The Eighteenth International Symposium on Artificial Life and Robotics 2013 (AROB 18th '13),
Daejeon Convention Center, Daejeon, Korea, January 30-February 1, 2013

A describing method of latency tolerant hardware for a pure ANSI-C/C++ based high-level synthesis technology

Akira Yamawaki[1] and Seiichi Serikawa[1]

[1]Kyushu Institute of Technology, Japan
(Tel: 81-93-884-3255, Fax: 81-93-884-3214)

[1]yama@ecs.kyutech.ac.jp

Abstract: The image processing is important for the robotics and its hardware implementation is required in order to realize a small and low-power device with the appropriate performance where the high performance computer cannot be used due to the cost, size and power limitation. To reduce the burden of such hardware development, the high-level synthesis (HLS) technologies that automatically convert the algorithmic description to hardware have been proposed and developed. The combination of the memory latency hiding and data process pipelining is very important to extract the hardware performance maximally. However, nobody shows clearly how to describe the hardware behavior to generate such hardware. This paper shows a generic describing method for HLS technology based on ANSI-C/C++ that can realize the combination of the memory latency hiding and data process pipelining. The experimental results show that our method can be applied easily to the intuitive C program. The logic simulation and an FPGA implementation reveal the effects to the performance improvement and the hardware increase induced by our method.

Keywords: high-level synthesis, latency hiding, pipelining, hardware, image processing.

1 INTRODUCTION

The image processing, e.g. the object tracking, the object recognition, and so on, is important for the robotics. Hardware implementation of the image processing is required in order to realize a small and low-power device with the appropriate performance where the high performance computer cannot be used due to the cost, size and power limitation.

However, the hardware is developed by an expert using hardware description language (HDL), spending a significant burden. At last the large development load can realize the image processing hardware optimized to performance and power consumption.

To reduce such burden of hardware development, the high-level synthesis (HLS) technologies that automatically convert the algorithmic description to HDL program have been proposed and developed [1-5]. Most HLS tools employ the C language as the design entry. However, the design entries are C-like language, which is not pure ANSI-C/C++. Also, some HSL tools provide their own grammar suiting to represent the hardware nature. That is, the developers have to learn the dedicated C-like language although they need not to learn HDL.

In contrast, some HLS tools supporting ANSI-C/C++ as a design entry are provided recently [6], [7]. Instead of C-like language that describes parallelism and clock boundary of the hardware explicitly, the HLS tools supporting ANSI-C/C++ automatically extract them from the sequence of statements and loops in a pure C/C++ program.

Actually, the parallelism however cannot be extracted enough. So, the developer must indicate parallelism by pragmas explicitly. Typical and simple examples showing how to insert pragmas are shown by the venders of the HLS tool. But, some describing methods which are expected to generate a sophisticated hardware to improve the hardware performance are not shown mostly. For example, it is to combine the memory latency hiding and data process pipelining. This combination is very important to pull the performance of the hardware maximally.

Using an HLS tool (Vivado HLS of Xilinx [6]) supporting pure ANSI-C/C++, this paper shows a generic describing method of ANSI-C/C++ program that can realize the combination of the memory latency hiding and the data process pipelining. In this method, the memory loading part, the data processing part and the memory storing part are described isolated by FIFO passing the stream data. While the memory loading part and the memory storing part transfer the data to the memory, the data processing part processes the stream data via FIFO simultaneously. Thus, the memory access is overlapped with the data processing efficiently and naturally.

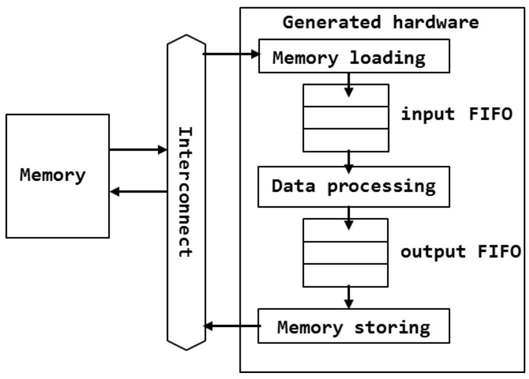

Fig. 1. Hardware model

In a case study to 3x3 image filter, we show that our method can be applied easily to the intuitive C program. In addition, through the logic simulation and implementation to an FPGA, we confirm the effects to the performance improvement and hardware increase induced by our method.

