Hardware Acceleration with Multi-Threading of Java-Based High Level Synthesis Tool

Yuto Ishikawa
Tokyo University of Agriculture and Technology
ishikawa@nj.cs.tuat.ac.jp

Keitaro Yanai
Tokyo University of Agriculture and Technology
yanai@nj.cs.tuat.ac.jp

Keisuke Koike
NIKON CORPORATION
koike@nj.cs.tuat.ac.jp

Takefumi Miyoshi
WasaLabo, LLC.
6-5-20 Minaminaruse, Machida
Tokyo 194–0045 Japan
miyo@wasa-labo.com

Hironori Nakajo
Tokyo University of Agriculture and Technology
2–24–16 Naka-cho Koganei,
Tokyo 184–8588 Japan
nakajo@cc.tuat.ac.jp

ABSTRACT

In this research, we attempt to speed up the computational fluid dynamics (CFD) and the convolutional neural network (CNN) using JavaRock-Thrash thread function of the high-level synthesis tool with an FPGA. In the two-dimensional heat equation, by using the thread function of the high-level synthesis tool, up to a 12.13 times speedup compared to single-threaded processing is obtained with multi-threading, up to a 29.0 times speedup against Vivado HLS is achieved. In the convolution process, the process of passing 11×11 filters on 2-dimensional data of 33×33 described with 484 threads results in a speedup of 95.0 times compared to the processing time at Vivado HLS.

Keywords

Reconfigurable computing, FPGA, hardware acceleration

1. INTRODUCTION

In recent years, researches on hardware acceleration using an FPGA (Field Programmable Gate Array) have been attracting attention. In addition to hardware description language (HDL), hardware design for an FPGA has been using high-level synthesis tool which enables hardware design with a higher abstraction language such as Java language or C language [1][2][3].

Especially, Convolutional Neural Network (CNN) is an important technology in applications such as handwritten character recognition and voice print analysis, and 90% of the execution time of CNN is occupied with convolution processing[4]. In this research, we have implemented and evaluated hardware acceleration of Computational Fluid Dynamics (CFD) and Convolutional Neural Network on FPGA using thread function of JavaRock-Thrash.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

In the rest of this paper, the section 2 introduces the related research. In the section 3, high level synthesis tool of C language and Java language system and explanation of high level synthesis language are explained. In section 4, targeted applications of hardware acceleration for high-performance computing by an FPGA are described. Section 5 shows the results and considerations of hardware acceleration. In the section 6, we summarize our research.

2. RELATED WORKS

JAXA ported their FORTRAN programs of CFD, which are practically used on supercomputers, into HDL against the technical problem in practical CFD code implementation of an FPGA[5]. In their research, technical problems in FPGA design are experimentally extracted.

Speed up stencil calculation using Vivado HLS as a high-level synthesis tool has been tried[6]. This study reveals the cases where the optimization directives such as pipelining and inline unrolling that are functions of Vivado HLS are effective and non-effective. Fine grain parallelism is secured using pipelining and inline expansion. However, since the coarse grain parallelism is not secured, the parallel processing effect which an FPGA is good at is not fully utilized.

In a research by Chen Zhang et al. on high-speed convolution calculation using high-level synthesis[7], implementation of deep Convolutional Neural Network using Vivado HLS is also attempted. About 90 % of calculation time in the Convolutional Neural Network is occupied by a calculation in a convolution layer, in this paper, to speed up convolution processing is focused. The feature of the proposed method in this paper is that the circuit resources and memory bandwidth are optimized for maximum use. As a result, the implemented CNN accelerator performs 61.62 GFLOPS, which is about 17 times faster compared with the execution time on a CPU. Though this research also includes loop pipelining and inline expansion to exploit fine grained parallelism, coarse grain parallelism is not secured.

3. HIGH LEVEL SYNTHESIS

There are many high-level synthesis tools including commercial and research use. MaxCompiler [8], Sea Cucumber and others are included as Java language based systems. C languages based systems include Vivado HLS, Impulse C, Stream C[9], CyberWork Bench [10] and others. There are the following differences when using a high level synthesis tool of C language based system and Java language one.

