 HT. The forward 4×4 IT first operates on each 4×4 block X and produces a 4×4 block Y as follows:

$$\boldsymbol{Y} = C_i X C_i^T \tag{1}$$

where

$$\mathbf{C}_i = \begin{bmatrix} 1 & 1 & 1 & 0.5 \\ 1 & 0.5 & -1 & -1 \\ 1 & -0.5 & -1 & 1 \\ 1 & -1 & 1 & -0.5 \end{bmatrix}. \tag{2}$$

If the current MB uses intra prediction 16×16 mode, 16 luma DC values from the previous forward 4×4 IT are grouped into one 4×4 block X and transformed as follows:

$$\boldsymbol{Y} = H_f X H_f^T \tag{3}$$

where

$$\mathbf{H}_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{4}$$

TABLE I
Multiplication Factor (MF)

QP mod 6	Positions (0, 0), (2, 0) (1, 1), (1, 3)	Positions (1, 1), (1, 3) (3, 1), (3, 3)	Other Positions
0	13 107	5243	8066
1	11 916	4660	7490
2	10 082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

The chroma Cb and Cr DC values are grouped into 2×2 blocks separately and transformed by

$$\boldsymbol{Y} = H_t X H_t^T \tag{5}$$

where

$$\mathbf{H}_t = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{6}$$

The above three transformations can be implemented by simple shift and addition operations. The H.264/AVC transform multiplication scaling is integrated in the quantization process. Therefore, H.264/AVC requires less complicated operations for transformations than previous standards [30].

B. Quantization

H.264/AVC adopts two different quantization procedures for residual data from 4×4 IT and DC coefficients from 4×4 or 2×2 HT. The quantization procedures for 4×4 residual block is described as follows.

Each 4×4 block Y needs to be quantized individually as follows:

$$Z_{ij} = \text{round}(Y_{ij} \cdot \frac{PF_{ij}}{Q_{step}}) \tag{7}$$

where Y_{ij} is a coefficient of the transform described above, $Qstep$ is a quantization step size, Z_{ij} is a quantized coefficient, and PF_{ij} is a scaling factor from the transform stage.

In H.264/AVC, 52 $Qsteps$ are stored in a table indexed by a quantization parameter (QP) (0–51). In order to avoid division operations, the above equation can be simplified as follows:

$$Z_{ij} = \text{round}(Y_{ij} \cdot \frac{MF}{2^{qbits}}) \tag{8}$$

where

$$\frac{PF_{ij}}{QStep} = \frac{MF}{2^{qbits}} \tag{9}$$

and

$$qbits = 15 + \text{floor}(QP/6). \tag{10}$$

The above equations can be further simplified in integer arithmetic as follows:

$$|Z_{ij}| = (|Y_{ij}| \cdot MF_{ij} + f) \gg qbits \tag{11}$$

$$\text{sign}(Z_{ij}) = \text{sign}(Y_{ij}) \tag{12}$$

where $f = 2^{qbits}/3$ for intra blocks or $f = 2^{qbits}/6$ for inter blocks. MF_{ij} is the MF depending on QP and the pixel position in the 4×4 block as shown in Table I.

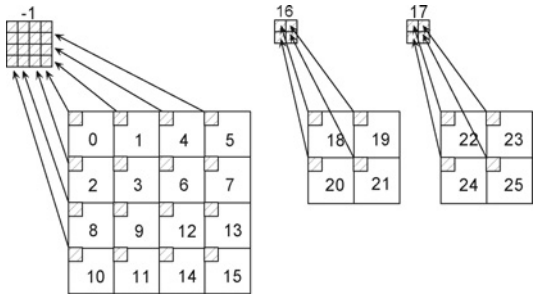


Fig. 5. Scanning order of residual blocks within a MB.

TABLE II
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 three 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

For intra 16×16 luma and chroma DC blocks, the transform coefficients $Y_{D(i,j)}$ are quantized to produce a block of quantized DC coefficients as follows:

$$|Z_{D(i,j)}| = (|Y_{D(i,j)}| \cdot MF_{(0,0)} + 2f) \gg (qbits + 1) \quad (13)$$

$$\text{sign}(Z_{ij}) \quad = \quad \text{sign}(Y_{ij}). \quad (14)$$

$MF_{(0,0)}$ is the MF for position (0, 0) in Table I and f, $qbits$ are defined as before.

C. CAVLC Encoding

The CAVLC encoder is used for encoding transformed and quantized residual coefficients of one video MB in the processing order as shown in Fig. 5. A maximum of 27 blocks must be encoded within one MB. Block "−1" contains 16 luma DC coefficients if the current MB is encoded in 16×16 intra mode. Blocks 16 and 17 are formed by the DC coefficients of two chroma components.

The CAVLC encoder can be partitioned into scanning and encoding phases. 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 II shows. The complexity of CAVLC mainly comes from the context-adaptive encoding of the first and third elements, *coeff_token* and *levels*. The *coeff_token* is encoded for the total number of nonzero coefficients and trailing ones. Five different variable-length coding (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 ten nonzero coefficients and less than three trailing ones, in 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 [31].

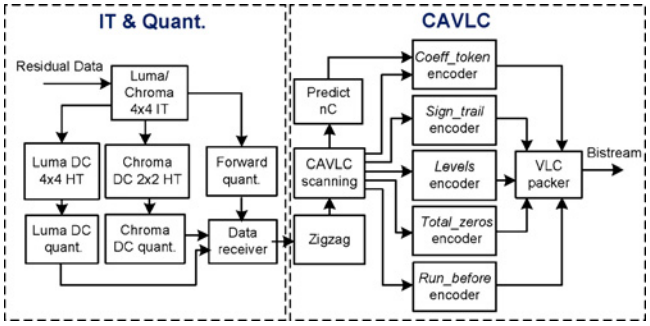


Fig. 6. Data flow diagram of the proposed H.264/AVC residual encoder.

IV. PROPOSED PARALLEL RESIDUAL ENCODER

Fig. 6 shows the data flow of the proposed parallel residual encoding kernel. The input residual data are sent to the shared 4×4 IT module. Then the transform coefficients are forwarded to the alternating current (AC) quantization, chroma DC, and luma intra 16×16 HT and quantization modules separately. All the quantized coefficients are collected by the data receiver module and sent to the CAVLC encoder. In the CAVLC encoder, the zigzag and CAVLC scanning block are the first phase of processing. Then corresponding syntaxes are distributed to five different encoding units in parallel. The packing unit collects and packs the final codes into an output bitstream. 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. Once the mapping and communication patterns are determined, coding for the small-memory processing array is similar to writing codes for a sequential machine. However, an efficient parallel mapping of this application on a fine-grained architecture still requires overcoming some challenges in terms of memory usage, mapping, and throughput optimization. The following section describes our approach to these problems.

