Hardware-Software Co-Design for Accelerating "Find Face"

Authors Chih-Yung Liang(梁智湧), 0116229

***Abstract*—This study introduces several methods for accelerating the program "Find Face", and discusses how each method help to the performance and consume the hardware resource.**

Keywords—co-design; block ram; pipeline; master IP

I. INTRODUCTION

"Find Face" is a program that find positions of four faces, 32 pixels wide and high, in a group picture, 1920 pixels wide and 1080 pixels high. Witten in *C*, it is originally processed only by software, and is slowed down by a portion of its implementation. By profiling the behavior of this program[1], it can be realized that the hot spot comes from one of its sub-procedure. With hardware cooperation, this sub-procedure is accelerated by several mechanisms, making it run many times faster than the original version. In this report, this hot spot along with mechanisms accelerating it will be described in detail. In the end, a conclusion with respect to the acceleration are discussed.

II. THE HOT SPOT OF "FIND FACE"

By the result of behavior profiling[1], the hot spot of "Find Face" comes from its sub-procedure, *match*. It is due to *match* processes a time consuming sub-procedure *compute_sad*, which loops 1024 times, for every position in the group picture. As the implementation, while *compute_sad* iterates a region 32 pixels wide and high to obtain the sum of absolute difference (SAD)[2] of that region, *match* memorizes the minimal one of the results, also called *cost*, as the position declared to be where the face is. Taking advantage of parallel computing, the hardware may accomplish *compute_sad* in only few clock cycles[3], and make it the reason to migrate this portion from software to hardware.

III. THE HARDWARE ACCELERATION CONCEPT

Although it sounds reasonable to migrate only *compute_sad* to hardware, with the assumption that only *compute_sad* is parallelizable, leaving the remaining part of match in software is still not perfect enough because doing this would require lots of memory communication between hardware and software. For the reason that memory communication has huge costs on performance that flaw the acceleration, the design should have memory communication as less as possible. As a result, the principle of the acceleration starts with migrating the whole *match* to hardware. The following describe the acceleration concept step by step.

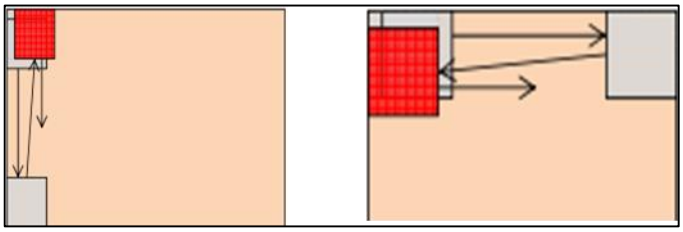

Fig. 1 The left one illustrates round-robin top-down scanning path. The right one illustrates round-robin left-right scanning path.

A. *Implement the whole **match** in hardware[4]*

To minimize the memory communication between hardware and software, instead of controlling the flow in the software, it is better to implement the complete *match* in hardware. The software will not send any pixel data to the hardware; however, it gives the memory addresses of all 5 pictures, including one group picture and four face pictures, to have the hardware get those pixel data by itself. After sending these addresses, all the software have to do is to trigger the hardware and wait for the finish response from the hardware.

To fulfill this concept, namely, to prevent from depending on the software, the hardware must have the ability to read pixels of a given address from memory, so *master IP* is chosen to implement the hardware. On the other hand in this implementation, due to the insufficient hardware resource to parallel compute SAD of four faces, these faces are searched separately and sequentially. For each face, the searching flow is same as the original one in the software, iterating every region of position, row by row and column by column, to get the minimal SAD. The minimal SAD, along with the position where it occurs, of each face is memorized, and all these values, accompanied with a finish signal, are sent back to the software when the *IP* finishes searching all the faces.

B. *Improve the utilization of group data[5]*

Since the SAD computation requires all pixels of the face and the region of the group to be accessible asynchronously for performance, pixels are preloaded into distributed RAMs (dRAM[1]). Because distributed RAMs are constructed by lots of LUTs, it is resource consuming that only two RAMs of 1024 bytes of this kind are created for storing pixels of the face and the region of the group. Thus, there must be some data dropped from the group RAM and some new data inserted into it for each iteration computing SAD.

