# Accelerating Real-time Processing of Articles by Using an OpenCL-based FPGA for the OSS Syntactic Parser SyntaxNet

Yoshiki Kurokawa, Yuichiro Aoki, Yuki Kondo, Yaoko Nakagawa
Research and Development Group, Center for Technology Innovation - Digital Technology
Hitachi, Ltd.
1-280, Higashi-Koigakubo, Kokubunji, 185-8601, Tokyo, Japan
Email: yoshiki.kurokawa.ee@hitachi.com, yuichiro.aoki.jk@hitachi.com, yuki.kondo.fe@hitachi.com, yaoko.nakagawa.gn@hitachi.com

***Abstract*— To improve customer satisfaction is necessary to provide services that enable real-time responses to complaints for call-center operations. The real-time parsing of complaint documents is more important for the real-time responses. Using the Open Source Software (OSS) syntactic parser SyntaxNet as a vehicle, a high-speed method using FPGA and OpenCL to achieve throughput of 700 words/s (required for real-time processing) is proposed. According to the results of the SyntaxNet analysis, matrix size (which changes dynamically according to the progress of the analysis) was found to be a performance determining factor. The proposed method was evaluated using public data, and the evaluation results confirmed throughput of 661 words/s, which almost met the requirement. As a result, the prospect of realization of a real-time complaint document analysis service for call centers was obtained.**

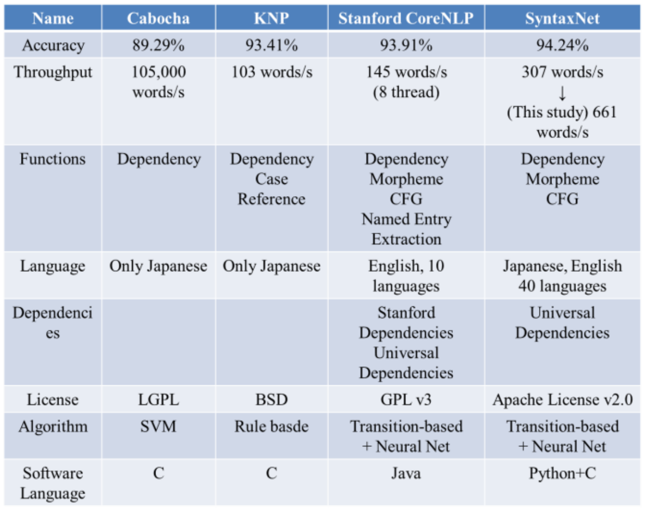
***Keywords-FPGA; OpenCL; Syntactic parser.***

## I. INTRODUCTION

For all companies, customer complaints point out problems with products and services of companies, and they provide important information for developing products and services with higher quality. In general, a complaint from a customer is first classified and extracted from a large amount of inquiry information. The content of the complaint is then analyzed, and countermeasures are investigated by customer-complaint analysis. Parsing is the most-important process for classification and analysis of complaints. The result of parsing is the input for a series of analysis processes such as grasping meaning, classifying sentences, and summarizing contents [1]. Referring to the “three-second rule”, [2] that is the rule of the response time of a web site, the parsing complaint sentences part takes at least one second of the entire complaint-classification process (taking three seconds). Since the size of the complaint text is unknown, the text size is assumed as a general text size. The general text size of English-language news articles and magazine articles is an average of 500 words and 900 words, respectively [3]. If a general article is assumed as a news article or a magazine article, so general text size is assumed 700 words length, that is taken between 500 words of a news article and 900 words of a magazine article. The required processing throughput would be 700 words/s to process one document per second. Therefore, the goal of this study is to increase the processing throughput of the syntactic parser to 700 words/s.

