Data Reuse Analysis of Local Stereo Matching

Tsung-Hsien Tsai, Nelson Yen-Chung Chang and Tian-Sheuan Chang
Department of Electronics Engineering, National Chiao-Tung University
1001 Ta-Hsueh Rd., Hsinchu 300, Taiwan, R.O.C.
{thtsai, ycchang, tschang}@twins.ee.nctu.edu.tw

Abstract—External memory bandwidth and internal memory size have been major bottlenecks in designing VLSI architecture for real-time stereo matching hardware because of large amount of pixel data and disparity range. To address these bottlenecks, this work explores the impact of data reuse on disparity-order and pixel-order along with the partial column reuse (PCR) and vertically expanded row reuse (VERR) techniques we proposed. The analysis suggest that a disparity-order reuse with both PCR and VERR techniques is suitable for low memory cost and low external bandwidth design, whereas the pixel-order reuse with both techniques is more suitable for low computation resource requirement.

I. INTRODUCTION

Computational binocular stereo is one of the major topics in computer vision, which is useful in many fields, such as intelligent robots, autonomous vehicles, 3D scene reconstruction, and multiview video coding [1]. The goal of computational binocular stereo is to estimate the displacement between each corresponding pair of pixels in the left and right views. The displacement is referred as disparity and the process is referred as disparity estimation. The most efficient way to estimate disparity is referred as local stereo matching [2] which finds the stereo correspondence by means of local matching windows.

Local stereo matching is the most prominent approach in implementing a real-time stereo matcher. For more complicated local stereo matching algorithms having better stereo matching performance, VLSI hardware implementation is inevitable. However, these powerful stereo matching algorithms often demands enormous data storage and memory bandwidth. To resolve such storage and bandwidth issues, data reuse techniques must be employed. Previous work [3] [4] have studied the data reuse in simpler local stereo matching algorithms based on block matching. However, the data reuse of more complex and robust local stereo matching [5] [6] algorithms with additional cost aggregation step has not been considered.

Unlike the simple local stereo matching algorithms, the more complex aggregation-based stereo matching algorithm poses a challenge in data reuse. The data reuse order and direction within different steps in the aggregation-based stereo matching algorithms may sometimes interfere with each other and result in even larger internal memory size or more bandwidth. To ensure the success of a hardware architecture for real-time aggregation-based stereo matcher, it is necessary to clarify the how different data reuse order and direction would affect the overall performance.

Motivated by the aforementioned reason, this work investigated disparity-order and pixel order data reuse for the initial matching cost computation. In addition, we proposed a partial column reuse (PCR) and a vertically expanded row reuse (VERR) techniques for the cost aggregation. The impacts of different combinations of data reuse orders and techniques on internal memory size and external memory bandwidth are analyzed. The analysis and comparison result suggest that if the internal memory size is the most important requirement, disparity-order reuse with PCR technique is the best candidate. On the other hand, if external memory bandwidth is deemed more important, disparity-order reuse with both PCR and VERR techniques should be more suitable.

The rest of the paper is organized as follows. Section II explains the flow of aggregation based method. Section III analysis the data reuse problem and explaining how the data reuse methods function. Section IV will compare the performance of different solutions. Then, we conclude in Section V.

II. REVIEW OF AGGREGATION BASED ALGORITHMS

According to Scharstein and Szeliski [7], the flow of aggregation based algorithms for stereo matching consist of three steps: cost computation, cost aggregation, and disparity computation. The detail of each step is explained in the following subsections.

A. Matching Cost Computation

Matching cost computation computes the initial matching cost of each pixel at different disparities. The matching cost is usually computed from luminance. There are many different similarity measures that can be used as the matching cost. The simplest similarity measure only requires one reference and target pixel to compute, such as the commonly used absolute-difference (AD). Other similarity measures requires a support window to take information of neighboring pixels into consideration, examples include sum-of-absolute-difference (SAD), adaptive support weight SAD, zero-mean SAD (ZSAD), census hamming distance [8], and mutual information [9]. The size of the window is arbitrarily selected depending on the performance requirement. Larger window often result in more accurate disparity map with blurry data.
ANALYSIS OF DATA REUSE METHODS

boundary, whereas smaller window result in opposite way. Here we define the matching cost of a pixel at (x,y) in the left reference image with disparity d be

$$C_{(x,y)} = Matcher(L(x,y), R(x-d, y)),$$  (1)

where \(L(x,y)\) and \(R(x-d,y)\) represents the center pixels of the support window in left and right images respectively, and \(Matcher()\) is the similarity measure function. For each pixel at (x,y), if the disparity range is 32, there would be 32 matching costs for (x,y) with each associated to a disparity value.

