International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation. July, 2009. Greece.

# Customizing Wide-SIMD Architectures for H.264

S. Seo, M. Woh, S. Mahlke, T. Mudge
Department of Electrical and Computer Engineering
University of Michigan, Ann Arbor, MI 48109
Email: swseo,mwoh,mahlke,tnm@umich.edu

S. Vijay, C. Chakrabarti
Department of Electrical Engineering
Arizona State University, Tempe, AZ 85287
Email: vijays,chaitali@asu.edu

***Abstract*—In recent years, the mobile phone industry has become one of the most dynamic technology sectors. The increasing demands of multimedia services on the cellular networks have accelerated this trend. This paper presents a low power SIMD architecture that has been tailored for efficient implementation of H.264 encoder/decoder kernel algorithms. Several customized features have been added to improve the processing performance and lower the power consumption. These include support for different SIMD widths to increase the SIMD utilization efficiency, diagonal memory organization to support both column and row access, temporary buffer and bypass support to reduce the register file power consumption, fused operation support to increase the processing performance, and a fast programmable crossbar to support complex data permutation patterns. The proposed architecture increases the throughput of H.264 encoder/decoder kernel algorithms by a factor of 2.13 while achieving 29% of energy-delay improvement on average compared to our previous SIMD architecture, SODA.**

## I. Introduction

In the past decade, mobile devices have rapidly proliferated. Today's devices not only support advanced signal processing of wireless communication data, but also multimedia services such as video encoding/decoding, interactive video conferencing and image manipulation. All of this requires a powerful processor which has to be very power-efficient.

H.264 is a state-of-the art video compression standard of ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). This standard provides higher quality video with lower bit rates than earlier standards and has been adopted in many of current and next generation video applications. For instance, both the Bluray Disc and HD-DVD format ratified H.264 as one of three mandatory video compression codecs for High Definition DVD, and the Digital Video Broadcast (DVB) also selected the use of H.264 for broadcast television.

Most mobile processors today combine general-purpose processors, digital signal processors and hardwired ASICs to satisfy the high-performance and low-power requirements. However, such a heterogeneous platform is inefficient in terms of area, power and programmability. Earlier, we have developed SODA (Signal Processing On-Demand Architecture) [1], wide-SIMD low-power programmable platform for wireless communications. In this paper, we present a programmable architecture that has been optimized for H.264. This is also a wide-SIMD architecture like SODA with features that exploit the characteristics of the H.264 kernel algorithms. The customizing features include support of multiple SIMD widths to increase the SIMD utilization efficiency, diagonal memory organization to avoid memory access conflict, bypass and buffer support to reduce the register file (RF) power consumption, fused operation support to speed up the processing, and a fast programmable crossbar to support complex data shuffle operations. The proposed architecture is similar to AnySP [19], but customized more for video codecs. The customized architecture is implemented in the RTL Verilog model and synthesized in TSMC 90nm using Synopsys physical compiler. The results show that the customizing features increase the processing throughput by a factor of 2.13 while achieving 29% of energy-delay improvement over SODA.

The rest of the paper is organized as follows. Section II gives a brief overview of H.264 encoder/decoder. Section III introduces SODA, the SIMD-based high-performance DSP processor for wireless communications. Section IV introduces the new architectural features incurred by H.264 algorithms and Section V describes the modified processing element (PE) architecture. Section VI shows how H.264 kernel algorithms are mapped on the modified SIMD architecture. Section VII presents the throughput and power analysis of the augmented architecture. Section VIII introduces the related work and Section IX concludes the paper.

## II. H.264 CODEC

Video compression is being actively considered for mobile communication systems because of the increasing demand of multimedia services on mobile devices. In this paper, we focus on H.264 because it is representative of contemporary video coding standards and achieves better performance than earlier standards such as MPEG-1, MPEG-2, MPEG-4, and H.263.

Fig. 1 shows the block diagram of H.264 encoder and decoder. The encoder includes two dataflow paths: a forward path (left to right) and a reconstruction path (right to left) [2]. The dataflow of the decoder contains the reconstruction path (shown in shaded blocks).

The H.264 encoder processes an input frame or field $Fn$ in macroblock units. Each macroblock is encoded using inter-prediction or intra-prediction. In the inter-prediction mode, the predicted $P$ macroblock is formed by motion-compensated prediction from previously encoded frames, and in the intra-prediction mode, $P$ is predicted by the current frame. The $P$ macroblock is subtracted from the current macroblock to produce a residual block $Dn$ that is transformed, quantized,

978-1-4244-4501-1/09/$25.00 ©2009 IEEE

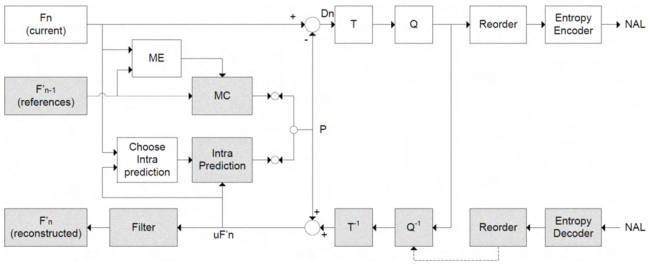


Fig. 1. H.264 encoder/decoder reference design. ME: Motion Estimation, MC: Motion Compensation, T: Transformation, Q: Quantization, NAL: Network Abstract Layer. Grey area represents functional blocks of the H.264 decoder, which is the subset of the H.264 encoder [2].

reordered, and entropy encoded. The entropy-encoded coefficients with header information that includes prediction modes, quantizer parameter, motion vector information, etc. form the network abstract layer (NAL) bitstream.

The H.264 decoder receives the compressed bitstream from the NAL. The entropy decoder decodes the bitstream, and after reordering it, the quantized coefficients are scaled and inverse transformed to generate residual block data $Dn$. Using the header information in NAL, the decoder selects prediction values using either motion compensation or intra-prediction. The predicted block is added to the residual block to generate unfiltered block data $uFn$ which is filtered by a deblocking filter and stored as reconstructed frame or field.

