平成２６年度修士論文

複数FPGAボードを用いたビッグデータ分割処理に関する研究

大学院情報システム学研究科
情報ネットワークシステム学専攻

学 籍 番 号 :　1352008
氏 名:　　工藤　龍
主任指導教員: 吉永 努 教授
指 導 教 員 :　大坐畠　智 准教授
指 導 教 員 :　入江　英嗣 准教授
提出年月日 :　平成２７年２月２７日

(表紙裏)

目 次

図 目 次

表 目 次

第1章　序論

IoT（Internet of Things）[1] の進展に伴い，これまで以上に大規模なデータが収集されるようになり，蓄積の速度が上がっている．このようなデータはビッグデータと呼ばれ，それらを高速に扱う新たな仕組みが必要となる．近年，MapReduce を始めとする分散処理を利用して，複数ノードで並列に処理を行う仕組み（並列化）が研究されている [2, 3].

蓄積された大規模データベースを扱う問題の例として，生物データベースの配列類似性検索問題が挙げられる [4]．配列類似性検索は，多数の塩基配列がレコードとして格納されたデータベースから，クエリ配列の類似文字列を探し出す問題である．文字列を比較する計算は文字列アラインメントと呼ばれる．文字列のアラインメントに用いる検索アルゴリズムとして BLAST(Basic Local Alignment Search Tool)[5, 6] が主に使われている．しかし，塩基配列を格納したデータベース [7] の規模は年々増加しており，それにともなって検索に要する時間も長大化している．このため，複数ノードによる分散処理で並列にスコアリング計算を行う．分散処理を適用した実装として mpiBLAST[8] がある．mpiBLAST は MPI(Message-Passing Interface) を用いて処理を並列化することで処理の高速化を図っている．また，計算時間を短縮するため，各ノードに分配するデータ量を一定にし，負荷分散を図っている．こうしたデータの分散はホスト PC 上のプロセッサとネットワークに高い負荷を掛ける．

本研究では，PC クラスタの各ノードにストレージとネットワーク I/F を搭載する FPGA ボードを導入し，これを用いてデータ分割する手法を提案する．FPGA に実装するユーザロジックは，専用ネットワークから入力されるデータについて，ヘッダ情報から格納するデータを選択することで，ホスト PC の負荷を上げることなく，データの分散を実現する．

以降，2 章で関連研究について述べ，3 章で mpiBLAST のデータ分割について説明する．4 章では専用ハードウェアを用いたデータ分割の提案手法と実装について述べ，5 章で実験結果を示し，ソフトウェア実装の mpiBLAST と比較する．最後に，6 章で結論を述べる．

第2章　背景

2.1　FPGA(Field-Programmable Gate Array)

FPGA は，ユーザが書き換え可能なデバイスとして CPLD (Complex Programmable Logic Device) と並んで多く出荷されているデバイスである．LUT (Look-Up Table) を用いて書き換え可能なロジック回路を実現し，細粒度に配置することにより，大規模な回路が実装可能となる [9, 10]．また，近年のテクノロジーの進化によって，FPGA の高集積化，高性能化，低消費電力化，低コスト化が進み，カーナビやテレビなどのさまざまな電子機器で使用されるようになっている．

2.1.1　FPGA の構造

書き換え可能な回路を実現する基本要素は，SRAM や Anti-Fuse ROM などで構成された LUT である．n 入力 m 出力の LUT とは入力 n ビット，出力 m ビットの任意の論理関数を実現するための小容量のメモリである．多くの商用 FPGA は 6 入力 1 出力の LUT で構成されており，FPGA ではこのメモリに値を書き込むことで任意の論理関数を実現できる．本研究で用いた ALTERA 社の Stratix IV シリーズは，LUT を SRAM によって構成した SRAM 型 FPGA である．SRAM 型 FPGA は一般的な CMOS 半導体プロセスで製造されるため，最新の CMOS 半導体プロセスを利用することができ，高集積化プロセスによる回路規模の増大の恩恵を受けることが可能である．

図 2.1.1 は，現在の主要な商用 FPGA で採用されている Island-Style と呼ばれるアーキテクチャである．このアーキテクチャでは，

- Logic block
- Connection block
- Switch block

の 3 つのブロックで FPGA を構成する．Logic block は小規模の論理を真理値表回路で実現するブロックであり，隣接する connection block への配線 (図 2.1.1 では 1 本線で表現) を持つ．Connection block は Logic block の入出力と配線領域との接続の切り替えを行うスイッチである．また，Switch block は配線同士の接続を切り替えるスイッチである．Connection block および Switch block は，主にパストランジスタを用いたスイッチとその制御用のコンフィギュレーションメモリで構成される．図 2.1.1(a) は FPGA を構成する基本的なひとつのブロックであり，これを多数並べることで図 2.1.1(b) のように大きな回路を構成する．

2.1.2　FPGA アーキテクチャの一例: ALTERA 社 Stratix IV シリーズ

ここでは，FPGA のアーキテクチャの一例として，ALTERA 社の Stratix IV[11] アーキテクチャについて述べる．

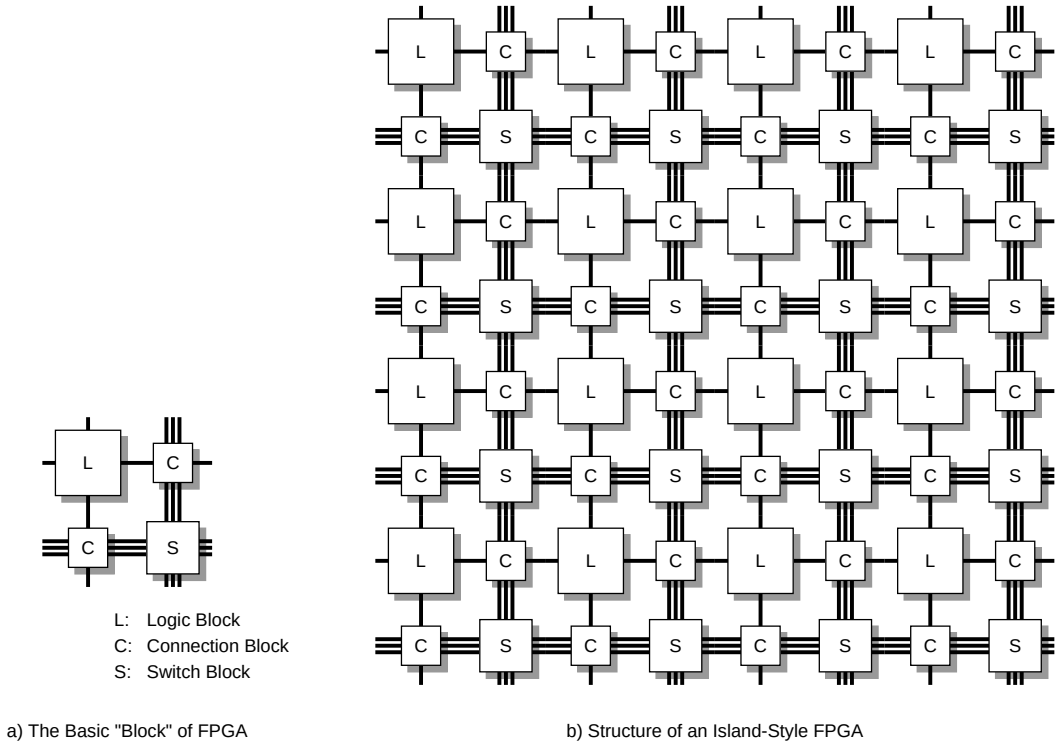

図 2.1.1: Island-Style FPGA の基本ブロックと全体の構造

前節で述べた logic block は，LUT だけで構成されるものではなく，実際には図 2.1.2 に示されるような複雑な構造になっている．

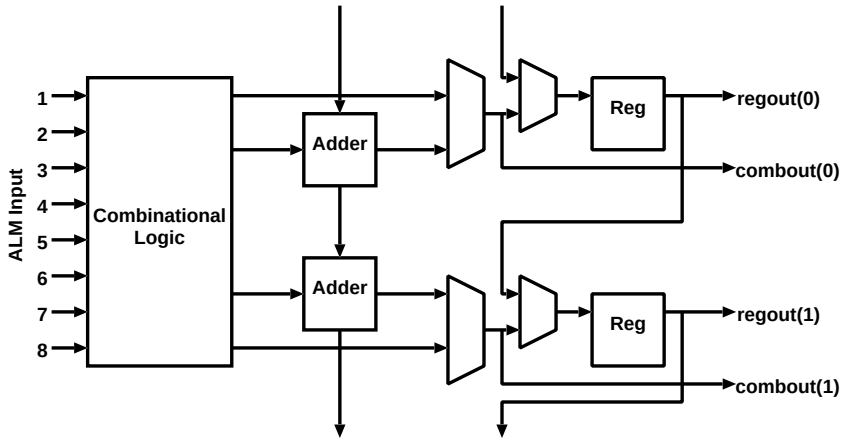

図 2.1.2: Stratix IV FPGA の ALM の構成

ALM の構成と活用

図 2.1.2 は，前節の logic block に相当する ALM (Adaptive Logic Module) の構成であり，ひとつの ALM には 8 本の入力を持つ分割可能な LUT，2 個の専用全加算器，キャリ・チェイン，共有演算チェイン，レジスタ・チェイン，2 個の専用レジスタが含まれる．これらの機能により，ALM を複数の LUT に分割して使用することができる．

図 2.1.3 に示すように，以下のように分割することができる．

- 1 つの Stratix IV デバイス ALM で任意の 6 入力 LUT を実装することができる（図 2.1.3(a) に示す）．この場合，1 つの 7 入力 LUT として動作する．
- 2 つの独立した 4 入力，またはそれより小さい LUT を実装することができる（図 2.1.3(b) に示す）．この場合，2 つの LUT の入力はそれぞれ独立して動作する．
- 1 つの 5 入力 LUT と 1 つの 3 入力 LUT を実装することができる（図 2.1.3(c) に示す）．この場合，2 つの LUT の入力はそれぞれ独立して動作する．
- 1 つの 5 入力 LUT と 1 つの 4 入力 LUT を実装することができる（図 2.1.3(d) に示す）．この場合，LUT 間の入力のうち 1 つは共通で，それ以外は独立した 2 つの LUT として動作する．
- 2 つの 5 入力 LUT を実装することができる（図 2.1.3(e) に示す）．この場合，LUT 間の入力のうち 2 つは共通で，各 5 入力 LUT に対して最大 3 つの独立した入力として動作する．
- 2 つの 6 入力 LUT を実装することができる（図 2.1.3(f) に示す）．この場合，LUT 間の入力のうち 4 つは共通で，各 6 入力 LUT に対して最大 2 つの独立した入力として動作する．
- 7 入力 LUT を実装することができる（図 2.1.3(g) に示す）．この場合，1 つの 7 入力 LUT として動作する．