B. Cost Aggregation

In this step, the initial cost of a pixel and its neighboring pixels within an aggregation window are summed together. This aggregated cost includes the influence of the neighboring pixels to enforce the smoothness constraint [7]. The cost aggregation can be iteratively performed to include the influences from farther pixels. The aggregation of iteration \(t+1\) can be defined as

$$C_{(x,y,d)}^{t+1} = \frac{1}{R_{(x,y)}} \sum_{i=1}^{r} \sum_{j=1}^{r} C_{(x+d(i-1), y+j)}^{t} \cdot \lambda_{(x,y,i,j)},$$  (2)

$$R_{(x,y)} = \sum_{i=1}^{r} \sum_{j=1}^{r} \lambda_{(x,y,i,j)},$$  (3)

where \(\lambda_{(x,y,i,j)}\) is the weighting of each pixel within the window of size \(r \times r\) pixels, and \(R_{(x,y)}\) is the weighting normalization term which is the summation of the weightings. The weighting gives different degree of influence to different neighboring pixels to achieve better disparity result. Examples of algorithm using weighted aggregation includes non-linear diffusion [10], outlier-rejection [6], and adaptive support weight [5].

C. Disparity computation

The disparity of a pixel is determined in a winner-takes-all (WTA) manner. The disparity with the minimal aggregated cost is selected as the disparity for this pixel.

III. ANALYSIS OF DATA REUSE METHODS

On implementing aggregation based method under real-time constraint, there are many solutions to the data reuse issue. We will use the hardware architecture shown in Fig. 1 to explain different solutions.

A. Data Reuse Problem

In the matching cost computation, if data reused along the disparity axis is preferred, the computation of all the matching costs for a pixel is computed before jumping to the next pixel. This allows the data within the matching cost support window to be reused. However, the cost aggregation sums the initial matching costs of the same disparity together, which would prefer initial costs to be output along the spatial (X-Y) plane than the disparity axis. As a result, to compute the aggregated cost within an aggregation window, all the matching costs at each disparity must be stored before the aggregation can be performed. These initial matching costs that need to be stored before the aggregation of an aggregation window can be represented with a cuboid in the disparity-spatial (D-X-Y) space. The volume of this cube represents the memory size needed to store the initial costs. One way to reduce the storage requirement is to avoid the conflict in data reuse direction. For instance, change the reuse direction in matching cost computation to the X-Y plane so that it meets the direction in cost aggregation. Although doing so removes the conflict between the matching cost computation and the cost aggregation, the conflict between the cost aggregation and the disparity computation exists. To determine the disparity of a pixel, the disparity computation needs to have all the matching costs at each disparity for that pixel. However, the aggregated costs are generated in the X-Y plane first, which is different from the preferred direction of the disparity computation. Consequently, additional storage would be required to store the aggregated costs. These conflicts in data generation and reuse directions play a key role in determining the storage requirement. Therefore, it is important to derive the best data reuse strategy which resolves these conflicts so that the storage requirement can be minimized.

B. Matching Cost Computation Data Reuse

The data reuse in the matching cost computation can be categorized into two types according to the reuse order. The details of these data reuse method are explained below.

1) Disparity-Order Reuse

The disparity-order reuse reuses the data in the matching window of different disparities. Fig. 2(a) illustrates how disparity-order reuse works. When we compute the disparity of a pixel in the left image, the matching window in the right image would slide leftward within the disparity range. In other words, the matching cost of different disparities for a pixel in
the left image is first computed. Then the matching cost computation of the next pixel in the left image is performed. With the disparity-order reuse, the overlapped data within the matching window in the right image shown in Fig. 2(a) can be reused to compute the matching cost at different disparities. As a result, if the pixel data are stored in external memory, there is no need for repeating accesses of the overlapped pixels. Hence, the bandwidth requirement to external memory can be reduced. However, the order of matching cost generation is different from the order of the matching cost consumption in the following cost aggregation step. This would result in additional memory storage requirement.

2) **Pixel-Order Reuse**

Similar to the disparity-order reuse, the pixel-order reuse reuses the data overlapped by the neighboring matching window in both left and right images.

Fig. 2(b) illustrates the detail of pixel-order reuse. The matching cost of the same disparity for each pixel is first computed. Then the cost of the next disparity for each pixel is computed. As a result, the matching window in the left and the right images both slides synchronously with the same disparity offset. With the pixel-order reuse, the overlapped data within the matching windows shown in Fig. 2(b) can be reused. Therefore, the pixel-order reuse can also reduce the external memory bandwidth requirement. In contrast to the disparity-order reuse, the order of matching cost generation is the same as the order of the cost consumed by the following cost aggregation step. Hence, the buffer size between the two steps can be reduced. However, the data reuse can only be exploited during the cost computation of one single disparity. There is no data reuse between the computations of different disparities. Once all the computation of the previous disparity has been completed for all the pixels in the whole image, pixel data have to be read from the external memory again. Unless all the previously read pixel data could be stored within the internal memory, otherwise repeating external memory accesses are inevitable.

C. **Cost Aggregation Data Reuse**

In addition to the data reuse in the matching cost computation, there are two data reuse methods in the cost aggregation. The details of these two data reuse methods are explained as follows.