- Java language has a thread function as a specification, but it is not in C language. The specification makes it easy to exploit coarse-grain parallelism in designing efficient circuits in an FPGA.
- Java is an object-oriented language and C language is a procedural one. Due to the object-oriented characteristics, since Java classes and objects can correspond to modules and sub-modules of HDL, circuit description becomes easy.

Because of the features of Java language, a high-level synthesis tool based on Java language is compatible with FPGA development.

In this research, we have adopted JavaRock-Thrash[11] as a high-level synthesis tool focusing on a thread function in particular, and implemented various high-performance calculations using multi-threading.

3.1 Java-based high-level synthesis tool: JavaRock-Thrash

JavaRock-Thrash is a high-level synthesis tool that generates Verilog HDL from Java source code executable as software. The reason for using JavaRock-Thrash of the high-level synthesis tool in this research is as follows.

- Java thread can be used to describe parallel operation
- Optimizing directives (e.g. loop unrolling) are available
- Supporting single/double precision floating point operation

Features of JavaRock-Thrash, are shown in the next section.

3.2 Features of JavaRock-Thrash

Because JavaRock-Thrash has not been extended from the original Java language specification, hardware can be designed with the conventional Java language description. However, the tool does not support recursion, generation of dynamic instances etc. In addition, JavaRock-Thrash has a function dedicated for hardware implementation by annotation. For example, when a loop statement is unrolled, the source code is described as shown in Listing 1.

Listing 1: Description of loop unrolling

```
1 @JRThrashUnroll(
2     unrollNum = 4,
3     loopVariableName = "i",
4     unrollType = JRThrash.copyLoopVar)
5 public void multStream(){
6     for(int i=0;i<128;++i){
7         c[i] = a[i] * b[i];
8     }
9 }
```

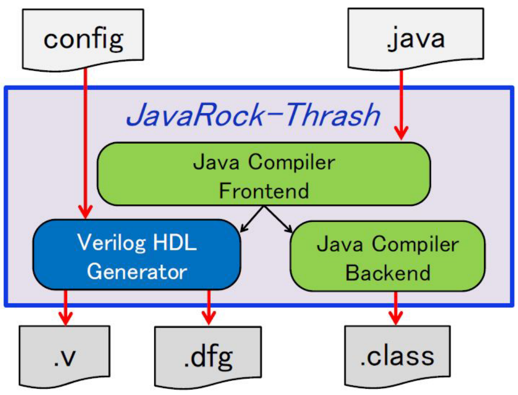

Figure 1: Compile flow of JavaRock-Thrash

Loop unrolling is conducted by the annotation described in the 1st to 4th lines. In addition, it is possible to specify the unrolling number of the loop, the specification of the loop variable and the type of loop unrolling as options. This loop unrolling improves fine grain parallelism.

Figure 1 shows the compilation flow of JavaRock-Thrash.

JavaRock-Thrash generates Java class files and Verilog HDL files using Java source files and config files as input files. In the config file, setting for generating the circuit such as parameter of arithmetic logic IP etc. is described. Since the class file can be executed on a Java Virtual Machine, it is possible to verify operation of the generated circuit.

Next, a description example of a hardware design using a Java thread is described. By using Java threads, since multiple modules operate in parallel, spatial parallelism is secured. The following source code shown in Listing 2 is an example of description for parallel processing using Java thread.

