Accelerating BLAST Computation on an FPGA-enhanced PC Cluster

Masato Yoshimi, Celimuge Wu, Tsutomu Yoshinaga
Graduate School of Informatics and Engineering, University of Electro-Communications
Chofu, Tokyo, JAPAN 182-8585
Email: {yoshimi,clmg,yosinaga}@is.uec.ac.jp

***Abstract*—This paper introduces an FPGA-based scheme to accelerate mpiBLAST, which is a parallel sequence alignment algorithm for computational biology. Recent rapidly growing biological databases for sequence alignment require high-throughput storage and network rather than computing speed. Our scheme utilizes a specialized hardware configured on an FPGA-board which connects flash storage and other FPGA-boards directly. The specialized hardware configured on the FPGAs, we call a Data Stream Processing Engine (DSPE), take a role for preprocessing to adjust data for high-performance multi- and many- core processors simultaneously with offloading system-calls for storage access and networking. DSPE along the datapath achieves in-datapath computing which applies operations for data streams passing through the FPGA. Two functions in mpiBLAST are implemented using DSPE to offload operations along the datapath. The first function is database partitioning, which distributes the biological database to multiple computing nodes before commencing the BLAST processes. Using DSPE, we observe a 20-fold improvement in computation time for the database partitioning operation. The second function is an early part of the BLAST process that determines the positions of sequences for more detailed computations. We implement IDP-BLAST (In-datapath BLAST), which annotates positions in data streams from solid-state drives. We show that IDP-BLAST accelerates the computation time of the preprocess of BLAST by a factor of three hundred by offloading heavy operations to the introduced special hardware.**

***Keywords*-In-datapath Computing, Interconnected FPGA boards, Flash storage, Computational Biology, BLAST**

I. Introduction

The popularity of various web services and increasing use of sensor technologies have led to massive amounts of data being generated every day. Over the past decade or so, big data applications have converted these vast quantities of data into profitable commodities. Numerous organizations have begun utilizing big data in application fields such as decision making, risk management, and advertising. Several scientific fields deal with such big data, e.g., computational biology, atmospheric science, astronomy, and other interdisciplinary research areas [1]. Notably, the combination of the scalable computing framework of MapReduce [2] and distributed file systems based on GFS [3] promotes the use of big data. In addition, the scale of computing systems is also expanding, enabling access to data in remote storage. The speed at which the accumulated data are growing, which is much faster than the advance of computational devices, requires current computer systems to be redesigned.

Especially in the field of computational biology, the speed of genomes accumulating database is faster than Moore's Law. The number of genomic sequences is doubling almost every 12 months[4]. This factor means that we are required to tackle with reconsidering the architecture of the computing system to improve the throughput to extract significant data from massive amount of no importance data.

Several projects have explored novel computer systems that can treat large amounts of data more efficiently [5] [6]. The key feature of these systems is the installation of field programmable gate array (FPGA)-based dedicated hardware that can be directly attached to the storage or network. The FPGA not only permits part of the user functions to be offloaded, but also allows Near Data Computing by filtering the data before loading the main memory.

Our research group has proposed a computer system with an FPGA-based, tightly coupled accelerator that connects an FPGA to flash storage and a network. The hardware module (Data Stream Processing Engine, DSPE) trims and aggregates the data streams passing through the FPGA [7] [8]. Even though the mechanism of in-datapath computing has shown favorable performance compared to software-based distributed frameworks, the evaluated applications were simple, single clock-cycle operations for reading data from storage to DRAM on the FPGA board. To increase the availability of such a mechanism, more flexible operations must be supported and performance evaluations conducted on practical applications.

In this paper, we apply the computing mechanism for mpiBLAST, which is an software-based parallel implementation of the sequence alignment for genomic data for PC-Cluster. Even though several research reported FPGA-based implementations to accelerate BLAST with an FPGA on a single computing node, we extend the mechanism to PC-Cluster utilizing direct connection among FPGA boards.

There are two novelties in this paper. First is a mechanism of partitioning database utilizing the direct data transmission between the FPGA-boards. The genomic database is divided and stored into each storage which is equipped on the FPGA-boards via network between FPGA-boards for parallel computation. Second, we propose a first-in-first-out (FIFO)-based mechanism that permits DSPE pipelined operations with a controller for solid-state drives (SSDs) to offload an early part of mpiBLAST to reduce the computation

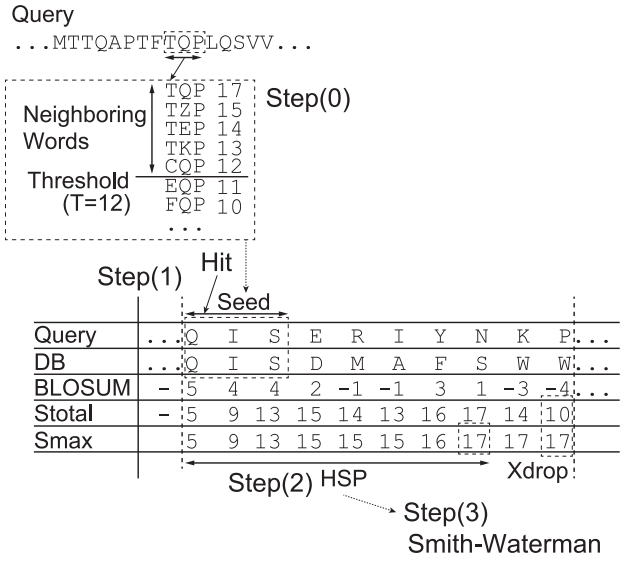

Figure 1. Computational step in BLAST

in the microprocessor. Through experimental evaluations, we confirmed that our in-datapath computing accelerates database partitioning by a factor of 20. Additionally, we also confirmed that the overall computation time of mpiBLAST by a factor of three by offloading preprocesses.