The computational requirements of H.264 video codec depends on video resolution, frame rate, and compression level. For mobile phone applications, the videos are encoded in the QCIF format (176 x 144) at 15 frames per second (fps). On the other hand, Bluray videos are encoded in 1080p (1920 x 1080) at 60 fps interlaced. The H.264 standard also defines several profiles, which use different compression algorithms. In this paper, we focus on the baseline profile. We study the following algorithms: intra-prediction, deblocking filter, motion compensation - interpolation, and motion estimation because these algorithms contribute the most to the processing time and power consumption.

## III. Wide SIMD Architecture, SODA

In this section, we present a representative SIMD architecture, SODA [1]. The architecture was initially designed to support wireless protocols such as WCDMA and IEEE 802.11a. Since both communication and multimedia processing are supported by today's handsets, SODA is selected as the base architecture in this study.

The SODA multiprocessor architecture is shown in Fig. 2. This system consists of multiple data processing elements (PEs), one control processor, and global scratchpad memory, all connected through a shared bus. Each SODA PE consists of 5 major components: 1) a 32-way, 16-bit datapath SIMD pipeline for supporting vector operations. Each datapath includes a 2 read-port, 1 write-port 16 entry register file, and a 16-bit ALU with multiplier. Intra-processor data movements are supported through the SIMD Shuffle Network (SSN); 2) a

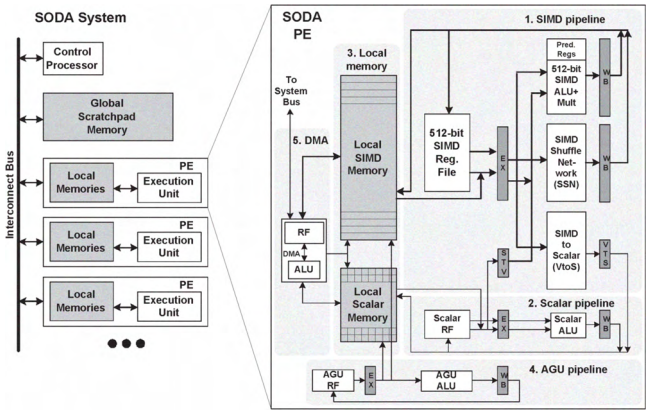


Fig. 2. Wide-SIMD architecture: SODA [1]

16-bit datapath scalar pipeline for sequential operations. The scalar pipeline executes in lock-step with SIMD pipeline with SIMD-to-scalar and scalar-to-SIMD operations to exchange data between two pipelines; 3) two local scratchpad memories for the SIMD pipeline and the scalar pipeline; 4) an AGU (Address-Generation-Unit) pipeline for providing the addresses for local memory accesses; and 5) a programmable DMA (Direct-Memory-Access) unit to transfer data between scratchpad memories and interface with the outside system (inter-processor data transfer). The SIMD pipeline, scalar pipeline and the AGU pipeline execute in VLIW-styled lock-step manner, controlled with one program counter (PC).

## IV. H.264 Algorithm Analysis and Design Decisions

In this section, we analyze key algorithms in H.264 and propose several architectural design decisions to improve the processing performance and power efficiency. This analysis led to the introduction of the following customizing features: 1) multiple SIMD widths 2) diagonal memory organization, 3) bypass and temporary buffer support (partitioned RF), 4) fused operation, and 5) programmable crossbar.

### A. Multiple SIMD Widths

| Algorithm | Kernel Operation | SIMD Workload | SIMD Width | TLP Level |
|---|---|---|---|---|
| Intra-pred (dec.) | 13-tap filter | 75.48 % | 16 | Med. |
| Intra-pred (enc.) | 13-tap filter | 91.06 % | 16 | High |
| Deblocking Filter | 3,4,5-tap filter | 86.61 % | 8 | Med. |
| Interpolation (MC) | 2,4,6-tap filter | 81.59 % | 8 | High |
| Motion Estimation | SAD (16) | 62.46 % | 16 | High |

TABLE I
Kernel operations, SIMD workload, required SIMD width, and the amount of thread level parallelism (TLP) for H.264 encoder/decoder algorithms

Table I shows the workload profiling for the key H.264 kernel algorithms. The other important computational kernels such as transform, quantization, and entropy coding are not included in this study because the transform/quantization kernel is easily parallelizable and is not the performance

bottleneck, and the entropy coding is completely sequential and can be mapped only to a scalar processing unit. The available data level parallelism (DLP) expressed in terms of SIMD workload, natural SIMD width, and the thread level parallelism (TLP) for the key parallel H.264 algorithms are presented in Table I. The SIMD workload consists of the arithmetic and logical computations that can be mapped to the SIMD pipeline. The scalar workload represents the instructions that are not parallelizable such as loop control and address generation, which run on the scalar pipeline and the AGU pipeline respectively. The overhead workload includes all the instructions that support SIMD computations such as SIMD memory operations and memory alignment operations.

As can be seen in Table I, most of the H.264 kernel algorithms can exploit the SIMD datapath, but the required SIMD width varies. While the deblocking filter and interpolation have SIMD width of 8, intra-prediction and motion estimation have a SIMD width of 16. Kernels such as intra-prediction mode decision and motion estimation have high TLP, which means that independent threads corresponding to different macroblocks can be mapped onto the SIMD datapath. For these kernels, the wide-SIMD pipeline helps to increase the processing performance. Kernels such as intra-prediction and deblocking filter are not easily parallelizable, and a wide SIMD width does not guarantee higher performance. Therefore, even though it is easier to design SIMD architectures with a fixed SIMD width, we propose to support multiple SIMD widths to maximize the SIMD utilization.