Listing 2: Description example of Java thread

```
class Top{
    final SubClass subA = new SubClass();
    final SubClass subB = new SubClass();

    void start(){
        //start thread processing
        subA.start();
        subB.start();

        try{
            //wait for thread finishing
            subA.join();
            subB.join();
        }catch(Exeption e){}
    }
}

class SubClass extends Thread{
    public void run(){
        ...
    }
}
```

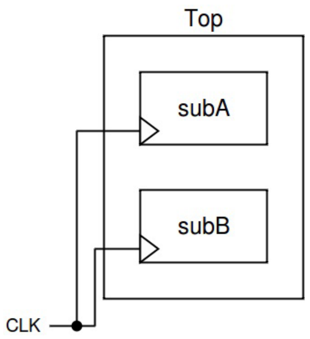

Figure 2: Module example using threads

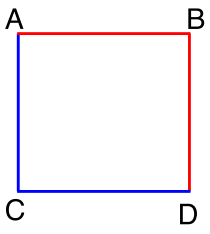

Figure 3: Heat plate conduction

In this description example, two SubClass modules are instantiated inside the TopClass module by declaring objects subA, subB of class SubClass by the class Top. Since these two SubClass modules operate in parallel, spatial parallelism can be exploited by using Java threads. When the source code of Listing 2 is compiled with JavaRock-Thrash, the module is shown in Figure 2.

As the same clock is supplied, subA and subB are processed in parallel synchronously. As a caveat, the circuit resources of subA and subB are not shared. For example, subA can not directly access the memory defined by the module of subB. In such a case, a shared memory in the Top module should be created to be available for subA and subB modules for communication.

4. EVALUATED APPLICATIONS

In this section, we describe two-dimensional heat equation used in computational fluid dynamics as an example of high-performance computation and convolution processing in convolution layer of Convolutional Neural Network. We also describe a method of implementing these calculation processes using the thread function of the high-level synthesis tool.

4.1 Computational Fluid Dynamics (CFD)

In this study, hardware acceleration of CFD using FPGA with high-level synthesis tool's thread function has been implemented and evaluated. The targeted fluid equation is a two-dimensional heat equation in which stencil calculation is used. In the two-dimensional heat equation adopted in this research, a thin plate which conducts heat well as shown in Figure 3 is assumed.

We have adopted a simulation program which calculates temperature change due to the passage of time when the sides AB and BD are heated and the sides AC and sides CD are cooled. Stencil calculation, which is one of methods for finding approximate solutions of partial differential equations, is used in the two-dimensional heat equation. An example of two-dimensional stencil calculation is shown in Figure 4.

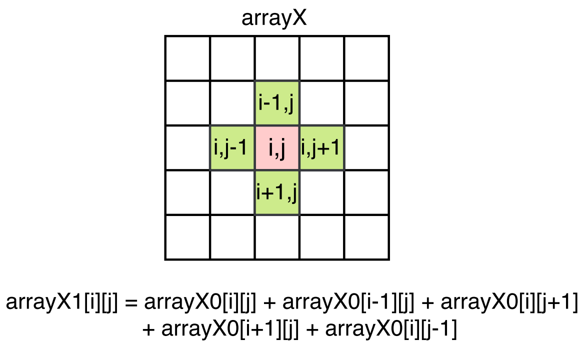
arrayX
i-1,j
i,j-1
i,j
i,j+1
i+1,j
arrayX1[i][j] = arrayX0[i][j] + arrayX0[i-1][j] + arrayX0[i][j+1] + arrayX0[i+1][j] + arrayX0[i][j-1]

Figure 4: Stencil calculation

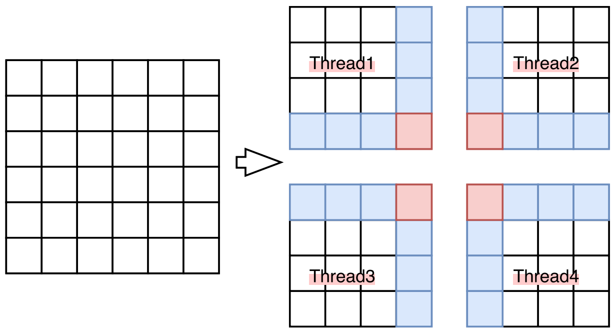

Figure 5: Array partitioning for stencil calculation

Let arrayX of the current time be arrayX0 and arrayX of next time be arrayX1. To obtain the value of arrayX1[i][j], the current time value arrayX0[i][j], arrayX0[i][j+1], arrayX0[i+1][j], arrayX0[i][j-1] and arrayX0[i-1][j] are used.