II. BLAST and Acceleration Technique

This section gives an overview of the BLAST algorithm and various acceleration techniques using parallel computing and specialized hardware.

A. Overview

BLAST (Basic Local Alignment Search Tool) is a famous algorithm for sequence alignment in bioinformatics. Sequence alignment obtains the similarity score between two sequences of nucleotides or proteins. BLAST is used to extract similar sequence fragments by comparing a query sequence with many sequences in a database. Although the Needleman–Wunsch and Smith–Waterman algorithms can be used for sequence alignment, rigorous algorithms are computationally intensive. The advantage of BLAST is that it drastically reduces the number of arithmetic operations by introducing heuristics and statistics for bioinformatics while maintaining accuracy. In this paper, we consider the sequencing of a protein.

As shown in Fig. 1, BLAST performs four steps to obtain the alignment between two sequences: (0) *Generating neighborhood words* initializes the query sequence to fine-grained letters to obtain the starting position of alignment, (1) *Word Matching* searches for candidates of this starting alignment, (2) *Ungapped Extension* trims the candidates by rough alignment, and (3) *Gapped Extension* computes the detailed alignment using the Smith–Waterman algorithm.

These steps reduce the number of operations by extracting candidate sequences that are examined in more detail. In this paper, BLAST is regarded as a hierarchical filtering algorithm to reduce the computational load. Further details on the BLAST algorithm can be found in [9].

In Step 0, the query sequence is converted to neighborhood words, which is a list of k letters ($k = 3$ for protein sequence) satisfying the similarity score of some least threshold $T(= 12)$. After Step 0, the sequence alignments from Steps 1–3 are repeated to filter the database sequences according to the neighborhood words. Step 1 searches for neighborhood words from the database sequence to obtain *Hits* with respect to the start position for successive steps. Step 2 (called Ungapped Extension) computes high-scoring segment pairs (HSPs) from all *Hits*. These HSPs are rough similarity scores used in Step 3. BLAST adopts a two-hit method, whereby Step 2 is only executed for two seeds that are within some specified distance of one another. The resulting HSPs are sorted in order of their significance. In Step 3, BLAST computes the detailed alignment, including the *gap*, for each HSP using the Smith–Waterman algorithm.

B. Parallel computation for BLAST

There have been several studies on the parallelization of BLAST to accelerate its performance [10] [11]. Ref. [11] classified the parallelism of BLAST into three levels: a fine-grained intra-query level, medium-grained intra-database level, and coarse-grained inter-query level.

The easiest way to parallelize BLAST is at the coarse-grained level by executing multiple BLAST processes of individual queries for the same database simultaneously. However, while such higher throughput may enhance the efficiency of a computational system, the performance of a single query should also be improved.

C. mpiBLAST

mpiBLAST[10], which is an MPI version of the well-known NCBI-BLAST implementation, runs on a PC cluster to exploit the medium-grained approach. BLAST can be easily parallelized by partitioning the database into multiple computing nodes, since the computation of each alignment is independent. The developers of mpiBLAST focused on the relationship between the length of the database sequences and the computation time. mpiBLAST introduces a novel partitioning method to distribute sub-database sequences to each computing node for parallel execution, thus balancing the computation time in each node.

III. Related Work

A. Future Architecture for Data Centers

Computing platforms for big data analytics require good performance and cost efficiency. Several industries have announced projects to develop a new computing infrastructure for data centers [12] [13]. For example, Firebox

[14] is a next-generation computer architecture consisting of multiple fine-grain components of SoCs and memory modules connected to high-radix switches. The prototype for this architecture, DIABLO, was developed using an FPGA-based system. Intel has announced the Rack Scale Architecture based on a new concept of disaggregating computer components into pooled processors, storage, and networking resources [13]. The machine, proposed by Hewlett-Packard, aims to utilize configurable fine-grained processing cores and memory pools by connecting them with a photonic network. They insist that future data centers require such computer systems, and plan to release them by the 2020s [12].

Our approach is within the context of developing a novel computer system by adopting specialized hardware. The primary feature of our approach is to accelerate applications by offloading part of the computation to the datapath of the transmission among the main memory of the host-PC, the flash storage, and the network.

B. FPGA-based Accelerators for Data-intensive Applications

There are several FPGA-based accelerators for big data. IBM has released Netezza [5], which integrates a considerable number of blade servers as a data warehouse appliance. Each server equips an FPGA between the main memory and the storage to extract data without increasing the processor load. Netezza compiles a user-defined query to distribute commands, and discards unnecessary data before loading from the storage. In [15], the authors adopted Netezza for chromosomal microarray analysis. They confirmed that the active disk mechanism of Netezza worked well compared to parallel computation on a PC cluster and Hadoop.

The trend analysis and machine learning of big data workloads, which involve the frequent scanning of all data, have led engineers to develop near-data processing, i.e., performing computations near the data source. Adding computing elements to the storage for offloading operations is a valuable technique for next-generation storage [16] [17].

Several computing systems have been developed for specific applications [18] [6]. For instance, [18] reports a computing system called BlueDBM to support DRAM on FPGA boards that are directly attached to the network and flash storage. This system was confirmed to maintain the same performance as a PC cluster with a large amount of main memory. Microsoft developed a computing system called Catapult [6] that equips FPGA boards connected to one another for their web service. The system executes each sub-sequence of the PageRank operation in the manner of a systolic array. These computing systems are currently in the demonstration phase. Although these projects involve various interconnected FPGAs, the novelty of our work is the development of an acceleration methodology that not only offloads whole computations to FPGAs, but also combines the computation with communication, as shown in Section II.