Most of the appreciation of syntactic parser, immediate

TABLE I. SYNTACTIC PARSER COMPLISON



| Name | Cabocha | KNP | Stanford CoreNLP | SyntaxNet |
|---|---|---|---|---|
| Accuracy | 89.29% | 93.41% | 93.91% | 94.24% |
| Throughput | 105,000 words/s | 103 words/s | 145 words/s (8 thread) | 307 words/s ↓ (This study) 661 words/s |
| Functions | Dependency | Dependency Case Reference | Dependency Morpheme CFG Named Entry Extraction | Dependency Morpheme CFG |
| Language | Only Japanese | Only Japanese | English, 10 languages | Japanese, English 40 languages |
| Dependenci es | | | Stanford Dependencies Universal Dependencies | Universal Dependencies |
| License | LGPL | BSD | GPL v3 | Apache License v2.0 |
| Algorithm | SVM | Rule basde | Transition-based + Neural Net | Transition-based + Neural Net |
| Software Language | C | C | Java | Python+C |

processing is required at an edge computer. Processing at the edge computer requires a hardware accelerator with low power consumption and excellent processing capability. Among hardware accelerators, a Field-Programmable Gate Array (FPGA) is known to have low power consumption and high power efficiency. As a logic circuit, an FPGA enables offload processing, so it is structurally power efficient. This study’s purpose is to speed up OSS application SyntaxNet by FPGA, and to check offload feasibility.

The contributions of this study are that after probing SyntaxNet, we found that most of the SyntaxNet execution time is spent on matrix multiplication in Section II, to address that issue, a method for matrix multiplication with a high-speed external device FPGA, is proposed in Section III, and by evaluating the SyntaxNet execution performance, it is shown that SyntaxNet with FPGA can process a general sentence in about 1 second in Section IV, then the execution time was shortened and SyntaxNet was accelerated.

## II. STATE OF THE ART

SyntaxNet [4] is an open-source syntax analyzer announced by Google in 2016. To clarify the position of SyntaxNet used in this study as a parser, it was compared with other parsers, namely, Cabocha [10] and KNP [11] [12] as parsers dedicated to Japanese and Stanford CoreNLP [13] and SyntaxNet as parsers widely used for other languages. The features and performances of those parsers are compared with SyntaxNet in TABLE I. Processing speed was calculated by measuring execution time ourselves. The other parameters are based on previously reported research. In terms of processing speed, Cabocha surpassed the other parsers with throughput of 105K words/s, KNP achieved 105 words/s, Stanford CoreNLP achieved 145 words/s, and SyntaxNet before speedup achieved 307 words/s. However, Stanford CoreNLP uses 8 threads, while the others use one thread. Accuracy of Cabocha is about 4% lower than the other parsers. KNP Stanford CoreNLP, and SyntaxNet all achieve accuracy of over 93% and show similar values for processing speed and accuracy. For Japanese, Cabocha and KNP are often used, but it looks like they are properly used according to accuracy and function. Stanford CoreNLP is popular for English and other languages, but SyntaxNet uses the same syntax rules as Stanford CoreNLP, and it surpasses the others in terms of number of supported languages and performance, so it may be used in the future. Therefore, we think that our study's speeding up SyntaxNet is relatively fast and the study is effective.

## III. SyntaxNet

The SyntaxNet uses a transition-based algorithm [5] for syntactic parsing and a neural network for the decision process. SyntaxNet is overviewed in Figure 1. Parsey McParseface, a model running on SyntaxNet has demonstrated an analysis accuracy of 97.52% [4]. The interior of SyntaxNet is largely divided into a part that executes a transition-based algorithm (written in C ++) and a part that uses python and tensorflow (written in C ++) to execute judgments the next processing by using a neural network.

The transition-based algorithm is a kind of parsing algorithm that uses a state machine, stack, and buffer to parse sentences. First, a sentence is input to the sentence buffer, stack one word to stack at a time from the first word of the sentence, make a judgement on the top two words of stack, and one of the three actions is selected as a result of the judgment. This judgement and action are repeated after all the words disappear from the stack. After disappearing all words, an action sequence and dependency relationship of words is appeared. The process is completed with the result of the relationship of their words. In the Figure, the transition-based part performs other jobs except judgment.