A. IT and Quantization

1) *Memory and Algorithm Optimization:* Since the proposed encoder works at the 4×4 block level, most of the time a 16-word memory is required for storing streaming data. Thus, the 4×4 IT can be directly implemented on one AsAP processor in 97 cycles to process each 4×4 block (without configuration overhead). As for the quantization, we use look-up tables to implement computations such as $QP/6$, QP mod

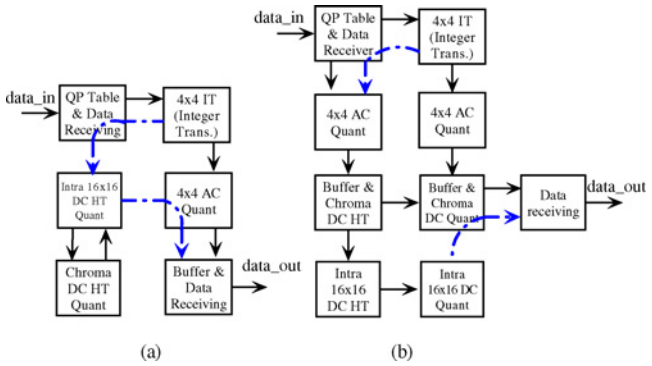


Fig. 7. Two mappings of IT and quantization. (a) Non-optimized. (b) Optimized with almost two times higher throughput.

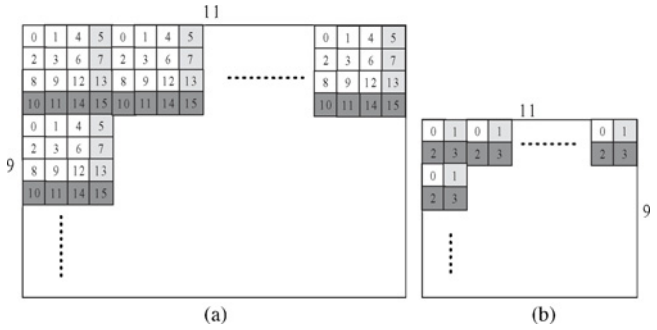


Fig. 8. MBs in a QCIF frame. (a) Luma. (b) Chroma Cb or Cr.

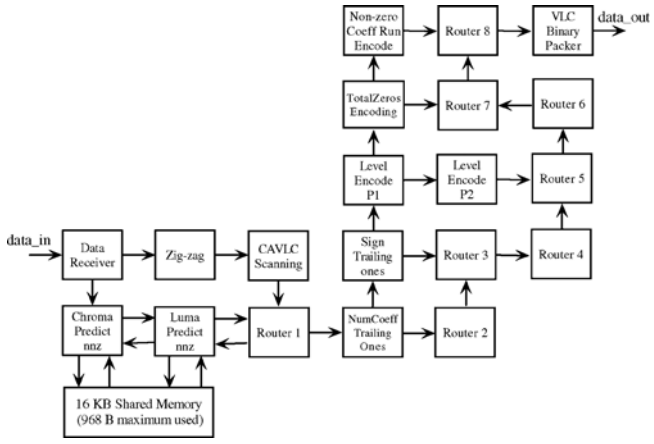


Fig. 9. 20-processor CAVLC straightforward mapping done manually without long-distance interconnection.

6, $2^{qbits}/6$, and $2^{qbits}/3$. Another problem for quantization is that the size of intermediate values exceeds 16 bits due to the large size of MFs. This can be solved by using the 40-bit accumulator to store the intermediate values so that a maximal precision is preserved during the quantization procedure.

If the MB is in intra 16×16 prediction mode, the luma DC (block -1) are first sent to the CAVLC encoder as shown in Fig. 5. This will break the natural task-level pipeline because the DC values cannot be fully collected until all the luma AC blocks within one MB are transformed and quantized. Thus, we need to buffer a maximum of 256 quantized luma AC values to reorganize the block order. We can compress two 8-bit AC values into one 16-bit word so that the buffer tasks can be implemented on one processor. Similarly, a maximum 64 quantized chroma AC values must be buffered so that the chroma DC values can be sent first (blocks 16 and 17 in Fig. 5).

2) *Mapping and Throughput Optimization:* Fig. 7(a) shows a 6-processor direct mapping of the IT and quantization procedures on the array processor. The two dashed lines represent long-distance links. The chroma DC HT and quantization procedures are implemented in a single processor. This fine-grained mapping creates an application level pipeline so that the major transform and quantization tasks are running in parallel. Our initial evaluation shows quantization is the bottleneck of this mapping. In order to support HDTV 1080p at 30 f/s, the 4×4 AC quantization processor needs to operate at 2.14 GHz. Fig. 7(b) shows a 9-processor mapping. We have duplicated the 4×4 *AC Quant* unit to double the throughput of the quantization tasks. The transformed coefficients are sent from the 4×4 *IT* processor alternately to the two 4×4 *AC Quant* processors. The chroma DC HT and quantization are implemented in two processors which also buffer half of the luma and chroma AC blocks within one MB. The intra 16×16 DC HT and quantization are running on two processors independently. The 9-processor mapping doubles the throughput with three extra processors and simple code duplication and re-mapping.

We can further parallelize the IT and quantization due to the vast data parallelism available in the transform and quantization operations. The residual encoder is parallelized in a way similar to a software pipeline. Therefore, the throughput of the encoder depends on the slowest task. Since the IT and quantization are fast enough for 1080p video encoding, we do not need to further improve this part. In the following section, we focus on the parallelization of the CAVLC encoder which may be slower than the IT and quantization tasks in the case that a test video sequence contains many nonzero residual data.

B. CAVLC Encoder

Compared with IT and quantization, the CAVLC algorithm is intrinsically serial due to the dependencies among 4×4 blocks within one MB and the neighboring MBs within a single video frame. However, task level parallelism can still be exploited by distributing different tasks among processors [32].

1) *Memory Optimization:* In the CAVLC encoder, the *coeff_token* symbol (refer to Table II) is encoded with a table look-up based on the number of nonzero coefficients (TotalCoeff) and trailing ± 1 values (TrailingOnes). In H.264/AVC, 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. 8 shows the organization of MBs within one quarter common intermediate format (QCIF) frame. The gray and dark gray blocks are data-dependent blocks between neighboring MBs. As MBs 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 MB needs

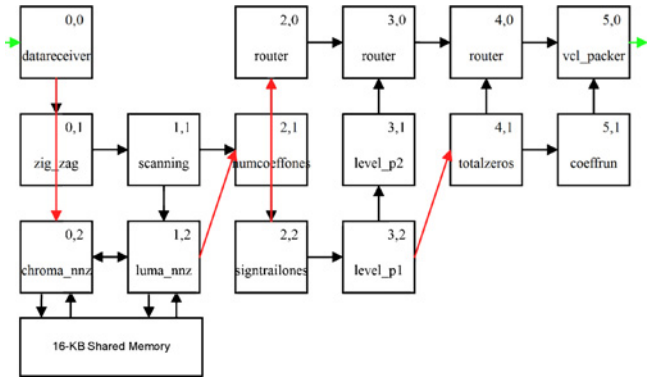


Fig. 10. 15-processor CAVLC mapping performed by an automatic task mapping tool [33].