1) **Partial Column Reuse**

The partial column reuse method reduces the local memory size in the cost aggregation by distributing the computation of aggregated cost to each column. Instead of computing the aggregated cost after all the initial costs in an aggregate window are available, the PCR computes the partial sum of a column after the initial costs of this column are available. As a result, the size of the local memory can be reduced. Moreover, the partial sum of each column can contribute to the aggregated cost of multiple overlapped windows. Storing partial column cost requires less local memory size than storing all the initial matching costs in a column.

Fig. 3 illustrates an example of the PCR with a 5x5 aggregation window size. An aggregated cost requires the partial sum of five initial cost columns. With PCR, the current partial column sum in Fig. 3(a) can be reused to contribute to the aggregated cost of windows 1 to 5.

2) **Vertically Expanded Row Reuse**

The vertically expanded row reuse reduces the bandwidth requirement to the initial cost memory by deliberately access additional rows of initial costs. When the aggregation finishes processing the current row and jumps to the next row, the overlapped data between the windows at the previous row and the current row have to be read from the initial cost memory again. Fig. 4 shows an example of the situation that the data are overlapped. To avoid accessing the already accessed costs, the VERR vertically expand the rows of initial costs to be read so that they can be reused to compute multiple rows of aggregated cost at once.

Fig. 4 shows how VERR can reduce the overlapped data. Without the VERR, most of the data in the windows are overlapped for many times. These overlapped data are read repeatedly multiple times. However, with the VERR, the portion of overlapped data becomes much smaller than the case without the VERR. Moreover, the overlapped data in the VERR case only overlap once. This implies that with the
TABLE I. ANALYSIS OF DISPARITY-ORDER AND PIXEL-ORDER DATA REUSE WITH PCR AND VERR methods

<table>
<thead>
<tr>
<th>Section</th>
<th>Property</th>
<th>Disparity-Order</th>
<th>Pixel-Order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Original</td>
<td>+PCR</td>
</tr>
<tr>
<td>Step 1</td>
<td>Internal Memory Size (KBytes)</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>Bandwidth Requirement from External DRAM (MB/Bytes/sec)</td>
<td>3.3</td>
<td>3.2</td>
</tr>
<tr>
<td>Step 2</td>
<td>Internal Memory Size (KBytes)</td>
<td>563.2</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>Bandwidth Requirement from Cost Computation Engine (MB/Bytes/sec)</td>
<td>6.5</td>
<td>158.7</td>
</tr>
<tr>
<td>Step 3</td>
<td>Internal Memory Size (KBytes)</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Total</td>
<td>Bandwidth Requirement from External DRAM (MB/Bytes/sec)</td>
<td>565.7</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td>Real-time Constraint (30 fps)</td>
<td>Meet</td>
<td>Meet</td>
</tr>
</tbody>
</table>

V. CONCLUSION

This paper explores the impact of disparity-order and pixel-order data reuse in the matching cost computation and proposed the partial column reuse (PCR) and vertically expanded row reuse (VERR) techniques for the cost aggregation for local stereo matching architectures. The analysis and comparison conclude that the architecture using the disparity-order reuse with both the PCR and VERR techniques is suitable for the design of low memory cost with high computation resource. On the other hand, the architecture using pixel-order reuse with VERR technique requires less computation resource, but needs large internal memory in storing the aggregated cost. It is up to the designers to adopt the reuse combination that meets best to their constraints and requirements. Further study on impact of various reuse parameter settings shall be conducted in the future.

REFERENCES


VERRU, the repeating accesses of the overlapped data would be fewer than the case without the VERR.

Fig. 5 plots the relationship between the average access count of an initial matching cost and the value k given an aggregation window size of 25x25. The value k represents the number of expanded rows. It can be observed that the average access count decreases as k increases. This suggests that with more rows expanded, less bandwidth is needed. However, increasing the value of k will also increase the local memory size and computing resource requirement.

IV. COMPARISON

TABLE I compares the estimated memory size and bandwidth requirement of the disparity-order and pixel-order reuse methods. The target disparity image is 352x288 pixels large with 64 disparities. The real-time constraint is 30 fps. The architecture is assumed to operate at 100MHz clock with a 32-bit data port to the external memory. The size of support window in the matching cost computation and cost aggregation are 9x9 and 25x25 pixels respectively.

From TABLE I, only original pixel-order and pixel-order reuse with PCR technique fail the real-time constraint because of enormous external memory bandwidth requirement. This is because applying PCR with the pixel-order reuse alone would limit the cost aggregation throughput due to column-based computation. For reuse combinations that meet the real-time constraint, if minimal internal memory size is required, the disparity-order reuse with PCR is the best candidate. On the other hand, if external memory bandwidth is deemed more important, the disparity-order with both PCR and VERR is more suitable. However, the disparity-order has large bandwidth requirement from the matching cost computation engine, which implies large computation resource requirement. Hence, if the computation resource is the concerning issue, the pixel-order reuse with both PCR and VERR should be adopted.