DEIM Forum 2021 B13-3

FPGAを用いたラベル付きグラフに対する正規パス問合せアクセラレータの改良

三浦　賢人† 　天笠　俊之†† 　北川　博之††

† 筑波大学システム情報工学研究科コンピュータサイエンス専攻　〒305–8577 茨城県つくば市天王台 1 丁目 1-1
†† 筑波大学 計算科学研究センター　〒305–8577 茨城県つくば市天王台 1 丁目 1-1
E-mail: †miura.k@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし　グラフ構造は身の回りの様々なデータを表すのに効果的なデータ構造である. ビッグデータ分析などの普及に伴って， 現在様々な分野でグラフ構造データが用いられている. そのようなグラフ構造データからユーザの望むデータを抽出する方法の一つとして，正規パス問合せ (RPQ) が存在する. RPQ はエッジにラベルが付与されたグラフに対して行われる問合せであり，指定されたエッジの並びをもつパスをグラフ内から探索し，そのパスの始点・終点ノードを返す. ここで課題としてあげられるのは，RPQ 評価の高速化である. 近年のデータ分析での対象データの大規模化の傾向から RPQ の対象となるグラフも大規模化が予想されており，これらの大規模なグラフに対しては実行に多大な時間を要することが想定される．このような課題を解決するため我々は FPGA に着目した．FPGA は任意の論理回路をプログラミングによって自由に実装できるデバイスであり，その性能上の特徴は各回路の並列性を利用した並列度の高い処理が可能なことである. 本研究では，RPQ 評価をパイプライン的に処理するための手法とその FPGA 実装を提案する．性能調査のための実験では，RPQ で指定されるパスの長さに比例した実行時間を示し，比較手法と比べ最大で約 3 桁の高速化が得られた．また，あるグラフに対して同一のパス長の RPQ を実行する場合では実行時間がほぼ一定になることや和演算や繰り返し演算などを含む様な複雑な RPQ に対しても実行時間が影響されないといった特徴を確認した.

キーワード　FPGA, 正規パス問い合わせ, ハードウェアアクセラレータ, グラフデータベース

1 はじめに

グラフ構造は身の周りの様々なデータの関係性を表すために非常に便利なデータ構造である．我々の日々の生活の中ではスマートフォンや各種センサーなどによって多種多様なデータが取得されており，ソーシャルネットワークや linked open data, 化学化合物ネットワークといった現実世界のエンティティ同士の関係性はグラフ構造として変換され保持されている.

これらの様なグラフデータからユーザの望むデータを取得するための手法の一つとして，正規パス問合せ (RPQ) [1] が存在する．RPQ はエッジにラベルが付与されたグラフを対象とした問合せであり，指定されたエッジの並びを持つパスをグラフないから探索し，その始点・終点ノードを結果として返す.

ここで課題となるのが，RPQ 評価の計算時間である．近年のデータ分析での対象データの大規模化の傾向から，RPQ の対象となるグラフも大規模化が予想されており，現実世界に存在する様な多種多様かつ大規模なグラフに対しては実行に多大な時間を要することが想定される.

このような処理の高速化の分野で広く用いられているのが, GPU や FPGA などのハードウェアアクセラレータである. 従来の CPU のみの環境に GPU や FPGA を組み合わせたヘテロジニアスな計算機を構築することで高速化を目指すといった手法である.

RPQ 評価の高速化において，我々は特に FPGA に着目した．FPGA は任意の論理回路をプログラミングによって自由に実装できるデバイスであり，実装された各回路の並列性を利用したパイプライン処理を得意とする.

以上の点を踏まえ本研究では FPGA を用いた RPQ 評価の高速化手法を提案する．非決定性有限オートマトン (NFA) を利用した RPQ 評価と Frontier を用いた並列幅優先探索を組み合わせた演算パイプラインを考案し，その FPGA 実装を行った.

性能評価のための実験では，RPQ で指定されるパスの長さに比例した実行時間を示し，比較手法と比べ最大で約 3 桁の高速化が得られた．また，あるグラフに対して同一のパス長の RPQ を実行する場合では実行時間がほぼ一定になることや和演算や繰り返し演算などを含む様な複雑な RPQ に対しても実行時間が影響されないといった特徴を確認した.

本稿の構成を以下に示す．第 2 節で前提知識，第 3 節で関連研究を紹介する．第 4 節で提案手法について述べ，第 5 節で実験の結果を示す．最後に第 6 節で本研究のまとめを述べる.

2 前提知識

2.1 正規パス問合せ (RPQ)

正規パス問合せ (RPQ) は，ラベル付きグラフ中の任意の 2 ノード間に指定したパスが存在するかを調べる問合せである. 処理対象となるラベル付きグラフを $G = (V, E, \mathcal{L})$ とする．V

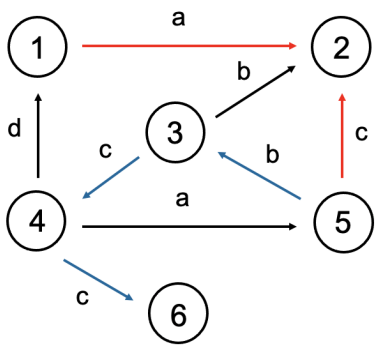


図 1　RPQ の例 ($R = a \circ b \circ b$ を適応)