Next, we describe the method of stencil calculation using threads. Consider stencil calculation with 4 threads as an example. The division method is shown in Figure 5.

The shaded area in the array after partition of Figure 5 is called as sleeve area. In the stencil calculation, values of adjacent regions are required, so data for Thread 2 and Thread 3 are required for calculation in Thread 1. Therefore, by storing the required data in the sleeve area, multithread processing of the stencil calculation is realized. In this study, two dimensional heat equations are processed with a maximum of 32 threads.

4.2 Convolutional Neural Network (CNN)

CNN is drawing attention especially in the field of image recognition. CNN is an extension of the classical multilayered perceptron, but models the knowledge in the structure of the visual cortex of the brain. The structure of CNN is shown in Figure 6[12].

From the Figure 6, it is structured to repeat the convolution layer that extracts features of two-dimensional data and the pooling layer that summarizes amount of features.

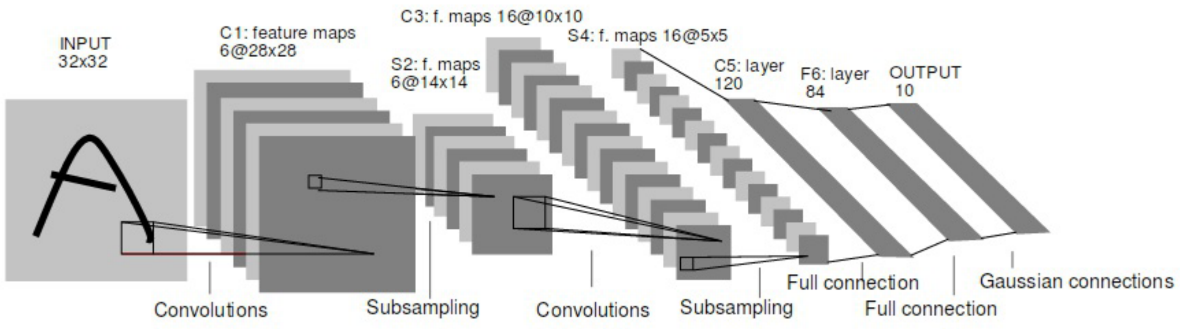

Figure 6: Convolutional Neural Network

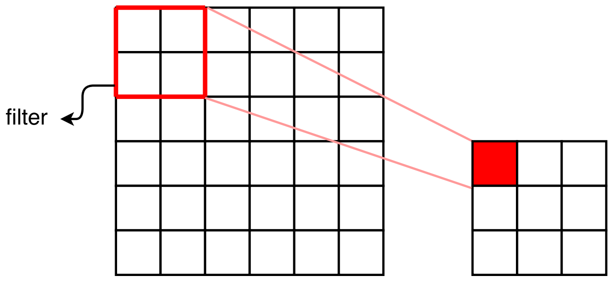

Figure 7: Outline of convolution (1)

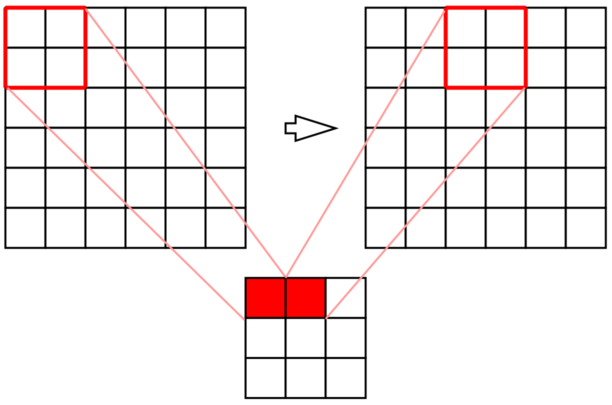

Figure 8: Outline of convolution (2)

For CNN, convolution processing of the convolution layer occupies 90% of the total execution time. Therefore, in this research, convolution processing using JavaRock-Thrash thread function is performed.

