A Fine-grained Parallel Implementation of a H.264/AVC Encoder on a 167-processor Computational Platform

Zhibin Xiao, Stephen Le and Bevan Baas

Department of Electrical and Computer Engineering, University of California, Davis

***Abstract*— The emerging many-core architecture provides a flexible solution for the rapid evolving multimedia applications demanding both high performance and high energy-efficiency. However, developing parallel multimedia applications that can efficiently harness and utilize many-core architectures is the key challenge for scalable computing. We contribute to this challenge by presenting a fully-parallel H.264/AVC baseline encoder on a 167-core asynchronous array of simple processors(AsAP) computation platform. By exploiting fine-grained data and task level parallelism in the algorithms, we partition and map the dataflow of the H.264/AVC encoder to an array of 115 small processors coupled with two shared memories and a hardware accelerator for motion estimation. The proposed parallel H.264/AVC encoder is capable of encoding video sequences with variable frame sizes. The encoder presented is capable of encoding VGA (640 x 480) video at 21 frames per second (fps) with 931 mW average power consumption by adjusting each processor to workload-based optimal clock frequencies and dual supply voltages with less than 1dB loss in resolution.**

I. Introduction

In the past decades, multimedia systems evolve with the rapid development of VLSI technologies. More and more complicated image and video algorithms become feasible by upgrading the underlying hardware using newer process technology. Traditional video encoding architectures appear in three forms: application-specific processors, multimedia extensions to general-purpose processors, and multimedia co-processors. However, none of these methods achieve both high performance and flexibility for emerging multimedia standards. The 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 which results in high computational complexity. The emerging many-core approach has proven to be a feasible solution for real-time H.264/AVC video encoding. However, how to map such complex applications as H.264/AVC to many-core processors is challenging. Many coarse-grained parallel many-core approaches have been proposed for H.264/AVC encoding. Most of them exploit thread-level or frame-level parallelism in video encoding algorithms [3], [4], [5], [6].

In this paper, we propose an on-chip distributed processing approach to parallelize the H.264/AVC baseline encoder at the macroblock (16x16) and 4x4 sub-macroblock level on a fine-grained many-core platform. The proposed fine-grained parallelization exploits the existing locality and streaming nature of H.264/AVC encoding algorithms. Our work differs from previous research in that we apply a more fine-grained approach to exploit task-level parallelism in H.264/AVC encoding. We also take advantage of the globally-asynchronous locally-synchronous (GALS) and per-processor voltage and frequency scaling features of the target many-core system to further reduce the power consumption.

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. Section III presents the task-level parallelization approach to the H.264/AVC encoder. Section IV shows the performance analysis and results. Section V concludes the paper.

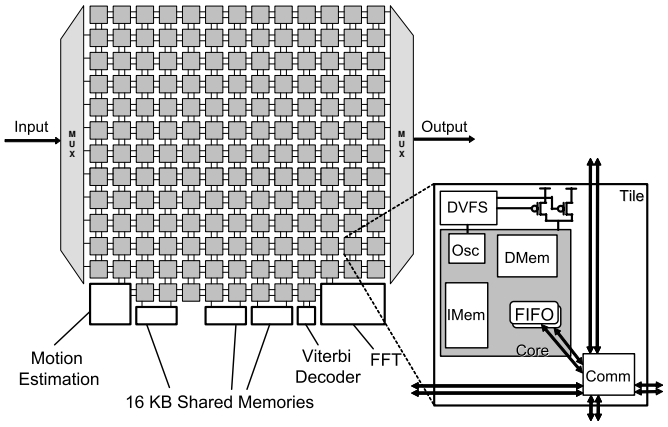


Fig. 1. Architecture of targeted many-core system.