はグラフ内の頂点の集合，E はグラフ内のエッジの集合，$\mathcal{L}$ はグラフ中のエッジに対する全ラベルの集合である．さらに，エッジ集合 E からラベル集合 $\mathcal{L}$ に対する写像関数 $\phi : E \to \mathcal{L}$ を定義する．ここで，$\ell \in \mathcal{L}$ とすると，以下のような RPQ 評価のセマンティクス R が定められる．

$$R ::= \epsilon \,|\, \ell \,|\, \ell^{-} \,|\, R \circ R \,|\, R \cup R \,|\, R^{i,j} \tag{1}$$

ここで，ϵ は空列，ℓ はラベルを順方向に，ℓ^{-} は逆方向にたどる表現であり，$\circ$ は結合，$\cup$ は和，$R^{i,j}$ は R の k 回 ($i \leqq k \leqq j$) の繰り返し $\underbrace{R \circ \cdots \circ R}_{k\,times}$ を表す．例として，図 1 のグラフを G_{ex} とし，RPQ : $R = (a \circ c^{-}) \cup (b \circ c^{1,2})$ を適用すると

$$R(G_{ex}) = \{(1,5),(5,4),(5,6)\} \tag{2}$$

となる．この RPQ 処理では図中の赤色と青色でハイライトされたパスが検出され，最終的にその始点・終点ノードの集合である $\{(1,5),(5,4),(5,6)\}$ が得られる．

2.2　FPGA

Field Programmable Gate Array (FPGA) は任意の回路をプログラミングによって繰り返し実装可能なハードウェアチップである．近年では FPGA 性能の向上により，開発者はより複雑な回路を FPGA 上に実装可能になっており，それに伴い FPGA を用いた開発は画像処理や数値計算，機械学習など様々な分野で活発に行われている．

従来，FPGA は Verilog や VHDL のようなハードウェア記述言語によってプログラムされることが一般的であり，大規模な回路を作成する場合の開発コストの高さが問題視されていた．しかし，近年では C/C++や OpenCL といったより高レベルな言語によって FPGA 上の回路をプログラムできる高位合成 (HLS) という技術が確立されてきている．本研究では HLS を使用して FPGA 上への回路の実装を行った．

3　関 連 研 究

3.1　FPGA を用いた高速化の事例

FPGA を用いた処理の高速化に関する研究は様々な分野で活発に行われている．例として，データベース内の文字列データに対する正規表現を含んだ問合せを FPGA を用いて高速化する Sidler らの研究 [2]，PageRank [3] や協調フィルタリング [4] などのグラフ分析処理を高速化するための FPGA アクセラレータ Graphicionado [5] などが挙げられる．機械学習の分野においても Owaida ら [6] が低レイテンシかつ高スループットな推論エンジンを FPGA を用いて開発している．また，商用システムの例では IBM 社の Netezza (現 PureData System for Analytics [7]) が FPGA を用いて大幅なスループットの向上を達成している．

3.2　RPQ の応用分野

RPQ の応用分野として Resource Description Framework (RDF) やグラフデータベースなどが挙げられる．例として Oracle 社の開発したグラフクエリ言語 PGQL [8] が RPQ に対応している他，openCypher プロジェクト [9] や World Wide Web Consortium (W3C) [10] もそれぞれが開発した Neo4j cypher [11] や SPARQL1.1 [12] に対して，RPQ への対応を進めている．

3.3　RPQ 評価の高速化に関する研究

RPQ 評価の高速化のためのアプローチとして，オートマトンベースとインデックスベースの 2 つのタイプの手法が挙げられる．

オートマトンベースの手法では与えられた RPQ を基にオートマトンを作成し，処理対象のグラフに対して適応していく．例として $R = (a \circ c^{-}) \cup (b \circ c^{1,2})$ が与えられた場合, 図 1 に示す様なオートマトンが作成され，グラフ中の全てのノードを始点ノードとしてこのオートマトンが適応される．Koschmieder らの研究 [13] では，グラフ中での出現頻度の低いラベル (レアラベル) を FPGA を用いてグラフを複数のサブグラフに分割することで探索スペースを削減し，より効率的なオートマトンによる探索を実現している．一方で本手法はレアラベルの存在を必要としており，また，そのラベルを適切に選択できるかどうかに性能が依存する．

インデックスベースの手法の代表例として Path Index [14] を挙げる．Path Index ではグラフ中のパスの登場を指定した長さ k まで事前にインデクシングする．図 1 を例として，$k = 2$ までのパスをインデクシングすると表 1 の様になる．この様に事前にインデクシングを行うことにより，長さが k 以下のパスはインデックスを参照することで即座に取得でき，長さが k よりも長いパスに対してもインデックス同士を結合することで効果的に結果を取得することができる．

Path Index は RPQ 評価の高速化の面では効果的だが，パスの長さ k が増えるに連れてインデクシングされるパスの数が増加し，ストレージサイズを圧迫してしまうといった問題点も存在する．

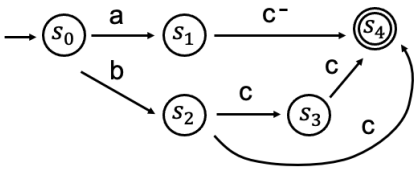


図 2　クエリから作成されるオートマトン ($R = (a \circ c^-) \cup (b \circ c^{1,2})$)

表 1　Path Index
(a) k = 1

パス	始点ノード	終点ノード
a	1	2
b	3	2
c	3	4
d	4	1
a	4	5
c	4	6
c	5	2
b	5	3

(b) k = 2

パス	始点ノード	終点ノード
$c \circ d$	3	1
$c \circ a$	3	5
$c \circ c$	3	6
$d \circ a$	4	2
$a \circ c$	4	2
$a \circ b$	4	3
$b \circ b$	5	2
$b \circ c$	5	4