Stratix IV デバイスの ALM の分割機能により，ハードウェアリソースを効率的に利用し，演算機や組み合わせ回路を実装できる．

Stratix IV FPGA 全体の構成

次に，Stratix IV FPGA の構成を図 2.1.4 に示す．Stratix IV は，チップ周辺にプログラマブル I/O Banks(IOB) が配置されている．IOB は CMOS インターフェイス以外にも LVTTL, HSTL, SSTL, LVDS, PCI, AGP など各種の信号規格に対応できるプログラマブルな構成で，DDR 入出力用のレジスタも備えている．

FPGA 上のロジック部は ALM が網目状に配置され，ALM の間は接続用の配線で接続される．ALM の間には，縦方向にメモリブロック (BlockRAM) および高性能デジタル信号処理ブロック（DSP Blocks）のストライプが数カ所配置される．メモリブロックや DSP Blocks などは頻繁に使われるが，LUT で構成した場合に大きな面積を必要とするため，これらを専用のハードマクロとして組み込むことにより回路面積を抑え，また，ハードウェアの動作周波数を向上させることが可能である．

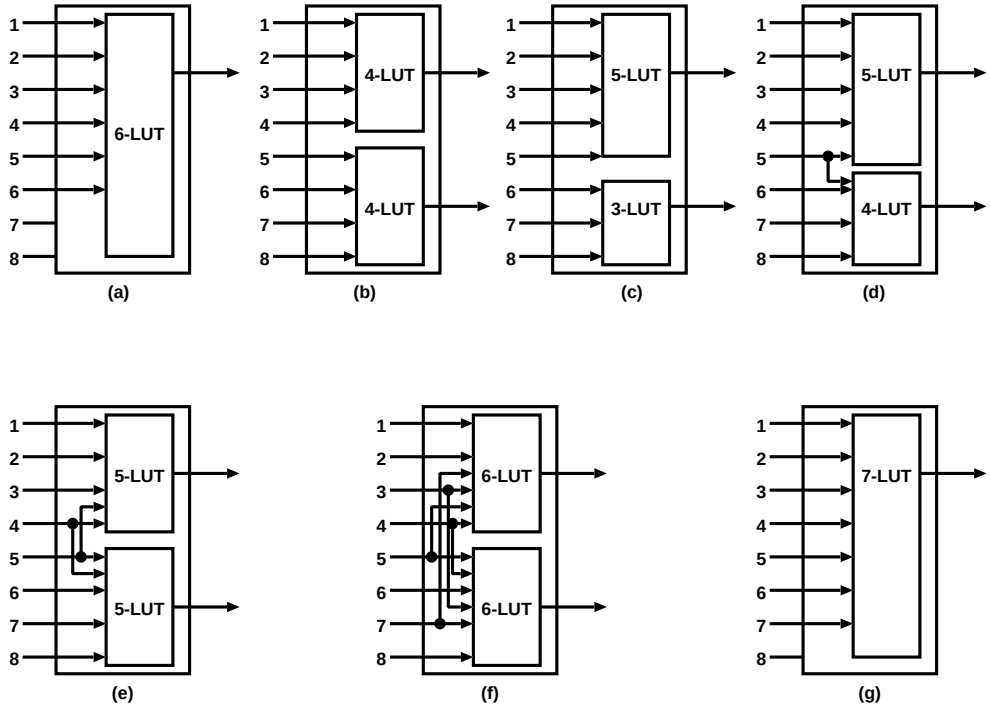

図 2.1.3: Stratix IV FPGA の ALM の分割

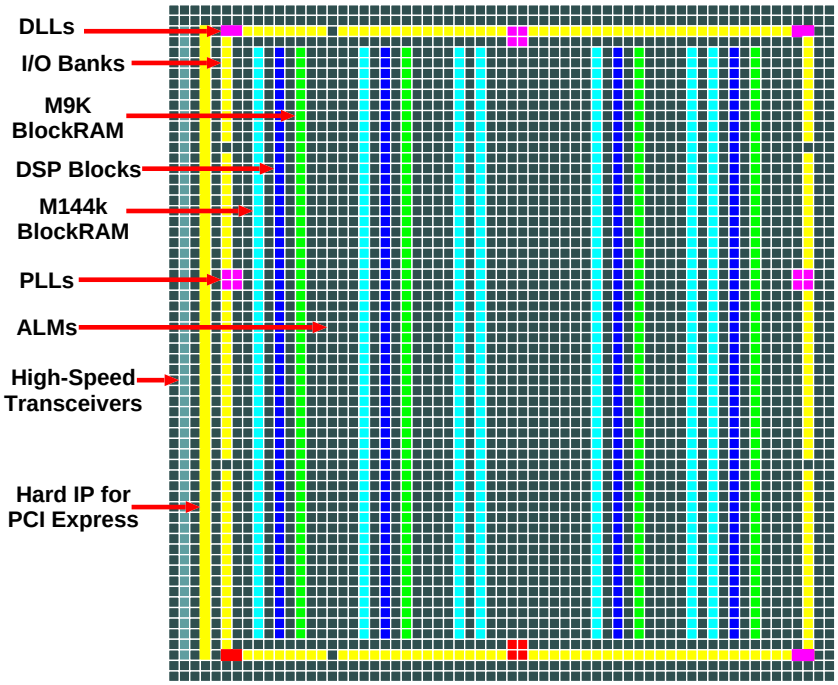

図 2.1.4: Stratix IV FPGA の構成

2.2 FPGA を用いた計算システム

2.2.1 FPGA を用いた計算システムの利点

従来, 科学技術計算を高速に行うには, ハードウェアから設計し，GRAPE[12] や MDM[13] などのような専用計算機を開発する必要があった. 専用計算機による計算処理ではひとつのチップ上に演算器を並べ，順番に接続することで，パイプライン実行が可能となる．計算をパイプライン化にすることで，高い計算性能を得られる．その一方，専用ハードウェアの問題としては，

- ある計算に特化しており，他の計算に利用できないため，資源の流用が困難
- システムをハードウェアから設計するため，コストが高い

といった点が挙げられる．

これに対して FPGA は，ハードウェア回路を構成するにより，専用ハードウェアの利点を持つ．また，ハードウェア回路をソフトウェア的に書き換え可能である．ハードウェアの高い性能とソフトウェアの柔軟性の両立が可能となる (図 2.2.1)[14]．さらに，FPGA の使用できるハードウェア容量が年々増加している．表 2.1 に示すように，2002 年にリリースされた 130nm プロセスの FPGA と，2010 年に発表された 28nm プロセスの FPGA とでは，その LE 数（Logic Element）に 10 倍以上の差がある．LE とは Stratix II まで使用された最小単位であり，LUT およびレジスタで構成される．現在では前述した分割可能 ALU によって構成される ALM に置き換わっている．大きなハードウェアリソースの FPGA を使用することで，より複雑な計算を実装することが可能になる．

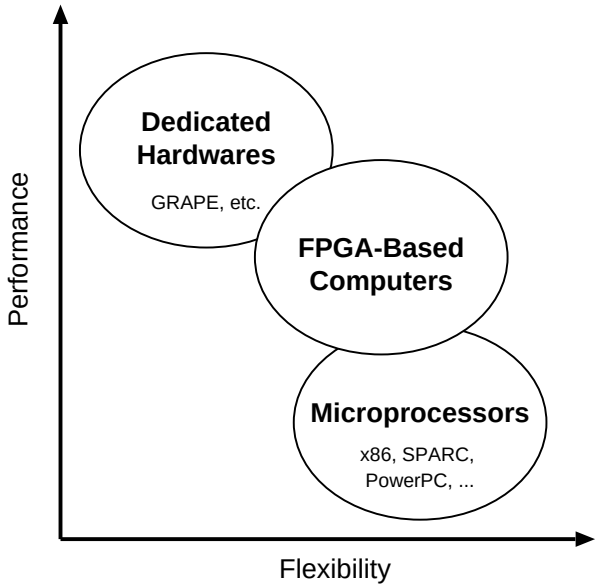

図 2.2.1: 性能と柔軟性

2.2.2　商用計算機への応用例

Netezza

Netezza とは IBM から販売されている商用データウェアハウスアプライアンスである [15]. Netezza の構成を図 2.2.2 に示す．Netezza は SMP ホストと S-Blades(Snippet Blades) および Disk からなる．SMP ホストは 2 台の高性能の LINUX サーバによって構成される．S-Blades はそれぞれ 8 コアの CPU と 8 チップの FPGA およびメモリで構成される．これらの CPU 1 コアと FPGA 1 チップとメモリで S-Processor（Snippet Processor）を構成する．Disk には磁気ディスクの HDD を複数使用している．S-Blades は 1 ラックに最大で 12 個搭載可能である．これは Netezza 1 ラックの中に 96 個の S-Processor 搭載していることになる．

これらのデータベースに特化した S-Processor が並列に動作することによって最大 145TB/時間の DB 処理を実現している [16]．また．Disk に格納されるデータはすべて FPGA によって圧縮される．

