CD-001

FPGA によるSPARQL問合せの高速化

仁木美来‡　山口佳樹‡　天笠俊之†

‡筑波大学　†筑波大学計算科学研究センター

1　はじめに

文書共有（Linked Data）[1] と文書公開（Open Data）[2] の双方の概念を併せ持つ技術的方法論として，LOD (Linked Open Data) への期待が高まっている．英国政府（data.gov.uk）や米国連邦政府（www.data.gov）を初め，我が国でも内閣官房主導によるデータカタログサイト（www.data.go.jp）など，LOD を利用した Web サイトが開設されている．また，総務省統計局では，LOD の公開度を 5 段階で表す 5 Star Open Data [3] の最高段階 (5 つ星) のデータを提供する試み（data.e-stat.go.jp）が進められている．

5 Star Open Data において，高い公開度（4 つ星ならびに 5 つ星）にランク付けされる LOD には Linked Data の観点をより意識した記述が期待されている．これは，RDF (Resource Description Framework) [4] 等を用いて記述することで，データ形式の標準化とデータの接続関係による情報空間を用いた探索等が容易になり，LOD の利用可能性をより高めることが期待されるからである．そして，RDF データへの問合せには，W3C (World Wide Web Consortium) により標準化された問合せ言語である SPARQL (SPARQL Protocol and RDF Query Language) [5] などが整備されつつある．

次に，RDF 形式のデータを扱う代表的なデータストア（データベースやファイルシステム）に，Jena [6]，RDF-3X [7]，gStore [8]，Virtuoso [9, 10] がある．これらは全て LOD 利用を加速かつ活発化させる魅力的な取り組みである．UniProt [11] (440 億トリプル) や PubChemRDF V1.6.1 beta [12] (1370 億トリプル) など有用かつ膨大な RDF データもこの流れを更に加速させると考えられる．一方で，これらデータベースの増大に応じて，問合せに要する時間が無視できないものとなっている．

この問題解決の試みとしてデータストアの演算部のハードウェア実装が挙げられる．文献 [17, 18] では，オンチップメモリを用いて SPARQL 問合せの JOIN を高速化を実現しており，一定の成果が得られている．しかし，増大するデータサイズへの対応やそれに伴う抽出結果行数の増加やその取扱い，また演算加速の拡張性については述べられていない．そこで本研究では，JOIN を伴わない問合せ検索に対象を絞り，ハードウェア加速の可能性および拡張性について改めて検証することとした．

FPGA-based SPARQL query acceleration
Mirai NIKI‡, Yoshiki YAMAGUCHI‡,
Toshiyuki AMAGASA†
‡University of Tsukuba
†Center for Computational Sciences, University of Tsukuba

まず本研究では，ハードウェア実装の事前準備として，RDF データサイズの圧縮を図った．一般に，トリプルデータは Internationalized Resource Identifier（IRI）で記述されており，同一のリソースが繰り返し出現するため冗長性が高い．そこで，RDF データの圧縮フォーマットである HDT (Header, Dictionary, Triples) [13, 14] などが採用している ID 化を導入した．具体的には，RDF データ内の全リソースに対して固有の ID を割り振り，内容を全て ID で記述することで RDF データサイズの圧縮を図る．そして，ID 化による高速化とハードウェア実装による高速化を切り分けるため，予備実験として CPU 上で ID 適用前と後のそれぞれの RDF データについて RDF データのトリプルストアである Virtuoso を用いて検証した．続いて，RDF トリプルストアの問合せ部の演算を FPGA (Field Programmable Gate Array) 上に移植し，ハードウェア実装による高速化可能性および並列演算による拡張性などについて検証を行った．ハードウェア実装による高速化では，予備実験と同様に Virtuoso を用い，その結果と比較した．本稿の実験では，単純なハードウェア化で 500 倍以上，演算部の並列化により 4000 倍以上の速度向上を達成することが確認できた．また，現在の実験条件下では，並列度に合わせてほぼ線形に演算性能の向上が期待できることも確認された．