### B. Diagonal Memory Organization

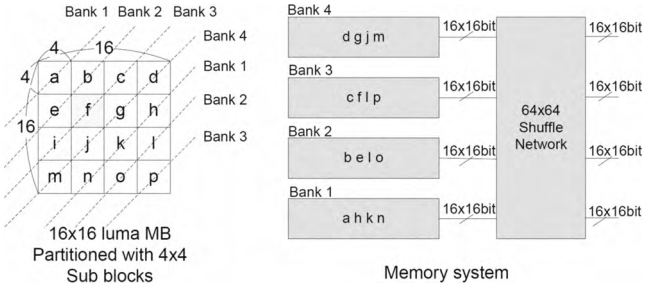


Fig. 3. Diagonal memory organization and shuffle network, which allows the horizontal and vertical memory access without conflict. The 64x64 shuffle network realigns 64 16-bit data.

Multimedia algorithms use two or three dimensional data unlike wireless signal processing algorithms that typically operate on single dimensional data. For example, the deblocking filter algorithm operates on horizontal edges followed by vertical edges. Row or column order memory access works well for one set of edges, but not for the other. A diagonal memory organization is more suitable here since blocks of pixels along a row or column can be accessed with equal ease.

Fig. 3 shows how a 16x16 macroblock is stored in the proposed diagonal memory organization. The 16x16 macroblock is broken into 4x4 sub blocks ($a$, $b$, ... $p$) each containing 16 pixels. Groups of sub blocks $(a, h, k, n)$, $(b, e, i, o)$, $(c, f, i, p)$, and $(d, g, j, m)$ are stored in separate memory banks. This allows neighboring blocks which share horizontal and vertical edges to be accessed at the same time.

### C. Bypass and Temporary Buffer Support

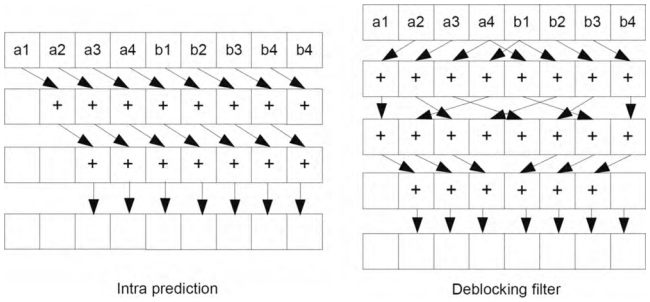


Fig. 4. Subgraphs for the inner loops for two H.264 kernels; The bypass path is not shown for simplicity.

Fig. 4 shows the subgraphs for inner loops of two H.264 kernel algorithms. We see that there exists large amount of data locality. Moreover, intermediate data do not need to be stored in the register file (RF) because the values are usually consumed by the very next instruction and all not used anymore. Thus, it is sufficient to store these values in a temporary buffer or bypass them. These features have been inspired by recent works in [3] and [4], which show that storing short-lived data and bypassing RF reduce the power consumption and increase the performance.

### D. Fused Operation

| Algorithm | Shuffle-ALU | Add-Shift | Sub-Abs | Neg-Add |
|---|---|---|---|---|
| Intra-Pred.(Enc) | 21.43 % | 7.14 % | 28.57 % | - |
| Intra-Pred.(Dec) | 30.77 % | 30.77 % | - | - |
| Deblocking Filter | 49.48 % | 16.49 % | - | - |
| Interpolation(MC) | 30.09 % | 3.76 % | - | 15.05 % |
| Motion Estimation | 24.04 % | - | 48.08 % | - |

TABLE II
INSTRUCTION PAIR FREQUENCY FOR H.264 KERNEL ALGORITHMS

Many operations in DSP algorithms occur in pairs or tuples. The most common example is the multiply followed by accumulate, which has been exploited by many architectures. Table II shows the breakdown of the most frequent instruction pairs of H.264 kernel algorithms. Among all pairs, the shuffle-ALU pair is heavily used because most of the time, data must be aligned before being processed by the SIMD datapath. The frequencies of add-shift and sub-abs pairs are also very high. The sub-abs instruction pair is used in the SAD (Sum of Absolute Differences) operations in motion estimation. The add-shift instruction pair represents the round operation, which is one of the most used operations in H.264 algorithms.

Based on this analysis, we propose to fuse the frequently used instruction pairs. This would increase performance and lower power consumption because unnecessary RF access can be significantly reduced.

### E. Programmable Crossbar

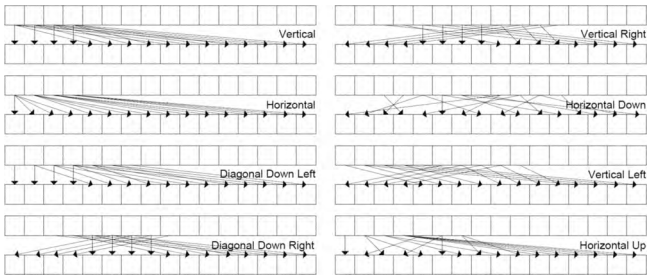


Fig. 5. Permutation Patterns for H.264 Intra-prediction Modes

Fig. 5 shows some examples of the SIMD permutation patterns that are found in H.264 intra-prediction algorithm. Even though the permutation patterns look very random, each H.264 algorithm - intra-prediction, deblocking filter, interpolation, and motion estimation - has a predefined set of shuffle patterns, and the number of distinct sets is typically less than 16.

Most commercial DSP processors and GPP multimedia extensions support some types of data permutations. These features are even more important in SIMD architectures for aligning data before the SIMD computation units. For instance, the perfect shuffle network in SODA [1] supports a few sets of permutations in one clock cycle. But, if complex permutation patterns are required, multiple instructions need to be executed. These additional clock cycles degrade the timing and power performance. To support complex data access patterns in H.264 algorithms, we propose small low-power programmable fixed pattern crossbars. We place one of these between memory and register file to align data before loading and storing, and another between the register file and SIMD functional units to shuffle data before processing.

