Acceleration of Optical-Flow Extraction Using Dynamically Reconfigurable ALU Arrays

Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi,980-8579, Japan
Email: {hasitha@kameyama, hariyama@, kameyama@ecei.tohoku.ac.jp

Abstract—An effective way to implement image processing applications is to use embedded processors with dynamically reconfigurable accelerator cores. The processing speed of these processors are not only depends on the parallelism, but also depend on the local memory utilization since the local memories are much faster than the global memory. In this paper, we accelerate the optical-flow extraction algorithm based on SAD calculation using a dynamically reconfigurable ALU array. We use the maximum parallelism and propose a memory allocation method to use the local memory effectively. The experimental results demonstrate that an image of size $640 \times 480$ can be processed in $15 \text{ms}$.

Keywords—DPGA, multi-context, memory allocation.

I. INTRODUCTION

Real time image processing has become a key area in the fields of high-safety vehicles, security systems, etc. High processing speed, low cost and low power implementation are important in these fields. Heterogeneous architectures such as [1] and [2] with dynamically reconfigurable processor (DRP) cores are a good platform to implement image processing applications. Unlike ASICs, the processing time of DRPs is decided by not only the parallel processing, but also the memory utilization. DRPs contain a large memory module (global memory) placed outside the accelerator core and several small memory modules (local memory) placed inside the core. The local memories are much faster than the global memory. Therefore, we have to copy the data from the global memory to the local memory periodically. This data transfer time is a critical problem. Since the data transfer time is linearly related to the data amount, it is very important to reduce the transferred data amount.

In embedded heterogeneous processors such as [1], the address generation units (AGUs) are integrated with the memory modules. Figure II shows the DRP architecture proposed in [1]. It is called Flexible Engine/Generic ALU Array (FE-GA). The AGU is placed inside the “LS” (Load/Store) cells and directly connected with the memory. The AGU produces only linear access patterns. Therefore, we need data duplication to implement more complex access patterns. This data duplication increases the data transfer time significantly.

II. ARCHITECTURE MODEL

We use the heterogeneous architecture proposed in [1] that has 4 homogeneous CPUs (SH cores) and 2 FE-GAs. Figure II shows the overall architecture of the processor. A detailed discussion on CPU cores is given in [3]. The data transfer among processor cores are done through the “SuperHyway” bus. An off-chip SDRAM of 128MB is attached to the SuperHyway to store a large amount of image data that are received from image devices such as cameras. The processing speeds of FE-GAs and CPUs are 300MHz and 600MHz respectively.

The architecture of FE-GA is shown in Fig.II. It has an array of 32 processing elements (PEs) or cells. These cells are dynamically reconfigured as adders, subtracters, logic gates etc. A PE can be connected to its four neighbors and those connections are also dynamically reconfigurable. The FE-GA has dynamically reconfigurable 256 sequences. The sequence manager can change the sequence after receiving an input from the PEs or after processing for
III. IMPLEMENTATION OF OPTICAL-FLOW EXTRACTION

A. Optical flow

The motion vector of a particle gives its direction and traveled distance after \( \delta t \) time as explained in [4]. The optical flow of an object in an image refers to the motion vectors of all the pixels of that object. In optical flow extraction, corresponding pixels between two images taken at time \( t \) and \( t + \delta t \) are searched. To find a corresponding pixel, a reference window of a particular pixel on the image at time \( t \) and a search area on the image at time \( t + \delta t \) are considered as shown in Fig.3. Different candidate windows are selected from the search area and SAD (sum of absolute differences) with the reference window is calculated. The DFG of SAD calculation for a reference and a candidate windows with \( n \) pixels each is shown in Fig.4. The similar the reference window to the candidate window is, the smaller the SAD becomes. Therefore, the candidate window when the SAD becomes minimum is selected as the corresponding window to the reference window.

B. PE allocation

As explained in Sec.III-A, optical flow extraction has 2 tasks, SAD calculation and minimum SAD search. Table I shows the time taken for each task when CPU is used. The SAD calculation takes more than 99% of the processing time.

Therefore, we accelerate the SAD calculation process using the FE-GA. In FE-GA, the maximum parallelism is 8 calculations which is limited by the number of PEs in the cell array. The PE allocation is shown in Fig.5. This is the maximum speed we can get using this cell array. Table II shows the processing speed of this implementation. Although the SAD calculation time reduced from 584ms to 11.06ms, the total processing time reduced by only 1.6 times. The data transfer time from the off-chip SDRAM to the CRAMs of the FE-GA has a huge impact on the total processing time.

C. Memory allocation

To solve the data transfer problem, we need to reduce the transferred data amount. As shown in Fig.3(b), all the candidate
windows overlap each other. Therefore, if we allocate the shared area effectively, we can reduce the transferred data amount and also the data transfer time. To explain this problem, we consider an example that has a search area and candidate windows of sizes $10 \times 10$ and $4 \times 8$ respectively. We consider there are only four memory modules or CRAMs.