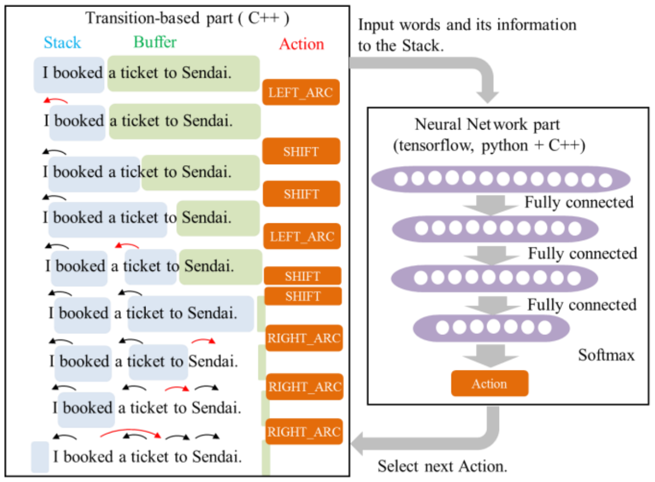


Figure 1. SyntaxNet algorithm

Information concerning the top-two words on the stack is sent to the neural network that performs only the judgment job, and the result of judgment is sent to the transition-based part. According to the result of execution-time analysis of SyntaxNet by Intel Vtune™ Amplifier, the tensorflow matrix-multiplication library Eigen [6] GEneral Block Panel (GEBP) uses 73% of the processing time, as shown by the pie chart in Figure 2. Since the GEBP is used for matrix multiplication, then matrix multiplication uses for 73% of the total processing time. In consideration of those results, the aim of the present study was to speed up matrix multiplication by the FPGA and improve the execution performance of SyntaxNet to 700 words/s.

## IV. Proposal

OpenCL [7] is chosen for implementing logics on the FPGA and for activating FPGA from the host computer. OpenCL is a framework for implementing multithreading, and its specification is managed by Khronos Group Inc. Intel uses OpenCL as a framework for implementing FPGAs [8]. OpenCL was chosen for the reason explained below.

The development costs for offloading to the FPGA are shared between design cost for the logic circuit and system for starting up the FPGA, and these two costs must be minimized. When OpenCL is used, the logic-circuit design can be created in a shorter period of time than the Hardware Description Language (HDL) design by compiling a C program to be run on the FPGA. The system design has been implemented so it does not have any costs. From the above consideration, it is considered that the development costs can be reduced by using OpenCL on FPGA.

Then we consider how to call the FPGA from host computer. It would be efficient to call the FPGA from Eigen. But Eigen is programmed as allowing multi-threaded operation. If the FPGA is called form Eigen, multiple calling would happen to one FPGA. Therefore, it is good place to call where the point calling Eigen routine currently.

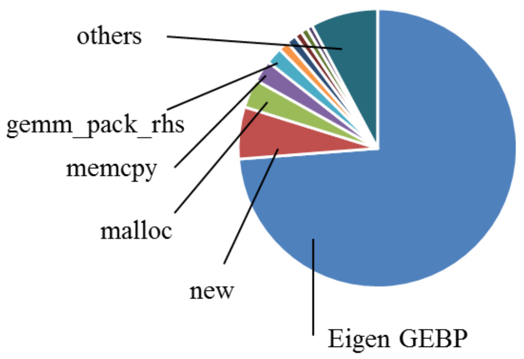


Figure 2. SyntaxNet Execution time analysis

```
for (int i=0; i < A_height; i++ ) do
    for (int j=0; j < B_width; j++ ) do
        float sum = 0.0 ;
        for (int k=0; k < A_width; k++ ) do
            sum += A[i][k] * B[k][j] ;
        end
        C[i][j] = sum ;
    end
end
```

Figure 3. Matrix multiplication algorithm

Another consideration, since data cannot be aligned in the program, Direct Memory Access (DMA) transfer to the FPGA is impossible with non-aligned data. A separate DMA buffer is added for DMA, and a memory-copy routine is added to copy data to the DMA buffer.

To minimize FPGA development cost, matrix-multiplication open-source sample code written in OpenCL was chosen for FPGA logic program, and the code was modified for turning the code performance. The routine for matrix multiplication is overviewed in Figure 3. Matrix multiplication is described by a triple loop [9]. The outer double loop specifies the position where the result, and the innermost loop calculates the inner-product of each data. In the OpenCL sample code, parallel processing of the innermost loop and the other row are performed. The high-speed internal memory size on FPGA is limited by FPGA chip size, then all matrix-data are placed on low-speed external DRAM, and partial data are copied to internal memory before performing partial matrix multiplication. Performances of the matrix multiplication on FPGA has different values depend on row and column size. Figures 4 and 5 show the performance of maximum and minimum data sizes used by SyntaxNet. According to Figure 4, performance of the FPGA did not change for any submatrix shape at minimum data size. Increasing the degree of parallelism increases the number of invalid area of matrix multiplication, but it does not improve matrix multiplication performance. On the other hand, according to Figure 5, the performance of the FPGA is almost constant at maximum data size even if the size of the column changes, but the performance increases in proportion to the size of row. Since the column side size is parallelized as much as possible when OpenCL is executed, the circuit configuration can only be slightly changed, and these circuits have the same performance. Even so, since row size is a parameter expressing how many elements are calculated in parallel, it is thought that doubling the number of submatrix rows doubles computation performance. Submatrix size above 64×64 could not be configured due to lack of FPGA resources, so 64×64 was considered to be the maximum. If execution time for each matrix size is focused on, it is clear that minimum matrix size takes about 6 ms, and maximum size takes about 150 ms. For that reason, maximum matrix size of 64×64 was taken as the parameter of the matrix-multiplication kernel.