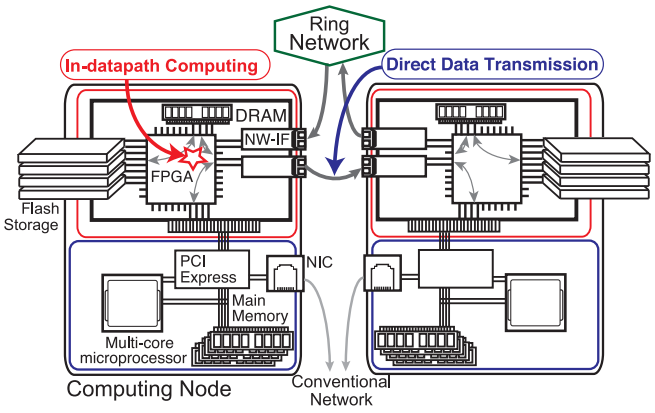

Figure 2. A computing platform with In-datapath Computing

C. FPGA-based Accelerators for BLAST

BLAST and related algorithms are regarded as primary applications to be accelerated by dedicated hardware such as FPGAs. A number of researchers have developed techniques to offload part or all of an operation to BLAST.

Mercury BLAST [19] is a hardware implementation that offloads the preprocessing step of BLAST to a single FPGA board. Mercury BLAST offloads word matching and ungapped extension to the FPGA board database when reading database sequences from magnetic disks in the style of a systolic array. The host PC need only execute gapped extension. Although the approach proposed in this paper (see Section V-C) is similar to Mercury-BLAST, the underlying computing scheme is different. We consider that in-datapath computing with DSPE should not block the data stream from the data source to maintain wire-speed. Since we can choose various accelerators for data in the main memory, in-datapath computing should not impede the speed at which data can be accessed.

IV. In-datapath Computing Platform

A. Overview

This section describes in-datapath computing platform we have proposed. Although our prototype environment includes several proprietary modules, the scheme can port to other FPGA boards.

Our research group has proposed an acceleration scheme for data-intensive applications using an FPGA-based accelerator [7]. Fig. 2 shows an example of the computing platform with two computing nodes. In the scheme, we suppose that PC clusters in which each node is equipped with an FPGA board transfer data between storage and other nodes.

The proposed scheme has two advantages compared to conventional many-core accelerators such as GPUs and the Xeon Phi: (a) offloading part of the computation to the

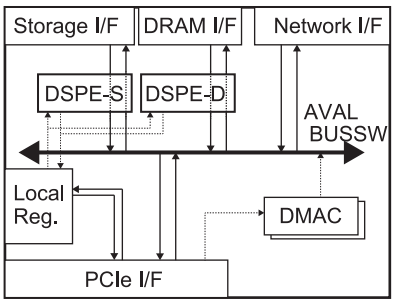

Figure 3. An architecture implemented in APX series

FPGA when loading data from storage or network into the main memory, and (b) direct data transmission between FPGA boards without going through the main memory by executing system calls in the host PCs. We refer to the scheme of (a) as *in-datapath computing*, whereby hardware modules configured in the FPGA apply operations in the data stream.

B. DSPE: Data Stream Processing Engine

In previous work, we adopted the AVALDATA APX-880A [20] FPGA board for an experimental proof of the scheme. This FPGA board, whose specification is listed on the left of Table I, equips four primary interfaces to the PCI Express, DRAM, optical network, and flash storage. As shown in Fig. 2, a proprietary bus switch called AVAL-BUS interconnects these interfaces, as in conventional bus systems provided by FPGA vendors. The optical network connects multiple FPGA boards to form a ring topology in the PC-cluster. The data stored on the flash storage can be transferred directly via the network by invoking Direct Memory Access (DMA) controllers from the host PC.

We have proposed the DSPE mechanism shown in Fig. 3 for in-datapath computing [7]. Two types of DSPE can be configured on the FPGA: (1) DSPE-S in front of the flash storage, and (2) DSPE-D at the DRAM interface. We can implement user logic according to the target application to apply logical and arithmetic operations to data streams passing through either DSPE. The user logic is controlled by instructions stored in the local register array, which can be accessed by the host PC via the PCI Express memory space. Both the network and storage throughput are around 1 [GB/s] in the APX-880A.

To validate the applicability of the scheme, we have reported two implementations: (1) simple word counting [7], and (2) group-by aggregation [8]. Evaluations confirmed that each application outperformed software-based distribution frameworks by offloading the comparison of values and accumulation of counters into the data stream of DSPE-S.

C. Motivation for Developing APX-7142

We encountered two problems with the previous APX-880A: (1) the devices are no longer effective following rapid progress in applications, and (2) the data granularity restriction of 128 MB for accessing the storage of SD cards is markedly different from conventional storage systems.

Moreover, previous evaluations [7] [8] did not confirm the applicability of the scheme for three reasons. First, only two operations are required by the host CPU—issuing an operation to transfer data from flash storage to DRAM, and reading results from the local register array. Second, utilizing the network for in-datapath computing has not been discussed, since both hardware modules are implemented on the DSPE-S. In this paper, we use DSPE-D. Third, the implementation of DSPE-S can accept operations that complete within a clock cycle using the simple I/O protocol. Section IV-E proposes an FIFO-based DSPE-S implementation in which more flexible operations require multiple clock cycles. Therefore, we developed a new hardware component in which an FPGA board is used to accommodate in-datapath computing.