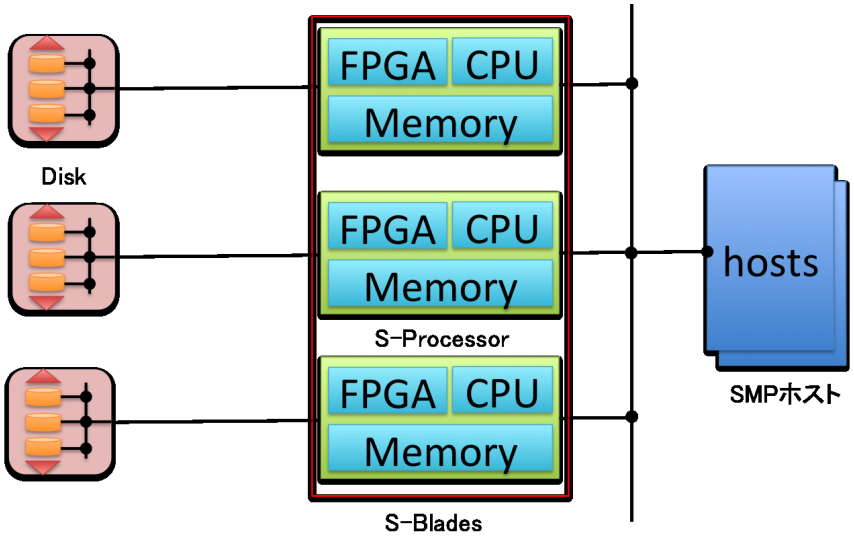

図 2.2.2: Netezza の構成

表 2.1: 各世代の FPGA のプロセス・LE 数と発売時期 (ALTERA 社)

プロセス	シリーズ	型番	LE 数 [Kb]	発売時期
130nm	Stratix	EP1S80	80	2002
130nm	Stratix GX	EP1SGX40G	40	2003
90nm	Stratix II	EP2S180	180	2004
90nm	Stratix II GX	EP2SGX130	130	2005
65nm	Stratix III	EP3SE260	338	2006
40nm	Stratix IV	EP4SE530	813	2008
28nm	Stratix V	5SGXBB	952	2010

Bing

Bing とは Microsoft が提供してウェブ検索エンジンである [17]．Microsoft では，2010 年から Bing 用に FPGA を搭載したサーバの開発が行われている．図 2.2.3 にサーバの構成を示す．1 つのサーバにストレージとしてハードディスクが 4 つ，SSD が 2 つ搭載している．ホストには 8 コアの Xeon 2.1GHz CPU が 2 つ，64GB のメモリおよび 10GbE が搭載している．また，サーバの先端には FPGA が搭載されたドータボードが配置されている．ドータボードには 2GB/s の SerialLite III リンクが 4 本付いており，隣のサーバと 6×8 の 2 次元トーラスネットワークを構成している．

FPGA に実装するハードウェアは 2 つのコンポーネントに分割されている．1 つは Shell と呼ばれ，外部または CPU との通信を制御するコントローラによって構成される．もう一つは Role と呼ばれ，アプリケーションの要求によって動的に処理用の回路を書き換えることができる部分である．以上の 2 つのコンポーネントに分けることにより，データセンタでの運用に必要な柔軟性を確保することができる．

また，Role に Bing 用の処理回路を書き込んだ 1632 台のサーバを用いたクラスタの実験では，既存のソフトウェアと比較して，スループットが 2 倍であることが報告されている．2015 年初期には Bing の実システムに投入するとしている．

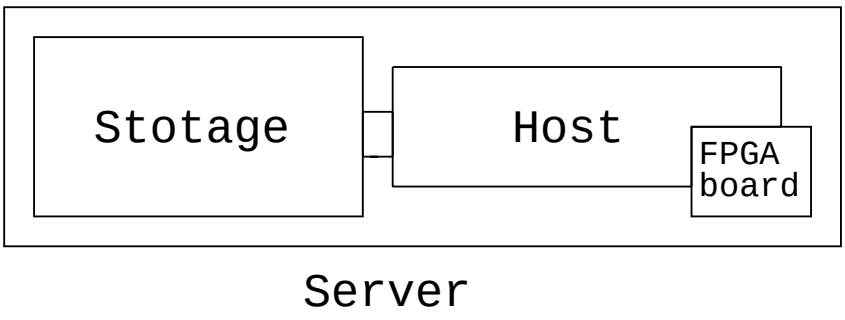

図 2.2.3: Bing の構成

2.3 Avaldata APX-880A

2.3.1 ハードウェアアーキテクチャ

本研究で実装に使用した FPGA ボードを図 2.3.1 に示す．また，そのストレージの拡張に用いる拡張ボードを図 2.3.2 に示す．APX-880A[18] はアバールデータ社が開発した高速ストレージボードである．本ボードは，光通信モジュールと SD カードコネクタを搭載しており，ネットワークとストレージへのデータ転送を高速に行うことができる．全ての機能はボード上の FPGA から制御可能である．また，ホストマシンからのアクセス経路として，PCI Express Gen2x4 を採用している．ホストマシンからも，データの転送や各種制御が可能である．

本ボード上の FPGA には，SD カードコントローラ，光通信コントローラ（GiGA Channel，後述），PCIe コントローラが実装されている．これらのモジュールは，1 つのバススイッチに接続されており，自由にアクセスすることができる．また，DMA コントローラが 2 つ実装されており，ホストマシンへの負荷をかけずに，データの転送を行うことができる．さらに，データ転送や計算

のバッファとして使用可能な SDRAM とそのコントローラも搭載している．ホストマシンからの制御に必要なすべてのレジスタは，レジスタマップモジュールにより，PCIe の空間にマッピングされている．バススイッチを経由した全ての経路は，1GB/s 以上の帯域で動作可能であり，データ転送上のボトルネックが存在しない．

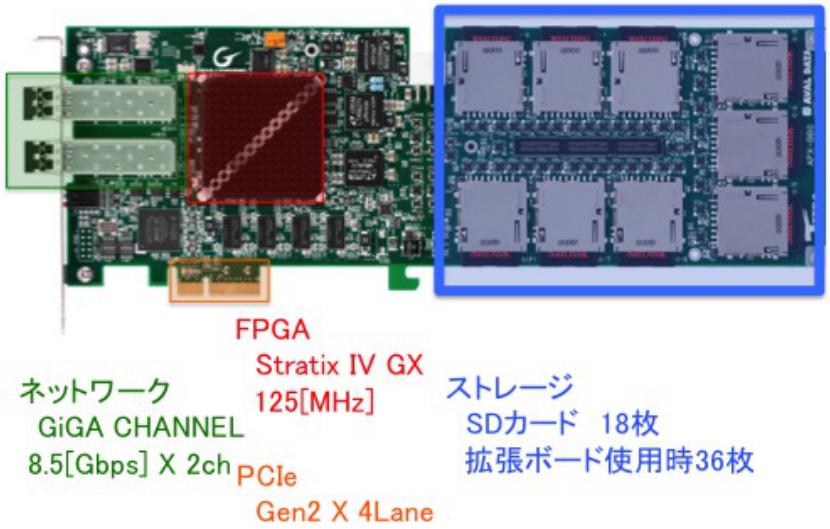

図 2.3.1: APX880A ボード

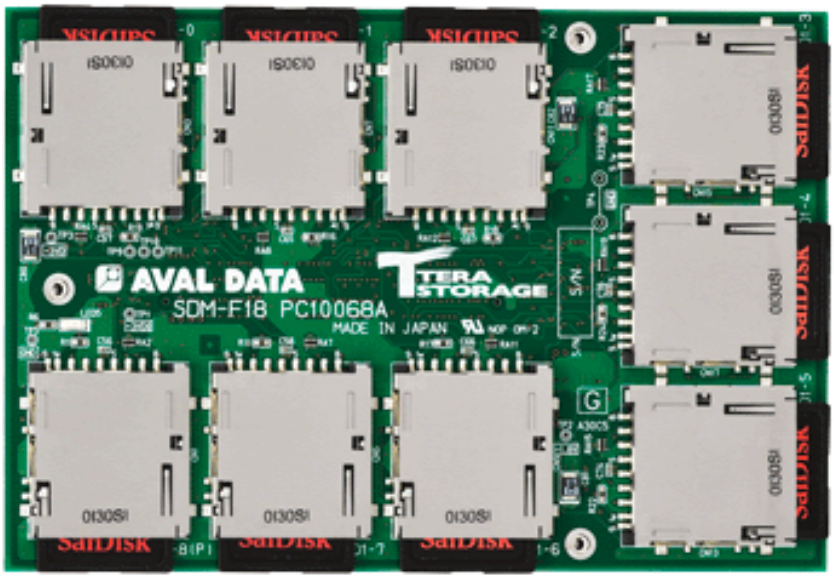

図 2.3.2: APX880A 拡張ボード

2.3.2 GiGA Channel

APX-880A は GiGA Channel を用いて FPGA 間通信を行うことができる．GiGA Channel とは，アバールデータが独自に開発した高速通信プロトコルであり，FPGA 用 IP コアとして提供・実装されている．本プロトコルを用いることで，通信のプロトコル処理はハードウェアで行われるため，FPGA による高速なデータの分散を伴うシステムを容易に構築することができる．GiGA Channel は，複数のノードをリング状に接続して使用する（図 2.3.3 に示す）．リング状に接続されたすべてのノードは，1 つの仮想的なアドレス領域を共有する（共有メモリ領域）．データの送信は，共有メモリ領域への書き込みという形式で行う．書き込まれたデータは，自動的に全てのノードに対して，アドレス情報とともに送信される．データの送信は，全てのノードが自由に行うことができる．また，全てのノードは受信したデータの扱いを自由に決定することが許される．例えば，現実のメモリに格納したり，ストリームデータとして使用したりすることができる．不要なデータと判断し，破棄することも自由である．なお，他ノードからデータを読み出す，という概念は存在しない．

GiGA Channel には，FPGA 間の高速通信とデータの分散を同時に行う機能が備わっており，その全ての機能を FPGA 上に実装することができる．そのため，ソフトウェアによる操作を，特定のノードへの割り込み通知（ドアベル）や，各種エラーの確認等に限定することができる．また，各ノードはデータの送受信を自由に行うことができるため，様々なシステムに応用することが可能である．

GiGA Channel では，データが全ノードへ自動的に送信されるため，通信情報量がノード数に比例して増加する．複数のノードが大量のデータを送信しようとした場合，通信帯域を超えてしまう場合があるが，自動的にフロー制御が行われるため，データが欠落することはない．

