A Pipelined Systolic Arrays Architecture for the Hierarchical Block-Matching Algorithm

Hyung Chul Kim and Seung Ryoul Maeng
Dept. of Computer Science, KAIST
373-1, Kusong-dong, Yusong-gu
Taejon 305-701, Korea
(82) 42-869-3552
hckim@camars.kaist.ac.kr

ABSTRACT
This paper presents a pipelined architecture for the hierarchical block-matching motion estimation algorithm (HBMA). The hierarchical style leads enormous computation and complex data flow between hierarchy levels. Each stage of the proposed architecture consists of a systolic array for block-matching and an interpolation unit for bilinear interpolation. The interpolation unit regulates also the data flow suitable for fully synchronous operation. The performance analysis shows that the proposed one gains nearly linear speedup, thus makes HBMA be operated in real time.

INTRODUCTION
The block-matching algorithm (BMA) is widely used in motion-compensated prediction which pursues only minimum prediction error. However, it is not appropriate in motion-compensated interpolation due to inaccurate vector field, where omitted frames should be reconstructed from temporally adjacent frames[1]. The accuracy of vector field implies that motion vectors should trace the actual motion of moving object as accurately as possible, and the group of pixels to which a motion vector is assigned should be as small as possible[2]. Moreover, motion estimation is required to be operated in real-time, i.e. 30 frames/sec.

The Hierarchical Block-Matching Algorithm (HBMA)[3, 4], a recursive motion estimation algorithm, has been developed to satisfy the accuracy constraint of motion-compensated interpolation. HBMA is fundamentally based on BMA and interpolation. The vector field generated by BMA on the top layer is formed on the coarse grid. The vectors estimated on the coarse grid are bilinearly interpolated to generate initial estimate of the vectors on finer grid of the next layer. Some vectors estimated on the higher layer are repeatedly updated as the recursion layer goes on. HBMA applies a different set of parameters to each recursion layer as shown in Table 1. The sizes of reference block and search at each layer are \( n_x \times n_y \) and \( (n_x + 2p_x) \times (n_y + 2p_y) \), respectively. The grid size at each layer is determined by the step size of \( s_i \).

<table>
<thead>
<tr>
<th>Parameters at phase</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum displacement</td>
<td>( p_i )</td>
<td>( \pm 7 )</td>
<td>( \pm 3 )</td>
</tr>
<tr>
<td>Reference block size</td>
<td>( n_i )</td>
<td>64</td>
<td>28</td>
</tr>
<tr>
<td>Step size ( s_i )</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

The general bilinear interpolation is given by

\[
d = d_1 h v + d_2 h^* v + d_3 h v^* + d_4 h^* v^*
\]  

(1)

where \( (\gamma)^* = 1 - \gamma \), \( d_i \) denote the vectors for the neighbor pixels used in interpolation, \( h \) denotes the horizontal offset, and \( v \) denotes the vertical offset.

There have been lots of researches including [5, 6] on parallel processing of BMA for real-time processing. Although, however, the computational complexity of HBMA is several times as high as that of BMA[7], there were no researches on parallel processing of HBMA to satisfy the real-time constraint owing to its control overhead and irregular data flow as pointed out other literatures [5]. We claim that the irregular data flow is caused from the repeatedly updated pattern of vectors and the bilinear interpolation. For bilinear interpolation, we can see an irregular data flow from the fact that four spatially adjacent vectors are required. Hence it is important in parallel processing of HBMA to regulate the irregular data flow suitable for synchronous operation of the whole parallel architecture.

This paper proposes a three stage pipelined systolic arrays architecture for HBMA in which a recursion layer of HBMA is mapped onto a pipeline stage. Our approach is mainly aimed at resolving the repeatedly updating of vectors with pipeline style. Each stage consists of a systolic array and an interpolation unit. The systolic array performs the full search block-matching algorithm that has been already designed in VLSI architecture[5]. We design the interpolation unit that consists of an interpolator, a set of latch arrays, and a latch array controller. The interpolator performs the bilinear interpolation. The purpose of the fixed size latch arrays and the latch array controller is resolving the irregular data flow caused by bilinear interpolation. It is verified that the proposed architecture operates in a fully synchronous way.
MAPPING HBMA TO A PIPELINED SYSTOLIC ARRAYS ARCHITECTURE

Overall Structure

The overall structure of the proposed architecture is shown in Figure 1. The systolic processing is used for the estimation procedure in each recursion phase of HBMA. The pipelined processing is used for unfolding the recursion phases. The estimation procedure and the interpolation procedure within a recursion phase are pipelined. The architecture has three pipeline stages, each stage of which consists of an estimation unit and one interpolation unit. The interpolation unit consists of the interpolator, a set of latch arrays, and a latch array controller. The interpolator, IPLTk, performs the bilinear interpolation, which takes the inputs from three latches and a latch array and generates the outputs to two latch arrays. The latch array controller, LACk, controls the output of latch arrays.