4　提 案 手 法

本研究では RPQ 評価を FPGA 上でのパイプライン処理によって高速化するための手法を提案する．提案手法は対象グラフ上のパスの探索と探索されたパスに対する RPQ 評価の 2 つのパートに分けられる．

4.1　提案手法の処理の概要

図 3 に提案手法の処理の概要を示す．全体の処理は Host 側 (FPGA と通信を行う PC) で行われる処理と FPGA 側で行われる処理に分けられる．ユーザから RPQ が与えられると，まず Host はそのクエリを解釈し，4.2 節で後述する RPQ 評価のためのオートマトンの設定データと処理対象のグラフデータを隣接リスト形式で FPGA へ送信する．オートマトンの設定データは FPGA チップ上の Block RAM (BRAM) に，グラフデータは FPGA ボード上のグローバルメモリにそれぞれ配置される．ここで，BRAM は FPGA チップ上の回路が 1 サイクル以内にアクセス可能な非常に高スピードなメモリであるが，一般的にその容量は小さい (数百 Kb ～ 数百 Mb 程度)．一方でグローバルメモリの容量は大きいがアクセスは低速である．

FPGA は対象グラフ中のパスの探索と探索されたパスに対してのオートマトンを用いた RPQ 評価を行い，RPQ を満たすパスは FPGA から Host へ送信される．

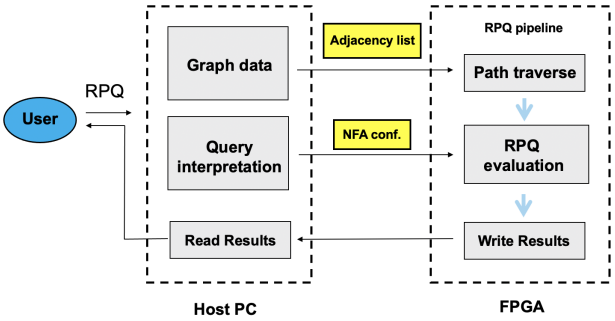


図 3　提案手法の処理の概要

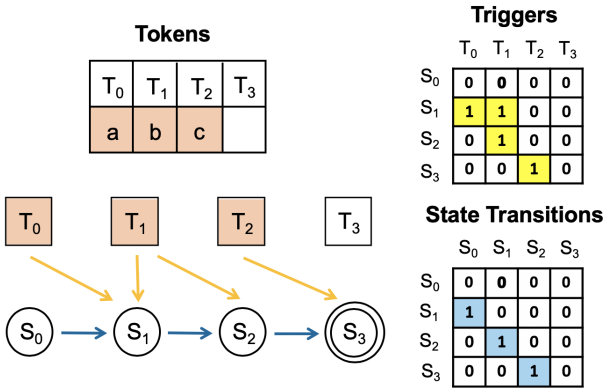


図 4　NFA の構造 ($R = (a \cup b) \circ b \circ c$)

4.2　RPQ 評価のためのオートマトン

本提案手法では FPGA 上に実装された NFA によって RPQ 評価を行う．関連手法で紹介した Sidler らの研究 [2] において，FPGA 上に任意の NFA を実装するための手法が提案されており，本研究ではそれらを参考に FPGA への NFA の実装を行った．

今回実装した NFA の構造を図 4 に示す．この NFA は Tokens, Triggers, State Transitions の 3 つの設定データを受け取ることで任意の NFA を実現する．Tokens は RPQ に登場するラベルの種類, Triggers はどのラベル (Token) がどの State への状態遷移を引き起こすか, State Transitions は State 間の遷移をそれぞれ指定する．ユーザから与えられた RPQ に応じてこれらの設定データを指定することで， FPGA 上の回路を書き換えることなく任意の RPQ の評価を行うことが可能である．一般に FPGA 上の回路の再合成は数時間程度の時間を要するためこれは大きなメリットとなる．

4.3　グラフ中のパスの探索

本研究では Frontier を用いた幅優先探索によって並列にグラフ中のパスを探索する．

図 5 に fontier を用いた幅優先探索の概要を示す．図はノード 1 を始点として深さ 1 まで探索を行った結果を表している．ここで,Current frontier は現在の深さで探索が行われるノード, Next frontier は 1 段下の深さで探索が行われるノードをそれ

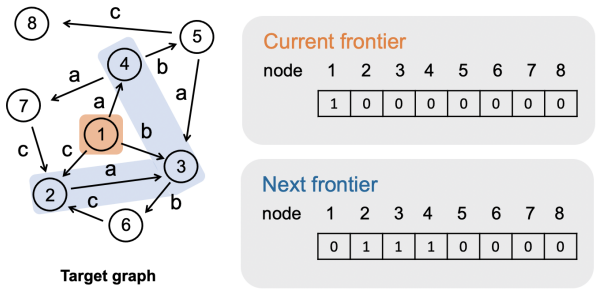


図 5　Frontier を用いた幅優先探索 (始点ノード：1)

ぞれ示している. 図の場合, 深さ 1 の探索 ではノード 1 の周辺ノード (2,3,4) が探索され，一段下の深さ 2 の探索では, Next frontier が Current frontier に置き換わり，ノード 2,3,4 の周辺ノードが探索される．この処理を繰り返すことで任意の深さまでの探索が可能である.

