A COARSE GRAINED AND HYBRID RECONFIGURABLE ARCHITECTURE WITH FLEXIBLE NOC ROUTER FOR VARIABLE BLOCK SIZE MOTION ESTIMATION

Ruchika Verma
Department of ECE
University of Arizona
Tucson, Az-85721
15206265149
ruchikav@ece.arizona.edu

Ali Akoglu
Department of ECE
University of Arizona
Tucson, Az-85721
15206265149
akoglu@ece.arizona.edu

ABSTRACT

This paper proposes a novel application-specific hybrid coarse-grained reconfigurable architecture with a flexible network on chip (NoC) mechanism. Architecture supports variable block size motion estimation (VBSME) with much less resources than ASIC based and coarse grained reconfigurable architectures. The intelligent NoC router supports full search motion estimation algorithm as well as other fast search algorithms like diamond, hexagon, big hexagon and spiral. Our model is a hierarchical hybrid processing element based 2D architecture which supports reuse of reference frame blocks between the processing elements through NoC routers. This reduces the transactions from/to the main memory. Proposed architecture is designed with Verilog-HDL description and synthesized by 90 nm CMOS standard cell library. Results show that our architecture reduces the gate count by 7x compared to its ASIC counterpart that only supports full search method. Moreover, the proposed architecture operates at a frequency comparable to ASIC based implementation to sustain 30fps. Our approach is based on a simple design which utilizes a high-level of parallelism with an intensive data reuse. Therefore, proposed architecture supports run-time reconfiguration for any block size and for any search pattern depending on the application requirement.

1. INTRODUCTION

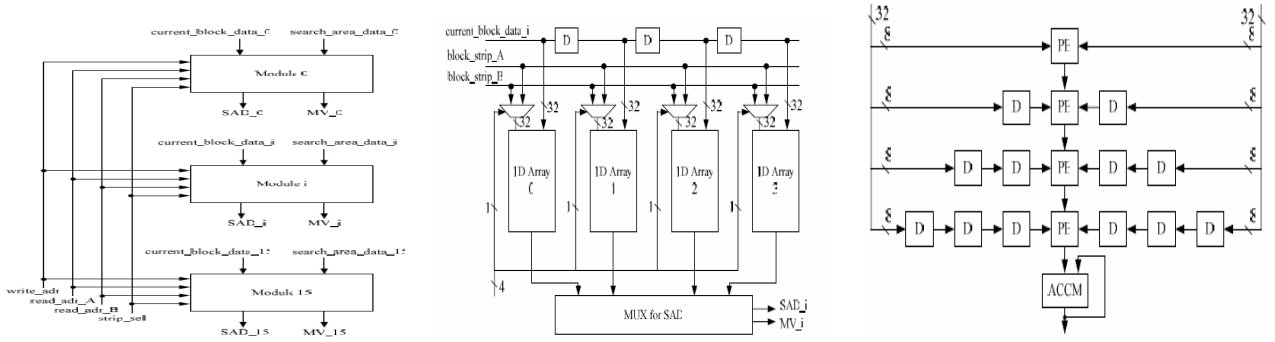
There is an increasing demand for multimedia processing solutions through flexible and highly parallel architectures. H.264 [1] video compression standard plays an important role in today's consumer market. Motion estimation (ME) is a key technique in most algorithms for video compression. It is one of the most compute intensive subroutines of H.264. Compared to fixed block-size ME (FBSME), H.264 supports VBSME [2] which provides better estimation of small and irregular motion fields and allows better adaptation of motion boundaries resulting in a reduced number of bits required for coding prediction. In motion estimation, each frame of a video sequence is divided into fixed number of non-overlapping square blocks. For each block in current frame, best matching block is searched within the previous frame. In most block matching algorithms, the sum of absolute differences (SAD) is used as the main metric. VBSME requires support for 7 block patterns: 16x16, 16x8, 8x8, 8x4, 4x8, 4x4. Supporting this feature is a challenging task in terms of resource utilization when implementing the application on hardware. Due to its highly parallel nature and algorithm's demand for a flexible solution, a reconfigurable architecture poses as an ideal candidate to respond to this compute intensive routine.

Based on the historical perspective, implementation of motion estimation has evolved through general purpose processors[3][4][5], ASIC[5][6][7][8][9], FPGA[10], and coarse grained reconfigurable architectures [11] [12] [13] [14]. Existing architectures either only support FBSME or implement VBSME with redundant hardware. While new processor advances including VLIW, SIMD, and out of order execution, have provided some advances in exploiting parallelism within applications, the processor's inherently sequential and generic architecture limits the ability to efficiently exploit potential parallelism within highly concurrent types of algorithms. Alternatively, application specific instruction set processors (ASIPs) allow designers to custom the microprocessor by adding custom instructions and execution units within the processor. However, such extensions are similar to VLIW or SIMD approaches and are still limited to the data access through the processor's register file.

Several ASIC based approaches have been proposed for variable block size block matching algorithms to reduce computational complexities. However some of these architectures [5] [6] [7] are not capable of processing all the block sizes specified by the H.264. As an alternative, Yap [8] promises to support all block sizes. Architecture is formed of 16 processing elements (PEs) interconnected as a 1D systolic array where each PE computes a 4x4 SAD. As an improvement, Ou [9] introduces a hierarchical 1D systolic architecture that employs partial SAD computation technique. Additionally in [9], a "*VBSME processor*", based on an adder tree structure, supports all block sizes. In overall, [9] achieves lower latencies with higher throughput compared to existing VBSME architectures. However partial SAD computation requires delay registers and extra accumulators; and "*VBSME processor*" consists of a fixed set of redundant adders. Therefore this architecture has high area overhead which can be improved by using a flexible routing architecture.