The rest of the paper is organized as follows. Section 2 shows a hardware model to which we map C program. Section 3 intuitively designs the hardware description in C to 3x3 image filter. Section 4 extends the intuitive C program mentioned above to combine the memory latency hiding and data process pipelining by our method. Section 5 evaluates the hardware overhead and the performance improvement induced by our method, compared with the intuitive C program. Finally, Section 6 concludes the paper.

2 HARDWARE MODEL

Fig. 1 shows a hardware model to which we map C program. The organization of this model is commonly used. Thus, any hardware designer can accept to employ this model.

This model is decomposed to the memory loading part, the data processing part and the memory storing part. These parts are separated by FIFOs. They run completely in parallel, communicating via FIFO streamly each other.

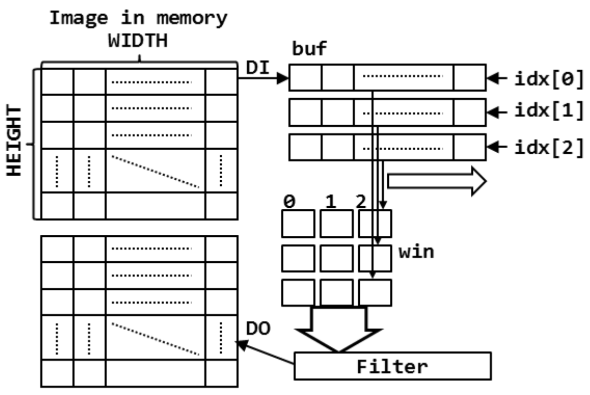

Fig. 2. Image of Data process

```
 1:void Top(uint32_t *DI,uint32_t *DO)
 2:{
 3:  int i, j;
 4:  static uint32_t buf[3][WIDTH];
 5:  static uint32_t win[3][3];
 6:  uint8_t  idx[3];
 7:  uint8_t  tmp;
 8:  uint32_t result;
 9:  idx[0]=0; idx[1]=1; idx[2]=2;
10:  for(i=0;i<HEIGHT;i++){
11:    for(j=0;j<WIDTH;j++)
12:      buf[idx[2]][j]=DI[i*WIDTH+j];
13:    for(j=0;j<WIDTH;j++){
14:      Win_Shift_Right(win);
15:      win[0][2]=buf[idx[0]][j];
16:      win[1][2]=buf[idx[1]][j];
17:      win[2][2]=buf[idx[2]][j];
18:      Filter(win,&result);
19:      DO[i*WIDTH+j]=result;
20:    }
21:    tmp=idx[0];idx[0]=idx[1];
22:    idx[1]=idx[2];idx[2]=tmp;
23:  }
24:}
```

Fig. 3. Intuitive C program

The memory loading part loads the data to be processed in a memory into the input FIFO. The data processing part gets the input data from the input FIFO, processes the input data and pushes the processed data into the output FIFO. The memory storing part pops the processed data from the output FIFO and stores them into the memory.

The data processing part processes the stream data through its own pipelined data path, while the memory loading and storing parts perform the memory accesses. Thus, the memory latency is hidden by the pipelined data processing.

This model is trivial but nobody has explained clearly how to write to generate such hardware. That is, since a generic describing method does not exist, the designer has to write the hardware behavior in C carefully by its own talent so that the hardware model as shown in Fig. 1 is generated.

3 INTUITIVE METHOD

To confirm that our method can be applied easily to the intuitive C program, we describe the case study of 3x3 image filter in this section. Fig. 2 depicts the data processing image of 3x3 image filter and Fig. 3 describes an intuitive C program of the hardware behavior.

For the input image, a column is loaded into the top of the internal ring buffer along the image height as the lines of 11 to 12 in Fig. 3.

To the buffer holding columns, a 3x3 image filter is performed as the lines of 14 to 18 in Fig. 3. 3 pixels in the 3 rows currently indexed in the buffer are pushed into the 3x3 window. Then, an image filter processes this 3x3 window. The result of the image filter is stored into the memory as the line of 19 in Fig. 3.

Once the 3x3 image filter finishes on the buffer, the indexes to banks of the buffer are updated to load a next column to new top entry as the lines of 21 to 22 in Fig. 3.