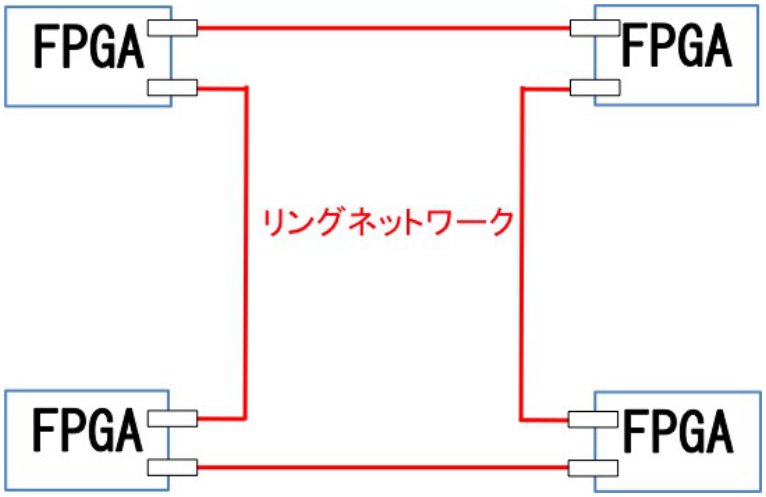

図 2.3.3: GiGA Channel の接続例

2.4 関連研究：データ分割

分割とはデータベースを複数の部分に分割することである．分割には大きく分けて “テーブルの分割”，“ノード間の分割”がある．テーブル間の分割は大規模なテーブルを複数に分けて格納することにより，データアクセス単位を小さくすることに用いる．一方，ノード間の分割は大規模なデータベースを分割し，複数のノードに分配することでノード間の並列処理を可能とする．適切な分割は，ノード間のデータ転送量を減らし，計算ノードがデータに対して並列アクセスによるスループットを向上させる．

ビッグデータ処理では，データベースのパフォーマンスはデータ分割に大きく影響される．また，処理時間の多くをデータ分割が占める場合もある [19]．データ分割を高速に行うため，プロセッサに組み込まれる専用ハードウェアが提案されている [20]．関連研究 [20] では，データに対するアクセスレイテンシを削減するために，専用ハードウェアにレンジデータ分割を実装している．専用ハードウェアを使用することによりソフトウェアと比較して，スループットが 9 倍向上すると同時に，消費電力も 9 分の 1 程度に削減できることが報告されている．

2.5 関連研究：FPGA を用いた BLAST の高速化

従来から，BLAST の処理の一部または全部を，FPGA で構成した専用ハードウェアにオフロードする研究が行われてきた [21, 22, 23]．関連研究 [22] では，BLAST の全処理を FPGA にオフロードすることに成功し，ソフトウェア実装に比べ約 790 倍の高速化を達成した．しかし，大規模なデータ計算に対する分散処理は実現していない．

そこで本研究では，ハードウェア支援によりデータ分散を高速に実現する計算機システムを実装し，後述する mpiBLAST の分散フェーズの計算時間を短縮する．

第3章　mpiBLASTとデータ分割の重要性

3.1　mpiBLASTの動作

mpiBLAST[8] は，NCBI BLAST[6] の並列実装である．mpiBLAST では，主に 2 つのフェーズに分けて，文字列のアラインメントを行う．1 つ目のフェーズは並列実行を可能にするためのデータベース配列の分割と BLAST 処理に適したデータにするためのフォーマットを行う mpiformatdb である．2 つ目のフェーズは文字列のアライメントを行う mpirun である．以下に mpiBLAST の動作について述べる．

まず，mpiformatdb コマンドを用いて，データベース配列を任意の数に分割し，それぞれについて内部で NCBI BLAST の formatdb を呼び出し，フォーマットを行う．当該データベース配列は共有ディレクトリに格納される．分割後の処理を高速に行うため，各ノードの処理時間を揃える必要がある．そこで，分割後のデータサイズを一定となるように分割を行う．また，データ分割処理は入力データが 1 つであるため，並列に行うことができていない．

次に Open MPI の mpirun コマンドを介して mpiblast コマンドを各ノードで実行する．mpiblast を実行すると，文字列のアラインメントを行うノードは分割されたデータベースのうち，自分の担当のファイルを共有ディレクトリから取得し，各々で文字列のアラインメントを行う．

mpiBLAST では少なくとも 2 プロセスがタスクのスケジューリングおよびファイル出力のコーディネートに使用されるため，実際に mpirun コマンドで実行するプロセス数は，BLAST 検索を行う場合に比べ 2 プロセス増加する．

3.2　ハードウェア支援による mpiBLAST のデータ分割

前節で述べたように，mpiBLAST はデータを分割するフェーズと，データの分配および文字列のアラインメントを行うフェーズの 2 つからなる．mpiBLAST を対象とした研究 [21, 22, 23] の多くは文字列のアラインメントをターゲットとしている．しかし，関連研究 [20] が主張するように，分散システムの処理時間としてデータを分散する部分が多くの割合を占めている．mpiBLAST においても，処理時間全体の約 60%がデータの分散が占めている．従って，mpiBLAST の全体の高速化を図るためには，データの分散を高速化する必要がある．このデータの分散に，専用ハードウェアを用いた高速なデータ分散システムを適用する．また，本研究はデータ分散のみを対象とするため，NCBI BLAST の fotmatdb はオフロードの対象外とする．

Flash ストレージ，ネットワーク，DRAM を搭載する FPGA ボードを用いて，FPGA 上にデータ分割のための専用ハードウェアを実装する．ネットワークから入力されるデータについて，ヘッダ情報から格納するデータをハードウェアが選択することで，ホスト PC の負荷を上げることなく，データの分散を実現する．

3.3 FASTA フォーマット

NCBI BLAST や mpiBLAST では，データベース配列およびクエリなどのシーケンスデータは，FASTA フォーマットで記述される．FASTA フォーマットは，ヘッダ行とシーケンス文字列の平文で構成される．ヘッダ行とは先頭の 1 行のことであり，一文字目は必ず“＞”から始まる．“＞”以降はそのシーケンス文字列の識別情報が記述される．

ヘッダ行とシーケンス文字列を合わせて，1 レコードとする．データの一例を図 3.3.1 に示す．

```
>gi|26355|emb|Z19607.1| HSAAAAAKP P, Human foetal Brain Whole tissue Homo sapiens cDNA, mRNA sequence
GATTACCCTATATCTACAATTNGAGGTAAAATAGAAGCAACACATAAAAGGGCCTATTTCTGCTACCATGATAAAAGGGCCTATTTCTGCTACCATGAATT
NGAGGTAAAATAGAAGCAACACATAAAAGGGCCAATTNGAGGTAAAATAGAAGCAACACATAAAAGGGCCAATTNGAGGTAAAA
>gi|26355|emb|Z19607.1| HSAAAAAKP P, Human foetal Brain Whole tissue Homo sapiens cDNA, mRNA sequence
GATTACCCTATATCTACAATTNGAGGTAAAATAGAAGCAACACATAAAAGGGCCTATTTCTGCTACCATGTCATATAATTGGCCTATTTCTGCTACCATGA
TAAAAGGGCCTATTTCTGCTACCATGAATTNGAGGTAAAATAGAAGCAACACATAAAAGGGCCAATTNGAGGTAAAATAGAAGCAACACATAAAAGGGCCA
GGCCTATTTCTGCTACCATGATAAAAGGGCCTATTTCTGCTACCATGAATTNGAGGTAAAATAGAAGCAACACATAAAAGGGCCAATTNGAGG
```

図 3.3.1: FASTA フォーマットのデータ例

第4章　設計と実装

4.1　専用ハードウェアによるデータ分割手法

本研究は高速なデータ分割を実現するために，PC クラスタの各ノードにストレージとネットワーク I/F を搭載する FPGA ボードを導入し，これを用いてデータ分割する手法を提案する．専用ハードウェアを用いた PC クラスタ（4 ノード）の構成を図 4.1.1 に示す．

FPGA ボードは，アバールデータ社製 APX-880A を用いる．APX880A の構成を表 4.1 に示す．APX-880A は，光通信モジュール（GiGA Channel）と SD カードコネクタを搭載しており，SD カードに格納されたデータの高速通信が可能である．GiGA Channel とは，2.3.2 節で述べたように，アバールデータ社が独自に開発した高速通信プロトコルである．リング状に接続されたネットワークを活かし，書き込まれたデータは，自動的に全てのノードに送信される．本研究で用いる PC クラスタは，図 4.1.1 のように GiGA Channel で相互接続した 1 つの送信ノードと複数（図 4.1.1 では 3 つ）の受信ノードで構成する．

送信ノードは，ストレージからデータベースを読み出し，各レコードのヘッダ行に分割ルールに従ってタグを付けて，タグ付けしたデータをリングネットワーク経由して送信する．受信ノードはタグを見て格納データを選択する．格納データはバッファ領域である DRAM に格納した後，ブロックサイズでストレージに転送する．

提案システムはデータ分割において高速であることを評価するために，3.1 節で紹介した mpiBLAST に適用し，ケーススタディとする．ケーススタディとして挙げた mpiBLAST のデータ分割処理に合わせるために，分割ルールはデータベースを全受信ノードに一定なサイズとなるように分割を行うこととする．

表 4.1: AVALDATA APX880A 仕様

FPGA Device	Stratix IV GX	
	EP4SGX230KF40C2	125 [MHz]
DRAM	DDR3(533MHz), 512MB	8.53 [GB/s]
Storage	18 SD Memory Cards×1 or 2	1.5 [GB/s]×2ch
	(up to 64[GB]×32 = 2[TB])	two SDs for parity
	unit to transfer	128[MB]
Network	Proprietary GiGA CHANNEL	
	Optical token ring network	8.5 [Gbps]×2ch
	unit to transfer	16[KB]
PCIe I/F	Gen2×4 Lane	2 [GB/s]
		(simplex)
Internal Bus	Proprietary AVAL-bus	128bits-width