Coarse-grained reconfigurable architectures RaPiD[14], MATRIX[12], ChESS[13], RAW[11] have been introduced to overcome some of the drawbacks of lookup table based fine-grained reconfigurable architectures, such as FPGAs. In general, coarse-grained reconfigurable fabrics are composed of high-level processing elements (PEs) with generic reconfigurable interconnect network. While fewer configuration bits are needed for the PEs, fully utilizing the functionality of each PE is difficult, leading to significant under utilization of coarse-grained fabrics. This can be avoided by tailoring the architecture to the computation characteristics of the algorithm with application specific hybrid grained processing elements interconnected through a tuned routing architecture.

978-1-4244-1694-3/08/$25.00 ©2008 IEEE



(a) SAD Modules **(b) PE Array** **(c) 1D Array**

Figure 1: Hierarchical 1D systolic array architecture with 4 processing elements (PEs) forming 1D array and 4 1D arrays forming a module

In sections 2 and 3 we give an overview of existing ASIC and coarse grained architectures and analyze their performance on VBSME with respect to processing power and resource utilization metrics. Section 4 introduces the new architecture, section 5 presents implementation results and section 6 outlines the conclusion.

2. ASIC BASED APPROACHES

ASIC based architectures can be broadly classified into different categories depending upon various metrics such as topology of processing elements and methodology for accumulation of SADs etc. Based on topology, the architecture can be classified as 1D and 2D systolic arrays. 1D architectures [8, 15, 16] consist of 1D systolic array of processing elements. They are simpler in structure but use a large number of registers for storing partial SADs and thus suffer in area and high latencies owing to the sequential computation and accumulation of SADs. 2D architectures [6, 7] consist of processing elements connected in a mesh-based architecture and SAD computations are direct-mapped. [6] does not support block sizes smaller than 8x8. This architecture uses a smaller 2D array. So, the partial SADs are calculated and added sequentially. This results in greater latencies. [7] does not support VBSME. It uses a large number of storage registers to store reference pixels. Also, loading of reference pixels in the propagation registers causes long latencies.

Based on methodology for accumulation of SADs, the architectures can be classified as partial and parallel sum SADs. In partial sum SAD architectures [6], reference pixels are broadcasted and SAD computation for each 4x4 subblock is pipelined. In this architecture, each processing element computes one pixel difference, accumulates it to the previous partial SAD and sends the computed partial SAD to the next processing element. This kind of architecture uses large number of storage registers due to the accumulation of partial SADs in each processing element. In parallel sum SAD architectures, all pixel differences for a 4x4 sub-block are computed concurrently and thus added in one clock cycle. In this architecture, reference pixels are reused between different processing elements which decrease memory bandwidth requirements. The direction of data transfer among different processing elements depends on the search pattern adopted.

In this paper, we discuss [9] in detail which promises to have lower latency and higher throughput over other existing VBSME architectures till date. [9] consists of 16 separate SAD "*modules*" (Figure 1a) to process sixteen 4x4 motion vectors. It also consists of a chain of adders and comparators, (VBSME processor), to compute larger SADs. "*PE array*" (Figure 1b.) which forms the computation element of each SAD module is constructed by cascading four 1D arrays (Figure 1c). Each 1D array consists of a 1D systolic array of 4 PEs. Each PE computes 1 pixel SAD. This circuit operates by scheduling the columns of the current 4x4 subblock "*current_block_data_i*" through a delay line, and broadcasting two sets of search block columns "*block_strip_A*" and "*block_strip_B*" on each clock cycle. Four block matching operations can be performed concurrently in one SAD module. The produced 4x4 SADs are then sent through a fixed series of adders and comparators to produce 4x4 motion vectors. The 4x4 SADs are also sent in parallel to four sets of adders and comparators to produce 4x8, 8x4 SADs. The two 8x4 SADs are then again sent through a set of adders and comparators to form 8x8 SAD. 16x8, 8x16, and 16x16 SADs are computed similarly. This adder/comparator based chain of events form the adder tree structure.

[9] has separate SAD modules for 4x4 subblock computations with separate input ports for loading current block and search region data. Thus, it does not support reuse of search data between modules. This increases the amount of data transactions from the memory. Also, the VBSME is supported by a fixed set of adders based on a large adder tree leading to resource wastage. An intelligent routing scheme that uses same set of adders to compute different motion vectors each time is needed to overcome the wastage. Also, this architecture does not support other fast search algorithms like diamond search, hexagonal search etc.

3. COARSE-GRAINED ARCHITECTURES

In general, coarse-grained reconfigurable fabrics are composed of high-level functional blocks with generic reconfigurable interconnect network. In this section we provide a brief review and analysis of ChESS, RaPiD and MATRIX architectures for implementing motion estimation algorithm. We carefully chose these three architectures for an investigation based on granularity. They vary in granularity from 4-bit, 8-bit to 16- bit processing elements and buses and thus help in providing a

broad view about mapping of motion estimation algorithm on coarse-grained architectures.

The ChESS [13] architecture is a reconfigurable arithmetic array targeted mainly for multimedia applications. The fundamental computation component is a 4-bit ALU with 16 instructions. The routing structure is based on 4-bit buses. Each ALU has a switchbox adjacent to it which serves as a crosspoint with 64 connections. Hence it needs about 64 bits to configure the switches and connections. Since each ALU has a corresponding switchbox associated with it the routing area consumes up to 50% of the total area. If we map the motion estimation algorithm on the ChESS architecture and try to exploit full parallelism, SAD computation for a 16x16 array would require a 512 ALU ChESS array.