II. The target platform and parallel programming methodology

A. The target AsAP platform

The target AsAP (Asynchronous Array of Simple Processors) architecture is a fine-grained many-core system which is composed of simple cores with small memories for high energy efficiency. Target applications of AsAP include multimedia and communication algorithms which can be partitioned into small tasks running separately on small and simple processors. As shown in Fig. 1, the system is composed of 164 16-bit homogeneous DSP processors, three dedicated hardware accelerators (Viterbi decoder, FFT and video motion estimator, and three 16-KB integrated shared memories, all of which have local oscillators and are connected by a reconfigurable mesh network [7].

1) The AsAP processor: Some of the key features of the AsAP processors are listed as follows:

- each processor is small, containing only 128-word of instruction and 128-word of data memory.
- the instruction set of the simple programmable processors adheres to a simple one-destination and two-source architecture. The two source operands are from either local data memory or two input buffers connected with neighboring or far-away processors.
- each processor has two sets of communication links in each direction for nearest-neighbor communication; long-distance communication can be statically configured without interrupting the processors in the middle.
- each processor has two input port with dual-clock FIFO.

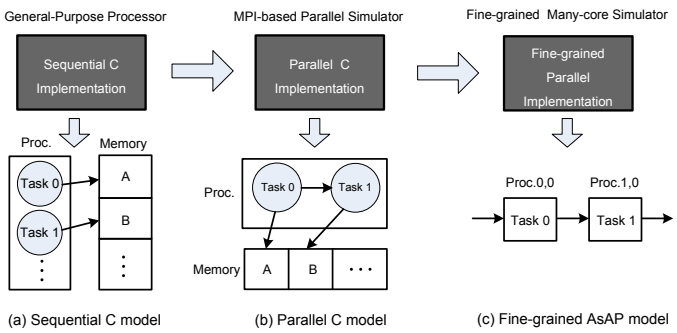


Fig. 2. There execution model of fine-grained program parallelization methodology

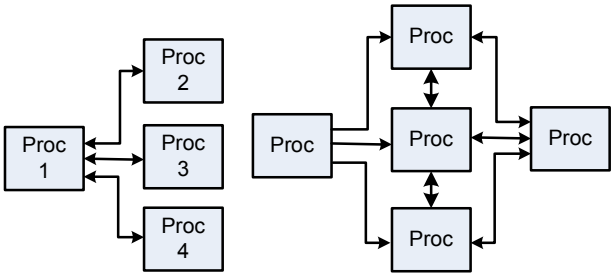


Fig. 3. Dataflow cases with three or more inputs per processor (a) simple case without feedback path (b) complex case with feedback path

- each processor's voltage and frequency can be dynamically scaled to achieve highest energy efficiency.

2) The Motion Estimation Accelerator: Motion estimation is a highly computational task and has been implemented as configurable hardware accelerator in our design. The motion estimation accelerator (ME_ACC) allows for communication with two neighboring AsAP processors for control. The controlling processor sets up the ME_ACC to perform SAD (Sum of Absolute Difference) computation by programming various registers and controls the ME_ACC to start, continue or stop the MV search. The ME_ACC performs SAD computation at the given search position and increments the search position index automatically. The search area and pattern in the ME_ACC are user defined allowing for different types of searches such as full search, 4-step, diamond as well as any custom ones. The accelerator currently allows for 4 sets of search patterns, each with 64 different programmable search locations.

B. Parallel mapping and programming methodology

Fig. 2 shows the parallel programming methodology for the proposed video encoder. The methodology is divided into three steps. We first implemented a bit-level verified sequential C video encoder, which uses a traditional shared memory model on general-purpose processors as shown in Fig. 2(a). Then the sequential algorithm is partitioned into multiple parallel tasks which are implemented with simple C programs separately as shown in Fig. 2(b). The H.264/AVC encoder can be divided into many tasks which can be combined by linking their inputs and outputs using a GUI-based mapping tool. We have developed a Linux-based parallel simulator based on message passing interface (MPI) library to verify the parallel C implementation. At the third step, the coarse-grained tasks are repartitioned to fit on the resource-constrained fine-grain parallel AsAP processors as shown in Fig. 2(c). By using the activity profile of the processors reported by the simulator, we evaluate its throughput and power consumption. This distributed processing approach is suitable for video applications with streaming features so that large shared memories are avoided and each processor can work on its own piece of data.