The convolution here refers to a calculation used for image processing. This convolution operation is the basis of image processing since it can make images smoother and sharper. It is convolution to take a small filter area on two-dimensional data and compress it as one feature amount. A schematic diagram of convolution is shown in Figure 7.

CNN repeats the process shown in Figure 7 while moving the filter like Figure 8. The shifting stride of the filter in this implementation is 2, and the filter of 2×2 is passed through for the two-dimensional data of 6×6. Figure 9 shows a calculation example of the convolution. The processing of Figure 9 is represented by source code shown in Listing 3.

Listing 3: Program example of convolution

```
for(int i_data=0;i_data<result_height;i_data++){
 for(int j_data=0;j_data<result_width;j++){
  for(int i_filter=0;i_filter<height_filter;i_filter++){
   for(int j_filter=0;j_filter<width_filter;j_filter++){
     result[i_data][j_data] +=
       data[i_data+i_filter][j_data+_j_filter]
       * filter[i_filter][j_filter];
   }
  }
 }
}
```

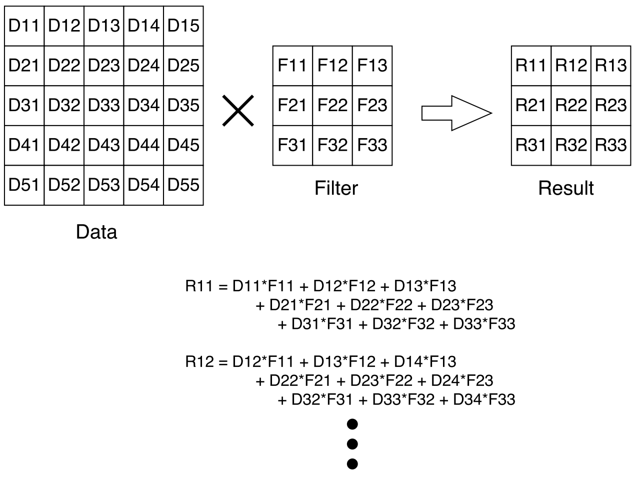
D11 D12 D13 D14 D15
D21 D22 D23 D24 D25
D31 D32 D33 D34 D35
D41 D42 D43 D44 D45
D51 D52 D53 D54 D55
Data
F11 F12 F13
F21 F22 F23
F31 F32 F33
Filter
R11 R12 R13
R21 R22 R23
R31 R32 R33
Result
R11 = D11*F11 + D12*F12 + D13*F13 + D21*F21 + D22*F22 + D23*F23 + D31*F31 + D32*F32 + D33*F33
R12 = D12*F11 + D13*F12 + D14*F13 + D22*F21 + D23*F22 + D24*F23 + D32*F31 + D33*F32 + D34*F33

Figure 9: Calculation example of the convolution

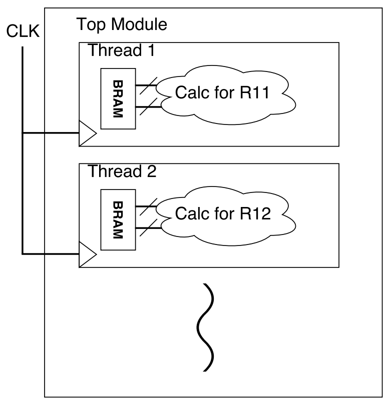

Figure 10: Modules for convolution with multi-threading

Next, a convolution operation method using a thread is explained by taking the case of Figure 9 as an example. In this case, parallel processing is performed by calculating each calculation to calculate the values of R11 to R33 for each thread. In the calculation, the number of threads is 3×3 pieces. A module diagram of thread convolution calculation is shown in Figure 10.

An example of source code of processing in each thread is shown below.

Listing 4: Program example in each thread for convolution

```
for(int i=0;i<heigth_filter;i++){
  for(int j=0;j<width_filter;j++){
      result += data[i][j] * filter[i][j];
    }
}
```

Since the processing is distributed to each thread, it can be seen that the calculation amount is reduced as compared with the source code in Listing 3.