Stage 0

Stage 1

Stage 2

Fig. 1. Overall structure of the proposed architecture

All of latch arrays have fixed sizes that are determined by the parameters of HBMA. The latch arrays and the latch array controller ensure the synchronous operation. Three different clocks are used in a stage to ensure also the synchronous data flow. The clock CLK is the base clock of the system and operates the estimation unit. The clock period of CLKk+1 is four times longer than that of CLKk+1. The design criteria of the clock periods and the sizes of latch arrays are discussed in [7].

Estimation Unit

The estimation unit performs full-search BMA. There have been lots of researches about the systolic array architectures for FSBMA [5, 6] due to the regularity of data flow and the simplicity of control of FSBMA. In this paper, we simply employ the Komarek's architecture[5].

Figure 2 illustrates a 2-D systolic array architecture of Komarek. In a steady state, the systolic array requires \( E_1 = n_i \cdot (2p_i + 1) \) cycles to generate a motion vector[5].

Interpolation Unit

The global structure of the interpolation unit is shown in Figure 3(a). The interpolation unit consists of an interpolator, a set of latch arrays, and a latch controller. Based on the feature of HBMA, \( s_i = \frac{1}{2}s_{i-1} \), as shown in Table 1, the following simplified equations of the bilinear interpolation are obtained from (1).

\[
\begin{align*}
d_a &= \frac{1}{2}(d_3 + d_4) \quad \text{where} \quad h = \frac{1}{2}, \quad v = 0 \\
d_b &= \frac{1}{4}(d_1 + d_2 + d_3 + d_4) \quad \text{where} \quad h = v = \frac{1}{2} \\
d_c &= \frac{1}{2}(d_1 + d_3) \quad \text{where} \quad h = 1, \quad v = \frac{1}{2}
\end{align*}
\]

The design of the interpolator is shown in Figure 3(b). The shifters do only the shift-right-once operations for division. The delay of one cycle, represented as a small dot on the link, makes the output preserve scan line order (see Figure 4). The timing of the outputs is illustrated with regard to the point when they are caught at the output latches LA2 and LA3. Assumptions used in the design is listed in [7]. The interpolator requires three cycles for all pipeline stage. LA2 and LA3 need only another cycle to catch all of four outputs of the interpolator, because they already handle three outputs when the interpolator does its job.

Fig. 2. Systolic array for the full search BMA (in case of \( n_i = 3, p_i = 2 \)) [5]
The input latches and the output latch arrays of the interpolator allow the data flow to be synchronous and be in the scan line order. As shown in Figure 4, four estimated vectors (say neighbor vectors) are generally required to perform the bilinear interpolation. In considering the scan line order, \( d_3 \) is the oldest vector and \( d_1 \) is the youngest one. Three vectors \( d_2, d_1, \) and \( d_0 \) are interpolated when the youngest vector \( d_1 \) is available from the estimation unit. Even though \( d_0 \) can be interpolated without \( d_1 \), it is interpolated only if \( d_1 \) becomes available in order to keep the output sequence of the interpolator. To provide these neighbor vectors to the interpolator in the systematic way within a cycle, it is necessary to keep them in four different registers respectively. Further, it is also necessary to keep \( (N_p/s_i - 2) \) estimated vectors (say asleep vectors) that are younger than \( d_3 \) and older than \( d_2 \) to accommodate the interpolation when the next younger vector than \( d_1 \) becomes available.

To fulfill these requirements, we employ three latches (R1, R2, and R3) and a latch array (LA1) of size \( (N_p/s_i - 1) \) vectors to provide four inputs at once to the interpolator. The latches R1, R2, and R3 keep the vectors \( d_1, d_2, \) and \( d_4 \) of Figure 4, respectively. And LA1 keeps both \( d_3 \) and all of the asleep vectors in the oldest-first order. After the next youngest vector becomes available, the data movement between latches takes place within a cycle along the dashed arrows in Figure 3(a).

The result of the interpolation is forwarded to LA2 or LA3 to preserve the scan line order. The destination is decided in accordance that it lies whether on the same line with the neighbor vector or on the newly generated row; \( d_5 \) and \( d_6 \) are forwarded to LA2 and \( d_7 \) is forwarded to LA3. It is caused from the fact that the density of the interpolated vector field is two times higher than that of estimated vector field. It is also caused from the condition that all of \( N_p/s_{i+1} \) vectors in a line should be contiguously supplied to the next pipeline stage to ensure the scan line order. Table 2 shows the required minimal sizes of LA1, LA2, and LA3.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Size of LA1</th>
<th>Size of LA2</th>
<th>Size of LA3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td>( N_p/s_i - 1 )</td>
<td>( N_p/s_{i+1} + 1 )</td>
<td>( N_p/s_{i+1} )</td>
</tr>
</tbody>
</table>

The Latch Array Controller (LAC) controls LA2 and LA3 to allow the contiguous flow of \( N_p/s_{i+1} \) vectors in a line. Figure 3(c) shows the state transition diagram of LAC. After the full signal of LA3 is arrived, LAC allows the output end of LA3 to be open (in state S0 and ctl=1) and keeps LA3 be open during \( N_p/s_{i+1} \) cycles. One vector is forwarded and the counter \( k \) is decremented for every cycle of \( CLK_i \). After \( N_p/s_{i+1} \) vectors are forwarded from LA3 during \( N_p/s_{i+1} \) cycles, LAC switches the control from LA3 to LA2 (in state S1 and ctl=0).