本稿の構成は，以下の通りである．2 章では，RDF データとその問合せ言語である SPARQL について述べ，3 章では高速化に向けた RDF トリプルデータの ID 化とその予備実験について述べる．4 章では，SPARQL 問合せのハードウェア実装について提案し，その実験結果について 5 章で議論する．最後に，6 章で本稿の結論と今後の課題を述べ，本稿をとじる．

2　LOD (Linked Open Data)

2.1　RDF データ

RDF データは，Web 上のリソースをコンピュータが容易に認識できるようにする枠組みに沿って作られたデータであり，主語，述語，目的語を 1 単位とするトリプルの集合で構成される．

ここでトリプルの具体例を示すと，

(主語) http://www4.wiwiss.fu-berlin.de/ProductType1,

(述語) http://www.w3.org/1999/#type,

(目的語) http://www4.fu-berlin.de/ProductType.

と表せる．目的語の最後のピリオドがトリプルの終端を示す．各要素は IRI を用いて記述されるが，これは各要素を一意に識別可能とするためである．

```
<http://www4.wiwiss.fu-berlin.de/ProductType1>
<http://www.w3.org/1999/#type>
<http://www4.fu-berlin.de/ProductType> .

<http://www4.wiwiss.fu-berlin.de/ProductType1>
<http://www.w3.org/2000/01/rdf-schema#label>
"Thing" .

<http://www4.wiwiss.fu-berlin.de/ProductType1>
<http://www.w3.org/2000/01/rdf-schema#comment>
"The Product Type of all Products" .

<http://www4.wiwiss.fu-berlin.de/ProductType2>
<http://www.w3.org/1999/#type>
<http://www4.fu-berlin.de/ProductType> .
```

Listing 3: RDF トリプルの例（ID 化前）

```
<1>
<2>
<3> .

<1>
<4>
"5" .

<1>
<6>
"7" .

<8>
<2>
<3> .
```

Listing 4: RDF トリプルの例（ID 化後）

2.2 RDF 問合せ言語（SPARQL）

SPARQL は RDF データに対する問合せ言語である．SPARQL 問合せの記述方式は SQL と類似しており，SELECT 句で抽出したい変数を指定し，WHERE 節で要素の一部を変数とした RDF トリプル（問合せトリプル）を列挙する．そして，問合せトリプルと RDF データ内のトリプルとでパターンマッチングを行い，変数部分以外が一致するデータを抽出する．Listing 1 に `http://www.w3.org/1999/#type` を述語とし，主語と目的語を抽出する SPARQL 問合せトリプルの例を示す．

```
select ?subject ?object
where {
    ?subject
    <http://www.w3.org/1999/#type>
    ?object .
}
```

Listing 1: SPARQL 問合せ例

Listing 1 の SPARQL 問合せトリプルを Listing 3 に問い合わせると Listing 2 の結果を抽出できる．

```
?subject
http://www4.wiwiss.fu-berlin.de/ProductType1
http://www4.wiwiss.fu-berlin.de/ProductType2

?object
http://www4.fu-berlin.de/ProductType
http://www4.fu-berlin.de/ProductType
```

Listing 2: SPARQL 問合せ例

3 RDF データに対する ID 化の適用

3.1 ID 化の導入理由

既存の RDF トリプルストアに対して SPARQL 問合せを行った場合，同じ RDF データを用いても抽出結果行数に比例して問合せ時間が長くなるという傾向がある．つまり，問合せにかかる時間は RDF データサイズだけでなく抽出結果のデータサイズにも依存している可能性が高い．そこで，RDF データ内の全要素を ID 化し，RDF データサイズはもちろん抽出結果のデータサイズも縮小することが高速化につながると考えられる．本稿では，上記の理由より，RDF データに対して ID 化を適用した．