To allocate the data effectively, we consider the access of the candidate windows. As shown in Fig.6(a), window 1 to 7 are accessed by moving the windows horizontally. In each movement, we access a new vertical line. For example, when window 2 is accessed, we need the data of the vertical line $[0.4$ to $7.4]$. Therefore, we allocate the data of the vertical lines one-by-one to CRAMs as shown in Fig.6(b). To access the data of the first window, we have to access address 00 to address 07 in 4 CRAMs. Similarly, we access the addresses 02 to 09 for the second window. To access the window 8, candidate window moves one pixel down. The difference between window 1 and window 8 is the horizontal line $[7.0$ to $7.3]$. All the other data are the same for both windows. We have the data of lines 1 to 3 and 5 to 7 in CRAM1 to CRAM3. Therefore, we allocate the data of lines 4 and 8 to the next available address (address 20) of CRAM0.

In this case, we have to access CRAM1 instead of CRAM0, CRAM2 instead of CRAM1, and so on. Therefore, we need to change the connections of the CRAMs and the cell array to realize this access pattern. To solve this problem, we use dynamic reconfiguration. Figure 7 shows the structure of sequences in FE-GA. The sequence 1 in Fig.7(a) is used to access the data of window 1 and the sequence 8 in Fig.7(b) is used for the window 8. Therefore, we share the data in CRAMs among different windows by dynamically changing the sequences.

The address generators in CRAMs create only linear addressing functions. However, for the horizontally moving windows, we need stride access to the CRAMs. For example, to access the pixels of window 2, we have to change the address from 07 to 02 in CRAMs. Therefore, we need a separate addressing function for each window. To change the addressing functions, we use the dynamic reconfiguration. Therefore, for each window, we use a separate sequence that contain a unique addressing function.

### IV. Evaluation

We used the processor board shown in Fig.8 for the evaluation. The FE-GA is programmed by hand and the CPUs are programmed by C type programing language. We use a search area and candidate windows of sizes $24 \times 24$ and $16 \times 16$ respectively. As a result, we need 81 sequences. The image size is $640 \times 480$. The gap between two reference windows is 20 pixels. Figure 9 shows the task allocation and tasks scheduling when one CPU and one FE-GA is used. Table III shows the processing time.

In this implementation, the SAD calculation in FE-GA takes the largest processing time. Since there is a data dependency in each task, we cannot use CPU and FE-GA simultaneously. However, the SAD calculations of different corresponding points are mutually independent and the tasks of a particular corresponding point do not need the data of another point. Therefore, if we use both FE-GAs to calculate different corresponding points, we can use one CPU and one FE-GA simultaneously. Figure 10 shows this implementation.

<table>
<thead>
<tr>
<th>Task</th>
<th>Allocated processor core</th>
<th>Execution time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data transfer from SDRAM to CRAM</td>
<td>CPU</td>
<td>13.10</td>
</tr>
<tr>
<td>SAD calculation</td>
<td>FE-GA</td>
<td>14.60</td>
</tr>
<tr>
<td>Read back from CRAM</td>
<td>CPU</td>
<td>3.58</td>
</tr>
<tr>
<td>Searching for minimum SAD</td>
<td>CPU</td>
<td>0.02</td>
</tr>
</tbody>
</table>

### Table II: Processing time using CPU and FE-GA: single context

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data transfer from SDRAM to CRAM</td>
<td>352</td>
</tr>
<tr>
<td>SAD calculation</td>
<td>11.06</td>
</tr>
<tr>
<td>Data transfer from CRAM to SDRAM</td>
<td>3.58</td>
</tr>
<tr>
<td>Searching for minimum SAD</td>
<td>0.02</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>367</strong></td>
</tr>
</tbody>
</table>
the SAD calculation for a particular reference window is finished, the searching of minimum SAD and the data transfer of another candidate window is performed by the CPU. At the same time, the other FE-GA can do the SAD calculation for another candidate window. Since the operations of the CPU are overlapped with those of FE-GAs, the total processing time is reduced.

Figure 11 shows the speed-up of different implementations using different number of CPUs and FE-GAs. According to the results, 20 times speed-up is achieved using a single FE-GA. The speed-up is 39 times for 2 FE-GAs. Table IV shows the processing time for each implementation. If the frame rate is 30fps, we can achieve a real-time processing using a combination of CPU and FE-GAs.

V. Conclusion

According to the processing time analysis for different tasks in Table I, the SAD calculation takes most of the processing time. Therefore, we implemented the SAD calculation in the FE-GA. We use the highest possible parallelism, that is 8. Since the FE-GA can access only its local memory, we have to copy the data from the off-chip SDRAM to the CRAMs of the FE-GA. Since this data transfer time is a huge performance bottleneck, we propose a memory allocation to share the data in CRAMs among candidate windows.

As shown in Figure 11, the use of a single FE-GA increases the processing time by 20 times compared to the CPU only implementation. If the best approach is used, the optical-flow calculations takes only 15 milliseconds. Therefore, this method can be used in real time image processing algorithms that use SAD calculations. Moreover, the power consumption of the whole chip is less than 1.4W compared to the high-end CPUs that have a power consumption of more than 50W. Therefore, heterogeneous processors with DRP cores can be used in mobile appliances for image processing.

ACKNOWLEDGMENT

This work is partly supported by Hitachi, Ltd.

REFERENCES