MATRIX [12] is yet another coarse-grained reconfigurable computing architecture composed of 2D array of identical, 8-bit functional units overlaid with a configurable network. Each functional unit consists of an 8-bit ALU, memory and control logic. MATRIX also has a generic routing architecture. Mapping the motion estimation algorithm on the MATRIX architecture for a 16x16 block would require a 256 ALU MATRIX array. The MATRIX array with 8-bit functional units would require one functional unit for one pixel difference calculation. The performance result for motion estimation algorithm if mapped on a 256 ALU MATRIX array is [(M x 0.8M)/256 x 17 x 17] clock cycles considering a frame size of M x 0.8M. Also, support for VBSME in these architectures would involve huge routing complexity which is difficult to implement owing to the generic nature of routing architecture.

RaPiD [14] is a coarse-grained architecture mainly targeted for DSP applications. It consists of 1D array of functional units (ALUs, multipliers, Registers and RAMs). The complete RaPiD array contains 16 of these cells. The functional units (cells) in RaPiD are interconnected using a set of ten segmented buses that run the length of the datapath. The buses in different tracks are segmented into different lengths. The implementation result for motion estimation as provided in the paper for a block size of 8x8 is 272+32M+14.45M2 clock cycles considering a frame size of M x 0.8M. The implementation details show that the available parallelism in the system has not been exploited to the fullest extent. For a 16x16 SAD, only the row-wise differences are computed in parallel. The column wise differences are computed sequentially which decreases the performance to a great extent. This is because the linear array form of architecture fails to exploit this parallelism. Also there is huge underutilization of resources as one cell comprising of 3 ALUs is being used for computing just one difference per clock cycle. Though no such information has been provided in the paper, even if we assume that the second ALU is being used for adding the differences in a pipeline, the third ALU remains unutilized. Also, the ALUs are 16 bit, and the SAD computation involves 8 bit operation, which again adds to underutilization. Also, the reference frames considered are 8x8. So, the architecture does not support SAD calculation for smaller block sizes. The algorithm does not require ten buses for routing. So it leads to underutilization of routing resources too. Also, the real time video performance on a standard 720 x 576 image is about 12 frames per second using 100 MHz clock. So, the performance is quite poor with a huge dissipation of energy. The paper does not talk about supporting VBSME by the architecture.

This analysis shows that existing generic coarse grained architectures are not capable of responding to the computation demands of the motion estimation algorithm. Hence, there is a need for application specific coarse grained architecture with intelligent NoC mechanism tuned for motion estimation algorithm.

4. PROPOSED ARCHITECTURE

4.1 *Hybrid Architecture Overview*

We propose a 2D architecture formed of three types of processing elements (CPE, PE2 and PE3) as shown in figure 2. Architecture contains 16 “CPE”s, 4 “PE 2”s and 1 “PE 3”. Each “CPE” (figure 3) has a PE 1, Network Interface (NI) and a NoC router and each node is labeled as “CPE (x,y)” where (x,y) represents grid coordinates. Each PE2 is represented as “PE 2(z)” where “z” identifies the processing element. A“CPE ” receives current and reference block from main memory through its 32-bit data input ports labeled as “*current_data_(x,y)*” and “*reference_data_(x,y)*” where x and y are [1,2,3 or 4].

In ME, given a current frame (CF) and its reference frame (RF), CF is first divided into non-overlapping macroblocks of size 16x16. Each macroblock associated with a search region in RF is then partitioned into 16 4x4 sub-blocks labeled as (x,y). “PE 1” is responsible for block–matching operations (SADs) of a 4x4 sub-block and its search region. “PE 1” generates a 4x4 SAD which is then fed into PE2. PE2 is capable of generating 4x8, 8x4 or 8x8 motion vectors based on the type of block size. Similarly PE3 is capable of generating 8x16, 16x8 and 16x16 motion vectors. Architecture by default supports full search (data transfer between adjacent CPEs) with zigzag pattern (Figure 12). Other fast search algorithms like diamond or hexagonal pattern search require reference blocks to be transferred between adjacent/non-adjacent processing elements. Hence, NI packetizes reference block data forming a message and based on the search pattern inserts the routing information in header packet of the message. Then, NoC router transfers data from source PE to destination PE depending on the routing information. This data transfer takes place through bi-directional edges between “CPE”s. In the following section, we introduce hybrid PEs and routing mechanism and discuss their operation principles. In section 4.5 we detail the functionality of all the modules using examples.

Memory Interface (MI): “CPE”s request “reference blocks” from main memory through MI. MI computes memory addresses of those blocks based on the search pattern. It also holds memory address of sixteen “current frames” that are already in the CPEs. MI receives “data_load_control” (16 bits) and “reference_block_id” (5 bits) signals where data_load_control identifies the CPE and reference_block_id identifies the requested block. MI then feeds the Main Memory with the address of that requested block.

4.2 Configurable Processing Element (CPE)

4.2.1 PE 1

Figure 3 shows the structure of PE1 (1,1). It is composed of 5 hybrid adders, sixteen 8 bit subtracters, sixteen 8 bit current pixel registers (CPR) and sixteen 8 bit reference pixel registers (RPR). Subtracters calculate absolute difference between a