ここで，周辺ノードの探索と Frontier の更新はノード毎に並列に行うことができるので，FPGA 上での回路の並列性を活かした探索が可能である.

4.4　パスの探索と RPQ 評価

本提案手法では, 効率的な RPQ 評価を行うため 4.2 節の RPQ 評価オートマトンと 4.3 節の Frontier を用いた幅優先探索を組み合わせた．提案手法のパスの探索と評価について図 6 に示す.

本提案手法では Frontier を 2 次元の配列として保持する．ここで，配列の横軸は node を縦軸は NFA のどの State が active かを表す. 例として, ノード 1 を始点としてパスの探索と RPQ 評価を行う場合，まず Current frontier 上のノード 1 の初期状態 S_0 に 1 がセットされる．次にノード 1 の周辺ノード (2,3,4) を探索するが, この際にノード 1 から周辺ノードに対して貼られているエッジのラベルが NFA の状態遷移を引き起こすかをチェックする．ノード 3 の場合，ノード 1 からラベル b のエッジが貼られており，このラベルは NFA 上で S_0 から S_1 への遷移を引き起こすので, Next frontier 上のノード 3 の S_1 に 1 がセットされる．ノードの 2 の場合，ノード 1 から貼られているエッジのラベルは c であり，これは NFA 上で S_0 からの遷移を引き起こさないのでノード 2 に対してはどの状態にも 1 はセットされない.

この処理を任意の深さ繰り返し, NFA の受理状態 (図中 S_3) が active になるノードがあった場合は，ノード 1 からそのノードまでのパスが RPQ を満たすことがわかる．例として図 7 は深さ 3 までの探索を行った場合で，ノード 2 と 8 の受理状態に 1 がセットされているので $\{(1,2),(1,8)\}$ が RPQ を満たす. この処理をグラフ中の全てのノードを始点として行うことで, RPQ を満たす全てのパスを探索することが可能である.

さらに，FPGA 上ではこの探索と評価が全てパイプライン的に実行される．ここで，ノード毎に行う処理を以下の 4 つに大別する.

- **Stage1:** Current Frontier の該当ノードを表す列に active な要素があるかをチェックする
- **Stage2:** 該当ノードの隣接ノードを読み出す
- **Stage3:** オートマトンでの状態遷移をチェックする
- **Stage4:** Next Frontier への書き込みを行う

この場合，図 8 に示す様にそれぞれのノードの各処理毎に並列に処理が行われる．ここで，Stage1 でノードが active な要素を持たない場合は図 9 に示す様にその後の Stage は実行されないが，この場合でも他のノードの処理には影響を与えない．上記の例では説明のため探索・評価の処理を 4 つに大別したが, 実際では処理はより細かく細分化されパイプライン化される.

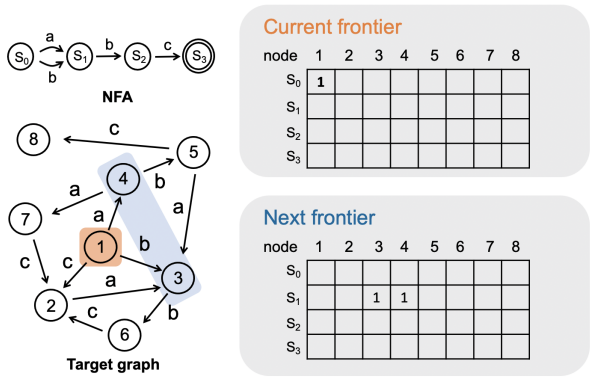


図 6　深さ 1 のパスの探索と評価 (始点ノード：1, $R = (a \cup b) \circ b \circ c$)

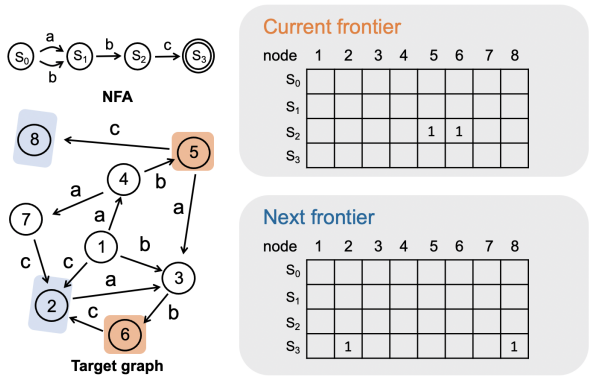


図 7　深さ 3 のパスの探索と評価 (始点ノード：1, $R = (a \cup b) \circ b \circ c$)

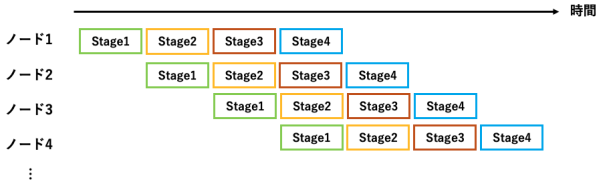


図 8　各処理のパイプライン化

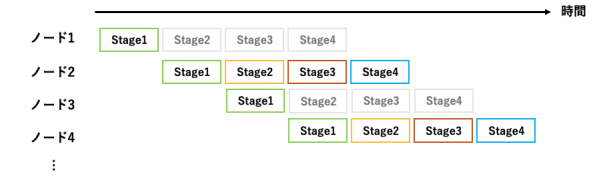


図 9　各処理のパイプライン化 (ノード 1 と 3 が active な要素を持たない場合)