## V. Proposed Architecture

In this section, we describe the customized wide-SIMD architecture which includes the features proposed in Section IV. Features such as configurable SIMD datapath, temporary buffer, bypass network and SRAM-based crossbar have also been incorporated in our recent architecture, AnySP [19]. The design of the functional unit and the multibank memory structure is, however, special to the proposed architecture.

### A. PE Architecture

Fig. 6 shows the proposed PE architecture. It is similar to SODA in that it consists of a SIMD pipeline, a scalar pipeline, and an AGU pipeline. The SIMD datapath consists of four groups of 16-wide SIMD units that can be functioned as eight groups of 8-wide, two groups of 32-wide or one 64-wide SIMD datapath. Each 16-wide 16-bit SIMD datapath consists of 16-wide 16-entry RF, 16 functional units (FUs) supporting fused instructions, partitioned 16-wide 4-entry RF (temporary buffer) and an adder tree that supports the summation of 2,4,8, and 16 elements. The 16-wide SIMD partitions are glued by multi-SIMD partition shuffle network and data within each 16-wide SIMD units can be shuffled using predefined shuffle

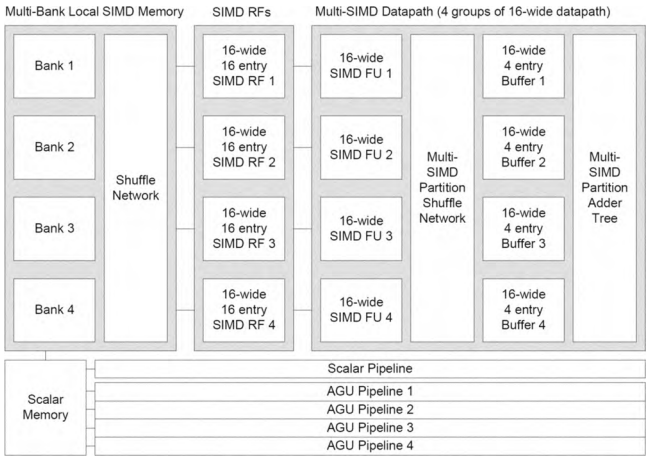


Fig. 6. PE architecture consists of multi-bank local SIMD memory, SIMD RFs, multi-SIMD datapath, scalar pipeline, four AGU pipelines dedicated to four 16-wide SIMD partitions, and DMA (not shown here)

patterns by a programmable crossbar. Also, multi-SIMD partition adder tree supports the function of the summation of 32 and 64 elements.

The local memory consists of four memory banks; each bank is 16-wide 16-bit 256-entries (8KB). The four AGU pipelines work for four local memory banks. The scalar and AGU pipeline share the same SIMD local memory using a scalar memory buffer which can be accessed sequentially. AGU pipeline also functions as scalar pipeline for each SIMD datapath. Details of these architectural features are described in the rest of this section.

### B. SIMD Partitioning

As described in Section IV-A, H.264 kernel algorithms have different natural vector widths. When the processor's SIMD width is smaller than the natural vector width, the performance drops because the natural vector has to be split into many small vectors and handling these vectors requires additional work. On the other hand, if the processor's SIMD width is larger than the natural vector width, some of the SIMD lanes are idle, thereby wasting power. Therefore, multiple SIMD partitioning is chosen to support both small SIMD-width algorithms having a large amount of TLP and large SIMD-width algorithms having little TLP.

As can be seen in Fig. 6, a 64-wide SIMD datapath is broken into four groups of 16-wide SIMD datapath units. This can be further broken into eight groups of 8-wide SIMD units. Each 16-wide SIMD datapath can be combined to exploit more data parallelism such as 32-wide and 64-wide with the support of the multi-lane shuffle network.

### C. SIMD Functional Units

Fig. 7 shows the 16-wide SIMD functional unit, which consists of a 32x32 shuffle network, a functional unit (multiplier, ALU, simple adder/subtractor) and a 16-wide adder tree. The shuffle network supports any permutation pattern using two 16-wide vectors. This shuffle network also stores

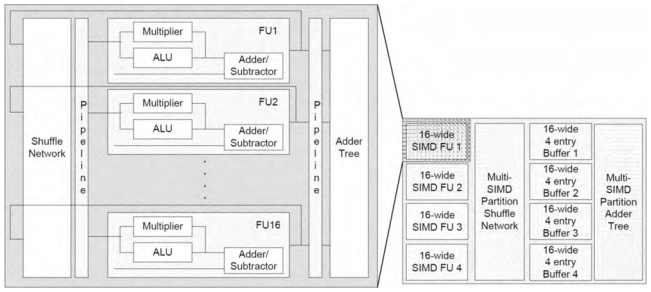


Fig. 7. 16-wide SIMD Functional Unit

a small number of shuffle patterns in the module to support fast permutation between 16 functional units. The functional units support instruction pairs such as multiplier-add and add-shift described in Section IV-D. A 16-wide adder supports the sum of 2, 4, 8, and 16 elements. The other characteristic of the functional unit is support of saturation arithmetic. For 2's complement signed 8-bit data, the results of the arithmetic units are saturated to +127 and -128, and for unsigned data, to 255 and 0. This saturation feature is very important for operations in the deblocking filter kernel.

### D. Temporary Buffer and Bypass Support

To alleviate the problem of high power consumption of register files (RFs), two techniques are applied: temporary buffer (partitioned RF) and data bypass network support. Each SIMD lane has a 4-entry temporary buffer that stores intermediate data (short-lived values) to decrease the amount of main RF accesses. This small RF consumes less power than the main RF and also helps to reduce register pressure of the main RF. Typical writeback stage is modified to support data forwarding bypass by explicitly directed instructions. Instructions dictate functional units where to fetch data (from main RFs, from temporary buffers or from bypassed data).