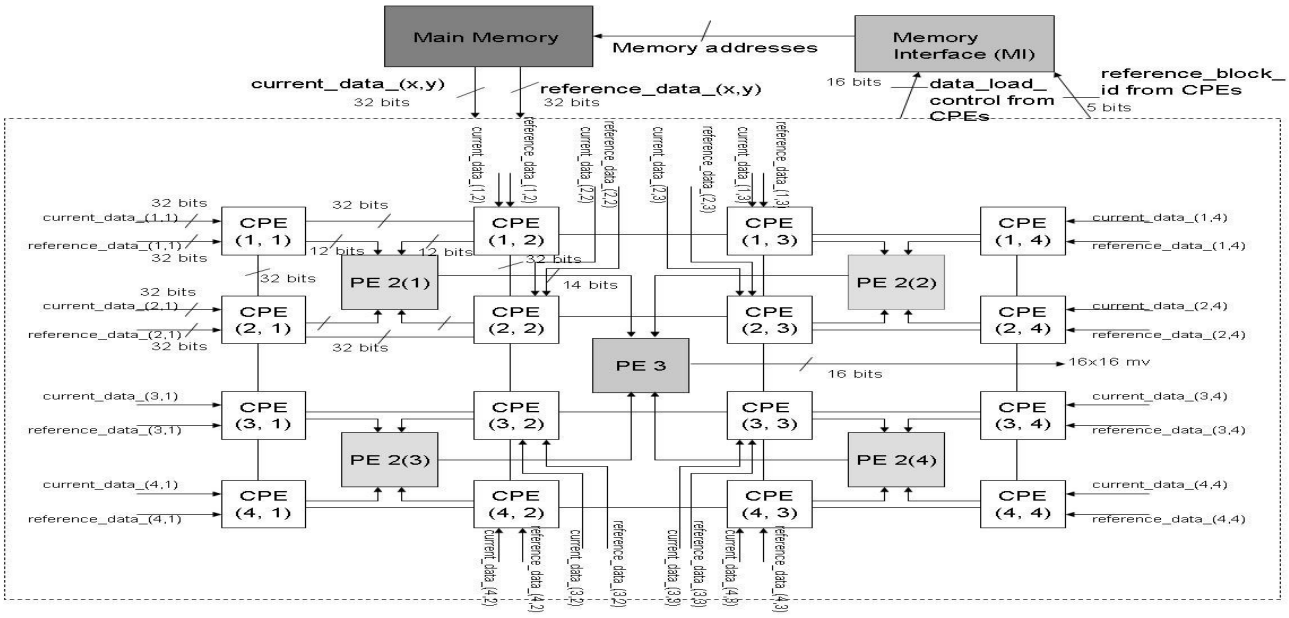


Figure 2. Proposed architecture: composed of 16 CPEs, 4 PE2s and 1 PE3. CPEs receive data from main memory through Memory Interface (MI) where MI computes memory address of the reference blocks

search pixel and a current pixel. Adders then generate 4x4 SAD and comparator (COMP) checks whether the current or the previous SAD is smaller and produces 4x4 motion vector. If block size is larger than 4x4, then current SAD is sent to PE2. For full search, demultiplexers are used to load the data into the RPRs based on the direction of movement in the search window to support ziz-zag pattern. For all other search patterns data is loaded through RPRs 1, 2, 3 and 4 and propagated through 16 subtracters.

4.2.2 Network Interface (NI):

As shown in figure 3, NI contains packetization, depacketization and control units. This unit performs the following three tasks:

- If PE1 has completed operating on a reference block, NI receives the block and inserts header information to that (specifying direction of movement, number of hops in the specified direction and size of the data) forming a message through packetization unit. NI first generates a "request" signal (1 bit) and sends it to the NoC router. When "acknowledgment" is received by NI then it sends the message to the router. Each packet is formed of 4 pixel data (32 bits) and 32 bits is reserved for header packet.
- If PE1 is the destination node, NI receives the message from NoC router and extracts reference block from incoming message through its depacketization unit and sends this data to PE1.
- If the reference block required by PE1 is not residing in any another PE1, then NI generates the "data_load_control" (1 bit) and "reference_block_id" (5 bits) signals through its "Control Unit" and feeds them to MI in order to request the data from Main Memory.

4.2.3 NoC Router

- If PE1 has completed operating on a reference block, after the handshaking mechanism described earlier in NI section, NoC router receives the packetized message through 32 bit 5-1 "*input multiplexer*" from NI. The packets are then stored in ring buffer. A "*ring buffer"* is used instead of a FIFO as data can be accessed from it in only one clock cycle irrespective of buffer size. To accomplish this, we use two pointers as shown in figure 3. The pointer *first_index* shows where the first data that comes in is stored. Thus data pointed to by *first_index* should be the first data that comes out of buffer. The pointer *last_index* shows where the last data that came in buffer is stored. Thus the location where next data that comes in is stored is *last_index + 1*. Suppose first packet is stored at location 1 and last packet is stored at location 5. The next packet that comes in after that will be stored at location 0 and the next packet that goes out will be the packet at location 1. Buffer sends first packet of message to "*header decoder"*. "*Header decoder"* follows x-y routing protocol and extracts the direction of data transfer from header packet. It also updates header packet with remaining "number of hops in the direction of data transfer". Based on the required direction of data transfer, message is forwarded to an adjacent router through 32 bit 1-5 "*output de-multiplexer*".
- When an adjacent router "B" has to send message to router "A", "*output controller*" of "B" first sends a "*request*" signal (1 bit) to "*input controller*" of "A". If "A" is not busy communicating with any other routers, then "*input controller*" of "A" checks available space in the buffer and sends an acknowledgement signal "*ack*" to "*output controller*" of "B". After receiving "ack", "B" starts sending message to "A" through 32 bit 5-1 "*input multiplexer*" of "A". The packets are then stored in "*ring buffer*" and forwarded to "*header decoder*" where the following actions are taken:
    - If the message has reached destination node, it is sent to depacketization unit of NI through "*output de-multiplexer*".
    - If the message has to be transferred to an adjacent router "C" ("A" is an intermediate hop), "*output controller*" of "A" sends a "*request*" signal (1 bit) to "*input controller*" of "C". After receiving "ack"