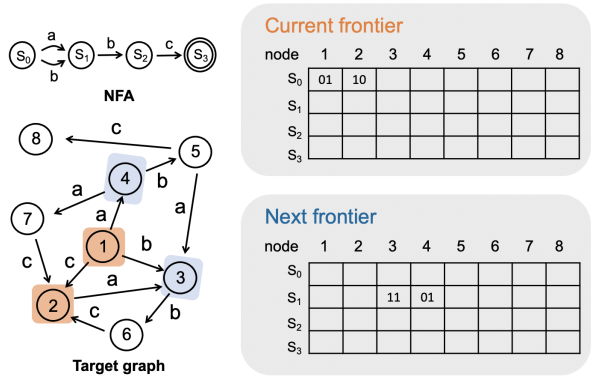


図 10　2 つのノードを始点とした深さ 1 までの並列探索 (始点ノード : 1, 2, $R = (a \cup b) \circ b \circ c$)

4.5　複数ノードを始点とした並列探索

4.4 節までに提案した手法ではグラフ中の各ノードの一つ一つを順番に始点としてパスの探索が行われる．本節では複数ノードを始点とした探索を一度に行い，より処理の並列度を高めるための手法を提案する．

図 10 は一度に 2 つのノードを始点として探索を行う場合を示す．4.4 節までとの違いは Frontier テーブルの各要素が 2bit の値となっている点である．ここで，この値のそれぞれの bit は各始点ノードと対応している．図から深さ 1 までの探索でノード 1 と 2 を始点とした場合にノード 3 の S_1 がアクティブとなり，ノード 1 を始点とした場合にノード 4 の S_1 がアクティブになることが分かる．さらにこの後深さ 2 の探索を行う場合，ノード 3 の S_1 に着目すると 4.4 節までの手法ではノード 1 を始点とした場合とノード 2 を始点とした場合で合計 2 回ノード 3 の隣接リストをグローバルメモリから読み出す必要があった．一方で図 8 に示す手法ではこれらの読み込みがまとめて 1 度で済むため，より効率の良いメモリアクセスが可能である．注意点として，Frontier の 1 要素を複数桁の bit で保持するため，当然テーブルのサイズは大きくなり，より多くの BRAM 容量を必要とする．よって適切な bit 幅は FPGA の BRAM 容量を考慮して決定する必要がある．

5　実　　験

5.1　比較手法と実験設定

本実験では 2 つの比較手法 Neo4j と Path Index を用いて提案手法の性能評価を行った．

Neo4j のクエリ言語 cypher では RPQ と等価なクエリを発行することが可能である．例として，RPQ:$R = (a \cup b) \circ (c^{1,2})$ を cypher を用いて表すと以下の様になる．

Cypher query (equal to $R = (a \cup b) \circ (c^{1,2})$) :

```
MATCH (n1:Node) - [:LabelA | :LabelB] ->
      () - [:LabelC*1..2] -> (n2:Node)
RETURN n1 AS src, n2 AS dst;
```

Path Index [14] では，実験に使用するグラフのサイズに応じてインデックス長 k を変化させる．また, [14] はインデックスを結合する際にクエリプランを決定するための数種類のアプローチを提案しているが，本研究では多くの場合で最も良い性能を示した minJoin アプローチを使用した．

提案手法では，(ノード数 × ステート数 × 1 要素の bit 幅) bits の Frontier テーブルが BRAM 上に配置される．本実験では上記のステート数を 16 に指定した．つまり最長で 16 ホップのパスを検索することができ，これは一般的な用途において十分な長さであると考える．

本実験で使用した実験環境を以下に示す．

表 2　実 験 環 境
(a) Host PC

CPU	Xeon E5-2660 v4 2.00GHz (x2)
OS	Linux version 3.10.0
Memory	DDR4 16GB x 8 (128 GB total)
Graph DB	Neo4j Enterprise 4.2.0
RDBMS	PostgreSQL 9.2.5

(b) FPGA

Board	BittWare 520N
FPGA	Intel Stratix 10 GX 2800
Memory	Four banks of DDR4 SDRAM x 72 bits 8GB per bank (32GB total)
I/O	PCIe Gen3 x16
BRAM	229 Mbits
MLAB	15 Mbits

5.2　データセット

本実験では Advogato [15] と Reddit Hyperlink Network [16] の 2 種類のデータセットを用いた．それぞれの詳細を以下に示す．

5.2.1　Advogato

Advogato は, フリーソフトウェアの開発者向けのオンラインコミュニティプラットフォームから作成された有向グラフで，プラットフォームを利用するユーザ同士の信頼関係をラベル付きエッジで表したものである．advogato の情報を表 3 に示す．

表 3　Advogato

ノード数	エッジ数	ラベルの種類
6,541	51,127	3

5.2.2　Reddit Hyperlink Network

Reddit Hyperlink Network は英語圏のウェブサイトであり多種多様なウェブサイトへのリンクを収集し公開する Reddit [16] から作成されたグラフデータセットである．ここで，グラフ中の各ノードは subreddit と呼ばれる Reddit 内で特定のドメインに焦点を当てたコミュニティを表しており，各エッジは subreddit 間に貼られたハイパーリンクを表す．様々な属性がこのエッジに付与されており，その中の一つにエッジの始点コミュニティから終点コミュニティに対してどの様な感情でリンクが作成されたかを表す属性が存在する．この属性は-1 から