TABLE II MEASUREMENT CONDITIONS

| Item | | Condition |
|---|---|---|
| Host | CPU | Xeon E5-2630 2.4 GHz |
| | Memory | 32 GBytes |
| | HDD | 1TB |
| FPGA board | | Nallatech 385A |
| | FPGA | Intel Arria 10 GX1150 |
| | DRAM | DDR-3 8GB |
| | interface | PCIe Gen3 x8 |
| Software | OS | CentOS version 7.4 |
| | OpenCL | version 17.0 |
| | Quartus | version 17.0 |
| | GCC | version 4.8.5 |

## V. Evaluation

Using the study up to the previous section, On the basis of the results presented in Figures 4 and 5, a matrix multiplication implemented on OpenCL on FPGA, and the performance of SyntaxNet of the implementation was evaluated and verified. The measurement conditions are listed in TABLE II.

A Nallatech 385A board uses the FPGA to improve performance. First, total execution time of SyntaxNet using FPGA matrix multiplication was measured three times and the average was taken.

The word throughput (which is taken as the performance

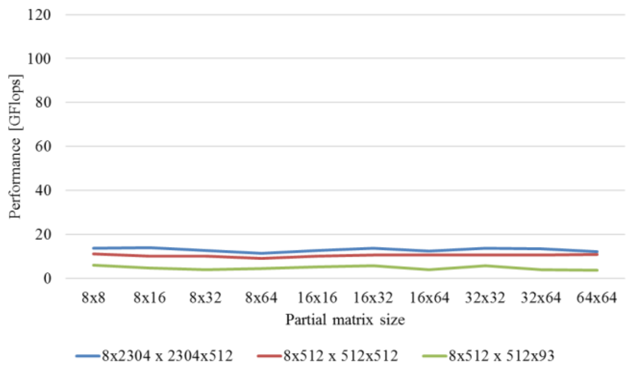


Figure 4. FPGA matrix multiplication performance (minimum)

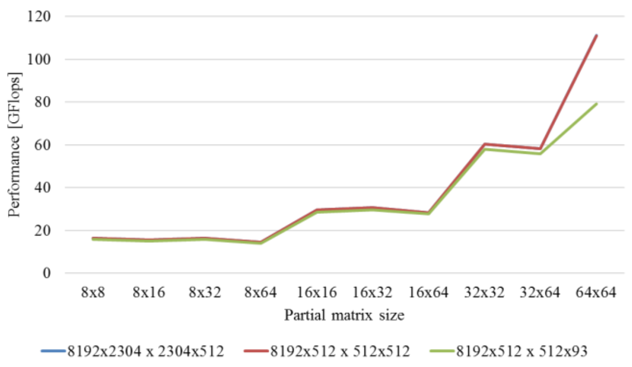


Figure 5. FPGA matrix multiplication performance (minimum)

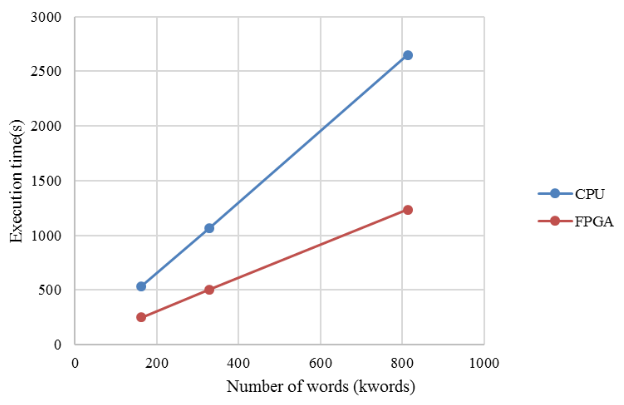


Figure 6. SyntaxNet performance ( execution time )