5. EVALUATION RESULTS

For evaluation, the language used for the high-level synthesis tool Vivado HLS is C language while Java is used for JavaRock-Thrash. In the description with Vivado HLS, pragmas of PIPELINE and UNROLL are added to the source code as the least and general optimization. Both of the performance of these two high level synthesis tools are compared to show the performance improvement of JavaRock-Thrash.

5.1 Evaluation conditions

As a specification of the program which implements two dimensional heat equation, the data size is changed to 128×128 and the number of threads varies 1, 2, 4, 8, 16, 32 and evaluated.

As a specification of the convolution operation program, the data size is 32×32, the filter size is 11×11, the stride number is 1 and the number of threads is fixed to single and 484.

For both 2D thermal conduction equation and convolution operation, the target FPGA is xcku035 of Xilinx's Kintex UltraScale series. Vivado 2016.3 is used for logic synthesis and simulation of Verilog HDL files.

The evaluation environment is as shown in the Table 1.

Table 1: Evaluation environment

CPU	Intel Core i5-4440 3.10GHz
Memory	8GB
OS	Ubuntu 16.04
Compiler	javac 1.8.0

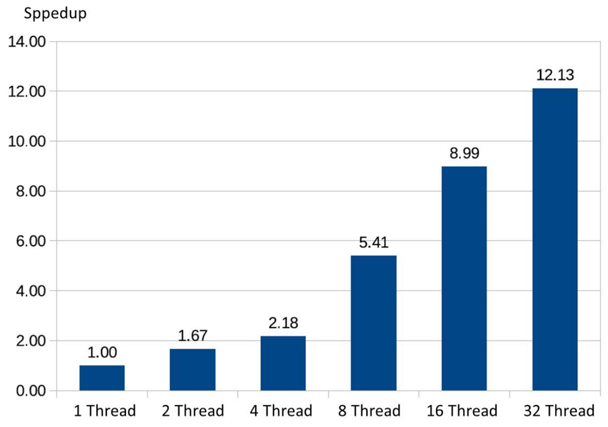

Figure 11: Results with JavaRock-Thrash

5.2 Evaluation results and study

Figure 11 shows the evaluation results of the two-dimensional heat equation program in JavaRock-Thrash. The speeding-up rate is compared with the processing time in a single thread. The result shows that the performance improves with each division into multiple threads.

Table 2 and Table 3 show the convolution operation implemented by JavaRock-Thrash and the result implemented with Vivado HLS. The speeding-up rate is a numerical value in comparison with Vivado HLS.

From Table 2, JavaRock-Thrash implementation for convolution using multi-threading performs about 95 times faster than Vivado HLS without thread function.

In convolution operation, in addition to the parallelizing effect by multi-threading, since data communication between threads is unnecessary, the data access to BRAM has not become a bottleneck.

Table 4 summarizes the data of the Table 2.

Table 2: Evaluation of Vivado HLS(C Language) and JavaRock-Thrash (Performance)

	Fmax [MHz]	Cycles	Exec Time [ms]	Speedup
CPU	-	-	0.35	-
Vivado HLS(C)	153	293643	0.19	-
JRT1 Thread	171	945740	0.55	0.35
JRT484 Threads	103	2435	0.002	95.0

Table 3: Evaluation of Vivado HLS(C Language) and JavaRock-Thrash (Resources)

	reg	LUT
CPU	-	-
Vivado HLS(C)	544	822
JRT1 Thread	280	224
JRT484 Threads	285115	184436

Table 4: Comparison of cycles and execution time

	Cycles ratio	Sppedup ratio
JRT1 Thread/ JRT484 Threads	388.3	175
JRT1 Thread/ Vivado HLS(C)	3.22	2.89
Vivado HLS(C)/ JRT484 Thread	120.5	95

Nevertheless, comparing the number of processing cycles and processing time in a single thread of JavaRock-Thrash with the implementation in Vivado HLS, the performance of JavaRock-Thrash is inferior. The reason is that the number of processing cycles in a single thread of JavaRock-Thrash is 3.22 times larger than that in Vivado HLS as seen from the Table 4.