We implement the in-datapath computing platform described in Section IV-B into AVALDATA APX-7142 [21], the specification of which is listed on the right of Table.I. There are two primary features of APX-7142: (1) faster overall data transfer due to the expanded bus-width, and (2) an Serial Attached SCSI (SAS) connector that branches to four SATA devices instead of multiple SD card slots.

D. SSD-based Flash Storage

Before designing an advanced DSPE-S, we must develop a SATA controller to connect the APX-7142 with SSDs, since the SAS connector of APX-7142 is essentially designed to connect digital measurement instruments. To reduce the development cost, we use the SATA-CORE design, a type of SATA2 (3 Gbps) interface [22] to connect off-chip SATA devices.

Fig. 4 shows the architecture of a SATA controller managed by the host PC. Two dual clock-in FIFOs act to buffer the different operating frequencies of the SATA device (200

Table I
SPECIFICATION OF PROTOTYPE FPGA BOARDS

Product	APX-880A	APX-7142
FPGA Device	Stratix IV GX EP4SGX230KF40C2 runs at 125 MHz	Stratix V GX 5SGXMA3K1F40C2N runs at 125 MHz
DRAM (DDR3)	533 MHz, 512 MB	800 MHz, 2.0 GB
Flash Storage	18 SD Cards×1 or 2 1.5 GB/s×2ch two SDs for parity	SAS connector extends 4 SATA ports scratch build in this paper
unit to transfer	128[MB]	4[KB]
Network	Proprietary GiGA CHANNEL Optical token ring network unit to transfer 16 [KB]	
	8.5 Gbps×2ch	14 Gbps ×2ch
PCIe I/F	2.0 Gen2×4 Lane	2.0 Gen2×8 Lane
Internal Bus	Proprietary AVAL-bus	
	128 bits-width	256 bits-width

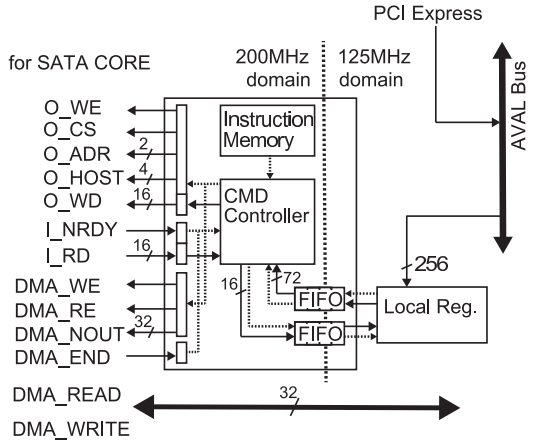

Figure 4. Architecture of a SATA controller

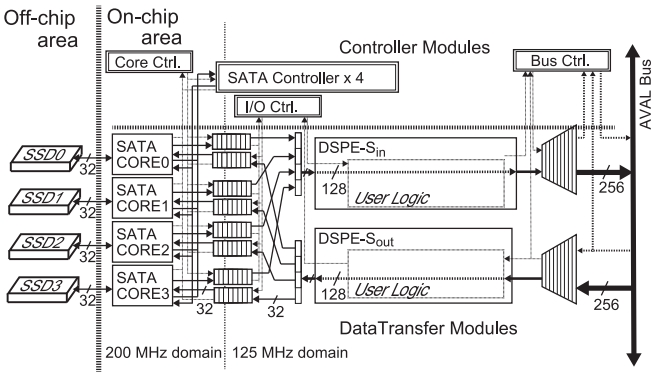

Figure 5. FIFO-based DSPE

MHz) and other hardware modules (125 MHz) in the FPGA. The SATA controller also requires a mechanism controlled by the host PC. We implement a scheme that triggers the DMA transmission from or to the SATA device over the SATA-CORE according to the address and block size parameters in the local register array. Therefore, the host PC instructs data transmission by configuring parameters in the local register array via PCI Express.

The command controller handles the SATA protocol according to instructions read from the instruction memory. The DMA data-bus connects to AVAL-BUS separately, as stated in Section IV-E.

E. DSPE with FIFO-based I/O

We propose an advanced DSPE with FIFO-based I/O to embed user logic requiring multiple clock cycles. Fig. 4 shows the architecture of the FIFO-based DSPE-S, including the SSD controllers explained in Section IV-D. DSPE-S for APX-7142 implements three types of sub-controller and two types of FIFO outside the user logic; the sub-controllers arbitrate the input and output of the FIFOs according to the data flow. The FIFO-based DSPE-S controls the user logic to implement various operations by introducing FIFOs that regulate the stream.

The FIFOs perform clock conversion and bit-width alternation between SATA-COREs (32 bits $\times$ 4 at 200 MHz) and AVAL-BUS in an FPGA (256 bits at 125 MHz). The SATA-side FIFOs also have the ability to align data arriving from four SSDs. Therefore, the data signals in DSPE-S are connected to the AVAL-BUS. Units of 4 [KB] of data communicate with the SSDs, since the controller accesses all four SSDs simultaneously, even though the unit of data is 1 [KB] for AVAL-BUS and a single SSD.

We also implemented software libraries to invoke DMA transmissions between interfaces. The FIFO-based interface of DSPE-S and software implementations make in-datapath computing possible. The architecture can be embedded in DSPE-S, as well as in the network and DRAM using similar FIFO-based flow control. Since the current development version of the driver for APX-7142 cannot access SSDs directly from the host PC, data streams from or to the SSDs are buffered in the DRAM on the APX-7142. Although the throughput to access storage is halved, the software development is still progressing.