Inserting pragma, the pipeline structure can be extracted by an HLS tool, Vivado HLS 14.3 [6]. For example, the inner loops of the line 11 and the line 13 are converted to the pipeline hardware. However, the parallel feature among the memory access and data processing cannot be extracted by any pragma. Although we cannot analyze the internal processes of the Vivado HLS in detail, the data dependency over some arrays may be a reason to obstruct the parallelization. That is, the C program must be reconstructed to extract the parallelism between the memory access and data process.

4 PROPOSED METHOD

Our proposed method is very simple and easy way to reconstruct the intuitive C program. Fig. 4 shows the C program of which the intuitive C program shown in Fig. 3 is reconstructed by our method.

The program basically consists of 3 functions which are the memory loading (*mem_load*), the data processing (*data_proc*) and the memory storing (*mem_store*). These functions are connected by the FIFOs. This structure is equal to that of Fig. 1.

Each function runs individually from the top to the bottom of the image. To make them run in parallel, the DATAFLOW pragma [6] is assigned to the function *Top* in order to extract the parallelism among the functions. In addition, the data dependency among the functions has to be cut. Thus, the streaming FIFOs, iFIFO and oFIFO which are bridges among the functions are defined in the function *Top*.

The *mem_load* pushes the pixel in the memory into the iFIFO until the iFIFO becomes full.

Simultaneously, the *data_proc* pops the iFIFO and stores the popped data into the top of the ring buffer while the iFIFO is not empty. Once the top of the ring buffer becomes full, the data_proc performs 3x3 image filter as same as Fig. 3. However, the processed data is pushed into the oFIFO instead of the memory. The pipelining is also performed to the inner loops of the same as Fig. 3 by the PIPELINE pragma [6].

In addition, the *mem_store* pops the oFIFO and stores the popped data into the memory in parallel.

```
void mem_load(uint32_t mem[HEIGHT*WIDTH],
                  stream<32>& iFIFO ){
  int i, j;
  for(i=0;i<HEIGHT;i++)
    for(j=0;j<WIDTH;j++)
      //Push iFIFO
      iFIFO.write(mem[i*WIDTH+j]);
}
void data_proc(stream<32>& iFIFO,
                  stream<32>& oFIFO){
  int i, j;
  static uint32_t buf[3][WIDTH];
  static uint32_t win[3][3];
  uint8_t  idx[3];
  uint8_t  tmp;
  uint32_t dat, result;
  idx[0]=0; idx[1]=1; idx[2]=2;
  for(i=0;i<HEIGHT;i++){
    for(j=0;j<WIDTH;j++){
      iFIFO.read( dat ); //Pop iFIFO
      buf[idx[2]][j]=dat;
    }
    for(j=0;j<WIDTH;j++){
      Win_Shift_Right(win);
      win[0][2]=buf[idx[0]][j];
      win[1][2]=buf[idx[1]][j];
      win[2][2]=buf[idx[2]][j];
      Filter(win,&result);
      oFIFO.write(result); //Push oFIFO
    }
    tmp=idx[0];idx[0]=idx[1];
    idx[1]=idx[2];idx[2]=tmp;
  }
}
void mem_store(uint32_t mem[HEIGHT*WIDTH],
                  stream<32>& oFIFO ){
  int i, j;
  uint32_t tmp;
  for(i=0;i<HEIGHT;i++)
    for(j=0;j<WIDTH;j++){
      oFIFO.read(tmp); // Pop oFIFO
      mem[i*WIDTH+j]=tmp;
    }
}
void Top(uint32_t *DI, uint32_t *DO){
  int i, j;
  static stream<32> iFIFO, oFIFO;
  mem_load (DI    , iFIFO);
  data_proc(iFIFO, oFIFO);
  mem_store(DO    , oFIFO);
}
```

Fig. 4. C program reconstructed by our me

Using our method, the memory loading and storing parts are accessing the memory while the data processing part is processing the data fast by the pipelining. That is, our method can naturally express the hardware which hides the memory latency to extract the performance of the pipelined hardware maximally.

5 EXPERIMENT AND DISCUSSION

5.1 Performance evaluation

To evaluate the performance of the hardware to which Vivado HLS 14.3 converts the C program, we have

Table 1. Hardware cost

Resolution	FIFO	LUT	FF	BRAM	Average
QVGA	256	1.47	1.56	1.33	1.45
VGA	512	1.63	1.48	1.33	1.48
720P	1024	1.53	1.51	1.44	1.49
SXGA	1024	1.49	1.48	1.44	1.47
1080P	1024	1.49	1.48	1.44	1.47

performed the logic simulation on the Modelsim 10.1c of Mentor Graphics. We set the clock frequency to 100MHz.