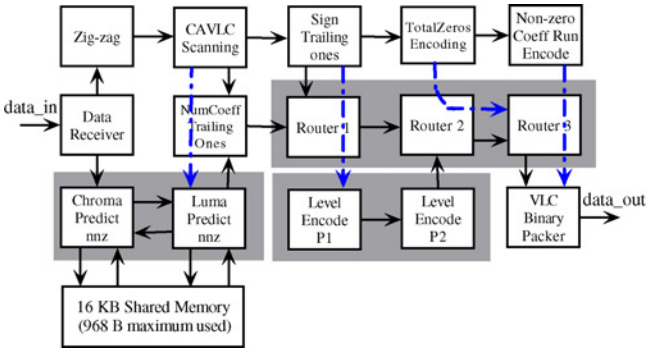


Fig. 11. 15-processor CAVLC mapping done manually with throughput identical to the mapping shown in Fig. 10.

only *nA* and *nB* from neighboring 4×4 blocks, the memory requirement can be reduced by maintaining a global memory of *upper_nA* and *left_nB* in one of the 16-kB on-chip memories of AsAP array. For one 1080p HDTV frame, the *upper_nA* contains 960 parameters and *left_nB* contains eight 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.

In our proposed CAVLC encoder, an arithmetic table elimination 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. 9. The P1 processor receives *level* information, sends *level* and *vlcnum* to P2 and updates the *vlcnum* each time.

We use look-up tables to encode the other symbols: *coeff_token*, *total_zeros*, and *run_before*. Most of the data in the VLC tables are less than 4 bits except for some entries in 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 data are zeros. Based on the above observations, we can divide the tables into smaller compressed tables and then determine which table to use at runtime with little extra computation. Our approach achieves a compression ratio of 4 so that the data tables of the tasks *coeff_token*, *total_zeros*, and *run_before* fit into one processor's 128-word 16-bit data memory.

2) *Dataflow Mapping:* As Fig. 6 shows, the CAVLC encoder can be easily partitioned into a number of independent serial and parallel tasks. When implementing the encoder on an array processor, each task is first mapped to a single processor to allow parallel execution. Each processor stores only a small amount of data (up to a 4×4 block data) for local computation. It is worth mentioning that the fine-grained partition step determines the throughput of the encoder since all of the tasks are implemented in a software pipeline style. In the following step, we need to map the fine-grained task graphs into the 2-D mesh array architecture. This mapping step can be conducted either manually or automatically by a customized AsAP mapping tool which aims to maximize nearest neighbor communication and insert as few number of routing processors as possible [33].

Fig. 9 shows a 20-processor straightforward manual 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. 9, the *nC* prediction unit is implemented on two processors for luma and chroma separately. The 16 kB shared memory supports two independent interfaces, which is ideal for this case.

The mapping in Fig. 9 is inefficient 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. 10 shows a compact 15-processor automatic mapping by the AsAP mapping tool which aims to map an algorithm with the shortest interconnection links and number of routing processors. The four long arrow lines represent long-distance links. The length of all the links are less than one processor. A saving of five routing processors shows the efficiency of the low overhead long-distance interconnection architecture [34]. With a little more manual optimization, we have another similar 15-processor mapping shown in Fig. 11, which is more regular and uses exactly a 5 by 3-processor array plus the shared memory. As shown in the shadow box of Fig. 11, compared to the CAVLC data-flow in Fig. 6, we added two more processors for the *nnz prediction* and *level encoding* and three routing processors which are required because of the constraints of two input ports per processor. Overall, the parallel mapping is very straightforward but effective once we have partitioned the algorithm well.

3) *Throughput Optimization:* The throughput of the 15-processor mapping can be further optimized by characterizing the workload of each processor and speeding up the

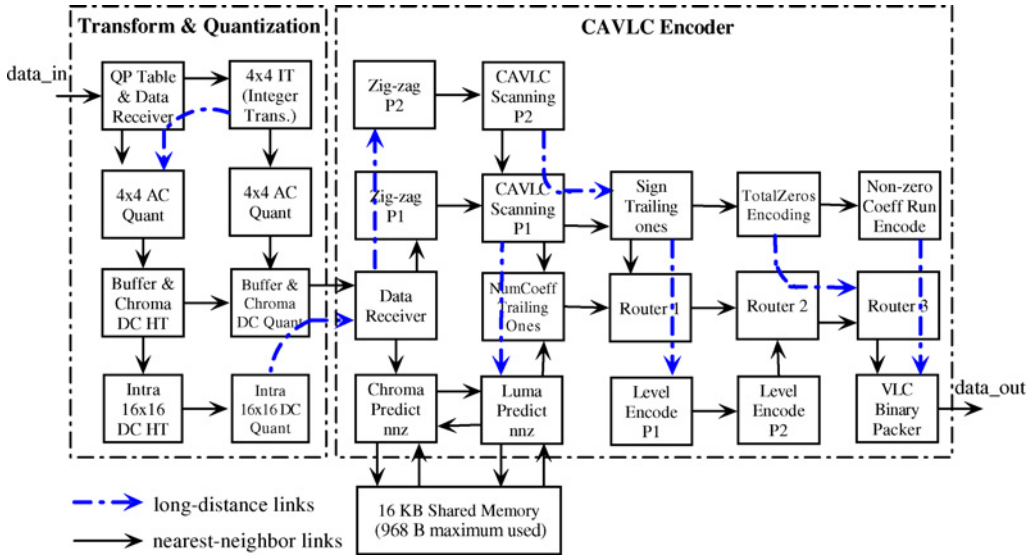


Fig. 12. Proposed 25-processor H.264/AVC residual encoder mapping.

processors in the critical data path. The noncritical processors add only latency to the system and do not affect the overall throughput. Since the processors stop once they finish their jobs, the processing time of one 4×4 block approximates the processor active time during the encoding.

Our evaluation shows the critical path of the CAVLC encoding includes zigzag reorder, CAVLC scanning, level encoding P1&P2, and VLC binary packing. We adopted three methods to optimize the mapping. First, the coding of these critical path processors are optimized by using AsAP's instructions and features such as block repeat, automatic address generation, and data forwarding. Second, the workload of VLC packing is re-mapped onto routing processors. The codes can be packed as soon as they are produced by each encoding unit. Third, we add another two processors to further parallelize zig-zag and CAVLC scanning procedures as shown by the CAVLC encoder in Fig. 12. These three optimizations triple the average throughput of the CAVLC encoder which can encode 1080p (1920×1080) HDTV at 30 f/s or higher for various video test sequences.

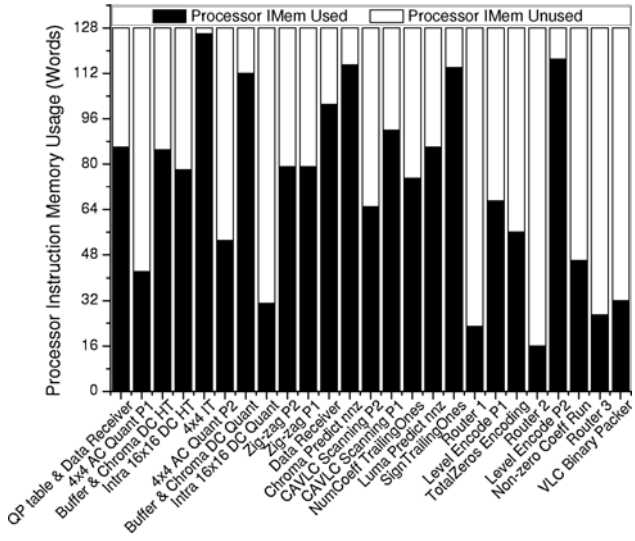


Fig. 13. Instruction memory usage of the proposed 25-processor encoder.