measure of SyntaxNet) is shown in Figure 6. For comparison, the performance achieved with one Central Processing Unit (CPU) thread is also shown. In terms of word throughput, processing time is decreased. The CPU processed 307 words/s, FPGA offload processed 661 words/s, and the performance ratio of above two was 2.15 times. As a result, 661 words/s was achieved which was almost the target performance 700 words/s, and then 700 words were processed in 1.05 seconds. As a result, the FPGA achieved 661 words/s, compared to the target of 700 words/s, and it could process 700-word article in 1.05 seconds.

A breakdown of the execution time of SyntaxNet based on the above-described measurements and analysis, and the performance ratio of CPU and FPGA offloading is given in Figure 7. In Figures 7(a) to (d), processing performances of CPU 1 thread and FPGA offloading are compared, and the performance ratio is shown. Peak performance of matrix multiplication measured by FPGA alone and performance of matrix multiplication by Eigen processing routine of 1 CPU is shown in Figure 7(a). The performance ratio is 8.53 times. SyntaxNet uses various matrix sizes for actual matrix multiplication. So SyntaxNet effective performance would be lower performance than the peak performance.

The ratio of effective performance due to the matrix size decreases by 6.56 times compared to that of Figure 7(b). It is necessary to process various sizes of large and small size, and processing of small matrix size degrades performance in

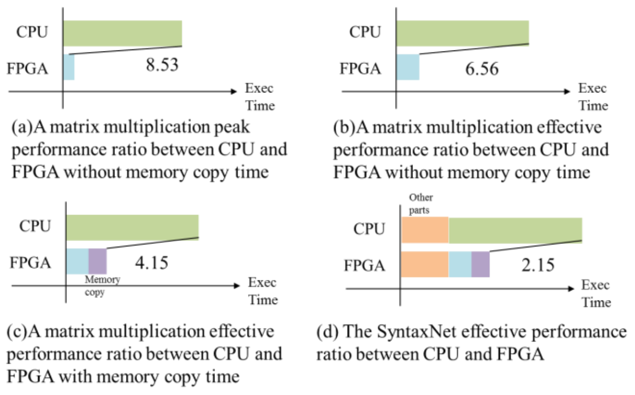

CPU
FPGA
8.53
Exec Time
(a)A matrix multiplication peak performance ratio between CPU and FPGA without memory copy time
CPU
FPGA
6.56
Exec Time
(b)A matrix multiplication effective performance ratio between CPU and FPGA without memory copy time
CPU
FPGA
4.15
Memory copy
Exec Time
(c)A matrix multiplication effective performance ratio between CPU and FPGA with memory copy time
Other parts
CPU
FPGA
2.15
Exec Time
(d) The SyntaxNet effective performance ratio between CPU and FPGA

Figure 7. SyntaxNet execution time breakdown

FPGA processing. It is necessary to process large and small matrix size, and processing of a small-size matrix degrades processing performance of the FPGA. As a result, effective performance is considered to decrease as a whole. Furthermore, memory processing is generated for FPGA processing. Therefore, when memory processing overhead is added, the performance ratio drops by 4.15 times compared to that shown in Figure 7(c). Then, in consideration of this result, the performance ratio becomes 2.15 times as shown in Figure 7(d) by adding other processing time of SntaxNet. Then offloading by FPGA cannot be achieved due to such overheads. It is difficult to measure these overhead previously. In consideration of the host-side software conditions and offload device characteristics, it will be necessary to make predictions.

## VI. DISCUSSION

Performance improvements are considered in the future. One of the methods for speeding up FPGA offloading is simultaneously executing DMA transfer and matrix multiplication by FPGA. However, as for SyntaxNet, the second-layer neural-network matrix multiplication is based on the result of matrix multiplication of the first-layer neural network. Therefore, the second-layer DMA transfer cannot be started until the first layer result is obtained. The second and the third layers are the same. Furthermore, a transition-based calculation is performed after the matrix multiplication of the third layer, and the transition-based calculation result is used to next neural network calculation. Therefore, in a loop that handles sentences, all matrix multiplications depending on the result need to be executed serially. Therefore, two SyntaxNet calculations of sentences need to be performed in parallel to hide the transfer time with the matrix multiplication time in FPGA. These calculation dependencies are eliminated by inserting irrelevant processing. As a result, the DMA transfer time can be hidden.