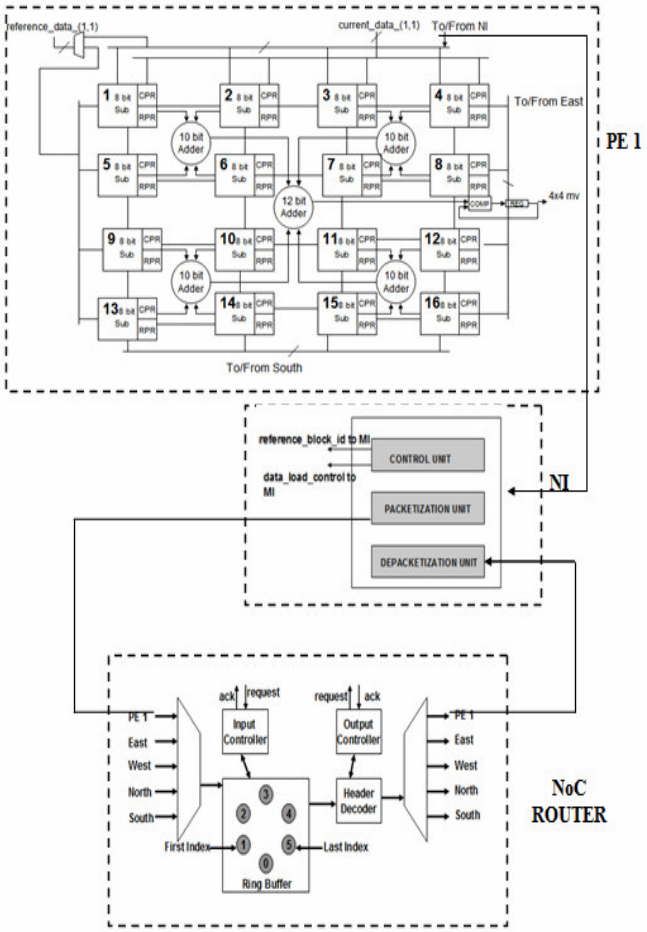


Figure 3. CPE comprising of PE1 at the corner of the architecture, Network Interface (NI), NoC Router

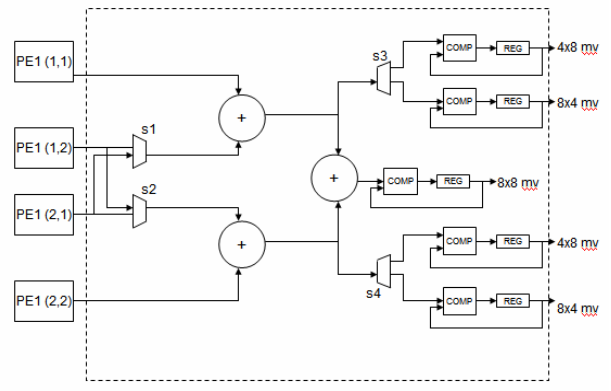


Figure 4. Processing element of type 2, PE2 (1)

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108
109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132
133	134	135	136	137	138	139	140	141	142	143	144

Figure 5. Current 48x48 Frame, and highlighted region(16x16 macroblock from current frame)

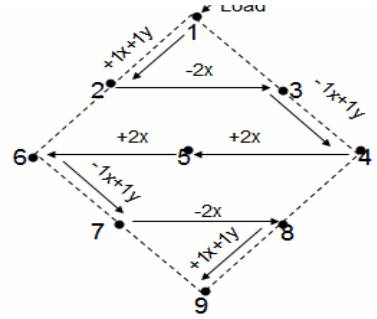


Figure 6. Data Transfer Patterns for Diamond search: direction and order of data transfer follows the numbering scheme

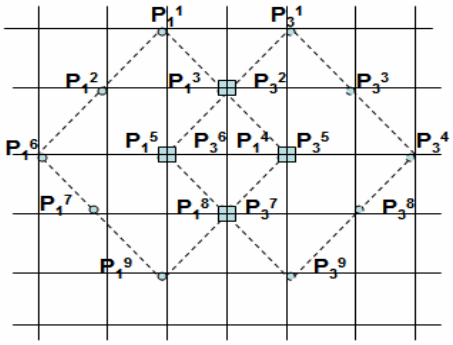


Figure 7. Reference Frames for PE1(1,1) and PE1(1,3), For P_x^y: x represents PE(1,x) and y represents order of data transfer

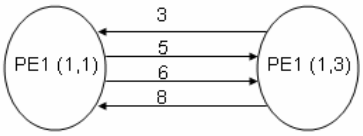


Figure 8. Data Reuse between PE1s where numbers represent cycles from Table 1

Table 1 Data load schedule for PE1(1,1), PE1(1,3) for diamond search

Clock Cycle→	1	2	3	4	5	6	7	8	9
Processing Element									
PE1 (1,1)	P_1^1	P_1^2	P_1^3= P_3^2	P_1^4	P_1^5	P_1^6	P_1^7	P_1^8= P_3^7	P_1^9
PE1 (1,3)	P_3^1	P_3^2	P_3^3	P_3^4	P_3^5= P_1^4	P_3^6= P_1^5	P_3^7	P_3^8	P_3^9

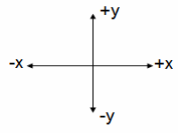


Figure 9. Coordinate conventions

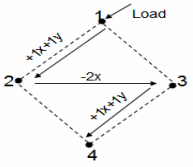


Figure 10. Data Transfer Patterns for Small Diamond search

from "C", "A" starts sending message to "C" through its 32 bit 1-5 "*output de- multiplexer*".

4.3 PE 2 and PE 3

Figure 4 shows processing element "PE 2". It consists of 3 adders, 2 multiplexers (mux), 2 demultiplexers (demux), 5 comparators and 5 registers. In PE 2, muxes "s1" and "s2" select the computation of either 4x8 or 8x4 motion vectors. Demuxes "s3" and "s4" direct 4x8 or 8x4 SADs to their corresponding comparators and registers. After computing either 4x8 or 8x4 SADs, 8x8 SAD is generated which is sent to PE 3 to process larger block SADs. PE 3 is similar to PE 2 with differences in granularity of adders. In PE 3, muxes "s1" and "s2" (Figure 4) select computation of either 8x16 or 16x8 motion vectors. Thereafter, demuxes "s3" and "s4" direct 8x16 or 16x8 SADs to their corresponding comparators and registers. Then, 16x16 SAD is computed from either 8x16 or 16x8 SADs. Now, we will discuss the routing patterns for different search algorithms.