The correct operation of the proposed architecture is verified by data flow analysis upon timing chart. The data flow analysis in detail is provided in [7].

**PERFORMANCE ANALYSIS**

In designing a pipelined architecture, the performance of an architecture for motion estimation is measured at least by the speedup and the feasibility of real-time processing. At first, we examine the total cycle time of the architecture for \( N_l \times N_p \) frame size. The fill-up delay of the stage \( i \) is the duration needed to start operations of the next stage \( i + 1 \). It is defined by the sum of the duration to fill up the input latches of IPLZ and that to fill the first-activated output latch array LA3. Let \( t \) be a clock period of the base clock \( CLK \) and \( t_i \) be a clock period of the clock \( CLK_i \). The fill-up delay is given by (2), where the coefficient of the second term means that a vector is supplied to LA3 for every two cycles of \( t_{i+1} \) as described in Figure 3(a).

\[
D_i = D_{fill\_up\_input\_latches} + D_{fill\_up\_LA3} = (N_p/s_i + 1)t_i + 2(N_p/2s_{i+1} + 1)t_{i+1} \tag{2}
\]
The total cycle time required for processing a frame (given by (3)) is the sum of the total delay and the duration required to flush all of results from the output latches of IPLT2. Let $D_{EO}$ denote the fill-up delay of the first estimation unit $E_0$.

$$T_f = D_{EO} + \sum_{i=0}^{2} D_i + T_{flush}$$

$$= 15(N_1N_p + 21N_p + 190) \cdot t \quad (3)$$

To analyze the speedup, a non-pipelined architecture is assumed, which has only one estimation unit and one interpolation unit that correspond to those in a stage of the proposed one, respectively. The estimation unit is an systolic array of size $n_0 \cdot (2p_0 + 1)$, the largest one among $EU_i$ of our architecture. The interpolator is similar to ours but it is not pipelined with the estimation unit and it has latch array of total size $N_1N_p/4 + N_p/2$, the minimal size for the bilinear interpolation. It operates with the base clock $CLK$. With the non-pipelined architecture, the next recursion phase can not start until a whole frame is processed including the estimation and the interpolation. Hence, the total cycle time of the non-pipelined architecture for a frame is given by

$$T_{\text{nonpipe}} = \sum_{i=0}^{2} (E_i + I_i) \cdot \text{Ref}_{i} \cdot t$$

$$= 38N_1N_p t \quad (4)$$

Therefore, the speedup of the pipelined architecture over a non-pipelined architecture is computed by (5) from (3) and (4).

$$S_{pipe} = \frac{T_{\text{nonpipe}}}{T_f}$$

$$= \frac{38N_1N_p}{15(N_1N_p + 21N_p + 190)} \quad (5)$$

By applying video formats ($N_1 \times N_p$ frame size), $f_{rate}$ Hz frame rate, and $k_{rate}$ frame skipping rate) and the worst case of the motion compensated interpolation application, where every other frame should be reconstructed by interpolation based on the motion vector ($k_{rate} = 2$), we obtain the following constraint for real-time processing.

$$T_f = 15(N_1N_p + 21N_p + 190) t \leq \frac{2}{f_{rate}} \quad (6)$$

Therefore, the required clock period $t$ of the base clock $CLK$ from (6) is

$$t \leq \frac{2}{15 \cdot f_{rate} \cdot (N_1N_p + 21N_p + 190)} \quad (7)$$

For three different video formats, the speedup and the required clock speeds are shown in Table 3 from (5), (7), and Table 1. The analysis shows that the proposed one gains nearly linear speedup since the theoretical maximum speedup that a pipeline can achieve is $k$ with $k$ stages in the pipeline. The required clock speeds of $t \leq 41$ ns (for the video conference format) and $t \leq 17$ ns (for the broadcast video format) are available in the current VLSI technologies. And for the HDTV format, the required clock speed of $t \leq 3.24$ is expected to allow the clock speed in the near future.

### Table 3. Result of performance analysis

<table>
<thead>
<tr>
<th>Video Format</th>
<th>$f_{rate}$ (Hz)</th>
<th>Speedup</th>
<th>Required Clock Speed (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>352x288</td>
<td>30</td>
<td>2.39</td>
<td>41</td>
</tr>
<tr>
<td>512x480</td>
<td>30</td>
<td>2.43</td>
<td>17</td>
</tr>
<tr>
<td>1408x960</td>
<td>30</td>
<td>2.50</td>
<td>3.24</td>
</tr>
</tbody>
</table>

**CONCLUSION**

This paper proposed a pipelined architecture for the Hierarchical Block-Matching Algorithm (HBMA). We design the interpolator for the bilinear interpolation and employ a set of latches arrays and the latch array controller for regulating data flow. The performance analysis shows that the proposed one gains nearly linear speedup. And thus it makes HBMA be operated in real time for the broadcast video format under the current VLSI technology.

### References