### E. Multi SIMD Partition Shuffle Network

Due to data access complexity in H.264 algorithms and proposed memory system, data needs to be shuffled within a SIMD partition or between SIMD partitions. The multi-SIMD partition shuffle network is placed next to four groups of 16-wide SIMD functional units to support data transfer between SIMD partitions. This large shuffle network also allows the processor to function as four SIMD pipelines connected in serial. This feature is useful when a signal processing algorithm have little TLP.

### F. Multiple Output Adder Tree Support

In some H.264 algorithms, the operation of wide vector inner sum ($s = v[0] + v[1] + ... + v[N-1]$) occurs frequently. Examples of this operation are matrix multiplication operation of DCT and SAD calculation for motion estimation. Though H.264 algorithms usually require the sum of 2, 4, 8, and 16 pixel values, the 64-wide multiple SIMD partition adder tree supports other output possibilities such as 32 and 64. The multiple outputs are stored back into temporary buffers and written back to the main RFs if necessary.

## VI. Mapping of H.264 Kernels

In this section, we describe how the main H.264 kernels are mapped onto the proposed architecture.

### A. Intra Prediction

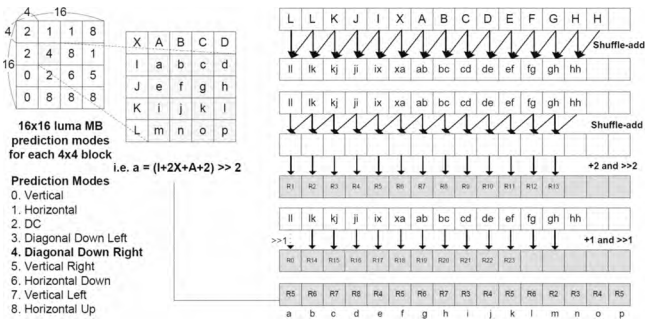


Fig. 8. Mapping a 16x16 luma macroblock intra-prediction process on the proposed architecture. Example of the Diagonal Down Right intra-prediction for a 4x4 sub block (grey block) is presented with fused operations.

In H.264 intra-prediction, there are nine prediction modes - Vertical, Horizontal, DC, Diagonal Down Left, Diagonal Down Right, Vertical Right, Horizontal Down, Vertical Left, and Horizontal Up. A 16x16 luma macroblock is broken into sixteen 4x4 sub blocks. The 16 prediction values $(a, b, ..., p)$ for each 4x4 sub block is calculated with neighboring pixels $(A, B, C, D, I, J, K, L, X)$ using 16 SIMD lanes. At the encoder, all the prediction modes are calculated and the best predicted one is chosen. At the decoder, the sub block is generated based on the prediction mode in the header information sent by the encoder.

| Prediction Mode | Shuffle Pattern |
|---|---|
| Diagonal Down Left | 7,8,9,10,8,9,10,11,9,10,11,12,10,11,12,13 |
| Diagonal Down Right | 5,6,7,8,4,5,6,7,3,4,5,6,2,3,4,5 |
| Vertical Right | 18,19,20,21,5,6,7,8,4,18,19,20,3,5,6,7 |
| Vertical Left | 19,20,21,22,7,8,9,10,20,21,22,23,8,9,10,11 |
| Horizontal Down | 17,5,6,7,16,4,17,5,15,3,16,4,14,2,15,3 |
| Horizontal Up | 16,3,15,2,15,2,14,1,14,1,0,0,0,0,0,0 |

TABLE III
SHUFFLE PATTERNS FOR SIX INTRA PREDICTION MODES FOR 4X4 LUMA

There is significant overlap in the computations of six of the modes. The other three modes, namely, Horizontal, Vertical, and DC mode, are computed using only a crossbar and an adder tree. Fig. 8 shows how to compute the partial intermediate values that are reused for the six prediction modes. 16 SIMD lanes are used to generate two sets of partial sums for a 4x4 sub block with fused operations such as shuffle-add and add-shift. After generating R0 to R23, these intermediate values are distributed to the 16 SIMD lanes by a shuffle network. Table III shows how to shuffle the partial sums for each prediction mode. The use of partial sums results in significant reduction in the number of instruction cycles in the encoder. The intra-prediction calculations in the encoder are very parallel and four groups of 16-wide SIMD datapath

can be utilized in parallel. However, in the decoder, there are dependencies in the processing order. For example, in Fig. 9, the $A6$ macroblock requires $A1$, $A2$, $A3$, and $A5$ macroblocks to be predicted first. Fig. 9 shows a processing order in which four macroblocks are processed at the same time, thereby utilizing all SIMD lanes.

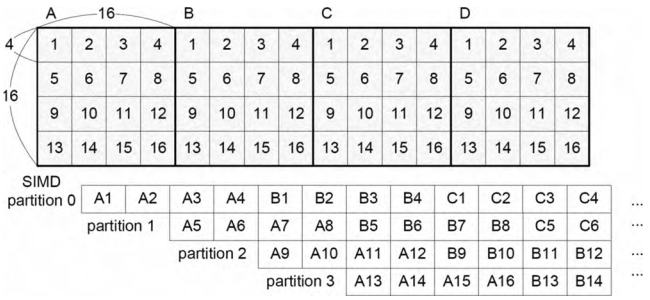


Fig. 9. Mapping macroblocks into SIMD partitions such that all SIMD lanes are utilized

## B. Deblocking Filter

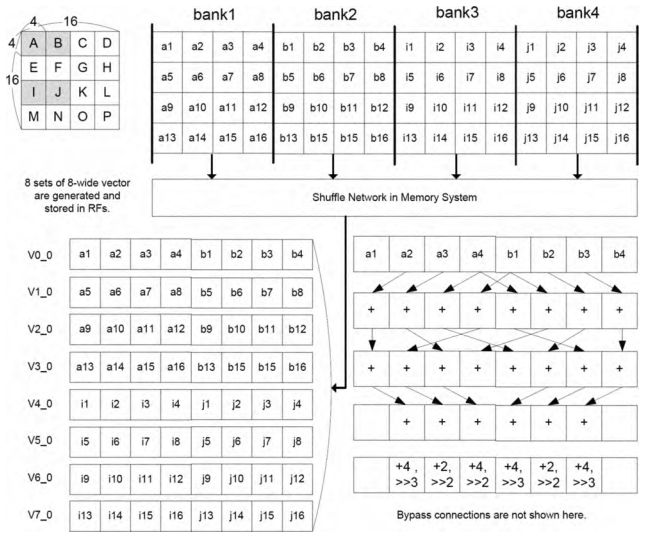


Fig. 10. Mapping a deblocking filter process when BS (Block Strength)=4.