RDF データの ID 化は，まず，RDF データ内の各トリプルに含まれる主語，述語，目的語の各要素に対して ID を割り当て，トリプル間で共通する要素に対しては改めて共通かつ使われてない最小の ID を割り当てる．Listing 3 に ID 化前の RDF データの例を，Listing 4 に ID 化後の RDF データの例をそれぞれ示す．

3.2 ID 化のソフトウェア実装

本稿における RDF データの ID 化の実装を Algorithm 1 に示す．RDF 問合せの頻度と比較したとき，RDF データの更新頻度は高くないため，Algorithm 1 の最適化（高速化）は本稿では行わなかった．

Algorithm 1 RDF データの IRI に ID を割り当てる

Input: Data(IRI)，Query(IRI)
Output: Data(ID)，Query(ID)

```
for iri ∈ Data(IRI) do
    // IRI-ID Map の生成
    if Map does not include (iri，count) then
        count ←count + 1
        Map(iri) ←iri，count
    end if
    // Data に ID を付与
    Data(ID) ← Map(iri)
end for
for iri ∈ Query(IRI) do
    //Query に ID を付与
    Query(ID) ← Map(iri)
end for
```

本稿の全ての実験では，この Algorithm 1 に従って生成した ID 化後の RDF データを用いている．予備実験およびソフトウェア実験では，ID 化後の RDF データは RDF トリプルストアの Virtuoso に格納され，Python で記述したコードから問合せを行っている．

3.3 予備実験（ID 化の検証）

BSBM (The Berlin SPARQL Benchmark) [16] で生成した 50k トリプルからなる RDF データを利用して，ID 化したデータと ID 化していないデータの問合せ時間を比較した．使用した RDF トリプルストアは Virtuoso であり，図 1 にこの予備実験結果を示す．

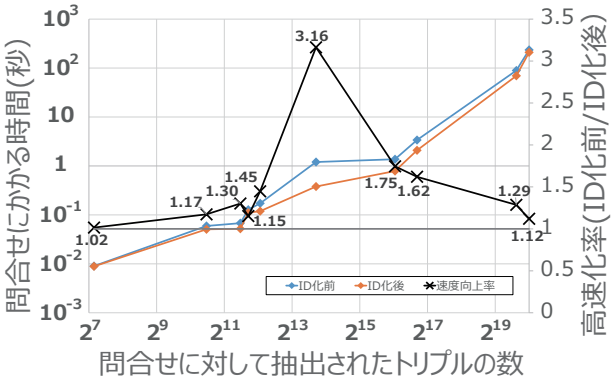

図 1: ID 化前と ID 化後の処理時間

この予備実験より，(1) 抽出されたトリプル数に比例して問合せに要する時間が増加していること，(2) ID 化により 10%程度以上の速度向上が期待できること，の 2 つが確認された．以上より，以降の章では，ID 化した RDF データを対象としたハードウェア加速について議論を行う．

4 SPARQL 問合せのハードウェア実装

本稿では，予備実験結果を参考に，ID 化した RDF データの問合せ処理をハードウェア実装した．ID 化は，記憶領域の削減に加え，扱うデータが固定長となりハードウェア化の際の回路使用量の見積もりや並列化時に複数の演算部の間の制御が容易になるなどの利点もある．本章では，4.1 節でパターンマッチの基本演算回路，4.2 節でその並列化について述べる．