V. Implementation

A. Overview

In this section, we propose a mechanism to accelerate mpiBLAST utilizing in-datapath computing on our FPGA-boards. We focus on two computational features of mpiBLAST: (1) time required to partitioning the database, and (2) computationally intensive operation of obtaining the start of Ungapped Extension. All hardware is implemented in VHDL and we executed synthesis and Place& Routes by Altera Quartus II versions of which are shown in Table IV.

B. Database Partitioning

Even though the database partitioning runs only once in advance of the computation, the time taken is not negligible, since the database is continually growing.

Table II gives examples of the computation time required for database partitioning using *mpiformatdb* on the computing environments listed in Tables III and IV. The computation time in Table II is derived by subtracting the time required for *formatdb* from the total time of *mpiformatdb*, since *formatdb* includes several format conversions not related to distributed operation. Database partitioning is a heavy CPU-bound operation, as the computational throughputs in Table II are less than one-twentieth of the I/O throughput of RAID0-SSDs. Even though these two databases are comparatively small, times of the order of several minutes may degrade the availability to use of parallel computing for mpiBLAST.

The in-datapath computing scheme is applied to accelerate the partitioning and distribute the mpiBLAST database by offloading the partitioning methodology of *mpiformatdb*.

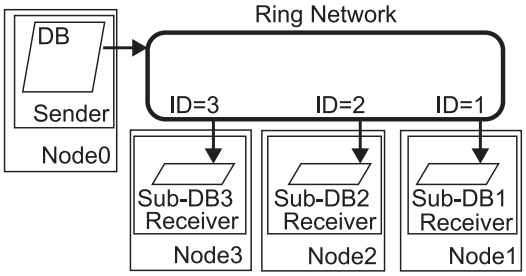

Figure 6. Database partitioning by distributing the database

Suppose that a computing node holds the database in the flash storage of the FPGA board. The sender node partitions the database and distributes the sub-database to other receiver nodes.

We designed the operation flow shown in Fig. 6, which is an example of a PC cluster consisting of a sender node and three receiver nodes. At the start of partitioning, the sender node sends the database to the ring network. The data is partitioned and taken to the receiver nodes. We implemented two hardware modules, DSPE-S and DSPE-D, to offload this process and achieve the database partitioning shown in Figs. 7 and 8.

The sender module modifies the database read from the flash storage and sends it to the GIGA channel. As shown in Fig. 7, the DSPE-S module at the output of the storage detects and modifies the head of each database sequence to indicate a receiver node ID. The module also sends a counter representing the total size of the database sequences to each receiver. The receiver node ID is determined by the minimum value of the counter. The sender module also adds padding to adjust the data size transferred through the GIGA channel.

The receiver module at the input from the GIGA channel

Table II
SOFTWARE-BASED TIME TO DATABASE PARTITIONING

DB name	Size [GB]	Time [s]	Throughput [GB/s]
est_human	5.157	133.9	0.0385
gss	27.973	640.7	0.0437

Table III
SPECIFICATIONS OF COMPUTING SYSTEM

No. of Node	4	
Product Name	DELL Precision Workstation T5610 D01T	
CPU	Intel Xeon E5-2630	2.60GHz (6C12T)
Memory	four 4 GB DDR3	16.0 GB
Network	Broadcom NetXtreme on board Ethernet	10Gbps 1Gbps
SSD	Crucial CT480M500SSD1 ×2(RAID0)	960GB
HDD	Seagate ST2000DM001	2.0TB

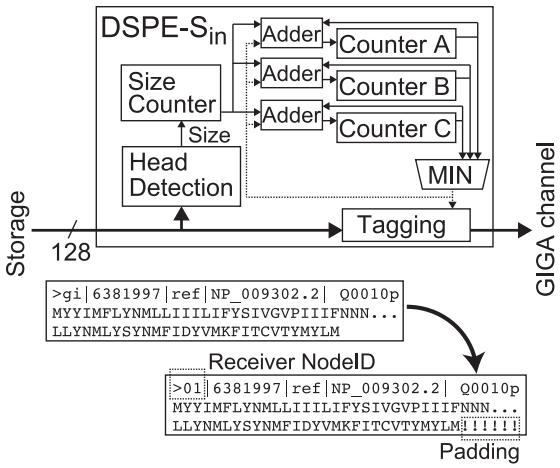

Figure 7. The sender module of database partitioning

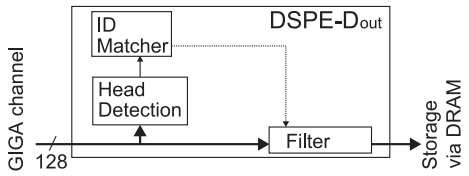

Figure 8. The receiver module for storing sub-database

also detects the head of each query and filters data with an ID that is equal to that of the receiver node. The filtered sequences are stored in the flash storage by passing through the DRAM.

These two modules reduce the computational load of scanning the database for partitioning and distribution by packing computations into the data-path between sender and receiver.

C. IDP-BLAST

After dividing the database, each computing node starts BLAST. In the BLAST algorithm, the first two steps can be executed while reading database sequences.