From the Table 4, in a single thread of JavaRock-Thrash and 484 threads, the processing cycle number ratio is about 175 times. As well as Coarse-grain parallelism reduced amount of calculation has brought improvement to speedup.

Although coarse grain parallelism can be improved by using multi-threading, another merit of using multi-threading is found. Comparing source code of Listing 3 and Listing 4, by using thread function, computational complexity has been reduced in each thread. Moreover, there are merits such as easy implementation and ease of implementation.

6. CONCLUSION

In this paper, for the hardware acceleration using an FPGA, JavaRock-Thrash of Java language based HLS with thread function has been adopted to implement and evaluate two-dimensional heat equation and convolutional operation program.

In the two-dimensional heat equation, performance improvement has been gained as compared with processing with a single thread by using thread function. In the convolution operation, when convolutional operation has been implemented using JavaRock-Thrash, the evaluated results shows superiority to Vivado HLS in processing cycle numbers and processing time.

Since JavaRock-Thrash is not extended from the conventional Java language, design with an FPGA can be developed with usual Java description. Therefore hardware acceleration by an FPGA of high-performance calculation program using high-level synthesis tool can be effective.

Especially, in the convolutional operation, using the multi-thread function has greatly improved the performance. In the future we need to tackle practical hardware acceleration by an FPGA of whole Convolutional Neural Network. Since there is a possibility that the circuit may not fit in the FPGA due to an increase in the circuit scale, it is necessary to consider architecture for efficient memory access considering data communication with the external memory.

Acknowledgements

This work was partially supported by MEXT Fund for Smart Space Technology toward Sustainable Society and by JSPS Grant-in-Aid for Scientific Research (C) Number 16K00078.

7. REFERENCES

[1] Xilinx. Vivado HLS: http://japan.xilinx.com/products/design-tools/vivado.html (last accessed on Apr 30, 2017).

[2] J. L. Tripp, P. A. Jackson and B. Hutchings: Sea Cucumber: A Synthesizing Compiler for FPGAs, in Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications (FPL'02) pp.875–885, 2002.

[3] D. Pellerin and S. Thibault: Practical FPGA Programming in C, first edition, Prentice Hall Press, 2005.

[4] J. Cong and B. Xiao: Minimizing computation in convolutional neural networks. In Artificial Neural Networks and Machine Learning-ICA NN 2014, pp.281–290, 2014.

[5] Naoyuki Fujita, Takashi Nakamura, Yuichi Matsuo, Katsumi Yazawa, Yasuyuki Shiromizu and Hiroshi Okubo: Feasibility Study of CFD Code Acceleration using FPGA, In Supercomputing, 2007.

[6] Hiroki Nakasone, Yasunori Osana, Yasunori Nagata: A feasibility study on implementing numerical applications on FPGAs using Vivado HLS, IPSJ SIG Technical Reports SLDM, Vol.2015, No.26, pp.1–6, 2015 (in Japanese).

[7] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, Jason Cong. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'15) pp.161–170, 2015.

[8] Maxeler Technologies, MaxCompiler White Paper: https://www.maxeler.com/media/documents/MaxelerWhitePaperMaxCompiler.pdf (last accessed on Apr 30, 2017).

[9] M. B. Gokhale, J. M. Stone, J. Arnold and M. Kalinowski: Stream-Oriented FPGA Computing in the Streams-C High Level Language, in Proceedings of the 2000 IEEE symposium on Field-Programmable Custom Computing Machines (FCCM '00) pp.49–56, 2000.

[10] NEC: Cyber WorkBench: http://www.nec.com/en/global/prod/cwb/ (last accessed on Apr 30, 2017)

[11] JavaRock-Thrash: http://sourceforge.net/p/javarock/javarock-thrash/ (last accessed on Apr 30, 2017)

[12] Y.LeCun, L. Bottou, Y.Bengio, and P.Haffner: Gradient-based learning applied to document recognition, Proc. of the IEEE, pp.2278–2324, 1998.