4.1 ハードウェア実装（基本演算回路）

ハードウェア実装の第一歩として Join を伴わない単純な SPARQL 問合せのパターンマッチングの回路を設計した．図 2 にそのブロック図を示す．

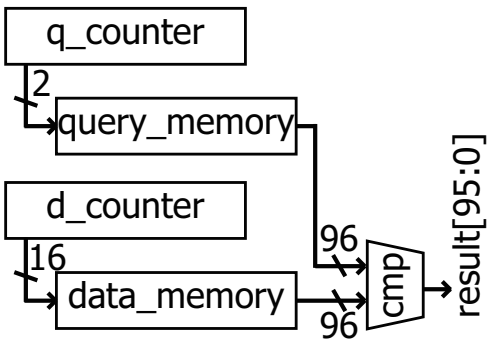

図 2: SPARQL 問合せの基本演算回路

この基本演算回路において，問合せトリプルおよび RDF データはそれぞれ query_memory と data_memory に格納されている．問合せトリプルは，q_counter を制御することで，query_memory から取り出すことができる．図 2 では問合せトリプルの数を 4 と仮定したためアドレスバスを 2 bit としているが，このバス幅は回路資源の許す限り広くすることが可能である．これと同時に，d_counter も制御され，data_memory から RDF データが読みだされる．図 2 では ID を 32 bit と仮定している．このため，トリプルデータは 96 (=32×3) bit となっている．読みだされた問合せトリプル (96 bit) と RDF データ (96 bit) は次段の比較モジュール（cmp）で演算処理される．この演算処理において，結果がマッチした場合，その内容を result として出力する．ここで，図 2 におけるタイミングチャートを図 3 に示す．

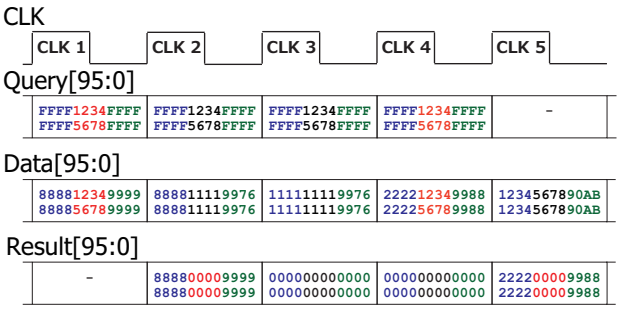

図 3: 基本演算回路 (図 2) のタイミングチャート例

まず，図 3 において，RDF トリプルは主語 (青:32bit)，述語 (黒:32bit)，目的語 (緑:32bit) から構成されているのがわかる．クロックの立ち上がり時に Query と Data の内容を比較し，次のクロックの立ち上がり時にその結果 Result の値として出力する．例えば，図 3 の CLK 1 と CLK 4 において，Query の述語と Data の述語が一致している（図 3 の赤字部分）．また，本稿において 0x FFFF-FFFF-FFFF-FFFF は変数を意味するので，この場合において値が一致したと考えることができる．そこで，それぞれの次のクロックである CLK2 と CLK5 において，問合せクエリ結果を出力している．

4.2 ハードウェア実装（並列回路）

演算処理性能を高めるには，単位時間あたりのデータ読み出し量を増やし演算部を並列化するか，単位時間あたりのデータ読み出しは増やさない代わりに同じデータを繰り返し複数の処理を割り当てるか，のいずれかが考えられる．本稿の応用アプリケーションはデータベースであり何れの方向でも高速化を実現するハードウェア構成が考えられるが，まずは，演算性能の基本的な拡張可能性を調べるため前者について調査を行った．ここで，図 4 に基本演算回路を 4 つ配置した構成（4 並列回路）のブロック図を示す．

図 4 の実装では，並列度が n のとき，データベースを n 個のサブデータベースに分割する．d_counter を用いて n 個のサブデータベース（data_memory）から同時にデータを読み出し，各サブデータベースに繋がっ