C. Programming constraints of AsAP platform

Three main differences in programming AsAP versus other chips or using MPI are the size of the data and instruction memory available and the number of input buffers per processor.

1) Data Memory: Video encoding is a highly memory-intensive application. Since each processor occupies 128 16-bit words of data memory, even if one macro block data is packed, it would not fit onto a single processor and would have to be split into at least two, with the luma data packed (two pixels per word) into one processor and the chroma data into another processor. Some processors are used solely for memory purposes, which would have to be separated from the computational processors. Multiple memory processors can be connected in a loop to form a FIFO like buffer.

2) Instruction Memory: The small instruction memory available for each processor is fairly adequate for simple tasks. However, programs need to be splitted up into smaller blocks for computationally-intensive tasks. This creates more parallelism if programs can be broken up in such a manner that the smaller blocks can be executed at the same time. The challenge is to find good breaking points in the programs where branching off to another processor would require little overhead because certain control information and data would be needed by both/multiple processors.

3) Limited Input Port: The fine-grained AsAP platform has limited number of inputs to both the chip and each individual processors. The AsAP chip has one external input and output for off chip communication. Due to the limited size of on-chip memories, the current and reference frames are stored off chip, when a processor requests a macroblock, it sends a request signal to off-chip. The request signals and encoded video output must share the same I/O port, requiring that control bits be sent to off-chip for determining where each output should be routed. In order to save buffer size, each processor only has two 64-word input FIFOs. Because of the limited instruction memory, many of the modules need to be broken up to smaller tasks, and at some later point combined again to re-construct the data, which creates input port congestions. As shown in Fig. 3, there are two cases in which the processors in the H.264 encoder requires there or more input ports. Fig. 3(a) shows a simple case without feedback path in the data flow. Fig. 3(b) shows a complicate case with feedback path. Generally, both cases can be solved by using processors for routing purpose to combine data from multiple sources or distribute data to multiple destination.

III. Fine-grained task-level parallelization of H.264/AVC Encoder

Fig. 4 shows the proposed H.264/AVC baseline encoder block diagram. The motion estimation is implemented with dedicated hardware motion estimator which supports several programmable search patterns and all H.264-specified block sizes. As shown in Fig. 4, an input frame is processed in units of macro-block which is composed of 16x16 luma pixels and 8x8x2 chroma pixels. This type of block-based video compression are very suited for the fine-grained many-core systems which exploit fine-grained task-level parallelism. An ideal data-flow application pass data among processors in a streaming style. However, data-dependencies and conditional execution complicate the data-flow control of H.264 encoding, thus requiring large memories for storing temporary data.