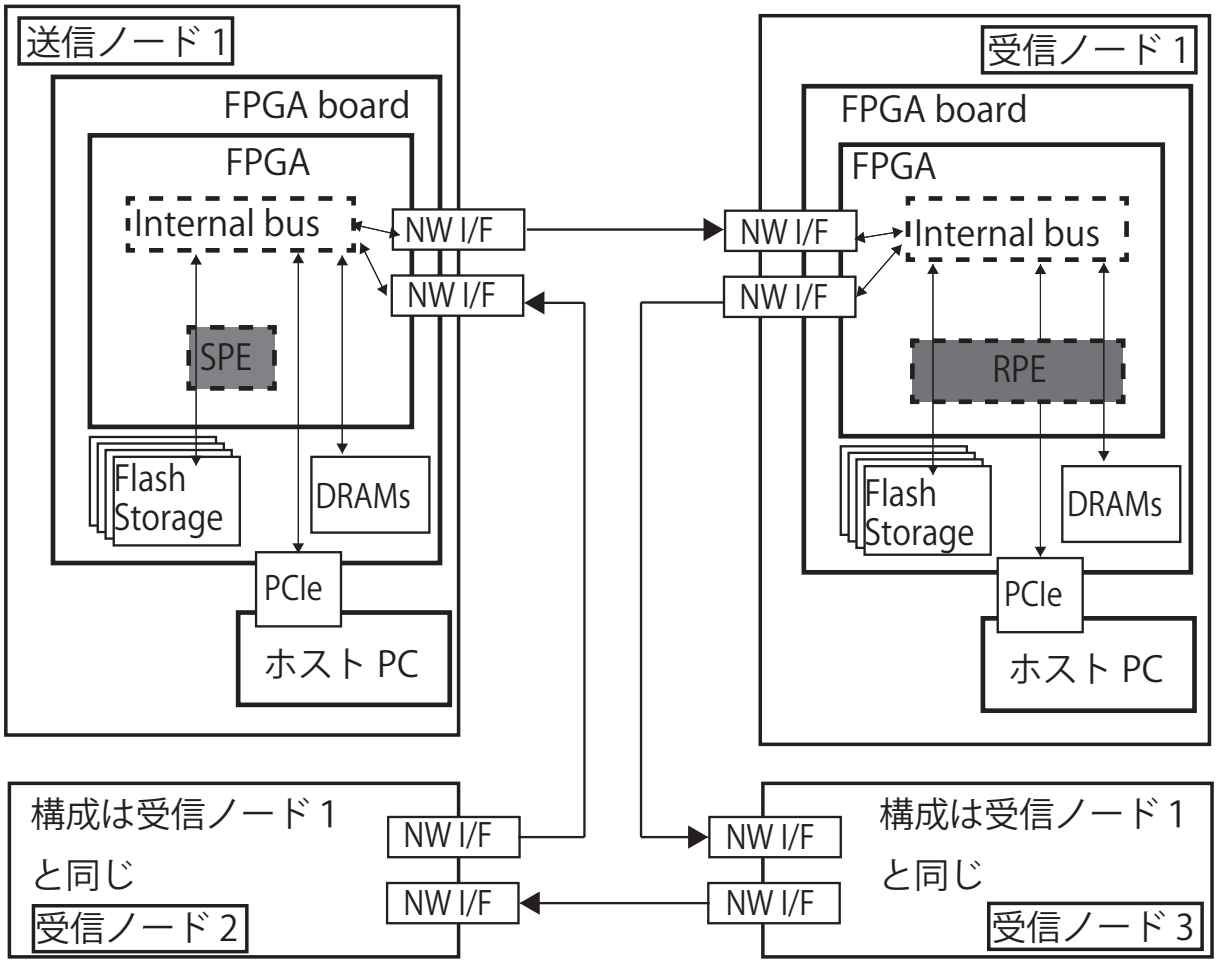

図 4.1.1: システムアーキテクチャ

4.2 設計

図 4.2.1 に，FPGA 内部に実現するハードウェア構成図を示す．ハードウェアは各種データ転送用モジュールとデータ分割を行うための専用モジュールで構成される．データベース配列のデータ分散をハードウェアで行うために，データの分割を行う専用モジュール“Send Processing Engine（SPE）”を送信ノードの FPGA 上に実装する．SPE のブロック図を図 4.2.2 に示す．そして，受信ノードの FPGA にはデータの取捨て選択を行う専用モジュール“Recv Processing Engine（RPE）”を実装する．RPE のブロック図を図 4.2.3 に示す．

SPE は FPGA ボード上のストレージからデータを読み出し，ネットワークに送出する際にデータのヘッダに対してタグ付け操作を行う．SPE では，それぞれの受信ノードに対して送信したデータサイズのカウンタを保持する．受信ノードのノード ID をデータのヘッダ部分（＞ gi | ）に付与し，サイズが最も小さいカウンタを所有する受信ノード向けにデータを送信する．そして，新たに送信した分のデータサイズを送信先のカウンタに加算する．

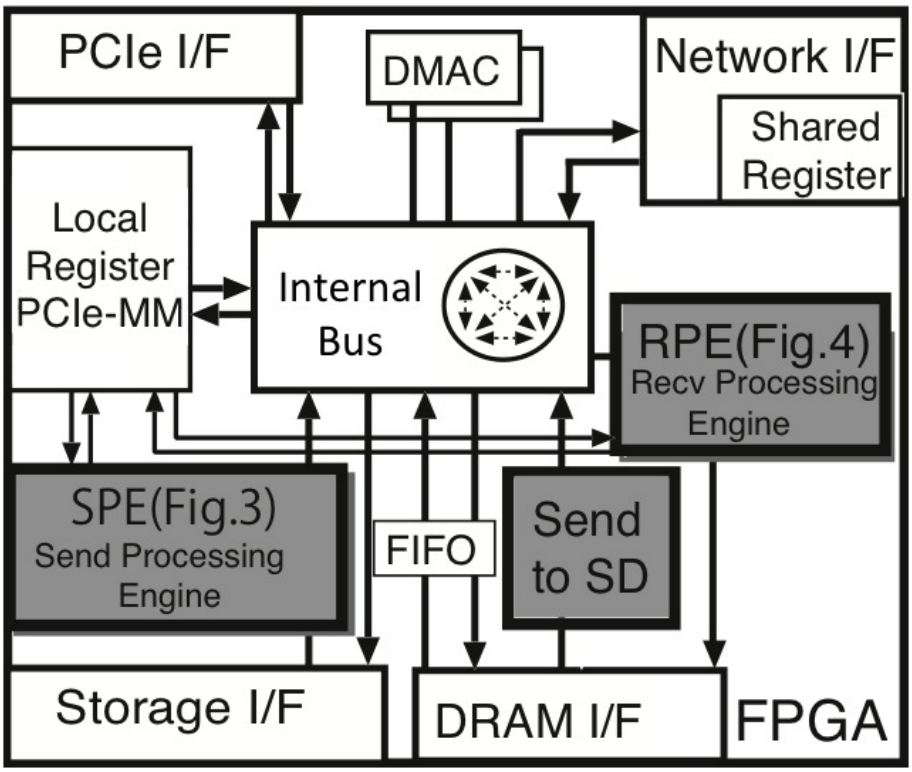

図 4.2.1: FPGA 内部に構成するハードウェア

RPE は FPGA ボード上の GiGA Channel からデータを受信する．データのヘッダ部分（＞ gi | ）に付与されているタグを監視し，データをフィルタリングする．Local Register に設定されたノード ID と一致した場合，データを DRAM に格納する．一方，一致しなければ受け取ったデータを無視する．また，Send to SD は RPE によって DRAM に格納されたデータを 128MB のブロックサイズでストレージに転送する．