Ref. [11] and our evaluation in Table VII indicate that the detection of *Hits* occupies 70–80% of the computation time of BLAST. The in-datapath computing mechanism proposed in this paper offloads the early BLAST computations to

Table IV
OPERATING SYSTEM AND SOFTWARE INSTALLATION

OS	CentOS 6.6 kernel-2.6.32-504.16.2.el6.x86_64
SW Compiler	gcc-4.4.7 boost-1.57.0 OpenMPI-1.8.1 -m64 -O3 -fopenmp
HW CAD	Altera Quartus II 10.1sp1 for APX-880A 13.1 for APX-7142
BLAST	NCBI-BLAST-2.2.30 mpiBLAST-1.6.0

the DSPE-S (which we refer to as In-Datapath BLAST, or IDP-BLAST). We offload the functions of *Word matching* and the former part of *Ungapped Extension*, which can be implemented as hardware and computed at wire-speed. Although Ref. [19] implemented similar functions in FPGA-based hardware, they did not consider wire-speed computation, because the input throughput was much less than the computing speed under their assumption that the storage format is magnetic disks.

We implemented IDP-BLAST as the DSPE-S from the flash storage, the architecture of which is shown in Fig. 9, to offload the detection of *Hits*.

The feature of our implementation is that a computation matrix for obtaining the hit positions is introduced instead of the *Generating Neighborhood word*. Since a large hash table is required to store neighborhood words, and the time required to search the hash table is uncertain, another implementation is needed to achieve wire-speed computation. We adopted an FPGA parallelism mechanism that distributes the computation of field programmable logic by matching the query sequence to database sequences directly. Although this idea suppresses the length of the query sequence in an FPGA, adopting a more fine-grained approach, such as that in Ref. [11], relaxes this restriction.

IDP-BLAST consists of a BLOSUM table, an adders and filters matrix, and a hit detection matrix, as shown in Fig. 9. The first matrix returns a score between a query letter and a letter of a database that has flowed from the flash storage. The second matrix calculates the similarity score by adding three letters, and filters scores that are larger than the threshold ($T = 12$). The third matrix further filters the *Hits* using the two-hit method. The hardware detects hits from database sequences and sets the position of the *Hit* within the data. We assume each letter is 8-bits wide to embed the index into the vacant bits of the data.

This computation is pipelined for three stages so as not to block the data stream from the SSDs. Therefore, the host PC can derive the *Hit* position at the same time as reading the database from the database sequence. Before implementation, we fixed the maximum query length to 512. This is considered practical, and is much longer than other FPGA-based implementations [23].

VI. Evaluation

A. Evaluation Environment

We evaluated the resource utilization and performance improvement in two implementations of BLAST. Database partitioning was evaluated using APX-880A, and IDP-BLAST was examined with APX-7142. Different hardware environments were used because of the degree of perfection required for networking with APX-7142. However, the performance can be compared, as the throughputs between the flash storage are similar in each case.

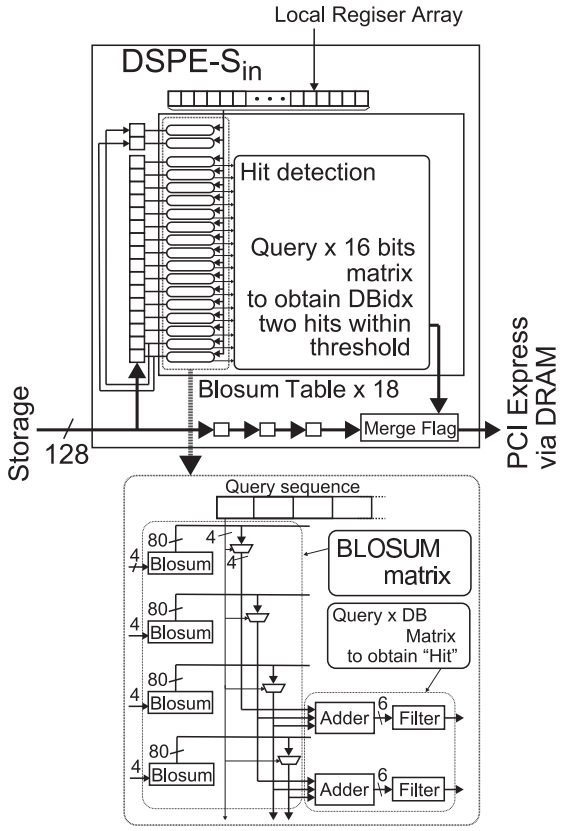

Figure 9. Detecting the *Hit* position

B. Resource Utilization

1) Database Partitioning: The logic resources consumed by the DSPEs for the database partitioning described in Section V-B are listed in Table V. The Sender and Receiver modules can be implemented with relatively small logic resources, and this further extends the user logic in the early computations of BLAST.

2) IDP-BLAST: Table VI lists the logic resources required for the DSPE hardware, including IDP-BLAST, described in Section V-C with the SATA controller in Section IV-E. Note that the FPGA generation of APX-7142 is

Table V
Logic Resources for Partitioning modules in APX-880A

	Capacity	Proposed	Sender	Receiver
Logic [%] Utilization	100	64	—	—
Combinational ALUTs	182400	74910	2892	271
Memory ALUTs	91200	750	0	0
Dedicated Logic Reg.	182400	80843	1738	386
Block RAM [Kb]	14283	5073	114	0

different from the database partitioning in Section VI-B1.

The results in Table VI offer two perspectives. The SSD controller explained in Section IV-E is comparatively small for extending the user logic in DSPE, even though it configures several FIFOs. In contrast, IDP-BLAST occupies larger resources because of the massive amount of adders and comparators.