As the result, the iterating way, or the scanning path, affects the performance a lot. The previous work[3] adopts the round-robin top-down scanning path, illustrated in Fig. 1. This scanning method drops and inserts a row of group pixels for

[1] Abbreviation for distributed RAM, not to be confused with Dynamic Random Access Memory (DRAM).

each iteration in the same column. When the iteration jumps to next column, all group pixels of this column have to be loaded again, although most of these pixels have been loaded before. In general, most pixels of group picture are loaded from memory for 32 times.

In consideration with the time costs for memory reading, it would obtain a large improvement on performance if all pixels in the group picture are read from memory only once. To fulfill this concept, 32 32bit-width block RAMs (BRAM) are used in the *IP*, each storing a whole row of pixels of the group picture, and the scanning path is changed to the round-robin left-right style, illustrated in Fig. 1. Before starting the iteration of a row, all 32 involved rows of pixels are loaded and each row is stored into a respective block RAMs; then, each iteration of a row can be simplified to loading pixels of a column from BRAMs into dRAMs and computing the SAD. When the iteration continues into next row, these BRAMs are shift up by one row to reuse those 31 already loaded rows of pixels, and the new row of pixels is loaded into the unused BRAM. With help of BRAM and the change to the scanning path, none of the pixels in the group picture is loaded from memory more than once, and the performance gets an effective improvement.

C. *Compute SAD in pipeline style*

The SAD computation must spend a couple of cycles to finish to prevent a long propagation delay exceeds the timing constraint. In other words, the output result retrieved from the computation is related to the input that is fed few cycles ago. After the hardware is accelerated by those methods previous mentioned, the cycle delay makes the SAD computation become the slowest portion of the execution, even slower than those involved with memory.

However, by taking advantage of pipeline implementation, these delay cycles can be completely eliminated. Because of the characteristic that the output retrieved after few cycles will not be influenced by any input not relating to it, the input pixels can be fed into the computation cycle by cycle, without immediately waiting for the output result, and the corresponding output SADs will be available after some amount of cycles also cycles by cycles. For the implementation, loading pixels from BRAMs and feeding pixels into the SAD computation are done simultaneously, meaning that the computing iteration of a row can be finished in only a little more than 1920 cycles.

D. *Load next row while computing SAD*

Because the SAD computing iteration of a row still consumes more cycles than loading a row of pixels into BRAMs, loading next row into BRAMs can be done at the same time SADs are computed. The second port and the remaining space of BRAM are used to accomplish this concept. Although this improvement will not greatly accelerate the program, as mentioned that the slowest part is still the SAD computation, it still makes difference on executing time when the program has already become very fast.

IV. The Overview of The Program

The software of the program is actually doing not much now, while most part of the program, except for image reading and median filtering, are migrated to hardware, and the data communication between software and hardware is left with a trigger/complete signal and few addresses for pictures. To realize the behavior of the hardware of this program, Fig. 2 illustrates the finite state machine of it, and the behavior is introduced by the following brief description to each state.

A. *IDLE*

Being in this state represents that the hardware *IP* has not received any request from the software since the previous execution completed, and the hardware *IP* will keep sending the complete signal as long as the state is not jumping away. The state jumps to *LOAD_FACE* when a trigger signal is set by the software.

B. *LOAD_FACE*

Pixel data of face are loaded from memory into the dRAM of the hardware *IP* in this state. As four different occasions entering this state, the corresponded face is chosen to be loaded. The state jumps to *LOAD_GROUP* when the face loading is completed.

C. *LOAD_GROUP*

In this state, the hardware *IP* loads pixel data of the first 32 rows of the group picture from memory to BRAMs. The state jumps to *BRAM_EXTRACT* when the loading is completed.

D. *BRAM_EXTRACT*

In this state, the first 32 pixels of each of the 32 rows to be computed are extracted from the corresponded BRAM to the dRAM in the *IP*. After that, the dRAM is ready for computation, and the state jumps to *MATCHING_COMPUTE*.