V. Simulation Results and Comparison

A. H.264/AVC Residual Encoder Implementation Results

Fig. 12 shows our proposed 25-processor fine-grained mapping for the H.264/AVC residual encoder. A total of eight processors are used for transform and quantization and 17 processors including one 16-kB shared memory (968 bytes maximum used for 1080p HDTV) are used for CAVLC encoding. There are eight long-distance links with a length of one processor. All other processors not included in the application mapping within the AsAP array (Fig. 1) are turned off to save power by halting their oscillators and disconnected them from the power grid with their individual power transistors. We may use the large number of unused processors to implement other workloads such as wireless communication or encryption for some applications such as a wireless security video encoding system.

Figs. 13 and 14 summarize the instruction and data memory usages for each processor among the 25 processors, respectively. Our implementation shows that 128 words of instruction and 128 words of data memory are more than enough for the H.264/AVC residual video encoding. Each processor of the 25-processor encoder uses an average of 72 words of instruction memory, which is 56.3% of all available instruction memory, and an average of 48 words of data memory, which is 37.2% of all available data memory.

In our proposed residual encoder, the throughput of the transform and quantization takes a maximum of 3960 cycles to encode one MB. The throughput of the CAVLC encoder is highly dependent on specific test video sequences and encoding QP value. In H.264/AVC, the coded block patterns (CBP) are used to determine the all-zero residual blocks which are not necessary to be encoded. Considering the CBP effects, we performed the simulations of our residual encoder using

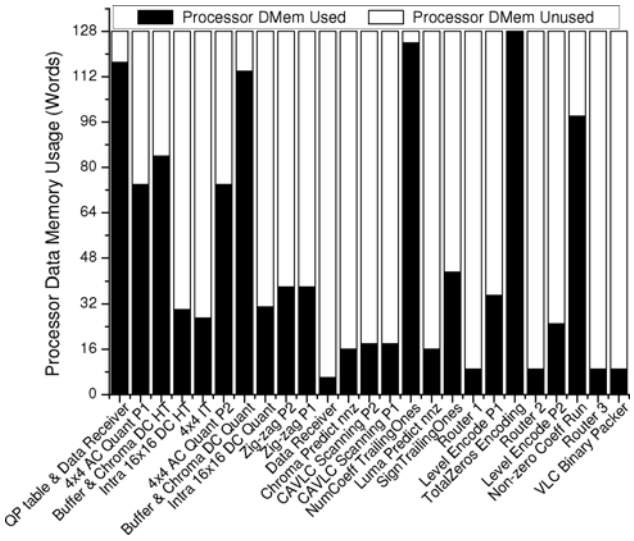


Fig. 14. Data memory usage of the proposed 25-processor encoder.

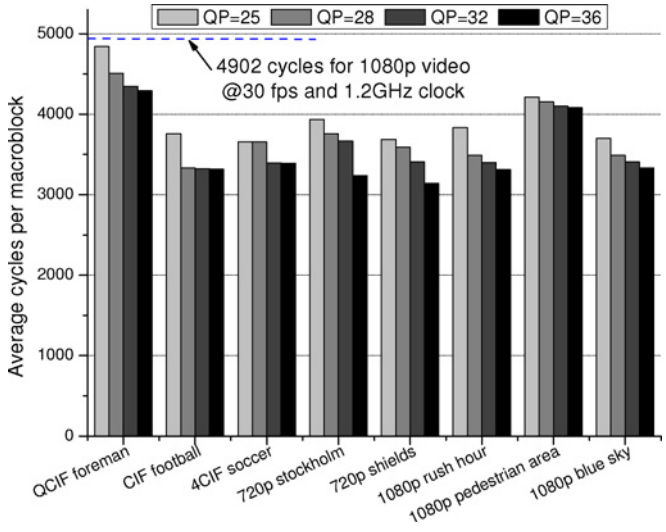


Fig. 15. Average cycles to encode one MB for test sequences with varying frame sizes and QP values.

eight test sequences with different frame size including QCIF *Foreman*, CIF *Football*, 4CIF *Soccer*, 720p *Stockholm*, 720p *Shields*, 1080p *Rush hour*, 1080p *Pedestrian area*, and 1080p *Blue sky*. All of these test sequences are encoded with four different QP values from 25 to 36.

We use the JM 12.4 reference software to encode original video sequences with a baseline setting. We collect the intermediate residual data after the intra and inter prediction in reference software and send them to our residual encoder as testing inputs. Simulation results are calculated by averaging the cycles of encoding one MB of one I type and one P type frame with a QP value from 25 to 36. If all the processors run at a maximum of 1.2 GHz with a supply voltage of 1.3 V, the encoder needs to encode one MB with less than 4902 cycles to support 1080p HDTV encoding. Fig. 15 shows the average cycles to encode one MB for all the tests. As shown in Fig. 15, all of the tests use less than 4902 cycles to encode one MB. The QCIF *Foreman* test sequences has the highest computation complexity and requires 4841 cycles to encode one MB at $QP = 25$. All of the other test sequences have a very steady encoding throughput in terms of average cycles per MB within a range of 3500–4200 cycles per MB. The results indicate that the encoder meets the real time requirement of 1080p HDTV encoding at 30 f/s.

B. Performance Evaluation

A more detailed analysis of processor execution reveals some interesting insights into the bottleneck of our design. Fig. 16 illustrates average processor activity of the encoder for encoding *Foreman* testing video with $QP = 25$. The activity of each processor (the amount of time spent executing, instead of stalling), is 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. 16 shows that the two 4×4 *AC Quant* processors are running all the time and they are both bottlenecks of our design in this case. The two processors intra 16×16 *DC HT* and intra 16×16 *DC Quant* are stalling on input for most of the time because

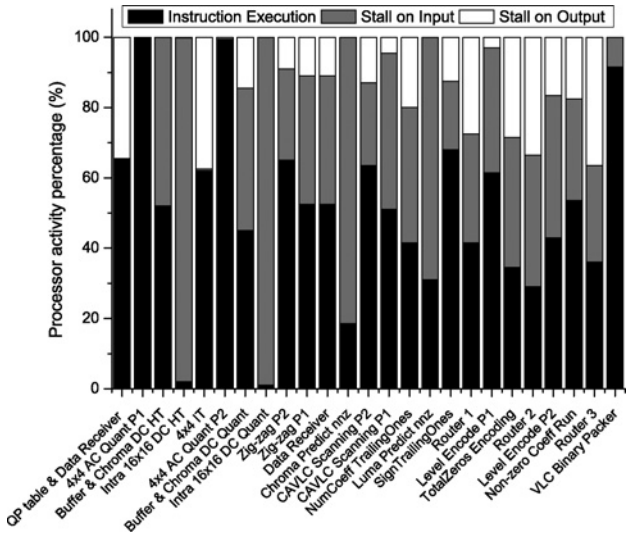


Fig. 16. Processor activity of the residual encoder while encoding QCIF *Foreman* at QP = 25.

the video frames are not encoded in intra 16×16 mode. The *QP Table & Data Receiver* and 4×4 *IT* processors stall on output for more than 30% of the whole encoding time because the downstream 4×4 *AC quant* processor is not fast enough to consume their outputs. Fig. 16 also shows that the *VLC Binary Packer* is busy most of the time due to the large volume of output bitstream which causes the other upstream processors in the CAVLC encoder to stall on output during execution. Most of the processors stall on input which indicates that at some time the source processors are providing data at a slower rate than the destination processor can consume it. The large amount of stall time in Fig. 16 shows a large slack for most of the processors, which provides a potential to reduce the clock rate and supply voltage to increase energy efficiency.