H.264 deblocking filter smoothes the block edges to reduce blocking distortion without affecting the real edges. Based on block strength, the function of this filter varies dynamically (three-tap, four-tap, or five-tap filter). Furthermore, there is an order in which the edges have to be filtered. Fig. 10 shows how deblocking filter process is mapped on the SIMD pipeline. To utilize all SIMD lanes, edges A-B, E-F, I-J, M-N are filtered in parallel. To avoid memory access conflict, sub blocks A,B,I,J (which belong to four different sub banks) are loaded first, followed by E,F,M,N, etc. The four groups of 16 pixel values are permuted by a shuffle network in the memory system to generates eight groups of horizontally aligned eight pixel values. Each SIMD partition exploits fused shuffle-add operations followed by round operations to produce filtered pixel values.

## C. Motion Compensation

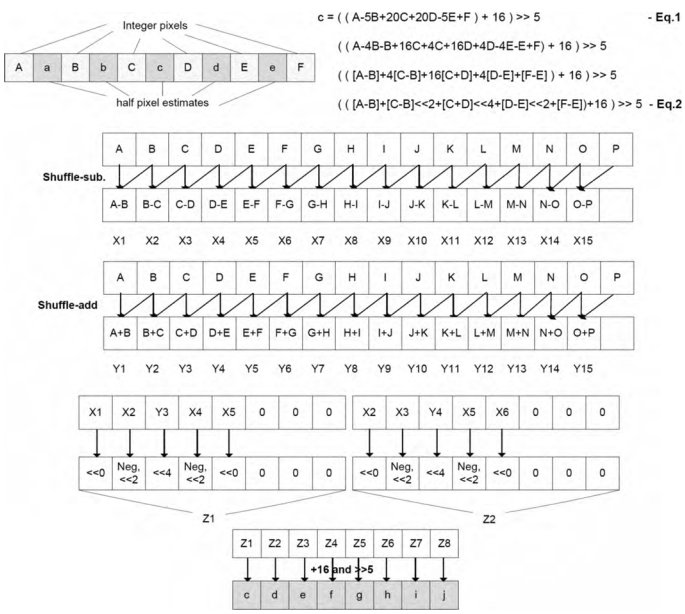


Fig. 11. Example of interpolation of motion compensation (half-pel).

In H.264, the size of the motion compensation block can be 16x16, 16x8, 8x16, 8x8, 4x8, and 4x4, and the resolution can be integer-pixel, half-pixel, quarter-pixel, or eighth-pixel. Because sub-sample positions do not exist in the reference frames, the fractional pixel data are created by interpolation. Half-pixel interpolations are derived by a six tap filter as shown in Eq.1 in Fig. 11. The equation is modified to reduce multiplications and to express the six tap filter in terms of partial sums and differences of the original pixel values. This helps in exploiting the re-usability of computations for subsequent half pixel interpolations (Eq.2) in Fig. 11. As can be seen in Fig. 11, the first row of a 16x16 block is loaded to SIMD RFs by using a shuffle network, and the partial sums and differences are stored in temporary registers. A subset of these values are shuffled and summed with an adder tree to obtain the half-pixel estimate. Eight groups of 8-wide SIMD datapath handle the interpolation process for each row. Once the half pixel estimates have been calculated for a particular row, we can use them to compute the quarter pixel values.

## D. Motion Estimation

Motion estimation of an $M$x$N$ block involves finding an $M$x$N$ sample region in a reference frame that closely matches the current block. An area in the reference frame of size $2M$x$2N$ centered on the current block position is searched, and the minimum SAD value is needed to determine the best match. Fig. 12 shows the mapping method for a 4x4 block (current frame) in an 8x8 search area in the reference frame. The pixels of the current 4x4 block $(a, b, c, ..., p)$

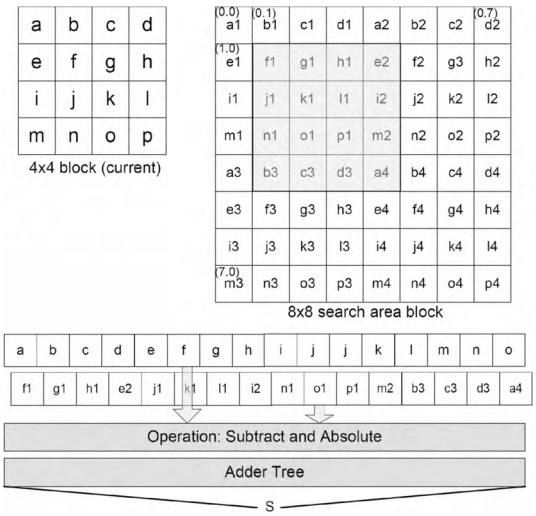


Fig. 12. Mapping a motion estimation process for a 4x4 block on the proposed architecture; The search area is 8x8.

are loaded from the memory to a SIMD register, and the pixels in the shaded 4x4 block $(f1, g1, h1, e2, j1, ..., a4)$ in the search area are obtained using memory loads and shuffles. The SAD value is calculated by a fused operation (sub-abs) and summation using the adder tree. The first SAD value is stored as the minimum SAD and is updated during subsequent computations. This process repeats for 25 possible positions in the 8x8 search area. The motion estimation process is highly parallel and four groups of 16 SIMD lanes are utilized to generate four SAD values at a time.