E. *MATCHING_COMPUTE*

This is the most complex state of the hardware *IP* where the

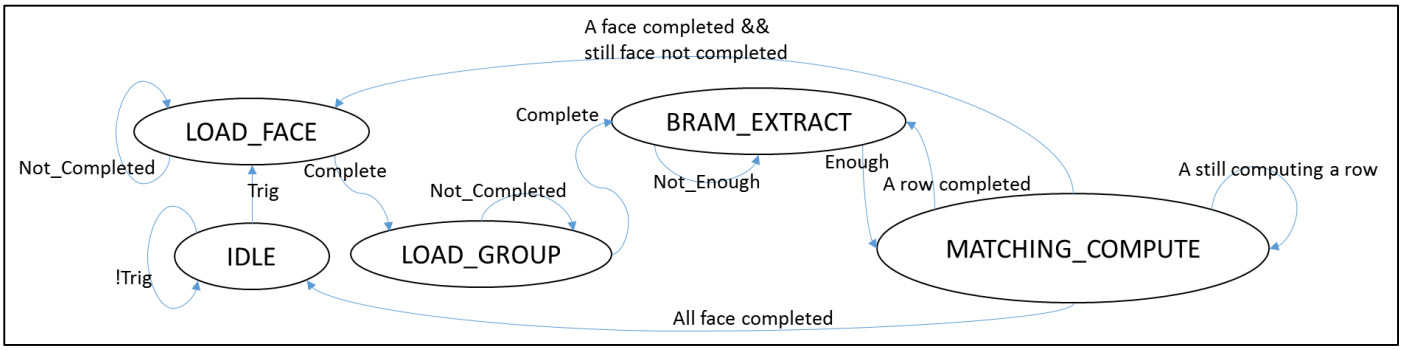

Fig. 2 The finite state machine of the hardware IP.

pipeline computation for SAD occurs. In this state, the SAD results is generated by the computation for each clock cycle while BRAMs keep extracting pixels into the dRAM in order to have the result after a couple of cycle. Each generated SAD value is compared to the minimal one of the searched face recorded until that cycle, and replace it if the generated one is smaller. At the same time the replacement occurs, the position of that result is also recorded. Notice that the minimal SAD of different faces are recorded separately, and the comparisons only involve the one relative to the searched face.

At the same time the computation occurs, the *IP* loads pixels of next row to be computed into the BRAM responsible for that row. After the computation of the row completes, the state jumps to *BRAM_EXTRACT* if there is still row to compute, jumps to *LOAD_FACE* if there is still face to search, or jump to *IDLE* if everything is completed.

V. The Execution and Performance Analysis

A. The execution

Fig. 5 is a snapshot of the executing result. As the snapshot shows, the found positions along with its *costs* are correct compared with the original version before acceleration, and the time spending on searching four faces is 80 ms.

B. The effect after applying each acceleration

Table 1 demonstrates the time spending on match after applying each acceleration. Some of these methods have the effect making the program run few times faster while some decrease the executing time for couples of microseconds. However, the combination of these acceleration methods makes the great performance.

Table 1 The executing time and acceleration effect of each method

Method	Executing time	Acceleration effect
Original	32 sec	X
A	1600 msec	95% reduced
B	770 msec	52% reduced
C	110 msec	6/7 reduced
D	80 msec	30 msec reduced

C. Performance analysis

The following roughly analyzes the reason for spending 80 ms to accomplish the *match*. Table 2 estimate the clock cycles required for each state.

Table 2 The estimated number of cycles required in each state

State	Reason	Cycles
LOAD_FACE	load 256 words	400
LOAD_GROUP	load 15,360 words	23,000
BRAM_EXTRACT	extract 9 words	9
MATCH_COMPUTE	iterate 1,978,624 regions	1,978,624
TOTAL		2,002,033

The numbers of cycles required for memory accessing are estimated by multiplying about 1.5 to the number of words (4 bytes) to be read. The factor 1.5 is realized by using ChipScope to observe the number of cycles needed for memory reading.

The total number of cycles listed in Table 2 is the estimated one for one face. Searching four faces may require 4 times of it, about 8,000,000 cycles, to accomplish. As a result, executing on the hardware with clock rate of 100MHz, 8,000,000 clock cycles spend 80ms to get the result of four faces searching, which seems to meet the real result executed on the machine..

VI. Hardware Resource Usage

Fig. 3 and Fig. 4 are parts of the report of the hardware generated by Xilinx Platform Studio. In additional to that, 32 block RAMs are also used in the design.