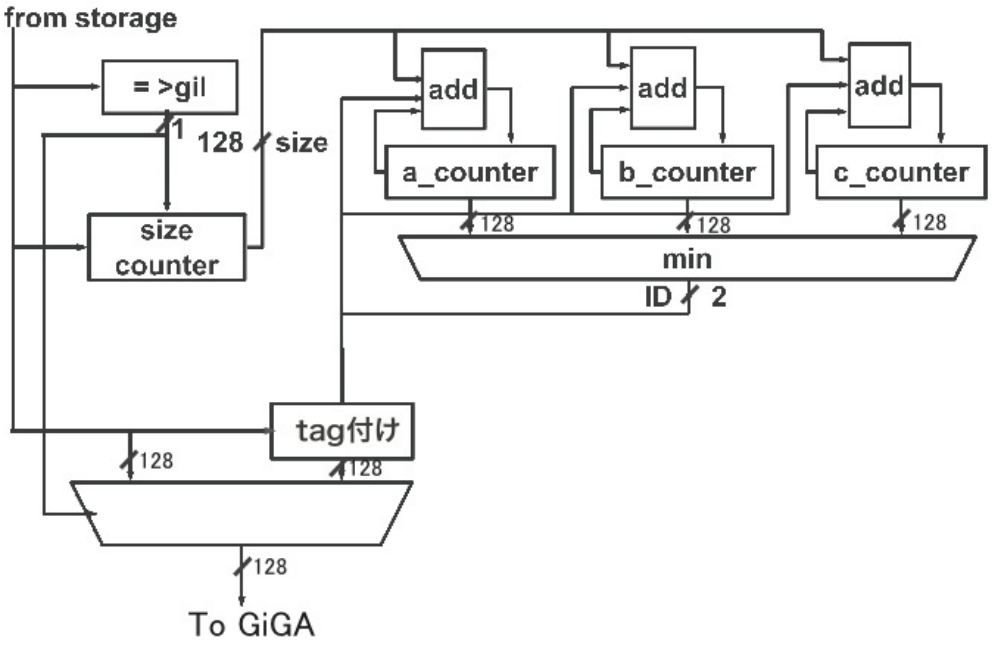

図 4.2.2: Send Processing Engine（SPE）のブロック図

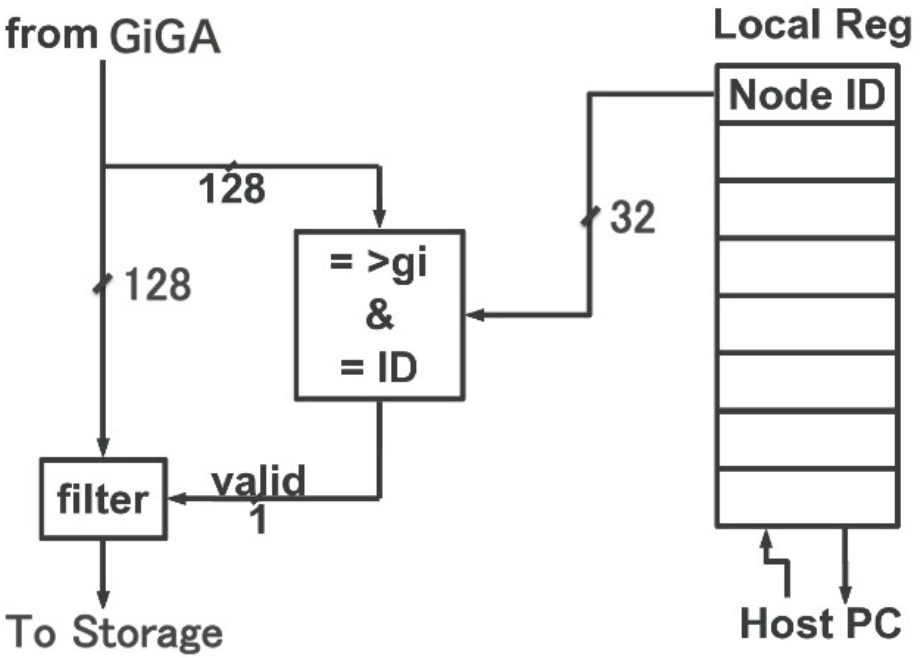

図 4.2.3: Recv Processing Engine（RPE）のブロック図

4.3 実装

データの転送や各種制御を行うために，既存のモジュールとして以下の機能を実装した（表 5.4 ではベースラインと表記）.

- ネットワークやストレージ，DRAM および PCI Express を制御するためのコントローラ，
- 各モジュール間のデータのやり取りに使用する BUS SWITCH，
- ホスト PC と制御情報をやり取りするための Local Register Array，
- ホスト PC の負荷を減らし，データの転送を制御する DMA (Direct Memory Access) コントローラ.

また，以上の既存モジュールに加えて，データのストリーム処理エンジンとして以下の機能を実装する（表 5.4 では追加回路と表記）.

- 分割データを一定にするために，送信データに対して，ヘッダにタグを付ける SPE モジュール，
- ヘッダのタグ部分を見て，データを選択する RPE モジュール，
- DRAM に格納されたデータをストレージである SD カードアレイに転送する Send To SD モジュール.

第5章　実験および評価

5.1　評価方法

データの分割に要する時間と，その際のホスト PC の資源利用率について評価を行う．評価に使用するデータベースは est_human，gss である．評価結果を従来の mpiBLAST と比較を行う．

評価環境は FPGA ボードを搭載したホストを用いて 4 ノードで構成する PC クラスタを使用する．クラスタに用いたホストの構成およびソフトウェア環境を表 5.1，表 5.2 に示す．また，ユーザロジックを組み込む前の FPGA ボードのストレージおよびネットワークのレイテンシを図 5.1.1，図 5.1.3 に示す．データサイズおよびレイテンシから計算した実行スループットを図 5.1.2，図 5.1.4 に示す．

表 5.1: 実験用ホストの仕様

No. of Node	4	
使用機種	Dell Precision T5610	
CPU	Intel Xeon E5-2630 v2	2.60[GHz]
メモリ	4GBx4 DDR3 RDIMM 1866MHz	16 [GB]
ネットワーク	Broadcom BCM957810A1008G	10[Gbps]
SSD	Crucial CT480M500SSD1	480[GB]
HDD	Seagate ST2000DM001	2[TB]

表 5.2: 実験用ホストのソフトウェア環境

OS	CentOS 6.6 kernel-2.6.32-504.1.3.el6.x86_64
SW Compiler	gcc-4.4.7
Compiler Option	-O2
HW CAD	Altera Quartus 10.1sp1
mpiBLAST	mpiBLAST-1.6.0

ストレージの実行スループットはホストに搭載した FPGA ボードのストレージに対してデータの読み書きスループットである．各データサイズのデータをホストで生成し，FPGA ボードのストレージに対して書き込みを行う．また，書き込まれたデータをホストまで読み出す処理を行う．図 5.1.2 からわかるように，ホストから FPGA ボードに搭載されたストレージに対して，書き込みは 900[MB/sec] 以上のスループットを実現している．また，読み込みに関しては 1000[MB/sec] 以上のスループットを実現している．これは PCIe で接続された SSD と同等の性能を示している．

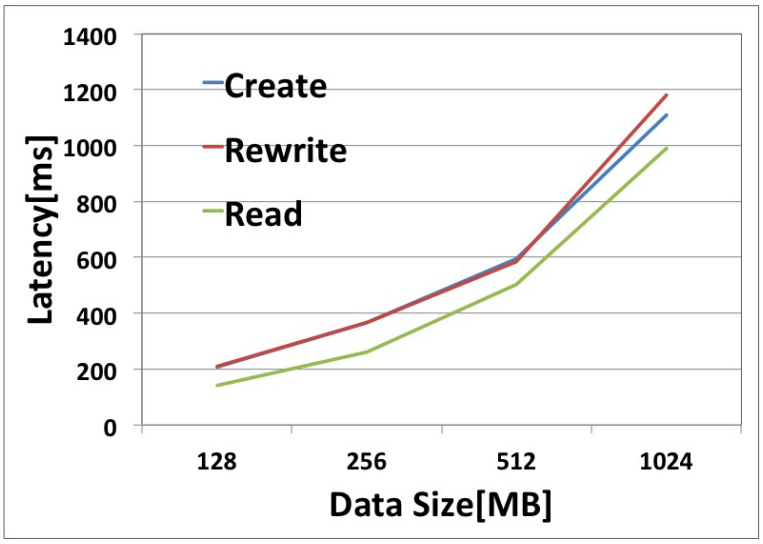

図 5.1.1: APX880A のストレージのレイテンシ

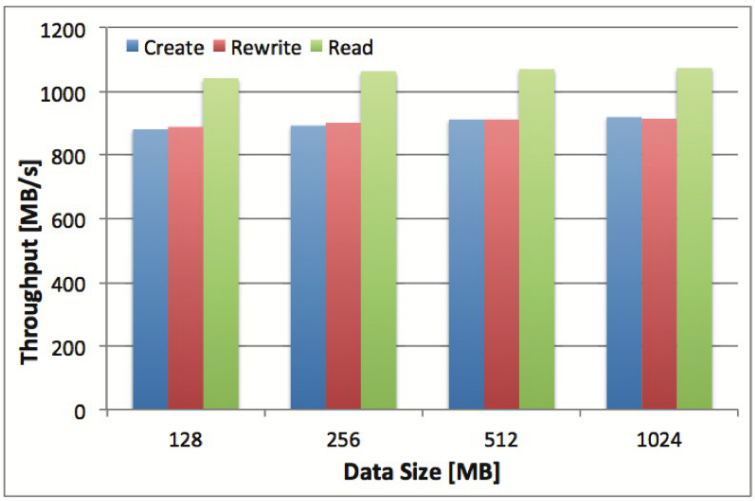

図 5.1.2: APX880A のストレージの実行スループット

GiGA Channel の実行スループットは，FPGA に搭載されたストレージに格納されいるデータを GiGA Channel によって他の FPGA のストレージに転送するときのスループットである．転送元での評価を Send，転送先での評価を Recv としている．図 5.1.4 からわかるように，Send は 1000[MB/sec] 以上のスループットを実現している．また，Recv は 900[MB/sec] 以上のスループットを実現している．表 4.1 からわかるように，GiGA Channel の理論転送速度は 17Gbps である．図 5.1.4 の評価では理論転送速度に対して十分な転送速度が得られていないのは，評価で FGPA に搭載されたストレージを使用しているためである．

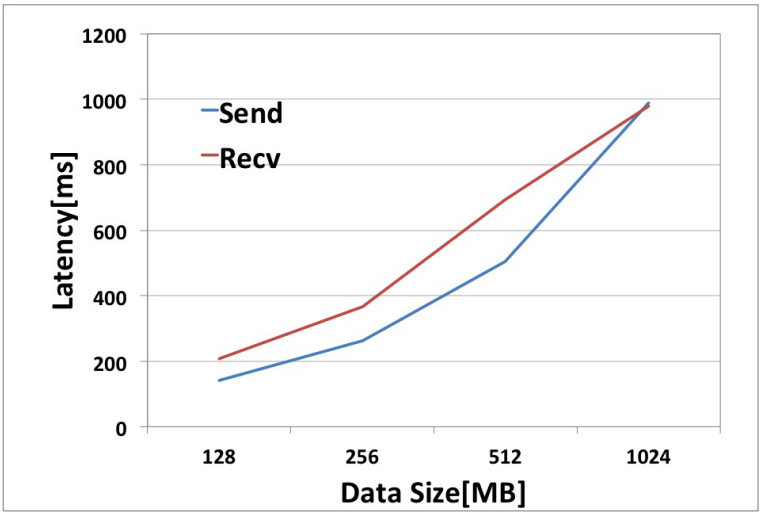

図 5.1.3: APX880A の GiGA Channel のレイテンシ

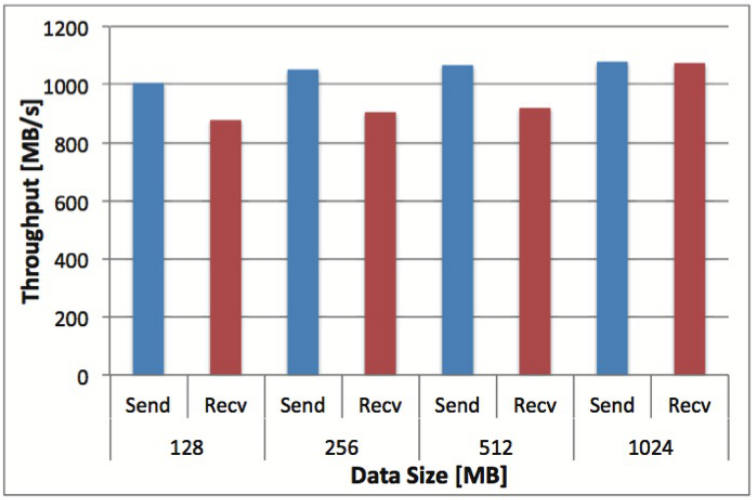

図 5.1.4: APX880A の GiGA Channel の実行スループット

FPGA の実装では，APX880A の仕様上，データバスは 128bit である．塩基配列のデータベースを用いた場合はデータのヘッダをデータバスの先頭に配置するため，“!” を用いてデータの Padding を行う．Padding を行ったデータベースを表 5.3 に示す．両データベースともに Padding によるデータの増加率は 1%前後と小さく，評価に対する影響が十分に小さい．Padding 後のデータの一例を図 5.1.5 に示す．