## VII. Results and Analysis

### A. Methodology

| | Components | Units | Area | | H.264 Decoder | |
|---|---|---|---|---|---|---|
| | | | Area $mm^2$ | Area % | Power mW | Power % |
| PE | SIMD Data Mem (32KB) | 2 | 4.88 | 34.15% | 5.14 | 6.29% |
| | SIMD Register File (16x1024bit) | 2 | 1.59 | 11.13% | 14.95 | 18.27% |
| | SIMD ALUs, Multipliers, and SSN | 2 | 2.25 | 15.75% | 22.43 | 27.41% |
| | SIMD Pipeline+Clock+Routing | 2 | 0.59 | 4.13% | 11.68 | 14.28% |
| | SIMD Buffer (128B) | 2 | 0.41 | 2.87% | 1.70 | 2.08% |
| | SIMD Adder Tree | 2 | 0.09 | 0.63% | 0.52 | 0.64% |
| | Intra-processor Interconnect | 2 | 0.47 | 3.29% | 4.67 | 5.71% |
| | Scalar/AGU Pipeline & Misc. | 2 | 0.61 | 4.27% | 6.72 | 8.21% |
| System | ARM (Cortex-M3) | 1 | 0.6 | 4.20% | 2.5 | 3.05% |
| | Global Scratchpad Memory (128KB) | 1 | 1.8 | 12.5% | 10 | 12.22% |
| | Inter-processor Bus with DMA | 1 | 1.0 | 7.00% | 1.5 | 1.83% |
| Total | 90nm (1V @300MHz) | | **14.29** | 100% | **81.81** | 100% |
| Est. | 65nm (1V @ 300MHz) | | **7.46** | | **25.76** | |
| | 45nm (1V @ 300MHz) | | **3.89** | | **20.36** | |

TABLE IV
Summary of Area and Power Running H.264 CIF video at 30fps

The RTL Verilog model of SODA processor [1] was synthesized in TSMC 180nm technology, and the power and area results for 90nm technology were estimated using a quadratic scaling factor based on Predictive Technology Model [5]. The proposed architecture was implemented in the RTL Verilog model and synthesized in TSMC 90nm using Synopsys physical compiler. The PE area is 25% larger than SODA's estimated 90nm PE area. The clock frequency is targeted for 300MHz, while SODA was targeted for 400MHz. The area and power breakdown of this architecture running H.264 CIF video at 30fps are presented in Table IV. This video encoder consumes about 81mW at 90nm, which is within the requirements for mobile video.

### B. Results

Fig. 13 shows the speedup of proposed architecture over SODA for the H.264 kernel algorithms. The improvement is broken into several architectural enhancements: wider SIMD width (from 32 to 64), fused operation, buffer+bypass support, and single cycle programmable crossbar. The wider SIMD width allows H.264 algorithms to operate on twice as many pixels and results in 72% of performance improvement over SODA. The fast programmable crossbars expedites the data alignment process, and accounts for another 25% improvement. The fused operations and buffer+bypass support also helps to boost the speed by about 16%. The energy-delay product for the H.264 kernel algorithms are presented in Fig. 14. On average, there is a 29% of energy-delay improvement due to lower clock frequency, reduced memory and register file access supported by crossbar, fused operation, and buffer+bypass support.

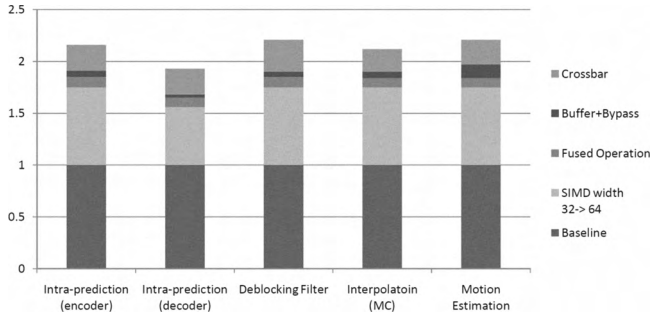


Fig. 13. Speedup over SODA for the key H.264 algorithms. The improvements are broken down into several architectural enhancements - wider SIMD width, fused operation, buffer+bypass support and fast programmable crossbar.

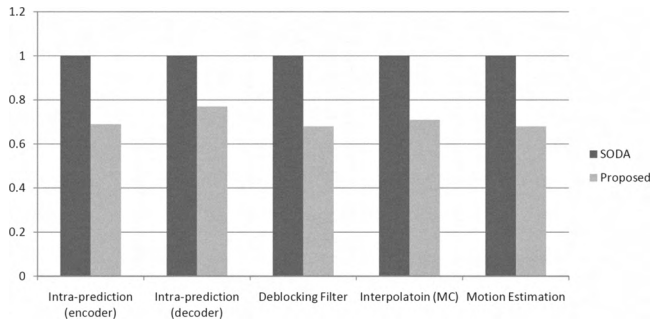


Fig. 14. Normalized Energy-Delay Product for H.264 kernel algorithms compared to SODA.

Table V compares the power performance of our architecture with state-of-the-art designs for H.264 baseline encoding. We use power consumption per pixel/sec (mW/(Mpix/s)) as the metric. Although the ASIC solution [17] outperforms the

programmable solutions, our proposed architecture has programmable flexibility and consumes significantly less power compared to TI's DSP solution.

| | **ISSCC2007** [17] | **TMS320DM6446** C64x+ DSP [18] | **This work** 2 PEs |
|---|---|---|---|
| Resolution | 720x480 | 720x480 | 352x288 |
| Technology | 130nm | 90nm | 90nm |
| Supply Voltage | 0.9V | 1.2V | 1.0V |
| Clock Freq. | 30MHz | 594MHz | 300MHz |
| Power consumption | 27mW | 415mW | 68mW |
| Power efficiency (mW/(Mpixel/sec)) | 2.6 | 40 | 22 |

TABLE V
COMPARISON WITH STATE-OF-THE ART H.264 ENCODERS.