TABLE III
POWER MEASURED AT 1.3 V AND 1.2 GHZ

Operation of	100% Active (mW)	Stall (mW)	Standby (mW)
Processor	62.0	31.0	0.13
Shared memory	4.3	NA	0.11
Nearest-neighbor communication	5.9	NA	~0
Long-distance communication one tile	12.1	NA	~0

C. Power Consumption Optimization

1) *Power Estimation:* One advantage of the target many-core system is that each processor its own oscillator. The clock can be totally halted when the processor stalls for a certain amount of time either because of input empty or output full. During a short stall, the clock can still be active which results in more power consumption than the case of a total standby with halted clock. The overall activity of processors allows us to estimate the total average power by

$$P_{Total} = \sum_i P_{Exe,i} + \sum_i P_{Stall,i} + \sum_i P_{Standby,i} + \sum_i P_{Comm,i} + P_{sharedmemory} \quad (15)$$

where $P_{Exe,i}$, $P_{Stall,i}$, $P_{Standby,i}$, and $P_{Comm,i}$ represent the power consumption of computation execution, stalling with active clock, standby with halted clock, and communication activities of the ith processor among 25 processors, respectively. $P_{sharedmemory}$ is the average power of the 16-kB shared memory. $P_{Exe,i}$, $P_{Stall,i}$, and $P_{Standby,i}$ are estimated as follows:

$$\begin{aligned} P_{Exe,i} &= \alpha_i \cdot P_{ExeAvg} \\ P_{Stall,i} &= \beta_i \cdot P_{StallAvg} \\ P_{Standby,i} &= (1-\alpha_i-\beta_i) \cdot P_{StandbyAvg} \end{aligned} \quad (16)$$

where P_{ExeAvg}, $P_{StallAvg}$, and $P_{StandbyAvg}$ are average power while the processor is 100% active in execution, stalling, and standby (leakage only); α_i, β_i, and $(1-\alpha_i-\beta_i)$ are the percentage of execution, stall, and standby activities of processor i, respectively. The communication power of processor i can be estimated as follows:

$$P_{Comm,i} = \sum_j (\delta_{ij} \cdot P_{CommActive,L_j} + P_{CommStandby,L_j}) \quad (17)$$

where δ_{ij} is the communication active percentage of link j; $P_{CommActive,L_j}$ and $P_{CommStandby,L_j}$ are the average power consumed by a link with a length L while the link is 100% active and standby. Table III shows the measured average power consumption of various functions at 1.3 V and 1.2 GHz. We have included two types of communication link power since the length of the long-distance communication links in our application are no more than one tile. As shown in Table III, all the components consume little standby power and the communication circuits consume nearly zero leakage due to their simplicity.

Based on the average cycles per MB data as we present in Fig. 15, Table IV lists the maximum frequencies to support realtime (30 f/s) encoding of all the eight test sequences. The processors only need to run as low as 15 MHz to encode QCIF *Foreman* sequence at 30 f/s. Among all the tests, the 1080p *Pedestrian area* video sequence requires the highest frequency of 1032 MHz for real-time encoding. Based on (15)–(17), and Table III, we can reasonably estimate the average power consumption of our residual encoder. We use the processor and communication activity data from the profiling of encoding all the eight test sequences at $QP = 25$. Table IV shows the power consumption of all the tests without voltage and frequency scaling which means all of the processors run at the same maximum frequencies and corresponding supply voltages. The QCIF *Foreman* real-time encoding consumes only 4 mW and the power number increase proportionally with the frame size. The encoder consumes 115–121 mW for 720p HDTV tests at 30 f/s and 433–544 mW for 1080p HDTV tests at 30 f/s.

TABLE IV
POWER CONSUMPTION OF RESIDUAL ENCODER RUNNING AT 30 F/S WITH AND WITHOUT STATIC VFS

Test	Frame Size	Max Frequency (MHz)	Power w/o VFS (mW)	Power w/ VFS (mW)	Power Change (%)
Foreman	QCIF	15	4.0	3.0	−25
Football	CIF	45	9.1	7.1	−22
Soccer	4CIF	174	32	27	−16
Stockholm	720p	425	115	78	−32
Shields	720p	397	121	89	−26
Rush hour	1080p	939	433	271	−37
Pedestrian area	1080p	1032	544	347	−36
Blue sky	1080p	905	447	260	−42

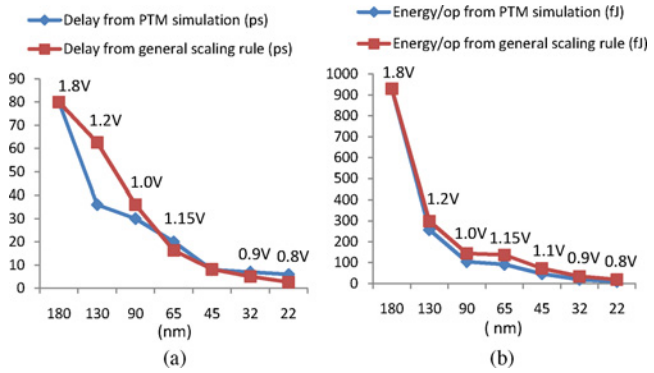


Fig. 17. (a) Delay and (b) energy per operation of an inverter driving a fanout of four based on SPICE simulation using PTM [35]; the general scaling rule assumes a v/s^2 reduction in delay and a $1/(sv^2)$ reduction in energy/op where s is the technology scaling factor and v is the voltage scaling factor [36]. (a)

2) *Power Optimization:* The power dissipation of our encoder can be further reduced by adjusting the frequency and voltage of each processor. Based on the processor activity number, each processor has an optimal operating frequency so that the processors can be active as much as possible. By running at these optimal frequencies, the power wasted by stalling and standby activities of the processors is eliminated. As shown in Fig. 16, in that case the two AC 4×4 *quant* processors must run at the highest frequencies and the other processors will run at lower frequencies.

Our platform supports two global supply voltage grids V_{ddHigh} and V_{ddLow}. The values of V_{ddHigh} and V_{ddLow} are variables for different test cases. The V_{ddHigh} is chosen to support the maximum frequency based on the voltage frequency curve presented in [27]. The V_{ddHigh} is set to 1.15 V for all the three

1080p video tests shown in Table IV. Based on our simulation, the two AC quantization processors are set to run at V_{ddHigh} for the three 1080p tests. The other processors can run at V_{ddLow} or V_{ddHigh} depending on their optimal operating frequency. If at V_{ddLow} the processor can reach its optimal operating frequency, the supply voltage is set to V_{ddLow}, otherwise V_{ddHigh} is chosen. To find the optimal V_{ddLow} we changed V_{ddLow} from V_{ddHigh} down to 0.65 V and chose the V_{ddLow} value which results in the minimum total power consumption.