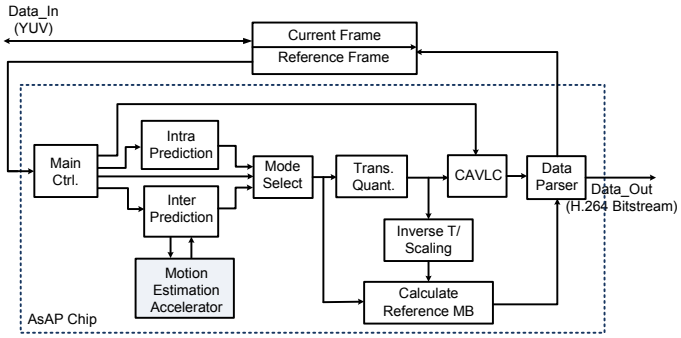


Fig. 4. H.264/AVC encoder block diagram

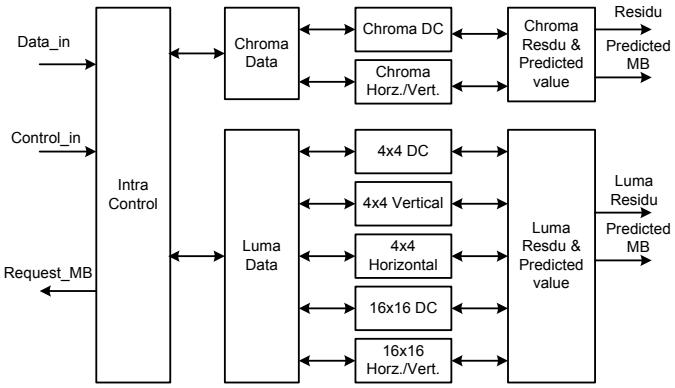


Fig. 5. H.264 intra-prediction data-flow diagram

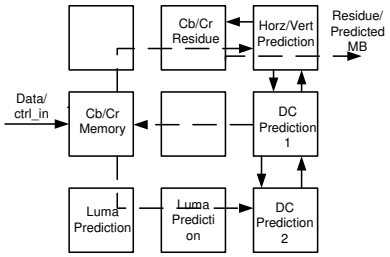


Fig. 6. Intra chroma prediction mapping on AsAP

A. The overview of H.264 encoder parallelization on AsAP

1) Memory organization: Three memory intensive tasks in the H.264 encoding are the current/reference frame management, motion vector management and non-zero coefficient management in entropy encoding. They arise from the fact that the encoding is based not only on the current macro-block but also previously encoded ones.

As shown in Fig. 4, the current/reference frame is stored off-chip, which allows the proposed encoder support flexible frame size. The off-chip memory is divided into three banks which holds the current encoding frame, the reconstructed current frame and the previously reconstructed frame in macro-block order with luma data followed by chroma data.

As macro-block are processed in raster scan order, a large memory is needed to store the motion vectors of the top and left blocks for motion vector prediction of current block and the number of the nonzero coefficients of those data-dependent blocks for CAVLC encoding. The H.264 standard supports sub-partitions of blocks for inter prediction, with two motion vectors per block this becomes a possible maximum of 32 motion vectors when using the smallest partition size(16 4x4 blocks). For motion vector prediction the preceding row of macro-block motion vectors must be saved. A maximum of 3840-word memory is required for the 1080p resolution. Similar to motion vectors, the number of non-zero coefficients has to be predicted in the CAVLC using the top and left block data. Because the CAVLC process is performed on 4x4 blocks, at least 4x120 word memory space has to be reserved for a frame of 1080p resolution requiring the use of on-chip shared memory.

2) Data-flow control: One of the greatest challenges of partitioning a program over such a large area is controlling the flow of data between processors. Ensuring that data is present when needed, and buffered when un-used is vital in preventing dead lock. Since video encoding is done on a macro-block basis, for intra prediction this requires each macroblock to go through the intra prediction process, integer transform, quantization, scaling, inverse transform, and reconstruction before the next macro-block can be predicted. At each step proper control information must be present to ensure accuracy. The chroma prediction process is much faster than luma prediction and the predicted value used must be buffered prior to being sent to the reconstruction blocks to prevent a dead lock situation at the integer transform. Basic macroblock and frame information is also sent along each stage to ensure accuracy and increase code reuse. Parameters such as frame width, frame height, macro block width, macro block number, encoding mode (intra/inter) and block mode are used at nearly every stage and transmitted to save limited size of instruction memory. Many processors can start some initial computation without all of the current data being present, this however requires that the control information be broadcasted to many processors via long distance interconnects creating an additional mapping issue.

B. Detailed parallelization of H.264/AVC encoder

The major encoding blocks of H.264 baseline encoder include intra-prediction, inter-prediction, integer transform, quantization and CAVLC encoding. Due to space limit, we only give a brief illustration of parallel mapping of inter-prediction and intra-prediction on AsAP. The detailed illustration of the CAVLC encoder can be found in [8].