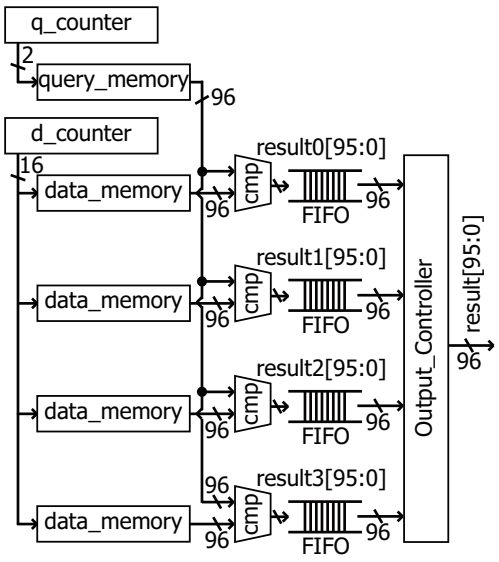

図 4: 基本演算回路の並列実装（4 並列回路）

た比較モジュール（cmp）で演算処理される．並列化時において，複数同時に，または連続して結果が返ってくることが考えられる．このため，比較モジュールの結果を直接出力するのではなく，一旦 FIFO へ保存し，出力制御モジュール（Output_Controller）により出力の調整を図る．本稿の回路は比較的シンプルであり，ハードウェア資源の許す限り，並列度を上げることが可能である．

5 実験

5.1 利用した RDF トリプル

予備実験と同様に BSBM [16] により生成した 5k トリプル，50k トリプル，500k トリプルの 3 種類の異なるサイズの RDF データを作成し，それを Algorithm 1 により要素を全て ID 化したものを実験に使用した．また，抽出結果の行数と抽出時間の関係を確認するため，抽出結果行数の異なる 4 つの SPARQL 問合せを用意して実験を行った．

5.2 計測環境

ソフトウェア計測環境は，Intel Core i7-4930K CPU @ 3.4GHz，Linux Mint 19，Virtuoso opensource 6.1.0 [15] であり，プログラミング言語とし Python3，ライブラリとして requests を利用した．

ハードウェア計測環境は，ソフトウェア計測環境に PCIe 経由で接続された FPGA ボード（Xilinx 社製 Virtex UltraScale+ VCU1525）を使用し，その開発環境として Vivado 2018.3，ハードウェア記述言語として Verilog-HDL を利用した．

5.3 ソフトウェア実験結果

ハードウェア実装に対する比較実験としてソフトウェア（Virtuoso）を利用し，データサイズの異なる 3 つの ID 化した RDF データに対する SPARQL 問合せ処理時間の変化を調査する．予備実験（3.3 節）は ID 化の効果を確認したものであり，本実験とは異なっている．

本実験結果を図 5，図 6，図 7 に示す．実験では，各パラメータに対して 30 回施行し，その平均をグラフとして示している．多少のばらつきが確認されたが，問合せ結果のデータサイズに対して線形に処理時間が増加していることが改めて確認された．