Table IV summarizes the estimated power consumption of encoding the eight video sequences at $QP = 25$ with voltage frequency scaling (VFS). As shown in Table IV, with VFS, the residual encoder only consumes 3 mW for QCIF *Foreman* encoding at 30 f/s. For the two 720p video tests, the encoder consumes 78–89 mW with VFS. On average, with V_{ddHigh} and V_{ddLow} at 0.85 V and 0.75 V, the encoder consumes 84 mW power dissipation for 720p video encoding at 30 f/s—an average reduction of 29% compared with the design without VFS. For the three 1080p 30 f/s video sequences, the encoder consumes 260–347 mW. On average, with V_{ddHigh} and V_{ddLow} at 1.15 V and 0.9 V, the encoder is capable of 1080p video encoding at 30 f/s with 293 mW power dissipation—an average reduction of 38.4% compared with the design without VFS. The results demonstrate the effectiveness of voltage and frequency scaling for video applications with dynamic workloads. It is also interesting to notice that as frame size increases, the power savings increase with voltage and frequency scaling. This is because HD video encoding has more unbalanced workloads among the encoding tasks, which provides more power-saving potentials for voltage and frequency scaling.

D. Performance Comparison

The H.264/AVC baseline encoder has been implemented on many DSP platforms. In order to fairly compare with other reference designs, we estimate the loading fraction of residual encoding in a full baseline encoder. Since this loading fraction is affected by many different variables such as processor architecture and test video sequences, we use a range to estimate the fraction number.

The CAVLC occupies 18.2% computation time of the full baseline encoder running on a general-purpose computer [43]. Our parallelized IT and Quant modules take around 56.2% computation time of CAVLC encoding. Since the other reported designs use VLIW, SIMD, or multiple-issue architectures which are very likely able to execute multiple instructions per cycle during the computation of IT and Quant, we estimate they double their performance while computing these workloads. In this way, we estimate the IT and Quant take about 5.1% computation time of the full encoder. Summing up the two fractions, the residual encoder is estimated to take 23.3% computation time of a full encoder. We added a fluctuation ranging from $\pm 3\%$ to roughly estimate the test sequence variation which is observed in our JM encoding tests over various test sequences from QCIF to 1080p frame sizes. Thus, we estimate the residual encoder takes about 20.3–26.3% of a full baseline encoder.

For a fair comparison, all of the reference data are scaled to 65 nm technology at a supply voltage of 1.15 V. We use a technology scaling rule justified by SPICE simulation of an inverter driving a fan out of four under different technology nodes and supply voltages with prediction technology model (PTM) [35] as shown in Fig. 17. We use the metrics of throughput (Mpixel/s), throughput per area [(Mpixel/s)/mm^2], energy per pixel (nJ/pixel) to compare the throughput, hardware efficiency, and energy efficiency of each design.

Based on the loading fraction and technology scaling rule, we estimate the residual encoder performance of published software H.264/AVC baseline encoders on two DSP platforms and two hybrid multicore architectures as shown in Table V-C2. Since the proposed residual encoder on AsAP is configurable and programmable, we include the performance data of our design encoding 1080p, 720p, and CIF at 30 f/s at different supply voltages as shown in Table V-C2. The energy per pixel of AsAP reduces as we reduce the frame size and supply voltages. A reduction of 36% and 52% energy per pixel are achieved for 720p and CIF video encoding compared to 1080p encoding.

For a fair comparison, we only compare the other designs with AsAP while encoding 1080p at 30 f/s because the other results are scaled to 65 nm and 1.15 V. As shown in Table V-C2, compared with the encoder on the TI DSP C642, the proposed residual encoder on AsAP has 2.9–3.7 times higher throughput, 11.2–15 times higher throughput per chip area, and 4.5–5.8 times smaller energy per pixel. Compared with ADSP BF562 DSP, our design has 2.3–3.0 times higher throughput and 5.6–7.2 times smaller energy per pixel. The IBM cell processor is a heterogeneous multicore architecture for high-end gaming and multimedia processing [44]. The reference design on Cell has 5.9–7 times higher scaled throughput than our design at a cost of 4.2–4.8 lower area efficiency than AsAP. The Cell processor power number is not available though AsAP should have far higher energy efficiency due to area alone. The customized signal processing on-demand architecture (SODA) is specially optimized for H.264 by introducing flexible SIMD width, diagonal memory organization and special fused operation instructions [40]. Compared to the customized SODA, our implementation achieves 2.8–3.6 times higher throughput and 4.5–5.9 times higher area efficiency. AsAP has similar energy efficiency compared to the SODA customized for H.264. SODA has not been fabricated and both area and power data are from synthesis results [40].

We also implemented the same residual encoder written in sequential C and compiled it with Intel C++ Compiler 9.1 on a state-of-the-art Intel Core 2 Duo P8400 computer running Windows XP SP2 with 3G bytes DDR3 RAM. To be fair, we doubled the performance estimation of our sequential implementation based on the fact the encoder could be potentially parallelized at the thread-level on the dual-core processor [13]. As shown in Table V-C2, the throughput of our design is around 4.7 times the scaled throughput of the design running on the P8400. Our results show a state-of-the-art general-purpose processor cannot meet realtime 1080p encoding requirement with around two orders of magnitude smaller throughput per area and around 93 times higher energy per pixel compared with our design on AsAP.

TABLE V
COMPARISON OF RESIDUAL ENCODER ON DIFFERENT SOFTWARE PLATFORMS

Platform	Arch.	Tech. (nm)	V_dd (V)	Area (mm^2)	Max Frequency (MHz)	Power (mW)	Throughput	Esti. Resi.[a] Throughput (Mpixel/s)	Est. Resi.[a] Energy (nJ/pixel)	Other Results Scaled to 65 nm and 1.15 V: Throughput (Mpixel/s)	Throughput/Area [(Mpixel/s)/mm^2]	Energy (nJ/pixel)
TI C642 [37]	8-way VLIW	130	1.2	72	600	718	CIF@24f/s	9.3–12.0	59.8–77.2	16.7–21.6	0.9–1.2	21.2–27.4
ADSP BF561 [38]	Dual-core DSP	130	1.2	NA	600	1110	CIF@30f/s	11.6–15.0	74–95.7	20.9–27.0	NA	26.2–33.9
Cell [39]	CPU + SIMD PE	90	1	221	3200	NA	1080p@31f/s	244–317	NA	366–476	2.8–3.2	NA
SODA[b] customized for H.264 [40]	CPU + SIMD PE	90	1	14.29	300	68	CIF@30f/s	11.6–15.0	4.5–5.9	17.4–22.5	2.3–3.0	3.9–5.2
Intel P8400[c] [41]	Dual-core CPU	45	1.1	107	2260	12 500[d]	1080p@12f/s	25	500	13.2	0.06	437.9
This paper AsAP[e]	Array (25 cores)	65	1.15/0.9	**4.6**	**959**	**293**	**1080p@30f/s**	**62.2**	**4.7**	**62.2**	**13.5**	**4.7**
			0.85/0.75	**4.6**	**411**	**84**	**720p@30f/s**	27.6	**3.0**	27.6	6.0	**3.0**
			0.675/0.675	**4.6**	**45**	**7.1**	**CIF@30f/s**	3.0	**2.3**	3.0	0.65	**2.3**

The original published data are included under different technology nodes and voltages. For comparison, data are scaled to 65 nm technology with a supply voltage 1.15 V assuming a $1/s^2$ reduction in area. Throughput and energy are scaled based on a scaling rule justified by the SPICE simulation (Fig. 17).