1) Intra Prediction: As mentioned before, the H.264 intra-prediction introduces dependencies between current macro-block and left, top and top right macroblocks. The proposed intra-predictor on AsAP supports 5 prediction modes for luma and 3 prediction modes for chroma, which reduce the dependencies between the current macroblock and the top right macroblock. The intra prediction process constitutes a rather large amount of computation. Fig. 5 shows a high level block diagram for the intra prediction module. Data_in and control_in contain information for the current macroblock being predicted, the request_MB signal is for requesting neighboring macroblock used for prediction. The residue output goes to a re-ordering processor for the integer transform process and the predicted macroblock goes to the reconstruction processor to be added to the reconstructed residue data. Fig. 6 shows the parallel mapping of chroma intra-prediction. The dash line represents the long-distance communication links. Since one macroblock contains only 8x8 and 8x8 chroma Cb/Cr blocks, only one processor is needed for storage while three are used for computation. To reduce the number of routing processors, data is automatically sent to the DC mode computation processors for computing the SAD for each mode and requested individually at the second pass for computing residue. The luma intra-prediction can be parallelized with a similar approach.

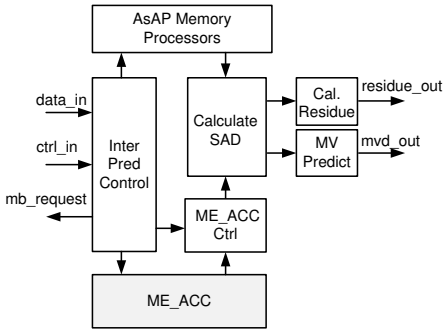


Fig. 7. Block diagram of inter prediction module on AsAP

	Custom Mapping	Mapping Tool
Number of Processors	115	147
Number of Memory Proc.	33	33
Number of Routing Proc.	21	53
Computational Proc.	61	61
Long Distance Links	48	52

TABLE I
COMPARISON OF CUSTOM LAYOUT AND PROPOSED MAPPING FROM ASAP ARBITRARY MAPPING TOOL

2) Inter Prediction: The inter-prediction is the bottleneck of the H.264 encoder which can be speeded up by a programmable motion estimation accelerator. The motion estimator basically consists of (a) a parallel array of processing elements for pixel level SAD operations; (b) a local memory to exploit data reuse to reduce the external memory access; (c) an I/O control unit. Fig. 7 shows the diagram of the proposed H.264/AVC inter-predictor. The motion estimator (ME_ACC) is capable of holding a 4x4 macroblock region for the search window. To speed up the prediction process, only a 3x3 search window is used. A modified diamond search algorithm is used for all block sizes. The modified algorithm uses only 5 search points as opposed to the nine points generally tested, and is repeated 4 times to find the best match. Although this process is not as accurate as a full diamond search the only drawback would be slightly higher entropy values to be encoded. Once the best set of motion vectors are computed, they are sent to a residue calculation processor. The data used for this prediction is read from the 11 AsAP memory processors that hold a mirror copy of the ME_ACC memory. The data-flow diagram of Fig. 7 can be also mapped to AsAP array in the same way as the chroma intra prediction module shown in Fig 6.

IV. IMPLEMENTATION RESULTS AND ANALYSIS

A. Resource Utilization

The proposed H.264 baseline encoder is implemented in sequential C, parallel C with MPI simulator and AsAP assembly on the AsAP chip simulator. The current implementation uses 115 AsAP processors, 2 shared memories and the motion estimator. Table I gives a comparison of overall processor number, memory processors, routing processors, computational processors and long distance communication links between the custom mapping and the initial mapping using the automatic mapping tool. The custom mapping saves 22% of number of processors by reducing the number of routing processors.

B. Processor Memory Usage

Fig. 8 shows the number of instruction memory used per processor in the full encoder. The majority of the processors use around 45 instructions on average and the main computation processors uses around 100 instructions. A greater usage of instruction memory would reduce the number of processors required and increase the percentage of time that each computation processor is active, however much of this space might not be used because nearly half the processors are used for routing and memory purposes. Increasing the amount of code in each processors may also decrease throughput due to less parallelization, but would also reduce the overhead of code splitting. Many of the processors in intra prediction if further divided would not provide any additional speed up through parallelization because of data dependencies.