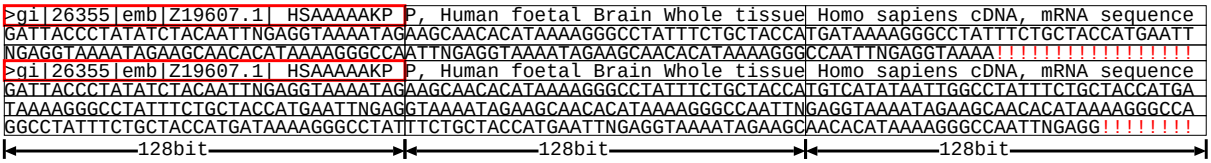

図 5.1.5: Padding 後のデータ例

表 5.3: 実験で使用するデータベース

	Padding 前 [GB]	Padding 後 [GB]
est_human	5.157	5.219
gss	27.973	28.240

提案システムの実験では，分割対象となるデータベースを送信ノードの FPGA に搭載されたストレージである SD に格納し，他のノードに転送する．転送と同時各ノードで分割処理が行われ，処理によって得られた分割されたデータは受信用の 3 ノードの FPGA のストレージに格納する．これに対して，ソフトウェア実装である mpiBLAST の実験では，データベースをホストの SSD に格納し，3.1 節で述べた mpiformatdb 処理を行う．処理によって得られた分割データをホストの 10 gigabit Ethernet を用いて他のノードに分散する．mpiBlast の実装上，mpiformatdb 処理と分割データを他のノードに分散する処理はパイプライン化されていない．また，処理を行う際に，分散処理に関係しない NCBI BLAST の formatdb を計測時間の対象外とするため，全体の実行時間から除外する．

5.2 評価

5.2.1 実行時間の評価

データサイズが 5.2GB の est_human および 28.2GB の gss に対するデータ分割実験で得られた実行時間の評価を図 5.2.1 に示す．前節で述べたように，ソフトウェアの評価を行う際に，分散処理に関係しない NCBI BLAST の formatdb を計測時間の対象外とするため，全体の実行時間から除外した．そのために，NCBI BLAST の formatdb の実行時間を単体で計測し，ソフトウェア評価である mpiBLAST の全体の実行時間から除外する．その結果，図 5.2.1 に示すように，両データベースに対して共に提案システムを用いた方が mpiBLAST に比べて実行時間を大幅に削減している．est_human に対して mpiBLAST がデータ分散処理に 133.9 秒要するのに対して，提案システムはわずか 4.9 秒で処理を終えており，27 倍の高速化が確認された．また，gss に対しても mpiBLAST は 640.7 秒であるのに対して提案システムは 26.6 秒で終えており，24 倍の高速化が確認された．さらに，データベースのサイズおよび実行時間から，APX880A を用いたシステムのスループットは，

$$5.2[GB] \div 4.9[s] = 1.061[GB/s] \tag{5.2.1}$$

$$28.2[GB] \div 26.6[s] = 1.060[GB/s] \tag{5.2.2}$$

となる．APX880A を用いたシステムでは，両データベースに対して 1GB/sec 以上で塩基配列データベースを分散することができる．この値は図 5.1.2 および図 5.1.4 に示す APX880A のストレージや GiGA Channel の実効スループットとも一致する．従って，実装したハードウェアは GiGA Channel の通信速度でデータ分散することが可能である．そして，ストレージや GiGA Channel の通信帯域を拡張することで，さらにスループットの向上が期待できる．

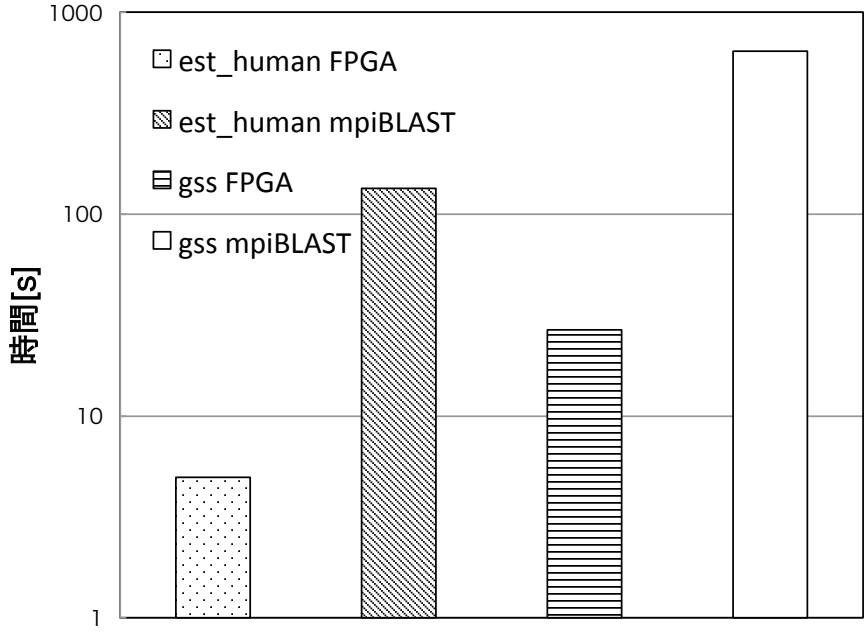

図 5.2.1: データ分割の実行時間

5.2.2 PC リソース使用率の評価

est_human に対するデータ分散を行うホスト PC におけるリソース使用率の評価を図 5.2.2 に示す．5.2.1 節で述べたように，mpiformatdb 処理には NCBI BLAST の formatdb 処理が含まれるため，全体の実行時間の評価から除外したが，PC リソース使用率の評価では除外できず，含まれている．そのため，図 5.2.2 に示す mpiBLAST に関する評価では，CPU 使用率，Disk I/O の使用率およびネットワーク使用率の横軸に示す測定時間には，NCBI BLAST の formatdb の実行時間が含まれている．図 5.2.2 に示すように，mpiBLAST の CPU 使用率は約 10%である．表 5.1 に示した通り，クラスタの各ノードに搭載されている CPU は Intel Xeon E5-2630 v2 であり，6 コア 12 スレッドの CPU である．また，3.1 節で述べたように，mpiformatdb はシングルスレッドのプログラムであるため，図 5.2.2 に示している約 10%の使用率は 1 スレッドを 100%使用していることを意味する．上記の NCBI BLAST の formatdb による影響があるものの，データ分散するためには PC ホストに対する負荷が高い．これに対して，提案システムは送信ノード 1 つと受信ノード 3 つで処理を分散し，ハードウェアが処理を制御している．このため，提案システムを用いた方が mpiBLAST に比べて CPU 使用率やディスク I/O，およびホストのネットワーク使用率を大幅に削減していることが確認された．

また，mpiBLAST で est_human を処理する際，ディスク I/O が示すように，最初に SSD からデータベースを読み出し，処理しながら結果を書き戻している．しかし，その後に他ノードへデータを分配するため，SSD から結果を読み出す処理が行われていない．表 5.1 に示した通り，クラスタの各ノードのメモリは 16GB であるのに対して，est_human のデータサイズは 5.2GB である．mpiBLAST でデータ分散を行う際に，最初の SSD からデータを読み出す以外の処理に関するデータはメモリに全て格納できる．このため，最後の処理であるデータを他のノードへの分配は，SSD から読み出す必要がなく，メモリから転送されることによる．これに対して，提案システムは前述のようにストレージの読み出し速度で分割を行うことができるため，データのサイズや位置を気にする必要がない．