4.4 Fast Search Algorithms and Case Study

Fast search algorithms like diamond, hexagon, big hexagon and spiral have been introduced to reduce the computation complexity of motion estimation by operating on reduced number of reference blocks. Number of reference blocks for diamond and hexagon searches are shown in figures 6 and 11. Diamond search requires (9+5n+4) reference blocks [17] and hexagon search requires (7+3n+4) [18] reference blocks where "*n*" is the number of execution iterations.

Diamond Search: As shown in figure 6, pattern consists of nine candidate search points (reference blocks) in first iteration. Numbers in this figure represent the order of processing the reference frames. Directed edges are labeled with data transmission equations derived for diamond search pattern based on the data dependencies. The grid co-ordinates followed for data transfer is shown in fig 9. An equation is represented with (+/-nx) and (+/-ny) where (+ or -) denotes the direction of data transfer on the x and y axis and n denotes the number of hops in that direction. Also, load indicates that reference block is loaded from the main memory.

We now explain SAD computation using diamond search pattern for a 16 x 16 current block on the proposed architecture. Figure 5 represents the current frame which is divided into blocks of size 4 x 4. Sixteen 4x4 current blocks 53, 54, 55, 56, 65, 66, 67, 68, 77, 78, 79, 80, 89, 90, 91, 92 are loaded into CPE (1,1), CPE (1,2), CPE (1,3), CPE (1,4), CPE (2,1), CPE (2,2), CPE (2,3), CPE (2,4), CPE (3,1), CPE (3,2), CPE (3,3), CPE (3,4), CPE (4,1), CPE (4,2), CPE (4,3), CPE (4,4) respectively. Each PE1 of the CPEs with a current block processes SAD on 9 reference blocks. PE1 (1,1) computes SAD for current block 53 against reference blocks 29', 40', 42', 55', 53', 51', 64', 66', 77'.

Figure 7 shows data dependencies and data transfer patterns for PE1(1,1) and PE1(1,3). Solid squares represent intersecting search points between the two and solid circles represent the other search points. Table 1 shows the reference block to be loaded in PE1s at different clock cycles. At clock cycle one, first search points (reference blocks P_1^1 and P_3^1) are loaded into both processing elements from main memory through MI. For this, NIs of CPE(1,1)) and CPE(1,3) set respective "*data_load_control*" signals to "high" indicating the CPE ids and "reference_block_id" output to 1 indicating the index of the search point on the diamond pattern (figure 7). Based on these inputs and address of the current frames, MI generates the addresses of the reference frames as per the search pattern and outputs it to the main memory. Main memory, then sends the requested data through corresponding "CPE" input ports. In this case first reference blocks for PE1(1,1) and PE1(1,3) are 29' and 31' respectively. Figure 7 shows that third search point (P_1^3) required by PE1(1,1) is same as second search point (P_3^2) of PE1(1,3). At clock cycle three, P_3^2 is transmitted from PE1(1,3)'s router to PE1(1,1). The NI of PE1(1,3) packetizes the reference block according to data transmission equation (Figure 6, Equation: +1x+1y, arrow labeled from point 1 to 2) and sends the packets to the router. The router of PE1(1,2) receives the packets, decodes the header packet and following x-y routing scheme, forwards the packets to the router of PE1(1,1). PE1(1,1) is the destination node hence its NI depacketizes the data to be processed within its PE1. Same events take place at cycles 5, 6 and 8 as shown in Figure 8. The events illustrated with Figure 8 take place for all sixteen 4x4 current blocks. The above data transfer pattern also dictates that at every clock cycle, every CPE is accessing a unique reference block. So, at no time slot, two CPEs compete for the same data and there is no case of resource conflict.

For every successive iteration, nine SADs are computed in each PE1. If the computed SAD minima is located at one of the vertices (1,4,6, and 9) or edges (2,3,7,8) of the diamond (Figure 6) , a new diamond pattern is formed with the minima as the new center for the next iteration. This again requires computation of SADs for next nine reference frames in the same order as in figure 7. But in this case, minima at the vertex leads to only 5 new reference frames and the minima at the edges lead to only 3 new reference frames for one iteration. In the last iteration, the minima is the center point (5). In that case, a small diamond is formed around the minima with 4 reference frames. The order of reference frames for loading the data for this case and the data transmission equations are shown in figure 10. Data transmission equations for hexagon, big hexagon and spiral search patterns are shown in figure 11. The spiral search is more compute-intensive compared to other fast search patterns. The width of the spiral search determines the number of reference frames to be compared. In this example, we carry out a 5x5 spiral search as shown in figure 11(c). So, the number of reference frames is 25.

Full Search: Now, we illustrate the functionality of the proposed architecture for full search. Table 2 shows "*data load schedule*" for PE1(1,1). As shown in table 2(a), at cycle one, four pixels of 4x4 current block are loaded into CPR registers 1, 2, 3, 4 through current_data_(1,1) port. At cycle two, these four pixels are shifted to CPR registers 5, 6, 7, 8 and next four pixels are loaded into CPR registers 1, 2, 3, 4. This continues for 4 cycles till a 4x4 current block is completely loaded into PE 1(1,1) as shown in table 2(b). Similarly, a 4x4 reference block is loaded into RPR registers of PE 1(1,1) in four cycles through reference_data_(1,1) port. All the PE1s are loaded with corresponding current and reference blocks in a similar fashion. At the end of the fifth clock cycle, 4x4 SADs are computed and sent to PE 2(1) (Table 3). Control signals to PE 2(1) select between computation of 4x8 SAD or 8x4 SAD at the end of sixth cycle. At the end of seventh cycle, 8x8 SAD is computed. Similarly, four 8x8 SADs are then sent to PE 3. In PE 3, control signals select between computation of 8x16 SAD or 16x8 SAD