1 の範囲での数値で表されており，本実験ではこの属性を 0.5 刻みで 4 分割し，強い負の感情，弱い負の感情，弱い正の感情，強い正の感情の 4 つのエッジラベルに変換した．今回我々が Reddit Hyperlink Network からしたグラフの詳細を表 4 に示す．

表 4　Reddit Hyperlink Network

ノード数	エッジ数	ラベルの種類
67,179	858,488	4

5.3　RPQ で指定するパスの長さと性能への影響

本実験では前述の 2 つのデータセットを用いて，RPQ で指定するパスの長さ $|R|$ を変化させた際の提案手法の性能調査を行った．本実験で対象とするクエリは和演算や繰り返し演算等を含まない結合演算のみの単純なクエリである．Path Index のパス長 k はストレージサイズ考慮した上で決定され，advogato では $k = 3$, Reddit Hyperlink Network では $k = 2$ を指定し，事前にインデックスを作成した．

実験の結果を図 11,12 に示す．実験の結果はパスの長さ毎に無作為に選択した 10 個のクエリを実行し，その実行時間の平均をとったものである．FPGA-$n$bits は提案手法の性能を表し，n は Frontier テーブルの 1 要素の bit 幅を示している．bit 幅の上限はグラフサイズと FPGA の BRAM 容量を考慮して決定され，Advogato : 8bit, Reddit Hyperlink Network : 4 bit を上限として定めた．

結果から分かる様に，比較手法である Neo4j と Path Index の実行時間はパスの長さ $|R|$ が大きくなるにつれて，指数関数的に増加しているが，一方で提案手法は $|R|$ に比例した性能を示している．$|R|$ が十分に大きい場合，提案手法は Path Index と比較して，Advogato で最大約 3 桁の高速化，Reddit Hyperlink Network で約 1 桁の高速化が得られた．本実験では実行に 6 時間以上かかる場合はタイムアウトとし，Neo4j は Advogato で $|R| = 4$, Reddit Hyperlink Network で $|R| = 3$ の場合でタイムアウトした．また，Adovogato での $|R| >= 9$, Reddit Hyperlink Network での $|R| >= 5$ では両比較手法がタイムアウトしたため実験を打ち切ったが，本提案手法は $|R|$ がそれ以上の場合でも変わらずパス長に比例した性能を得られることを確認した．

5.4　各パス長における実行結果の分散

5.3 節では RPQ で指定するパスの長さ毎にランダムに 10 個のクエリを選択しその平均実行時間を示した．本節では，それらの各パス長において 10 個のクエリの実行時間の分散について述べる．図 13 に Advogato でのパス長 4,8 の結果を，図 14 に Reddit Hyperlink Network でのパス長 2,4 の結果をそれぞれ示す．

これらの結果から分かるように，ベースライン手法である Path Index や Neo4j はクエリの実行時間の分散が大きく，最大で約 10 倍以上の差があることが確認できた．一方で，我々

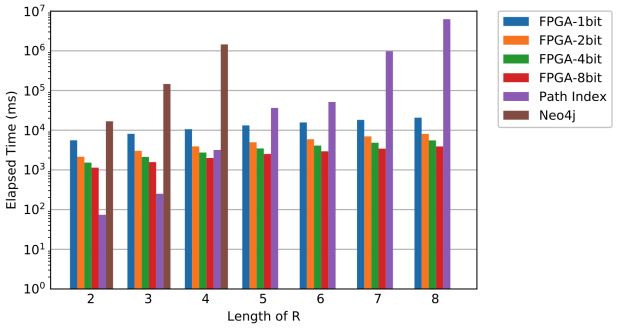


図 11　RPQ で指定するパスの長さと性能への影響 (advogato)

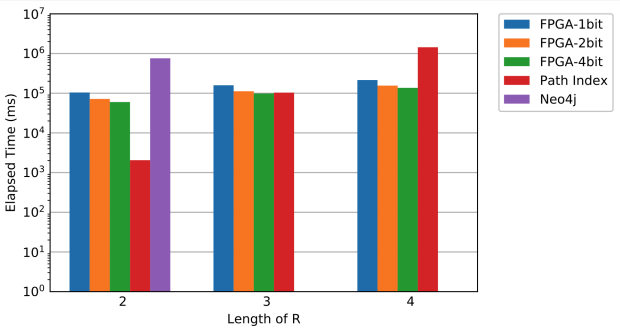


図 12　RPQ で指定するパスの長さと性能への影響 (Reddit)

の提案手法では実行時間はほぼ一定であることが分かる．

これらの特徴は RPQ の結果数に各手法が影響を受けるか否かに起因すると考えられる．2 つのベースライン手法では，RPQ で指定されたパスの途中にグラフ中での出現頻度の高いラベルが現れたり，グラフの探索途中に次数の大きなノードが現れた場合，結果の候補となるパスの数が増大するため計算コストが高くなり，より多くの実行時間がかかる．

一方で，我々の提案手法では 4.4 節で示した様に，各ノードにおける State の確認と周辺ノードの State の更新はパイプライン化されている．ここで，結果の候補となるパスの数が多いというのは Frontier テーブルにおける active なノード数が多いことと等しく，我々の手法では active なノード数が多い場合でも，それぞれのノードは並列に更新処理が行われるため，全体の計算コストは変わらず実行時間には影響を与えない．