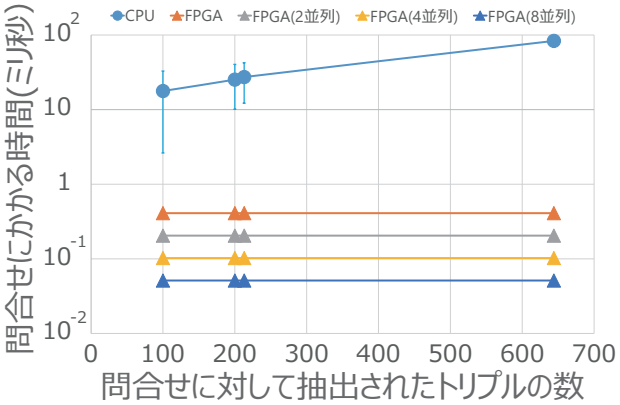

図 5: 実験結果（5k トリプル）

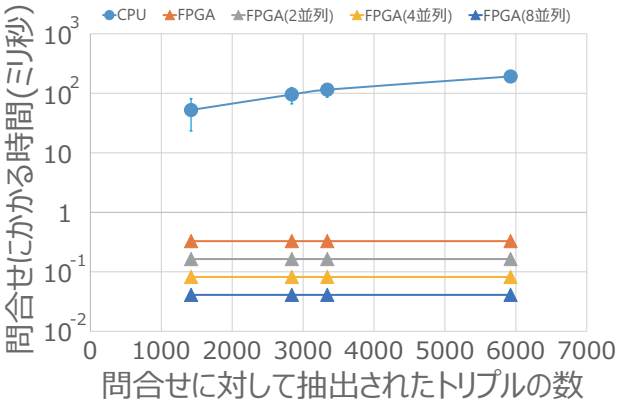

図 6: 実験結果（50k トリプル）

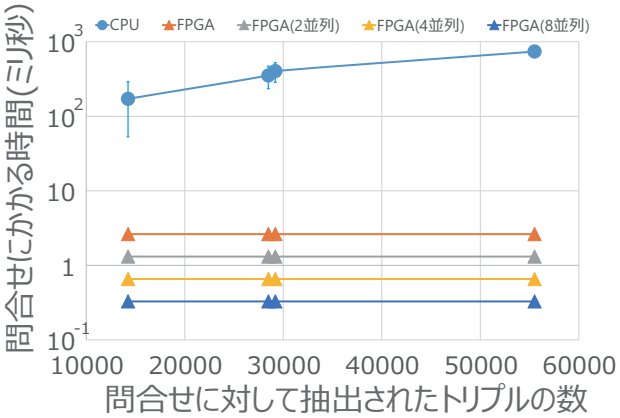

図 7: 実験結果（500k トリプル）

5.4 ハードウェア実験結果

ハードウェア実験において，1 並列（4.1 節に示す基本演算回路），2 並列，4 並列，8 並列（4.2 節に示す並列回路）の 4 つの回路を用意し，5k，50k，500k のデータに対して処理速度を比較した．このときの測定結果を図 5，図 6，図 7 にそれぞれ示す．

並列度が同じ場合，問合せに依存せずに一定の処理時間で処理が行われることが確認された．これは RDF データをメモリから読み出す時間により決定されるためであり，RDF データのサイズに着目するとそのサイズに対して線形に処理時間が増加していることが確認される．また，ソフトウェアと比較したとき，十分な高速化が得られていることがわかる．この詳細を 表 1，表 2，表 3 に示す．

表 1: SPARQL 問合せ速度比較（5k トリプル）

行数（行）	CPU(ms)	FPGA(ms)	CPU/FPGA
100	1.78	0.041	43.35
200	2.53	0.041	61.67
213	2.73	0.041	66.7
644	8.34	0.041	203.63

表 2: SPARQL 問合せ速度比較（50k トリプル）

行数（行）	CPU(ms)	FPGA(ms)	CPU/FPGA
1,420	23.94	0.33	73.05
2,840	28.98	0.33	88.44
3,340	42.06	0.33	128.36
5,925	69.27	0.33	211.39

表 3: SPARQL 問合せ速度比較（500k トリプル）

行数（行）	CPU(ms)	FPGA(ms)	CPU/FPGA
14,240	171.43	2.62	65.43
28,480	352.86	2.62	134.68
29,187	403.66	2.62	154.07
55,505	738.61	2.62	281.91

また，図 8，図 9，図 10 に並列化時の速度向上率を示す．基本演算回路を 1 として正規化した時，並列度に応じて理想に近い高速化が得られていることがわかる．

最後に，8 並列化におけるリソース使用量を表 4 に示す．演算回路はまだ十分に余っておりより複雑なロジックを導入することが可能なことがわかる．データサイズという点では，オンチップメモリについては既に 80%近く利用しているためこれ以上の実装は難しいが，HBM が搭載された新しい FPGA が登場している．これを利用し，扱えるトリプル数を増加させることは今後の課題である．

表 4: リソース使用量

リソース	回路使用量	使用可能量	使用率（%）
LUT	27,098	1,182,240	2.29
FF	9,902	2,364,480	0.42
BRAM	1,726	2,160	79.91
DSP48	0	6,840	0

6 まとめ

RDF データストアにおいて，データの ID 化による SPARQL 問合せ結果の抽出と FPGA の利用可能性について論じた．ソフトウェアでは ID 化による高速化がみられたが，問合せ結果の行数に依存して処理時間がかかった．一方で FPGA 上のチップオンメモリを利用した場合ではソフトウェアで ID 化した場合に比べて高速かつ問合せに依存せず一定の時間で，抽出することができた．今後の課題として，より複雑な問合せ処理ができるように回路を拡張する．また，より大きなサイズの RDF データに対しても処理できるように，オンチップメモリだけでなく外部メモリの利用を検討する．