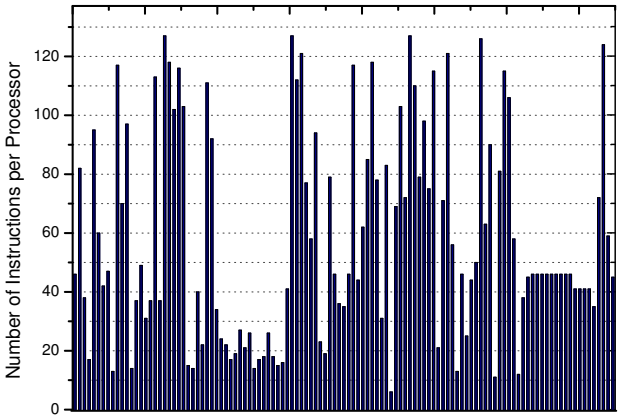


Fig. 8. Number of instruction memory words used per processor

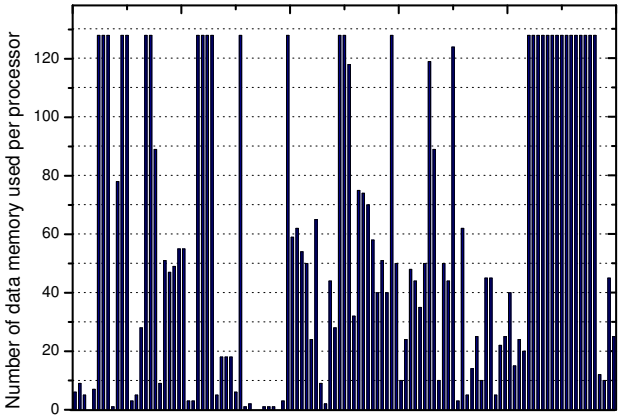


Fig. 9. Number of data memory words used per processor

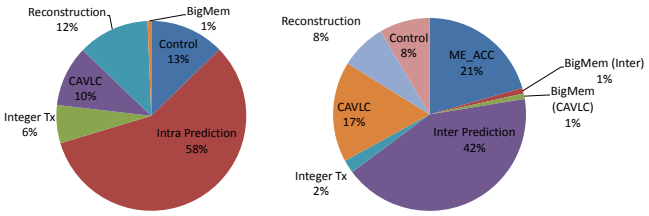
Fig. 9 shows the number of data memory words used by each processor in the full encoder. All the 33 memory processors use up the full 128-word data memory as shown in Fig. 9. The 21 routing processors use little data memory. The 61 computation processors use an average of 40-word of the 128-word available data memory.

C. Throughput and power consumption

The throughput of the proposed encoder is measured with the average cycles to encode one QCIF (176x144) frame which can be converted to frame per second at various voltages and maximum available frequencies. Table II shows the throughput and power number of H.264 encoder measured on AsAP chip. The performance of intra and inter encoder are reported separately. The power number

Voltage (V)	Max Freq. (MHz)	Intra fps	Inter fps	Power Intra (mW)	Power Inter (mW)
0.8	172	19	95	108.8	365.1
0.9	295	33	160	213.6	452.6
1.0	410	49	233	419.0	662.3
1.1	539	66	324	696.3	908.4
1.2	651	82	427	802.7	1059
1.3	798	96	478	947.5	1189

TABLE II
PERFORMANCE OF H.264 VIDEO ENCODER (QCIF FRAME) ON ASAP CHIP



(a) intra-prediction encoder (b) inter-prediction encoder

Fig. 10. Power distribution of major blocks in intra-prediction and inter-prediction encoder

in Table II is based on the condition that all of the processors are set to run at the same voltages and the maximum supported frequencies.