[a]The residual encoding throughput is estimated based on a loading factor of 20.3–26.3% of a full baseline encoder.
[b]SODA is not fabricated and data are from synthesis results [40].
[c]Measured results by implementing the same residual encoder on Thinkpad T400 Core 2 Duo PC.
[d]The P8400's typical power is not available, so 50% of TDP (25 W) is used based on benchmark data of a general-purpose processor [42].
[e]The AsAP's area includes 25 cores and one 16-kB shared memory. Three sets of supply voltages are used for 1080p, 720p, and CIF video encoding separately.

VI. CONCLUSION

We have implemented a high-performance parallel H.264/AVC baseline residual encoder on a fine-grained many-core system. The encoder is composed of IT, quantization, and CAVLC blocks. The 25-processor residual encoder is the first software implementation on a fine-grained many-core system that supports realtime 1080p HDTV encoding to the best of our knowledge. We exploited data and task level parallelism in the H.264/AVC algorithms at the fine-grained block level and utilized the benefits of the GALS architecture to reduce power dissipation based on the workload of each processor. The design achieved higher throughput, much higher throughput per chip area, and much lower energy per pixel than the exact same encoder implemented on a general-purpose multiprocessor. It also compared very well with published implementations on programmable DSP processors, thus demonstrating the great promise of fine-grained many-core processor arrays for use in video encoding.

REFERENCES

[1] JVT, *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC)*," document JVT-G050, Mar. 2003 [Online]. Available: http://ip.hhi.de/imagecom_G1/assets/pdfs/JVT-G050.pdf

[2] A. Joch, F. Kossentini, H. Schwarz, T. Wiegand, and G. J. Sullivan, "Performance comparison of video coding standards using Lagrangian coder control," in *Proc. IEEE Int. Conf. Image Process.*, Dec. 2002, pp. 501–504.

[3] T. Wiegand, G. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[4] S. Saponara, K. Denolf, G. Lafruit, C. Blanch, and J. Bormans, "Performance and complexity co-evaluation of the advanced video coding standard for cost-effective multimedia communications," *EURASIP J. Appl. Signal Process.*, vol. 2004, no. 2, pp. 220–235, Jan. 2004.

[5] R. B. Lee, "Subword parallelism with max-2," *IEEE Micro*, vol. 16, no. 4, pp. 51–59, Aug. 1996.

[6] S. Saponara, L. Fanucci, S. Marsi, G. Ramponi, D. Kammler, and E. M. Witte, "Application-specific instruction-set processor for Retinex-like image and video processing," *IEEE Trans. Circuits Syst. II*, vol. 54, no. 7, pp. 596–600, Jul. 2007.

[7] K. Iwata, S. Mochizuki, M. Kimura, T. Shibayama, F. Izuhara, H. Ueda, K. Hosogi, H. Nakata, M. Ehama, T. Kengaku, T. Nakazawa, and H. Watanabe, "A 256 mW 40 Mb/s full-HD H.264 high-profile codec featuring a dual-macroblock pipeline architecture in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, pp. 1184–1191, Apr. 2009.

[8] Y.-K. Lin, D.-W. Li, C.-C. Lin, T.-Y. Kuo, S.-J. Wu, W.-C. Tai, W.-C. Chang, and T.-S. Chang, "A 242 mW 10 mm^2 1080p H.264/AVC high-profile encoder chip," in *Proc. IEEE ISSCC*, Feb. 2008, pp. 314–615.

[9] H.-C. Chang, J.-W. Chen, C.-L. Su, Y.-C. Yang, Y. Li, C.-H. Chang, Z.-M. Chen, W.-S. Yang, C.-C. Lin, C.-W. Chen, J.-S. Wang, and J.-I. Quo, "A 7 mW to 183 mW dynamic quality-scalable H.264 video encoder chip," in *Proc. IEEE ISSCC*, Feb. 2007, pp. 280–603.

[10] S. Saponaraa, M. Martinab, M. Casulaa, and L. Fanuccia, "Motion estimation and CABAC VLSI co-processors for real-time high-quality H.264/AVC video coding," *Microprocessors Microsyst.*, vol. 34, nos. 7–8, pp. 316–328, Nov. 2010.

[11] C.-D. Chien, K.-P. Lu, Y.-H. Shih, and J.-I. Guo, "A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications," in *Proc. IEEE ISCAS*, May 2006, pp. 3838–3841.

[12] C. A. Rahman and W. Badawy, "CAVLC encoder design for real-time mobile video applications," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 64, no. 10, pp. 873–877, Oct. 2007.

[13] Y.-K. Chen, X. Tian, S. Ge, and M. Girkar, "Toward efficient multi-level threading of H.264 encoder on Intel hyper-threading architectures," in *Proc. 18th IPDPS*, Apr. 2004, p. 63.

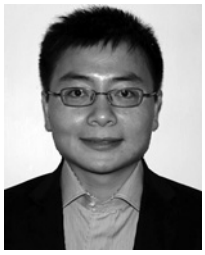
[14] M. Roitzsch, "Slice-balancing H.264 video encoding for improved scalability of multicore decoding," in *Proc. 7th ACM IEEE Int. Conf. Embedded Software*, Oct. 2007, pp. 269–278.

[15] A. Rodriguez, A. Gonzalez, and M. P. Malumbres, "Hierarchical parallelization of an H.264/AVC video encoder," in *Proc. Int. Symp. Parallel Comput. Electric. Eng.*, 2006, pp. 363–368.

[16] Z. Zhao and P. Liang, "A highly efficient parallel algorithm for H.264 video encoder," in *Proc. IEEE Int. Conf. Acou., Speech Signal Process.*, May 2006, pp. 489–492.

[17] S. Sun, D. Wang, and S. Chen, "A highly efficient parallel algorithm for H.264 encoder based on macro-block region partition," in *Proc. High Performance Comput. Commun.*, LNCS 4782. 2007, pp. 577–585.