## VIII. Related Work

There have been several architectural solutions for H.264/AVC. Many of them are specialized architectures for key kernels such as motion estimation, motion compensation [7], [9], interpolation [8] and deblocking [10], [11]. An important consideration in all these architectures is efficient memory access. For instance the deblocking filter architectures reduce the number of memory accesses by manipulating data stored in shift registers in [10] and using vector registers and VLIW processing in [11]. Reducing the overhead in memory accesses and data alignment in multimedia processing has been addressed in systems such as MediaBreeze [6] by adding hardware support for address generation, looping etc.

Efficient techniques for mapping H.264 onto multiprocessor platforms have been proposed in [12], [13], [14], [15]. While [12] focused on efficient partitioning of data, [13] proposed a high speed multithreading implementation of the H.264 video encoder. The implementation in [14] focused on efficient scheduling and memory hierarchy for the H.264 video encoder for HDTV applications. A hybrid task pipelining scheme which greatly reduced the internal memory size and bandwidth was presented in [15].

Recently a FGPA based architecture, Video Specific Instruction Set Processor, has been proposed in [16]. The architecture consists of hardware accelerators for inter prediction and entropy coding, and specialized instructions for a programmable processor for the rest of the kernels.

## IX. Conclusion

The mobile multimedia processor requires high-performance low-power solutions for high quality video and wireless protocols. General purpose processors, digital signal processors and ASICs are typically combined to meet this requirement. Such a heterogeneous solution is inefficient in terms of area, power, and flexibility. In this paper, we presented a software-hardware co-design case study of H.264 codec for a wide-SIMD architecture. Based on the characteristics of H.264 kernel algorithms, we proposed several key architectural enhancements including SIMD partitioning, diagonal memory organization system, bypass and temporary buffer support, fused operation support, and area and energy efficient programmable crossbar use. Our results show that we can achieve 2.13x speedup and 29% energy-delay improvement for the H.264 codec over a wide-SIMD architecture, SODA.

## Acknowledgment

This research is supported by ARM Ltd. and the National Science Foundation under grants CNS-0615261, CNS-0615135, and CCF-0347411.

## References

[1] Y. Lin et.al, "Soda: A low-power architecture for software radio," *Proc. of the 33rd Annual International Symposium on Computer Architecture*, pp. 89–100, June 2005.

[2] I. Richardson, "H.264 and MPEG-4 video compression," *WILEY*, 2003.

[3] N. Goel, A. Kumar, and P. Panda, "Power reduction in VLIW processor with compiler driven bypass network," *Proc. of the 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems*, pp. 233–238, Jan. 2007.

[4] K. Fan et.al, "Systematic register bypass customization for application-specific processors," *Proc. of IEEE 14th International Conference on Application-Specific Systems, Architectures, and Processors*, pp. 64–74, June 2003.

[5] Nanoscale Integration and Modeling Group, "Predictive technology model." *http://www.eas.asu.edu/ nimo/*

[6] D. Talla, L. John, and D. Burger, "Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements," *IEEE Transactions on Computers*, vol. 52, no. 8, pp. 1015–1031, Aug. 2003.

[7] R. Wang, J. Li, and C. Huang, "Motion compensation memory access optimization strategies for H.264/AVC decoder," *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp. v97–v100, Mar. 2005.

[8] R. Wang et.al, "High throughput and low memory access sub-pixel interpolation architecture for H.264/AVC HDTV decoder," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 3, pp. 1006–1013, Aug. 2005.

[9] S.-Z. Wang et.al, "A new motion compensation design for H.264/AVC decoder," *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 4558–4561 Vol. 5, May 2005.

[10] C.-M. Chen and C.-H. Chen, "Configurable VLSI architecture for deblocking filter in H.264/AVC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 1072–1082, Aug. 2008.

[11] P. Dang, "High performance architecture of an application specific processor for the H.264 deblocking filter," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, no. 10, pp. 1321–1334, Oct. 2008.

[12] R. G. E van der Tol, E Jaspers, "Mapping of H.264 decoding on a multiprocessor architecture," *Proc. of SPIE Conference on Image and Video Communications and Processing*, pp. 707–718, Jan. 2003.

[13] E. Q. Li and Y.-K. Chen, "Implementation of H.264 encoder on general-purpose processors with hyper-threading technology," *Proc. of SPIE Conference on Visual Communications and Image Processing*, vol. 5308, pp. 384–395, Jan. 2004.

[14] T.-C. Chen et.al, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 6, pp. 673–688, June 2006.

[15] T.-C. Chen, C.-J. Lian, and L.-G. Chen, "Hardware architecture design of an H.264/AVC video codec," *Proc. of Asia and South Pacific Conference on Design Automation*, Jan. 2006.

[16] S. D. Kim et.al, "ASIP approach for implementation of H.264/AVC," *Journal of Signal Processing Systems*, vol. 50, no. 1, pp. 53–67, Jan. 2008.

[17] T. C. Chen et.al, "2.8 to 62.7 mW low-power and power-aware H.264 encoder for mobile applications," *2007 IEEE Symposium on VLSI Circuits*, pp. 222–223, June 2007.

[18] M. Bhatnagar, "TMS320DM6446/3 Power Consumption Summary," *Texas Instruments Application Reports, http://focus.ti.com/lit/an/spraad6a/spraad6a.pdf*, Feb. 2008.

[19] M. Woh et.al, "AnySP: Anytime Anywhere Anyway Signal Processing," *Proc. of the 36th Annual International Symposium on Computer Architecture*, June 2009.