Fig. 5 shows the experimental results. The ORG of the horizontal axis means the intuitive version shown in Fig. 3. Also, the PAR indicates the version that is modified by our method shown in Fig. 4. The values on our method mean the depth of the input and output FIFOs. The vertical axis indicates the normalized execution time to the intuitive version. The data are measured over several resolutions of the input image. The filter was the average filter.

The result shows that our method can improve the performance compared with the intuitive version by the combination of the memory latency hiding and the data process pipelining. The depth of FIFO affects the performance improvement due to our method. The deeper, the better. However, the speedup is saturated by about 1.5 times. That is, the optimum depth exists to improve the performance maximally, suppressing the hardware cost.

5.2 Hardware Overhead

To improve the performance maximally while suppressing the hardware cost, the optimum point of the FIFO depth exists as mentioned above. Thus, we confirm that the hardware cost at the boundary point of the FIFO depth is compensated by the performance improvement compared with the hardware size of the intuitive version. To do this, we have implemented the hardware into an FPGA, Spartan-6 of Xilinx. The used implementation tool is ISE 14.3. Table. 1 shows the result of this evaluation.

FIFO means the depth of the i/o FIFOs. The LUT means the number of lockup tables realizing the combinational logic. The FF means the number of flip-flops. The BRAM means the number of embedded memories in the FPGA.

The result shows that the investment of hardware resources increased by our method is appropriate enough for the performance improvement of 1.5 times.

6 CONCLUSION

We have proposed a generic describing method of a latency tolerant hardware for a pure ANSI-C/C++ based high-level synthesis technology. Through the case study that uses 3x3 image filter, we have shown that our method can be introduced easily into the intuitive C program. Also, it has been confirmed that the performance improvement can be achieved by the moderate hardware investment.

As future work, we will evaluate our method to more application programs and apply it to more HLS tools.

REFERENCES

[1] Mitrionics (2008), Mitrion User's Guide 1.5.0-001, Mitrionics.

[2] Pellerin D and Thibault S (2005), Practical FPGA Programming in C, Prentice Hall.

[3] Lau D, Pritchard O and Molson P (2006) Automated Generation of Hardware Accelerators with Direct Memory Access from ANSI/ISO Standard C Functions, IEEE Symp. on Field Programmable Custom Computing Machines, pp.45-56.

[4] Mentor Graphics (2012), Handel-C Synthesis Methodology, http://www.mentor.com/products/fpga/han-del-c/

[5] Yamawaki A and Masahiko I (2011) High-level Synthesis Method Using Semi-programmable Hardware for C program with Memory Access, Engineering Letters, Vol.19, Issue 1, pp.50-56.

[6] Xilinx (2012) Vivado Design Suite User Guide High-Level Synthesis, Xilinx user guide, UG902 (v2012.2).

[7] Calypto (2012) Designing High Performance DSP Hardware Using Catapult C Synthesis and the Altera Accelerated Libraries, Calypto White Paper, WP-0004 05-2012, http://calypto.com.

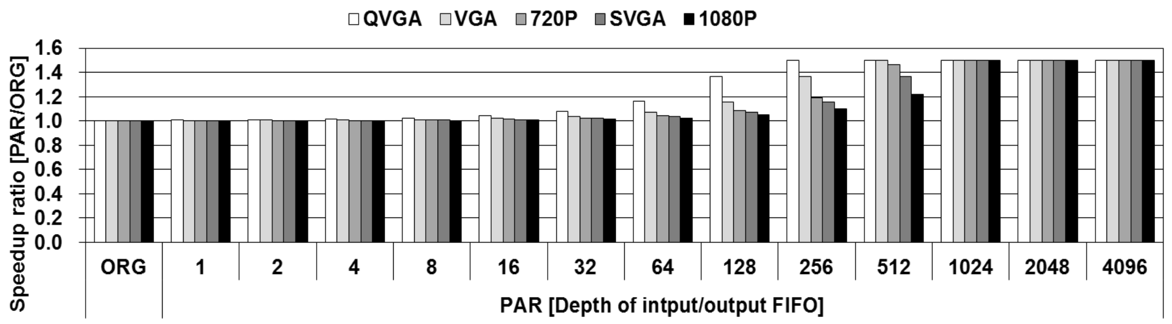

Fig. 5. Performance evaluation