5.5　クエリ内の演算子と性能への影響

本実験では RPQ に登場する様々なパターンのクエリを選択し実行時間を比較した．今回選択したクエリは以下の 5 つである (ここで l_i はグラフ中の任意のラベルを表す)．本実験では Advogato データセットを用いて実験を行った．

$q_1 = \ l_1 \circ l_2 \circ l_3$

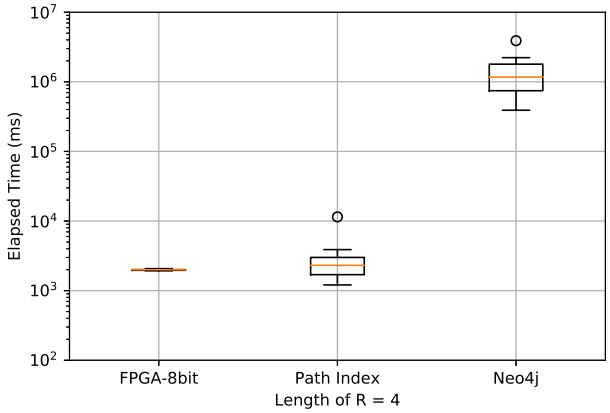
$q_2 = \ (l_1 \circ l_2 \circ l_3) \cup (l_4 \circ l_5 \circ l_6)$

$q_3 = \ (l_1 \cup l_2) \circ (l_3 \cup l_4) \circ (l_5 \cup l_6)$

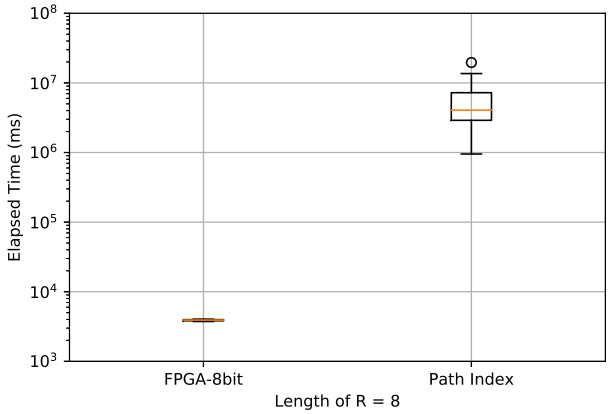
$q_4 = \ l_1 \circ l_2 \circ l_3 \circ l_4 \circ l_5$

$q_5 = \ l_1^{2,5}$

実行結果を図 15 に示す．実行結果はクエリ中のラベル l_i を

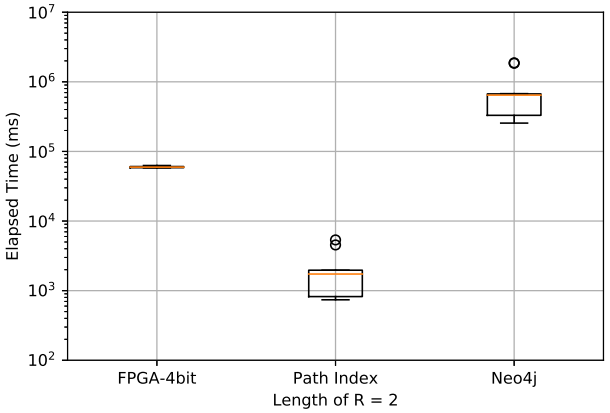


(a) $|R| = 4$ (Advogato)

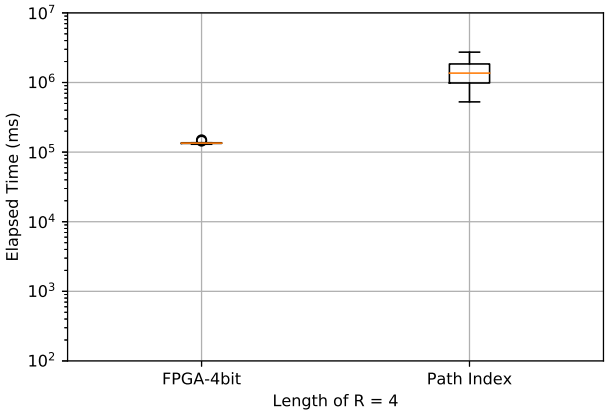


(b) $|R| = 8$ (Advogato)

図 13　ランダムに選択された 10 個のクエリにおける実行時間の分散 (Advogato)



(a) $|R| = 2$ (Reddit)



(b) $|R| = 4$ (Reddit)

図 14　ランダムに選択された 10 個のクエリにおける実行時間の分散 (Reddit)

無作為に選択しクエリを 10 回実行した平均をとったものである．ここで，$q_1 \sim q_3$ は $|R| = 3$ のパスを，q_4, q_5 は $|R| = 5$ のパスを探索するクエリであることに着目すると，比較手法は同じパス長のクエリでも和演算や繰り返し演算を含むことで実行時間が大きく変化するのに対し，提案手法はそれらの影響を受けずパス長の同じクエリでは実行時間がほぼ一定であることがわかる．

5.6　FPGA のリソース使用率

次に FPGA のリソース使用率について述べる．表 5,6 はそれぞれ Frontier テーブルのサイズを Advogato に最適化した場合と Reddit Hyperlink Network に最適化した場合の FPGA のリソース使用率を表している．

表から Frontier テーブルの 1 要素の bit 幅を増やすと BRAM の使用率が大幅に増加していることが見て取れる．この bit 幅は性能に直結しているため, 処理対象のグラフのノード数と使用する FPGA の BRAM 容量に応じて適切な bit 幅を選択することが性能向上のためには必要であると言える．