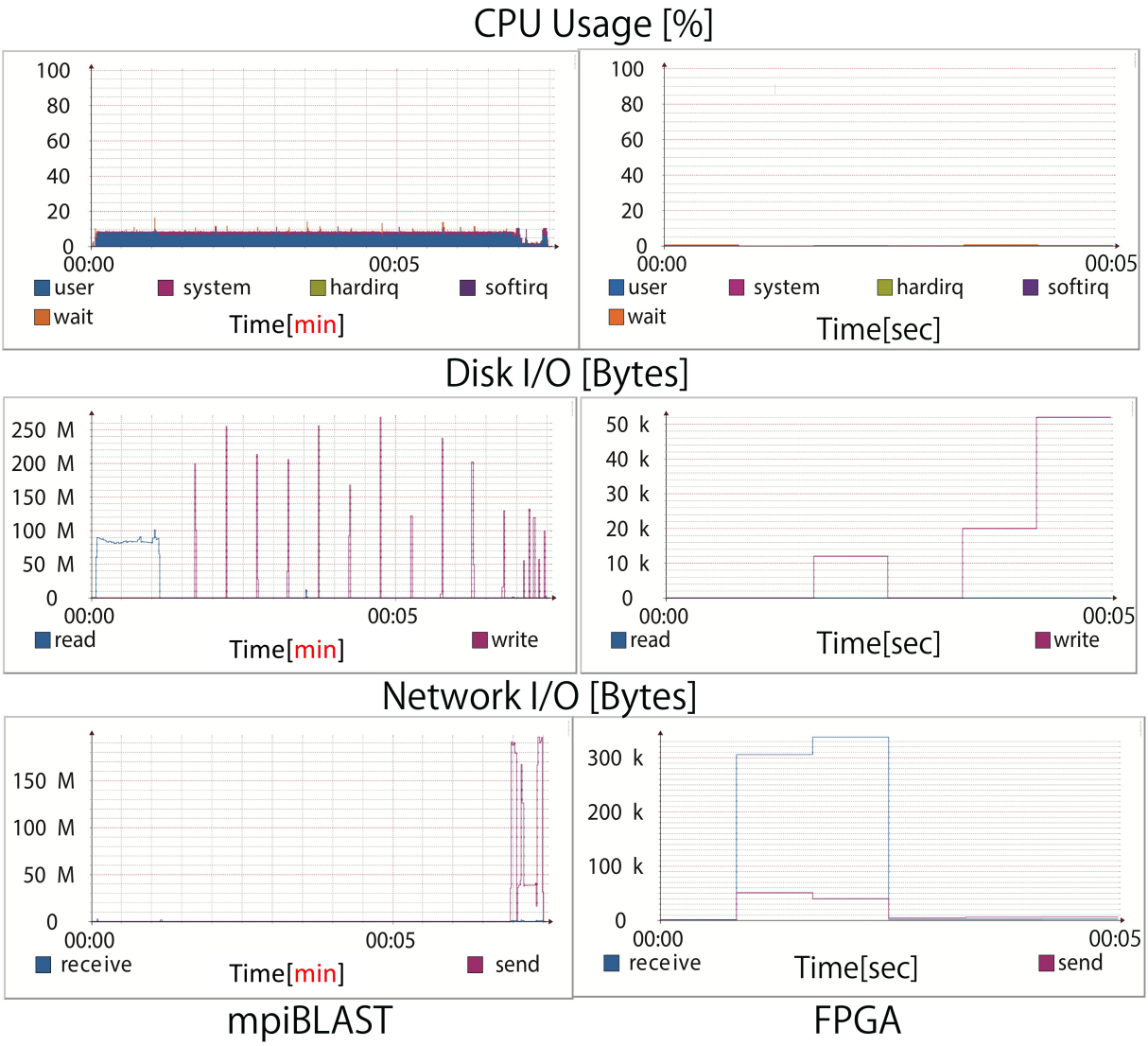

図 5.2.2: ホストのリソース使用率（est_human）

5.2.3 ハードウェアリソース使用率の評価

表 5.4 に，使用する FPGA のリソース量と 4.3 節で述べた既存回路（ベースライン），および我々が追加した SPE，RPE，Send to SD モジュールの回路規模を示す．また，追加回路の内訳を表 5.5 に示す．提案回路は既存のモジュールに加えて我々が実装したモジュールが消費したリソース使用率の合計である．表 5.5 からわかるように，追加回路の SPE および RPE は比較的に少ないリソースで実装することができ，FPGA リソースに余裕があることから他のユーザロジックも今後実装可能である．ベースラインから上昇した多くのリソースは追加回路と Internal Bus を接続するために消費されている．

表 5.4: ハードウェア全体のリソース使用量

	リソース	ベースライン	提案回路
Logic Utilization[%]	100	57	64
Combinational ALUTs	182400	65997	74910
Memory ALUTs	91200	695	750
Dedicated logic registers	182400	75558	80843
Total block memory[Kb]	14283	4797	5073
DSP block 18-bit elements	1288	0	0

表 5.5: 追加回路のリソース使用量

	Send to SD	SPE	RPE
Combinational ALUTs	1338	1554	271
Memory ALUTs	0	0	0
Dedicated logic registers	1012	726	386
Total block memory [Kb]	49	65	0
DSP block 18-bit elements	0	0	0

第6章　結論

本研究では，ストレージとネットワークを密結合するハードウェアが実装される FPGA ボードを用いて，データ分散処理をオフロードするシステムを提案した．提案システムを用いて，mpiBLAST の mpiformatdb フェーズをアプリケーションとして実データを使用した評価を行った．評価の結果，ソフトウェア実行の mpiBLAST に比べて，24〜27 倍の高速化を確認した．また，CPU 使用率，Disk I/O の使用率およびネットワーク使用率の大幅な軽減を示した．そのため，軽減したホストのリソースを他の計算に活用することが可能となる．

mpiBLAST の分散フェーズにおいて，図 5.2.2 から分かるように，分割処理と通信がパイプライン化していない．これ対して，専用ハードウェア上の高速なネットワークおよびストレージを活用することにより分割処理と通信を同時に行うことで，高速化を実現した．また，文字列の分割のような CPU にとって高負荷となる処理を専用ハードウェアにオフロードすることにより高速化を実現した．

今後はデータのサイズでデータ分割を行うだけでなく，BLAST の処理に適するようにデータを解析し，データ分割を行うシステムに拡張する．また，我々が次期プロトタイプボードとしているアバールデータ社が開発した高速光通信ボードである APX7142[24] では，ネットワークの通信性能が大幅に向上している．我々の提案システムを APX7142 に実装し，評価を行う．

謝辞

本研究を進めるにあたり，ご指導, ご助言をいただきました，情報システム学研究科情報ネットワークシステム学専攻ネットワークコンピューティング学講座，吉永努教授に厚く御礼申し上げます．ゼミでの議論や論文の添削を通し，多くの指導を頂きました．

同講座入江英嗣准教授，吉見真聡助教にも研究を進めるに当たり多くの助言をいただき，研究がより良いものとなりました．ありがとうございました．

株式会社アバールデータの寺田祐太氏，筆者が所属する FPGA 研究グループのオゲ ヤースィン先輩には研究・実装に関する多くのアドバイスを頂きました．また，ネットワーク研究グループの須戸里織氏には学会論文を提出する際に，お世話になりました．感謝の意を表します．

日常の議論を通し，多くの指摘を下さいました吉永研究室・入江研究室の先輩方，同期の皆様，後輩の皆様に感謝します．本当にありがとうございました．

参考文献

[1] Strategy, ITU and Unit, Policy. ITU Internet Reports 2005: The internet of things. *Geneva: International Telecommunication Union (ITU)*, 2005.

[2] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation*, pp. 137–149, 2004.

[3] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 29–43, 2003.

[4] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, Vol. 147, No. 1, pp. 195 – 197, 1981.

[5] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. *Basic local alignment search tool*, Vol. 215. 1990.

[6] J. Fassler and P. Cooper. NCBI: BLAST Help, 2011. http://www.ncbi.nlm.nih.gov/books/NBK62051/.

[7] NCBI: BLAST DATABASE. http://ftp.ncbi.nlm.nih.go/blast/db/FASTA/.

[8] mpiBLAST: Open-Source Parallel BLAST. http://www.mpiblast.org/.

[9] Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 1992.

[10] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-submicron FPGAs*. Kluwer Academic Publishers, 1999.

[11] Altera Corporation. Stratix IV Device Handbook. Mar. 2014.

[12] Atsushi Kawai, Toshiyuki Fukushige, Jun ichiro Makino, and Makoto Taiji. Grape-5: A special-purpose computer for n-body simulations. *Publ. of the Astronomical Society of Japan*, Vol. 52, pp. 659–676, Aug. 2000.

[13] 泰岡顕治, 薄田竜太郎, 戎崎俊一, 加藤健矢, 小林芳直, 成見哲, 古沢秀明, G. Elmegreen, B., D. McNiven, G., 大口晃司. 16 分子動力学専用計算機 mdm. 熱流体系および固体系のミクロシミュレーションに関する合同シンポジウム・分子動力学シンポジウム講演論文集, Vol. 2000, No. 5, pp. 31–32, feb 2000.

[14] Jürgen Becker and Reiner Hartenstein. Configware and morphware going mainstream. *Journal of System Architecture*, Vol. 49, No. 4-6, pp. 127–142, 2003.

[15] IBM. NETEZZA. www.ibm.com/software/data/netezza/.

[16] unisys. Netezza TwinFin シリーズ. http://www.unisys.co.jp/solution/netezza/twinfin.html.

[17] A. Putnam, A.M. Caulfield, E.S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G.P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P.Y. Xiao, and D. Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pp. 13–24, June 2014.

[18] AVAL DATA. 高速ストレージボード:APX880A. http://www.avaldata.co.jp/products/z1_embedded_zz/avalother_products/apx880/apx880.html.

[19] Changkyu Kim, Tim Kaldewey, Victor W. Lee, Eric Sedlar, Anthony D. Nguyen, Nadathur Satish, Jatin Chhugani, Andrea Di Blas, and Pradeep Dubey. Sort vs. hash revisited: Fast join implementation on modern multi-core cpus. Vol. 2, pp. 1378–1389. VLDB Endowment, August 2009.

[20] Lisa Wu, Raymond J. Barker, Martha A. Kim, and Kenneth A. Ross. Navigating Big Data with High-throughput, Energy-efficient Data Partitioning. In *Proceedings of SIGARCH Computter Architecture News*, Vol. 41, pp. 249–260. ACM, June 2013.

[21] P. Laczkó, B. Fehér, and B. Benyó. FPGA-based BLAST Prefiltering. In *Proceedings of the 14th International Conference on Intelligent Engineering Systems*, INES'10, pp. 278–281. IEEE Press, 2010.

[22] 石川淑, 田中飛鳥, 宮崎敏明. FPGA を用いた BLAST アルゴリズムの高速化. 情報処理学会論文誌, Vol. 55, No. 3, pp. 1167–1176, mar 2014.

[23] A. Jacob, J. Lancaster, J. Buhler, and R.D. Chamberlain. FPGA-accelerated seed generation in Mercury BLASTP. In *Proceedings of IEEE 15th International Confrence on Field-Programmable Custom Computing Machines*, pp. 95–106, April 2007.

[24] AVAL DATA. 高速光通信ボード:APX7142. http://www.avaldata.co.jp/products/z3_gigachannel/apx7142/apx7142.html.

発表論文

[1] Masato Yoshimi, Ryu Kudo, Yasin Oge, Yuta Terada, Hidetsugu Irie, Tsutomu Yoshinaga. “An FPGA-Based Tightly Coupled Accelerator for Data-Intensive Applications,” in Proceedings of IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs (MCSoc), 2014.

[2] Masato Yoshimi, Ryu Kudo, Yasin Oge, Yuta Terada, Hidetsugu Irie, Tsutomu Yoshinaga. “Accelerating OLAP workload on interconnected FPGAs with Flash storage,” International Workshop on Computer Systems and Architectures(CSA’14)

[3] 工藤 龍, 須戸 里織, オゲ ヤースィン, 寺田 祐太, 吉見 真聡, 入江 英嗣, 吉永 努. “複数 FPGA ボードを用いたビッグデータ分割処理の高速化,” 信学技報, vol. 114, no. 427, pp.193-198, Jan. 2015.