The other method for improving the performance of SyntaxNet is memory-copy reduction. In the current implementation of FPGA offload, the host cannot be used for DMA transfer from the array data area of the structure prepared by tensorflow because of data alignment. It is necessary to copy data to an area that aligned to 64 bytes. To eliminate this copy for speedup SyntaxNet, the memory area for the array data of the structure prepared by tensorflow must be aligned to 64 bytes. It is due to the specification of the PCIe bus of DMA transferring to the FPGA.

However, the FPGA matrix operation code divides matrix to submatrix. Then the code can calculate only the matrix size which is divisible by submatrix, but the code requires that a matrix of a certain size can be processed. When using the 64 × 64 submatrix, and matrix column size is not divisible by 64, for example, the column is 8, the remaining 56 parts should be 0 stuffed. It is necessary to make the matrix multiplication an accurate answer by the 0 stuffing process to manage. Since the 0 stuffing process perform memory copy, it can perform to change memory alignment to 64 bytes. In this study, the percentage of 0 stuffing that did not require 0-bit stuffing was estimated to be less than 5%. In the case of 95%, it needs memory copy because of 0 stuffing, then only 5% of memory copy would be eliminated by tensorflow modification. Then, it was found that this method had little prospect of performance improvement. It is thus concluded that the memory copy elimination method cannot improve the performance of SyntaxNet.

Another method to improve performance of SyntaxNet is implementing kernel code as a systolic array, which is a structure of logic circuit that repeats operations such as multiplication and addition while moving data. A systolic array is known as an efficient multiply-add operation method when there are many combinations of operations. The systolic array is improved calculation efficiency.

Especially in recent years, Google applied systolic array to matrix-calculation circuits for deep learning such as TPU [14]. Circuit logic diagrams of a matrix multiplier using a self-designed systolic array are shown in Figures 8 and 9.

Each arithmetic unit in Figure 8 is a simple one consisting of a multiplier and an adder. The arithmetic unit multiplies a value (A) sending from the left, and another value for multiplication (B) is held by the register. After that, the arithmetic unit adds the sent value (S). Each value coming from the left is sent to the right (An), and the value after addition is sent to the bottom (Sn). This arithmetic unit is arranged in two dimensions as shown in Figure 9. Its operation consists of four steps as follows.

(1) Set all the values of matrix B.

(2) Send the values in the order from the left to the right. The transmission one step down is started one cycle later, and the pattern is transmitted diagonally in space.

(3) Send A for the operation and wait for the result to appear in the lower buffer.

(4) Repeat (1) to (3) with the next data.

Data movement and calculation are performed at the same time by such operation, and multiply-add operation is efficient. We attempted to create this systolic array using OpenCL. The proposed systolic array circuit was created on the basis of OpenCL. In particular, a systolic-array code was written with OpenCL as the hardware shown in Figure 9, and the code looked like working. Then, the circuit ran on FPGA, but its performance was three digits lower than we expect.