VII. Conclusion

Although the hardware has a lower clock rate of 100MHz, it still has the performance much better than the software, which runs on a processor with clock rate 667MHz. That is the reason to implement some parts of a program on hardware. However, this advantage cannot be taken if there is too much communication between hardware and software to have the costs on performance. That's why the first acceleration method, which implement the whole match in hardware, removes almost all those communication and gains a great improvement

	Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1	Yes	TS_clk_fpga_0 = PERIOD TIMEGRP "clk_fpga_0" 100 MHz HIGH 50%	SET...	0.015ns 0.000ns	9.985ns	0 0	0 0
2	Yes	PATH "TS_axi_interconnect_1_reset_resync_path" TIG	SET...		1.874ns		0
3	Yes	PATH "TS_axi4lite_0_reset_resync_path" TIG	SET...		1.680ns		0

Fig. 3 The timing report

Device Utilization Summary (actual values)			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	32,582	106,400	30%
Number used as Flip Flops	30,112		
Number used as Latches	1		
Number used as Latch-thrus	0		
Number used as AND/OR logics	2,469		
Number of Slice LUTs	41,601	53,200	78%
Number used as logic	39,420	53,200	74%

Fig. 4 The hardware resource usage

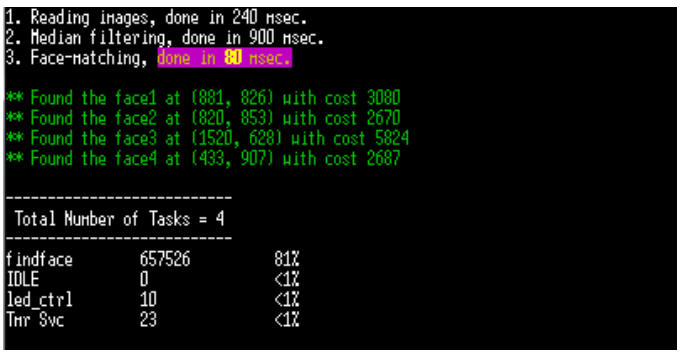

Fig. 5 The snapshot of the program execution.

on performance.

Either in software or hardware, tasks involved with I/O are always consuming the performance, including reading data from memory. In spite of that, some memory actions have to happen, at least once, in many situations. The only thing can be done for improve this is to reduce the repetitive access to same data. Caching is what helps doing this, but it is difficult to take benefit from it in the hardware; instead, it usually bothers the data accessing. As the result, the second acceleration method, which makes every pixel in the group picture loaded from memory only once for each face, achieves the best possible utilization of group pixels loaded from memory, and improves the performance the most.

To wait, to waste. As the third acceleration method, although the SAD computation cannot be completed in a single cycle, be good at using pipeline style computation can still make the best utilization of clock cycle that eliminate this disadvantage. On the other hand, the consideration of parallel computing is sometimes viewed as the reason that the hardware can beat the software a lot. In addition, not only parallel computing but also parallel doing computation and I/O can happen on the hardware. The fourth acceleration method take advantage of this, simultaneously computing SAD and reading pixels, and complete the last effort improving the performance.

As shown in the analysis previous made, after applying these acceleration, the time costs except for the computation is remaining only the time reading 4 1024-byte and 128 1920-byte memory regions, which is about 100,000 clock cycles or 1 microsecond. Since the utilization has reached the best, no more effective performance improvement can be made on the aspect relative with memory accessing, even discarding all the memory action. On the other hand, since the pipeline implementation has made the calculation for each position consume only amortized 1 clock cycle, only increasing the number of computing unit, which consumes hardware resource a lot, can help for acceleration. Considering the performance, the acceleration effect already made, and the hardware resource usage, the acceleration will be stopped here, for it is believed that the current design has a balance good enough to these factors.

Either the software or the hardware has jobs that are more suitable than the other, and both have the advantages doing its jobs. For example, the hardware has a better performance while the software has a better flexibility and a better development efficiency. The goal of hardware-software co-design is to take these benefits on developing programs, and a better hardware-software cooperation may lead to a better result, even not only on performance.

ACKNOWLEDGMENT

Thanks to the laboratory for providing the environment to develop this program, and thanks to teacher, teaching assistants, and classmates for inspiring me a lot.

REFERENCES

[1] Please refering to the report of lab1, Behavior Profiling on “Find Face”.

[2] Wikipedia, “Sum of absolute differences”, from http://en.wikipedia.org/wiki/Sum_of_absolute_differences.

[3] Please refering to the work done in lab3.

[4] The final project description document.

[5] The thought is inspired by 石奕心.