[18] T. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 8, no. 21, pp. 666–677, 1978.
[19] U. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany, "The Imagine stream processor," in *Proc. IEEE Int. Conf. Comput. Des.*, Sep. 2002, pp. 282–288.
[20] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, W. Lee, A. Saraf, N. Shnidman, V. Strumpen, S. Amarasinghe, and A. Agarwal, "A 16-issue multiple-program-counter microprocessor with point-to-point scalar operand network," in *Proc. IEEE ISSCC*, Feb. 2003, pp. 170–171.
[21] B. K. Khailany, T. Williams, J. Lin, E. P. Long, M. Rygh, D. W. Tovey, and W. J. Dally, "A programmable 512 GOPS stream processor for signal, image, and video processing," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 202–213, Jan. 2008.
[22] N.-M. Cheung, X. Fan, O. C. Au, and M.-C. Kung, "Video coding on multicore graphics processors," *IEEE Signal Process. Mag.*, vol. 27, no. 2, pp. 79–89, Mar. 2010.
[23] W.-N. Chen and H.-M. Hang, "H.264/AVC motion estimation implmentation on compute unified device architecture (CUDA)," in *Proc. IEEE Int. Conf. Multimedia Expo*, Apr. 2008, pp. 697–700.
[24] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. M. Baas, "An asynchronous array of simple processors for DSP applications," in *Proc. IEEE ISSCC*, Feb. 2006, pp. 428–429.
[25] Z. Yu, M. J. Meeuwsen, R. W. Apperson, O. Sattari, M. Lai, J. W. Webb, E. W. Work, D. Truong, T. Mohsenin, and B. M. Baas, "AsAP: An asynchronous array of simple processors," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 695–705, Mar. 2008.
[26] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, and B. M. Baas, "A 167-processor 65 nm computational platform with per-processor dynamic supply voltage and dynamic clock frequency scaling," in *Proc. Symp. VLSI Circuits*, Jun. 2008, pp. 22–23.
[27] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, A. T. Tran, Z. Xiao, E. W. Work, J. W. Webb, P. V. Mejia, and B. M. Baas, "A 167-processor computational platform in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.
[28] F. Vitullo, N. E. L'Insalata, E. Petri, S. Saponara, L. Fanucci, M. Casula, R. Locatelli, and M. Coppola, "Low-complexity link microarchitecture for mesochronous communication in networks-on-chip," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1196–1201, Sep. 2008.
[29] S. Agarwala, T. Anderson, A. Hill, M. D. Ales, R. Damodaran, P. Wiley, S. Mullinnix, J. Leach, A. Lell, M. Gill, A. Rajagopal, A. Chachad, M. Agarwala, J. Apostol, M. Krishnan, D. Bui, Q. An, N. S. Nagaraj, and T. Wolf, "A 600-MHz VLIW DSP," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1532–1544, Nov. 2002.
[30] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 598–603, Jul. 2003.
[31] G. Bjøntegaard and K. Lillevold, "Context-adaptive VLC (CVLC) coding of coefficients," document JVT C028r1.doc, JVT of ISO/IEC MPEG and ITU-T VCEG, May 2002.
[32] Z. Xiao and B. M. Baas, "A high-performance parallel CAVLC encoder on a fine-grained many-core system," in *Proc. ICCD*, Oct. 2008, pp. 248–254.
[33] E. W. Work, "Algorithms and software tools for mapping arbitrarily connected tasks onto an asynchronous array of simple processors," M.S. thesis, Office Graduate Studies, Univ. California, Davis, Sep. 2007 [Online]. Available: http://www.ece.ucdavis.edu/vcl/pubs/theses/2007-4
[34] Z. Yu and B. M. Baas, "A low-area interconnect architecture for chip multiprocessors," in *Proc. IEEE ISCAS*, May 2008, pp. 2857–2860.
[35] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm design exploration," in *Proc. 7th Int. Symp. Qual. Electron. Des.*, Mar. 2006, pp. 585–590.
[36] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.
[37] L. Zhuo, Q. Wang, D. D. Feng, and L. Shen, "Optimization and implementation of H.264 encoder on DSP platform," in *Proc. IEEE ICME*, Jul. 2007, pp. 232–235.
[38] S. Kant, U. Mithun, and P. S. S. B. K. Gupta, "Real time H.264 video encoder implementation on a programmable DSP processor for videophone applications," in *Proc. ICCE*, Jan. 2006, pp. 93–94.
[39] X. He, X. Fang, C. Wang, and S. Goto, "Parallel HD encoding on CELL," in *Proc. IEEE ISCAS*, May 2009, pp. 1065–1068.
[40] S. Seo, M. Woh, S. Mahlke, T. Mudge, S. Vijay, and C. Chakrabarti, "Customizing wide-SIMD architectures for H.264," in *Proc. 9th Int. Conf. SAMOS*, 2009, pp. 172–179.
[41] Intel. (2010, Oct.). *Intel Processor Specifications* [Online]. Available: http://ark.intel.com/Product.aspx?id=35569
[42] M. Butler, "AMD Bulldozer Core: A new approach to multithreaded compute performance for maximum efficiency and throughput," in *Proc. IEEE Hot Chips Symp. (Hot Chips 22)*, Aug. 2010.
[43] W. I. L. Choi, B. Jeon, and J. Jeong, "Fast motion estimation with modified diamond search for variable motion block sizes," in *Proc. Int. Conf. ICIP*, vol. 3. Sep. 2003, pp. 371–374.
[44] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation CELL processor," in *Proc. IEEE ISSCC*, Feb. 2005, pp. 184–185.



Zhibin Xiao (S'07) received the B.S. (with honors) and M.S. degrees in information science and electrical engineering from Zhejiang University, Hangzhou, China, in 2003 and 2006, respectively, where his research focused on high-performance multimedia processor design. He is currently pursuing the Ph.D. degree in electrical and computer engineering at the University of California at Davis, Davis.

His current research interests include high-performance many-core processor architecture, parallel video encoding implementations, and scalable memory system design.

Bevan M. Baas (M'99) received the B.S. degree in electronic engineering from California Polytechnic State University, San Luis Obispo, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1990 and 1999, respectively.

From 1987 to 1989, he was with Hewlett-Packard, Cupertino, CA, where he participated in the development of the processor for a high-end minicomputer. In 1999, he joined Atheros Communications, Santa Clara, CA, as an early employee and was a core member of the team which developed the first IEEE 802.11a (54 Mb/s, 5 GHz) Wi-Fi wireless local area network solution. In 2003, he joined the Department of Electrical and Computer Engineering at the University of California at Davis, Davis, where he is currently an Associate Professor. He leads projects in architecture, hardware, software tools, and applications for very large scale integrated computation with an emphasis on digital signal processing workloads. In 2006, he was a Visiting Professor with Intel's Circuit Research Laboratory, Hillsboro, OR. His current research interests include the 36-processor asynchronous array of simple processors chip, applications, and tools, a second generation 167-processor chip, low density parity check decoders, fast Fourier transform processors, Viterbi decoders, and H.264 video codecs.

Dr. Baas was a recipient of the National Science Foundation CAREER Award in 2006 and the Most Promising Engineer/Scientist Award by AISES in 2006. He was a National Science Foundation Fellow from 1990 to 1993 and a NASA Graduate Student Researcher Fellow from 1993 to 1996. He is an Associate Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS, and has was a Technical Program Committee Member of the IEEE International Conference on Computer Design in 2004, 2005, 2007, and 2008, on the HotChips Symposium on High Performance Chips in 2009, and on the IEEE International Symposium on Asynchronous Circuits and Systems in 2010.