5.7　考　　察

これまでの実験で述べた様に，本手法は処理対象のグラフのノード数と使用する FPGA の BRAM 容量によってその性能が

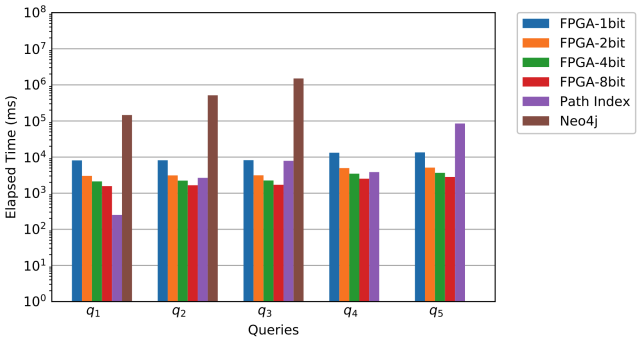


図 15　RPQ のパターンと性能への影響

表 5　FPGA のリソース使用率 (Advogato)

	Frontier テーブルの bit 幅			
	1bit	2bit	4bit	8bit
ロジック使用率	20%	20%	21%	22%
BRAM 使用率	9%	12%	17%	27%

左右される．現在での BRAM 容量は高性能な FPGA チップでも数百 Mb 程度であり，今後の FPGA 技術の発展を考慮しても，本提案手法を現実世界に存在する様な数百万 ～ 数億ノードの非常に大規模なグラフに対して適応することは難しい．

そこで現在我々はこの問題を解決するため，2 種類のアプ

表 6　FPGA のリソース使用率 (Reddit Hyperlink Network)

	Frontier テーブルの bit 幅		
	1bit	2bit	4bit
ロジック使用率	21%	21%	23%
BRAM 使用率	16%	30%	50%

ローチを検討している．1 つ目が Frontier テーブルを圧縮を行う手法である．この手法ではグラフの平均次数が小さい場合や RPQ で指定したラベルがグラフ中に稀にしか存在しない場合を考慮し，そのような場合では Frontier テーブルは非常に疎な行列になると考えられるので，テーブルを圧縮し FPGA の BRAM 容量を節約する．2 つ目が複数 FPGA を利用した並列処理を行う手法である．本提案手法では Frontier テーブルを用いた探索と NFA を用いた RPQ 評価を各ノード毎に行うことができるため，それらの処理を複数の FPGA に分散しより大きなグラフに対応できるのではないかと考えている．

6 ま と め

本研究では FPGA を用いた RPQ 評価の高速化手法を提案した．RPQ 評価オートマトンと Frontier を用いたグラフ中のパスの並列探索を組み合わせ，効率的な RPQ 評価を実現した．結果として，比較手法と比べ最大で約 3 桁の性能向上が得られ，あるグラフに対して同一のパス長の RPQ を実行する場合では実行時間がほぼ一定になることや和演算や繰り返し演算などを含む様な複雑な RPQ に対しても実行時間が影響されないといった特徴を確認した．今後の展望として，より大規模なグラフに対応するため FPGA の BRAM 容量を節約するための手法や，マルチ FPGA を用いた並列処理手法の開発を進める予定である．

謝　　辞

この成果は、国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP20006）および筑波大学計算科学研究センターの学際共同利用プログラムを利用して得られたものです。

文　　献

[1] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A Graphical Query Language Supporting Recursion. *SIGMOD Rec.*, Vol. 16, No. 3, pp. 323–330, December 1987.

[2] David Sidler, Zsolt István, Muhsen Owaida, and Gustavo Alonso. Accelerating pattern matching queries in hybrid cpu-fpga architectures. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pp. 403–415, New York, NY, USA, 2017. ACM.

[3] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web, 1999.

[4] D. Agarwal and B.-C. Chen. Machine learning for large scale recommender systems, 2011.

[5] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13, Oct 2016.

[6] Muhsen Owaida, Gustavo Alonso, Laura Fogliarini, Anthony Hock-Koon, and Pierre-Etienne Melet. Lowering the latency of data processing pipelines through fpga based hardware acceleration. *Proc. VLDB Endow.*, Vol. 13, No. 1, p. 7185, September 2019.

[7] Netezza performance server, December 2020.

[8] Oskar van Rest, Sungpack Hong, Jinha Kim, Xuming Meng, and Hassan Chafi. Pgql: A property graph query language. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, GRADES '16, pp. 7:1–7:6, New York, NY, USA, 2016. ACM.

[9] opencypher, December 4, 2020.

[10] World wide web consortium, December 4, 2020.

[11] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pp. 1433–1445, New York, NY, USA, 2018. ACM.

[12] Sparql 1.1 query language, December 4, 2020.

[13] André "Koschmieder and Ulf" Leser. Regular path queries on large graphs. In Anastasia Ailamaki and Shawn Bowers, editors, *Scientific and Statistical Database Management*, pp. 177–194, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[14] George H. L. Fletcher, Jeroen Peters, and Alexandra Poulovassilis. Efficient regular path query evaluation using path indexes. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France*, pp. 636–639, March 2016.

[15] Paolo Massa, Martino Salvetti, and Danilo Tomasoni. Bowling alone and trust decline in social network sites. In *In Proc. Int. Conf. Dependable, Autonomic and Secure Computing*, pp. 658–663, 2009.

[16] reddit: the front page of the internet, December 2020.