C. Performance of In-datapath Computing

1) Database Partitioning: We evaluated the computation time of database partitioning for two types of databases. The results are presented in Table III. Fig. 10 compares the performance of mpiBLAST on the PC cluster with that of our hardware for database partitioning. For both databases, the computation times achieved by the proposed method are a factor of 20 faster than those given by the conventional PC cluster. As the feature of in-datapath computing achieves computations with data transfer, we calculated the throughput of database partitioning according to:

$$\text{est\_human } 5.157[\text{GB}]/4.9[\text{s}] = 1.061[\text{GB/s}] \quad (1)$$

$$\text{gss } 28.2[\text{GB}]/26.6[\text{s}] = 1.060[\text{GB/s}] \quad (2)$$

Our computing system can partition and distribute the databases with a throughput of over 1.0 GB/s. This throughput is almost the same as the read performance of the flash storage of APX-880A [7]. Even though the networking and writing to storage throughputs are faster than this result, we have confirmed that database partitioning can be achieved by in-datapath computing using the hardware described in Section V-B. The hardware structure of the partitioning can be ported to other FPGA boards such as APX-7142, the hardware does not depend on any specific features of APX-880A.

2) IDP-BLAST: The performance of IDP-BLAST is the same as the throughput reading from the SSD to the host PC, as the hit positions are obtained when the database sequences are read from the SSD as described in Section V-C.

To evaluate the performance of IDP-BLAST, we considered the *env_nr* database provided by NCBI. Table VII lists the profile of a node of Table III. The *env_nr* database contains some 6.5 M sequences and has a size of approximately 1.29 [GB].

Table VI
LOGIC RESOURCES FOR IDP-BLAST IN APX-7142

	Capacity	Proposed	SSD ctrl.	IDP-BLAST
Logic [%] Utilization	100	75	—	—
Combinational ALUTs	128300	95778	7047.6	35042.5
Memory ALUTs	64150	1068	0	0
Dedicated Logic Reg.	128300	93933	7860	18692
Block RAM [Kb]	19140	4182	81.4	64

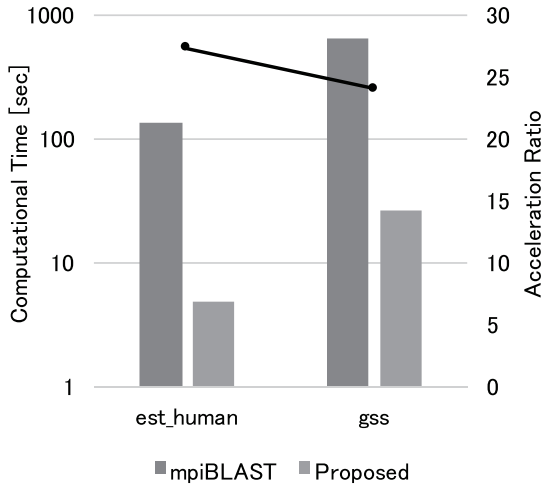

Figure 10. Performance of database partitioning

The total computation time for BLAST was about 948.47 [s], and Table VII indicates that word matching occupied over 70% of the computation time. Although BLAST executes several complex and flexible functions, accelerating the word matching step will improve the overall computation time.

Fig. 11 shows the throughput of data transfer among the main memory of the host PC, DRAM, and SSDs for APX-7142 configured with the DSPE-S described in Section V-C. The throughput between the host PC and SSDs is halved by the development version of the driver. However, the throughput can be improved to around 1 [GB/s] by upgrading the SATA-CORE from SATA2 to SATA3 and enhancing the driver.

The performance of IDP-BLAST is enhanced by offloading the *BlastAaWordFinder* function to APX-7142. Even though the ratio of computation time used by this function varies depending on the database sequences, it typically occupies 70–80% of the computation time. IDP-BLAST replaces the time required to execute this function with the time taken to read database sequences from the SSDs. The performance of IDP-BLAST is the same as the throughput

Table VII
PROFILES OF BLAST FOR ENV_NR

% Time	seconds	calls	function
70.86	665.56	19658001	BlastAaWordFinder
10.76	101.02	29807943	s_BlastSmallAaScanSubject
9.15	85.91	13314650	s_OutOfFrameGappedAlign
7.66	71.94	5335264	Blast_SemiGappedAlign
0.67	6.30	83866	ALIGN_EX

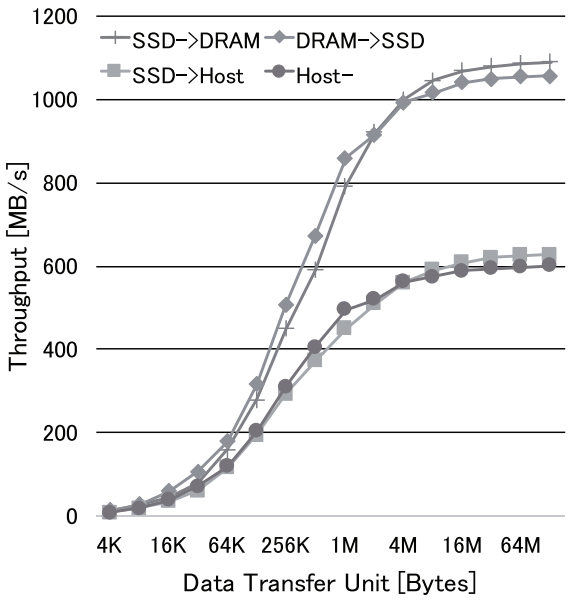

Figure 11. Throughput of SSDs for APX-7142

reading from the SSD to the host PC, as the hit positions are obtained when the database sequences are read from the SSD, as described in Section V-C.