Since AsAP processor can be set to run at different frequencies and two provided supply voltages, we can scale the processor frequencies and voltages based on the average processor activities data profiled by the simulator. In this way, processors can be active most of the time at their individual frequencies and voltages. We use the typical Foreman video sequences for testing purpose. The preliminary results show the encoder is capable of encoding VGA (640 x 480) video at 21 frames per second (fps) with 931 mW average power consumption by adjusting each processor to workload-based optimal clock frequencies and dual supply voltages with less than 1dB loss in resolution compared to reference C model. Since integer transform, quantization and CAVLC encoding are processed at a smaller block size (4x4 block), we can further exploit more fine-grained parallelism to achieve higher performance. In our implementation, the residual encoder (integer transform, quantization and CAVLC) can encode real-time 1080p HDTV at 30 frames per second (fps) with 424 mW average power consumption.

D. Power break-down analysis

The main blocks in the intra and inter prediction process of the H.264 encoder include prediction, integer transform and quantization, reference macroblock reconstruction, and CAVLC. The power breakdown helps to identify the power bottleneck for future optimization. Fig. 10 shows the power distribution of major blocks in intra-prediction encoder and inter-prediction encoder, separately. The majority of power consumed during the encoding process is from the prediction process. The intra prediction consumed 58% of the total power as shown in Fig. 10(a). Fig. 10(b) shows that the inter-prediction including ME_ACC consumes 63% of the total power. The CAVLC and the reconstruction consumes 22% of the total power for the intra encoder and 25% of the total power for the inter encoder. The integer transform and quantization consumes very little power compared with the other modules (6% in intra encoder and 2% in inter prediction encoder). Due to the memory constraint and the port limitation, 8% to 13% of power are used for the control processors.

V. CONCLUSION

In this paper, we have implemented an energy-efficient H.264/AVC encoder on a fine-grained many-core platform. The implementation utilizes an array of 115 small processors coupled with two shared memories and a hardware accelerator for motion estimation. The proposed parallel H.264/AVC encoder is capable of encoding video sequences with variable frame sizes. The preliminary implementation is capable of encoding VGA (640 x 480) video at 21 frames per second (fps) with 931 mW average power consumption with less than 1dB loss in resolution. Our parallel programming practices provides a new method of coding over a large number of simple processors allowing for a higher level of parallelization and energy-efficiency than conventional digital signal processors (DSP) while avoiding the complexity of implementing a full application specific integrated circuit (ASIC).

VI. ACKNOWLEDGMENTS

The authors gratefully acknowledge support from ST Microelectronics, NSF Grant 0430090 and CAREER Award 0546907, SRC GRC Grant 1598 and CSR Grant 1659, Intel, UC Micro, Intellasys, SEM, and a UCD Faculty Research Grant.

REFERENCES

[1] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transaction on Circuits Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

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

[3] Yen-Kuang Chen et al., "Towards efficient multi-level threading of H.264 encoder on Intel Hyper-Threading architectures," in *Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004.

[4] Michael Roitzsch, "Slice-balancing H.264 video encoding for improved scalability of multicore decoding," in *Proc. of the 7th ACM and IEEE International Conference on Embedded software*, 2007, pp. 269–278.

[5] A. Rodrłguez* et al., "Hierarchical parallelization of an H.264/AVC video encoder," in *Proc. of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, 2006.

[6] Shuwei Sun, Dong Wang, and Shuming Chen, "A highly efficient parallel algorithm for H.264 encoder based on macro-block region partition," *Lecture Notes In Computer Science*, pp. 577–585, 2007.

[7] Dean N. Truong, Wayne H. Cheng, Tinoosh Mohsenin, Zhiyi Yu, Anthony T. Jacobson, Gouri Landge, Michael J. Meeuwsen, Anh T. Tran, Zhibin Xiao, Eric W. Work, Jeremy W. Webb, Paul V. Mejia, and Bevan M. Baas, "A 167-processor computational platform in 65 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.

[8] Zhibin Xiao and Bevan Baas, "A high-performance parallel CAVLC encoder on a fine-grained many-core system," in *IEEE International Conference on Computer Design(ICCD '08)*, October 2008.