It would happen because the arithmetic unit has a lot of

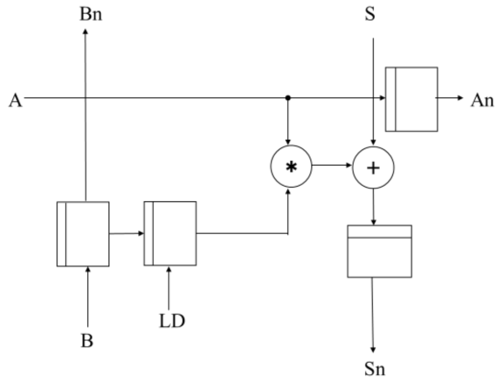


Figure 8. Systolic array arithmetic unit

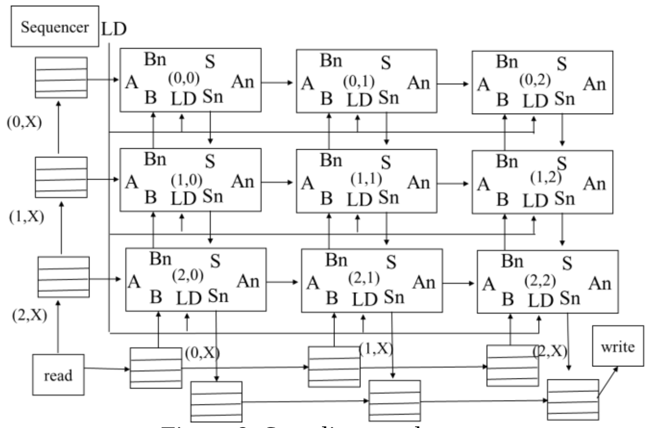


Figure 9. Systolic array layout

latency because of its floating-point multiplication, then OencCL compiler create a lot of processing latency in the array. The latency cannot be changed by changing the OpenCL code only. The OpenCL is designed to make hardware from an algorithm written in C. However, it is difficult to describe hardware itself like the systolic array.

## VII. CONCLUSION

To automatically classify and analyze customer complaints, we investigated whether it is possible to speed up the OSS syntax analyzer SyntaxNet with an FPGA, implemented FPGA offload, evaluated an actual machine, and obtained the following conclusions. We evaluated SyntaxNet with FPGA offload, and confirmed that SyntaxNet's execution performance was 661 words/s, almost achieved the target of 700 words/s, and processed sentences of 700 words in general size in about 1 second. The execution time of an FPGA offload machine was measured, and the measurement results confirmed that (i) the execution performance of SyntaxNet was 661 words/s (which almost achieved the target of 700 words/s) and (ii) sentences with size of 700 words (in general) could be processed in about 1 second. These results demonstrate that automatic text categorization and analysis can be immediately executed on a system with a reduced number of servers by speeding it up with power-saving FPGA acceleration. This FPGA offloading can be applied to all neural networks using matrix multiplication.

## ACKNOWLEDGMENT

The author thanks Dr. Tsuyoshi Tanaka for his assistance in writing this paper.

## REFERENCES

[1] D. Lin, Introduction to Natural Language Processing (NLP) [0nline]. Available form: https://www.slideserve.com/jory/introduction-to-natural-language-processing-nlp 2014.3.12

[2] Akamai Technologies, Inc. New Study Reveals Impact of Travel Site Performance. [Online]. Available from:

https://www.akamai.com/us/en/about/news/press/2010-press/new-study-reveals-the-impact-of-travel-site-performance-on-consumers.jsp 2010.06.14

[3] Forbes. Do You Read Fast Enough To Be Successful?. [Online]. Available from: https://www.forbes.com/sites/brettnelson/2012/06/04/do-you-read-fast-enough-to-be-successful/#3b3dd025462e 2012.06.04

[4] D. Andor, C. Alberti, D. Weiss, and A. Severyn, "Globally Normalized Transition-Based Neural Networks," arXiv 1603.06042v2, March 2016.

[5] J. Chang, J. Seefried, S. Taylor, and A. Brandner, "SyntaxNet: Google's Open-sourced Syntactic Parser," Department of Computational Linguistics University of Tubingen, January 2018.

[6] Eigen is a C++ template library for linear algebra. [Online]. Available from: http://eigen.tuxfamily.org/index.php?title=Main_Page 2019.08

[7] Khronos Group. OpenCL Overview. [Online]. Available form: https://www.khronos.org/opencl/ 2019

[8] Intel, Inc. Intel FPGA SDK for OpenCL Software Technology. [Online]. Available from: https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html 2019

[9] Z. Wang, B. He, W. Zhang, and S. Jiang, "A Performance Analysis Framework for Optimizing OpenCL Applications on FPGAs" IEEE, 2016.

[10] T. Kudo and Y. Matsumoto, "Japanese Dependency Analysis Using Cascaded Chunking," , in Japanese, June 2002.

[11] S. Kurohashi and M. Nagao, "KN Parser : Japanese Dependency / Case Structure Analyzer," unknown, 1994

[12] S. Kurohashi and M. Nagao, "Building a Japanese Parsed Corpus while Improving the Parsing System," unknown, 1998

[13] Stanford University. Stanford CoreNLP – Natural language software. [Online]. Available from: https://stanfordnlp.github.io/CoreNLP/ 2019

[14] N. P. Jouppi, C. Young, N. Patil, D. Patterson, et.al "In-Datacenter Performance Analysis of a Tensor Processing Unit", ACM/IEEE 44th, 2017