The computation times for *env_nr* in Table VII are derived from:

$$(948.47[\text{s}] - 665.56[\text{GB}]) + (1.29[\text{GB}]/600[\text{MB}]) = 282.91 + 2.15 = 295.06[\text{s}] \quad (3)$$

We have confirmed that introducing IDP-BLAST accelerates the word matching step by a factor of around $665.56/2.15 = 309.562$, with few constraints on practical applications. The constraint that limits the length of the query sequence can be overcome by dividing the query sequence into fragments, as described in Ref. [11]. Finally, IDP-BLAST reduces the overall computation time by a factor of:

$$948.47[\text{s}]/295.06[\text{s}] = 3.214 \quad (4)$$

Each host PC only finds hit positions and executes the latter part of ungapped extension and gapped extension after reading the database sequences. These remaining processes could also be offloaded to other many- or multi-core accelerators. Since a lot of researches can be applied to the DSPE-based implementation to accelerate gapped extention by FPGA or other many-core processors, over 300 times acceleration for by FPGA-based wire-speed computation of preprocess of BLAST contributes reducing the load of processsor and the computational time.

VII. Conclusion

In this paper, we introduced the acceleration hardware for mpiBLAST, which is a software-based parallel sequence alignment algorithm for computational biology. The hardware is configured on the FPGA-boards we are developing to adopt in-datapath computing for two parts of computing on mpiBLAST, database partitioning and preprocess of BLAST. Utilizing an in-datapath computing mechanism enables the scanning of data and the tagging and sequence alignment processes to be offloaded. We found that database partitioning and distribution results in acceleration by a factor of 20, and that IDP-BLAST offers threefold acceleration compared to computations on a conventional PC cluster. The latter hardware, which accelerates the ungapped extension of BLAST over 300 times, can be combined to other acceleration technique for gapped extension we do not focus on the paper.

In future work, we will further examine two approaches: the first extends the application area of in-datapath computing to data processing systems such as distributed databases, and the second involves integrating the individual implementations of BLAST presented in this paper for more practical uses in computational biology.

Acknowledgment

This work was partially supported by JSPS KAKENHI 26330061 and 63003088.

References

[1] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Information Sciences*, vol. 275, pp. 314 – 347, 2014.

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation*, 2004, pp. 137–149.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003, pp. 29–43.

[4] F. Meyer, "Genome sequencing vs. moore 's law: Cyber challenges for the next decade," *CTWatch Quarterly*, vol. 2, no. 3, pp. 20–22, 2006.

[5] G. S. Davidson, K. W. Boyack, R. A. Zacharski, S. C. Helmreich, and J. R. Cowie, "Data-centric computing with the netezza architecture," *SANDIA REPORT*, pp. 1–24, Apr. 2006.

[6] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *41st Annual International Symposium on Computer Architecture (ISCA)*, 2014.

[7] M. Yoshimi, R. Kudo, Y. Oge, Y. Terada, H. Irie, and T. Yoshinaga, "An FPGA-based Tightly Coupled Accelerator for Data-intensive Applications," in *IEEE 8th International Symposium on Embedded Multicore/Many-core SoCs*, 2014, pp. 289–296.

[8] ——, "Accelerating OLAP workload on interconnected FPGAs with Flash storage," in *Proceedings of 2nd International Workshop on Computer Systems and Architectures(CSA'14)*, 2014, pp. 1–7.

[9] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, Oct 1990.

[10] A. E. Darling, L. Carey, and W. chun Feng, "The design, implementation, and evaluation of mpiblast," in *The HPC Revolution*, 2003.

[11] K. Mahadik, S. Chaterji, B. Zhou, M. Kulkarni, and S. Bagchi, "Orion: Scaling genomic sequence matching with fine-grained parallelization," in *SC*, 2014.

[12] M. Whitman and M. Fink, "Hp labs: The future technology," HP Discover Las Vegas, 2014.

[13] J. Kyathsandra and E. Dahlen, "Intel rack scale architecture overview," INTEROP, May. 2013.

[14] K. Asanović and D. Patterson, "Firebox: A hardware building block for 2020 warehouse-scale computers," in *12th USENIX Conference on File and Storage Technologies*, Feb. 2014.

[15] J. A. Delmerico, N. A. Byrnes, A. E. Bruno, M. D. Jones, S. M. Gallo, and V. Chaudhary, "Comparing the performance of clusters, hadoop, and active disks on microarray correlation computations," in *Proceedings of International Conference on High Performance Computing*, 2009, pp. 378–387.

[16] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. R. Ganger, "Active disk meets flash: A case for intelligent ssds," in *Proceedings of the 27th International Conference on Supercomputing*, 2013, pp. 91–102.

[17] S. Kim, H. Oh, C. Park, S. Cho, and S.-W. Lee, "Fast, Energy Efficient Scan inside Flash Memory SSDs," in *Second International Workshop on Accelerating Data Management Systems (ADMS)*, 2011, pp. 1–8.

[18] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Annkcorn, M. King, S. Xu, and Arvind, "Bluedbm: An appliance for big data analytics," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2015, pp. 1–13.

[19] J. Lancaster, J. Buhler, and R. D. Chamberlain, "Acceleration of ungapped extension in mercury blast," *Microprocessors & Microsystems*, vol. 33, no. 4, pp. 281–289, 2009.

[20] AVAL DATA, "APX880," https://www.avaldata.co.jp/english_08/products/giga/tera_storage/apx880.html, 2011.

[21] ——, "APX7142," 2014.

[22] H. Sugahara, *The fundamentals of Serial ATA and Implementation on an FPGA (in Japanese)*, ser. Tech I. CQ Publishing Co.,Ltd., 2010, vol. 44.

[23] S. Ishikawa, A. Tanaka, and T. Miyazaki, "Accelerating blast algorithm using an fpga (in japanese)," *Journal of Information Processing Society of Japan*, vol. 55, no. 3, pp. 1167–1176, Mar. 2014.