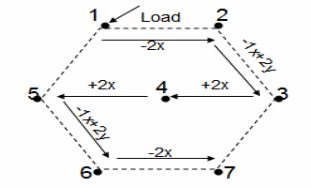
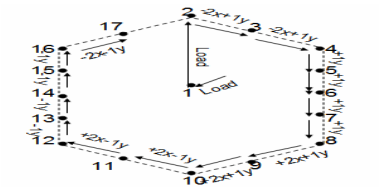
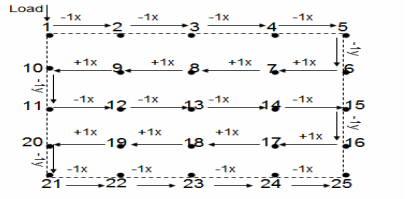


Figure 11. (a) Hexagon Data Transfer Pattern



(b) Big Hexagon Data Transfer Pattern



(c) Spiral Data Transfer Pattern

Table 2 (a) Data load schedule for PE1

Clock	Each block represents a CPR register with a current block pixel			
1	13, 13'	14, 14'	15, 15'	16, 16'
2	↓	↓	↓	↓
3				
4				

Table 2 (b) Data load schedule for PE1

Clock	Each block represents a CPR register with a current block pixel			
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

Table 3 Data flow for calculation of 8x8 SAD

Clock	PE 1(1,1)	PE1(1,2)	PE1(2,1)	PE1(2,2)	PE2(1)
5	SAD4X4	SAD4X4	SAD4X4	SAD4X4	
6					SAD4X8, SAD8X4
7					SAD8X8

Table 4 Data flow for calculation of 16x16 SAD

Clock	PE 2(1)	PE2(2)	PE2(3)	PE2(4)	PE3
7	SAD8X8	SAD8X8	SAD8X8	SAD8X8	
8					SAD8X16, SAD16X8
9					SAD16X16

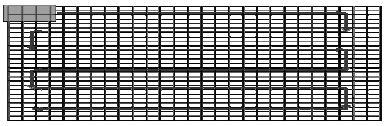


Figure 12. Zig-Zag Search Pattern

at the end of eighth cycle. At the end of ninth cycle, 16x16 SAD is computed (Table 4). Now, we present the performance results of our architecture and compare it with the ASIC based approach [9].

5. RESULTS

Proposed architecture was designed with Verilog-HDL description and synthesized by 90 nm CMOS standard cell library. Design contains about 82K gates. As shown in Table 5, resource usage (technology independent gate count) of our architecture is about one-seventh of its counterpart [9]. While [9] follows partial sum SAD approach and takes 4 cycles for producing the first 4x4 SAD, our architecture requires 5 cycles including 4 cycles of the "*data load schedule*" phase. Rest of the computations in our architecture and [9] are pipelined. Thus, [9] requires 265 cycles to produce a 16x16 motion vector for full search while our architecture requires 266 cycles. Table 6 shows the minimum clock rate that is required to sustain 30fps and 60fps frame rates by the proposed architecture and its counterpart [9]. As shown in table 6, for QCIF format, ASIC

Table 5 Resource usage comparison of the proposed architecture and its counterpart [9]

Properties	[9]	Our Architecture
Number of PE	256	21(16 of type PE 1, 4 of type PE 2, 1 PE 3)
Searching Region	16x16	16x16
Block size	4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16	4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16
Technology	180 nm	90 nm
Gate Count	597K	82K

Table 6 Required clock rates of the proposed architecture and its counterpart [9] for various frame sizes and frame rates using full search

Frame size	QCIF (176X144)	CIF (352X288)	4CIF (704X576)	16CIF (1408X1152)	SDTV (1280X720)	HDTV (1280X720)	SHDTV (1920X1080)
Frame rate(fps)	30	30	30	30	30	60	60
Clock rate(MHZ)[Ours]	0.80	3.20	12.83	51.32	29.16	58.32	131.22
Clock rate(MHZ)[9]	0.76	3.04	12.16	48.66	27.64	55.29	123.49

Table 7 Required clock rate of the proposed architecture using diamond search(DS), hexagon search(HS), big hexagon search(BHS) and spiral search(SS) for various frame sizes and frame rates

Frame size	QCIF(176X144)				CIF(352X288)				4CIF(704X576)				16CIF(1408X1152)			
	DS	HS	BHS	SS	DS	HS	BHS	SS	DS	HS	BHS	SS	DS	HS	BHS	SS
Frame rate(fps)	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Clock rate(best case)(MHZ)	0.30	0.26	0.49	0.59	1.23	1.04	1.99	2.37	4.94	4.18	7.98	9.50	19.76	16.72	31.93	38.01
Clock rate(worst case)(MHZ)	0.78	0.55	0.78	0.59	3.13	2.18	3.13	2.37	12.54	8.74	12.54	9.50	50.18	34.97	50.18	38.01

approach [9] needs to operate at 0.76 MHz to sustain the 30fps requirement, whereas our design needs a little faster clock. This is because the number of cycles required to compute 16x16 motion vector is larger in our design, (266 cycles) vs. [9](265 cycles). For clock rates slower than 0.8 MHz, our design won't be able to sustain 30fps. As shown in table 6, for all frame sizes, our clock rate is comparable with [9]. As expected, while the frame size gets larger, the clock rate needs to increase to respond to the more number of computations with the 30fps or 60 fps constraint.

Fewer number of search points for fast search algorithms overcomes the "router" cycles overhead and requires even lesser number of cycles than full search. Router's critical path consumes 6 cycles. Total data transmission cycles are the sum of router cycles and total number of hops. Since average number of hops for fast search patterns is 2, the routing cycle's overhead is 8 cycles per search point. Also, worst case number of reference blocks accounts from n=4 and best case comes from n=0 on a 32x32 search region. So, the best case and worst case number of search points are 13 and 33 for diamond search. Including the

routing overhead, diamond search takes 104 cycles and 264 cycles for best case and worst case scenarios respectively which is less than 266 cycles of full search. Similarly, the worst case and best case number of search points are 23 and 11 for hexagonal search and thus it takes 184 and 88 cycles respectively to produce a 16x16 motion vector. The worst case and best case number of search points are 33 and 21 for big hexagonal search and thus it takes 264 and 168 cycles respectively to produce a 16x16 motion vector. Also, the number of search points is constant for all the cases of a spiral search. For a 5x5 spiral search (Figure 11(c)), the number of search points is 25 and thus it takes 200 cycles to produce a 16x16 motion vector. Therefore, architecture supports fast search algorithms with operational frequencies less than the full search algorithm (table 6) as shown in Table 7. For QCIF format, our architecture operates at 0.8MHz for full search. Whereas, as shown in Table 7, for DS, HS, BHS and SS, it operates at 0.78, 0.55, 0.78, 0.59 MHz correspondingly for worst case scenarios. Since the number of search points for fast search types decrease significantly, our architecture can now operate at a slower clock rate even for the worst case scenarios to sustain the 30fps. Further, this pattern is observable for all frame sizes.

Thus, the clock rate of the proposed architecture is comparable to ASIC implementation for both full search and fast search algorithms with much lesser resources. Therefore, the frame size and frame rate supported by the circuit can be substantially extended subject to this clock rate constraint for high definition.

6. CONCLUSION

This paper proposed a new application specific reconfigurable hybrid coarse-grained architecture with intelligent NoC scheme. Existing coarse-grained architectures have generic processing elements and routing structure and therefore suffer in performance and resource utilization for motion estimation algorithm as compared to our architecture. The intelligent NoC scheme in our architecture supports VBSME for different types of fast search patterns like diamond search, hexagonal search etc. Hybrid grained processing elements support variable block sizes in motion estimation. Simplicity and highly parallel nature of the architecture with an excellent degree of data reuse makes our design suitable for run time reconfiguration. Architecture can easily be configured to run any fast search or full search algorithm with block size variations to respond to the video quality and/or timing constraints. Our 2D architecture supports huge reuse of search data between processing elements and thus reduces the memory transaction requirement which is missing in existing state of the art approaches. Proposed architecture uses hybrid grained processing elements to compute 4x8 or 8x4 and 8x16 or 16x8 motion vectors dynamically depending on the control signals supplied. Our architecture is highly flexible as it supports any block size for any search type which is a challenging problem from ASIC perspective.

7. REFERENCES

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

[2] Injong Rhee, et. al. "Quadtree-Structured Variable-Size Block-Matching Motion Estimation with Minimal Error," IEEE Trans. Circuits Syst. Video Technol., vol. 10(Feb.): p.42-50, 2000.

[3] Lappalainen, V. Hailapuro, A. Hamalainen, T.D. Nokia Res. Center, Tampere, "Performance of H.26L video encoder on general-purpose processor," The Journal of VLSI Signal Processing.

[4] S. Reader and T. Meng, "Performance Evaluation of Motion Estimation Algorithms for Digital Signal Processors," Tech. report, Stanford University, 1999.

[5] P. M. Kuhn, "Fast MPEG-4 Motion Estimation: Processor Based and Flexible VLSI Implementations," Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, vol. 23, pp 67-92, October 1999.

[6] J.F. Shen et al, "A Novel Low-Power Full-Search Block-Matching Motion-Estimation Design for H.263+," IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, no. 7, July 2001, pp.890-897.

[7] L. de Vos and M. Schobinger, "VLSI architecture for a flexible block matching processor," IEEE Trans. Circuits and Systems for Video Technology, Vol. 5, pp.417-428, 1995.

[8] S. Y. Yap and J. V. McCanny , "A VLSI architecture for variable block size video motion estimation," IEEE Transactions on CAS II, vol. 51, no. 7, July 2004.

[9] Chien-Min Ou, Chian-Feng Le and Wen-Jyi Hwang, "An Efficient VLSI Architecture for H.264 Variable Block Size Motion Estimation," IEEE Transaction on Consumer Electronics, Volume 51, Issue 4, Nov. 2005 Page(s):1291 - 1299

[10] Alex Soohoo, "FPGA Co-Processing Architectures for Video Compression," Altera Corporation.

[11] E. Waingold et al., "Baring it all to Software: RAW Machines," IEEE Computer, September 1997, pp. 86-93.

[12] E. Mirsky, A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," Proc. IEEE FCCM'96, Napa, CA, USA, April 17-19, 1996.

[13] A. Marshall et al., "A Reconfigurable Arithmetic Array for Multimedia Applications," Proc. ACM/SIGDA FPGA'99, Monterey, Feb. 21-23, 1999

[14] Carl Ebeling, Darren C. Cronquist, Paul Franklin, Chris Fisher, "RaPiD - A Configurable Computing Architecture for Compute-Intensive Applications," University of Washington Department of Computer Science & Engineering Tech Report TR-96-11-03

[15] K.M. Yang, M. T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," IEEE Trans. Circuits Syst., vol. 36, no. 10, pp. 1317–1325, Oct. 1989.

[16] Y.K. Lai and L. G. Chen, "A data-interlacing architecture with two dimensional data-reuse for full-search block-matching algorithm," IEEE Trans. Circuits Syst. Video Technol., vol. 8, no. 2, pp. 124–127, Apr. 1998.

[17] Jo Yew Tham et al., "A novel unrestricted center-biased diamond search algorithm for block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 8, no. 4, Aug. 1998.

[18] Ce Zhu et al., "Hexagon-based search pattern for fast block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 12, no. 5, May 2002.