謝辞

本研究の一部は，科学研究費補助金（JP17H01707）の助成を受けたものである．また，Xilinx 社より「Xilinx University Program」を通じて開発ソフトウェアの支援を受けており，ここに謝意を表する．

参考文献

[1] Tim Berners-Lee, “Linked Data”, https://www.w3.org/DesignIssues/LinkedData.html, 18 June 2009 (最終閲覧日：2019 年 6 月 20 日).

[2] Open Knowledge Foundation, “The Open Definition”, http://opendefinition.org/, (最終閲覧日：2019 年 6 月 20 日).

[3] Michael Hausenblas, “5 star open data”, https://5stardata.info/, 22 January 2012 (最終閲覧日：2019 年 6 月 20 日).

[4] Patrick J. Hayes and Peter F. Patel-Schneider, “RDF 1.1 Semantics”, W3C technical reports, W3C Recommendation 25 February 2014, https://www.w3.org/TR/rdf11-mt/.

[5] S. Harris, A. Seaborne, and E. Prud’hommeaux, “SPARQL 1.1 Query Language”, W3C technical reports, W3C Recommendation 21 March 2013, https://www.w3.org/TR/sparql11-query/.

[6] Brian McBride, “Jena: A semantic Web Toolkit”, IEEE Internet Computing, Vol.6, No.6, pp.55–59, Nov.-Dec. 2002.

[7] Thomas Neumann and Gerhard Weikum, “The RDF-3X engine for scalable management of RDF data”, The VLDB Journal, Vol.19, No.1, pp.91–113, Feb. 2010.

[8] Lei Zou et al., “gStore: a graph-based SPARQL query engine”, The VLDB Journal, Vol.23, No.4, pp.565–590, Aug. 2014.

[9] Orri Erling, “Virtuoso, a Hybrid RDBMS/Graph Column Store”, IEEE Data Engineering Bulletin, Vol.35, No.1, pp.3–8, 2012.

[10] Orri Erling and Ivan Mikhailov, “Virtuoso: RDF Support in a Native RDBMS”, Procs. of Semantic Web Information Management: A Model-Based Perspective, pp.501–519, 2010.

[11] UniProt Consortium, “UniProt: a hub for protein information”, Nucleic acids research, 43(Database issue), D204–D212, January 2015.

[12] PubChemRDF, “PubChemRDF”, v1.6.2b, 19 December 2018, `https://pubchemdocs.ncbi.nlm.nih.gov/rdf`, (最終閲覧日：2019 年 6 月 20 日).

[13] Mario Arias Gallego et al., “HDT RDF”, `http://www.rdfhdt.org/`, (最終閲覧日：2019 年 6 月 20 日).

[14] Miguel A. Martínez-Prieto et al., “Exchange and Consumption of Huge RDF Data”, Procs. of the Semantic Web: Research and Applications, pp.437–452, May 2012.

[15] Virtuoso Open-Source Edition, `https://github.com/openlink/virtuoso-opensource` (最終閲覧日：2019 年 6 月 20 日).

[16] Christian Bizer and Andreas Schultz, “The berlin sparql benchmark”, International Journal on Semantic Web and Information Systems, Vol.5, Iss.2, pp.1–24, June 2009.

[17] Stefan Werner et al., “Accelerated join evaluation in Semantic Web databases by using FPGAs”, Concurrency and Computation: Practice and Experience, Vol.28, Iss.7, pp.2031–2051, May 2016.

[18] Stefan Werner et al., “Hardware-accelerated join processing in large Semantic Web databases with FPGAs”, Procs. of 2013 Int’l Conference on High Performance Computing & Simulation, pp.131–138, July 2013.

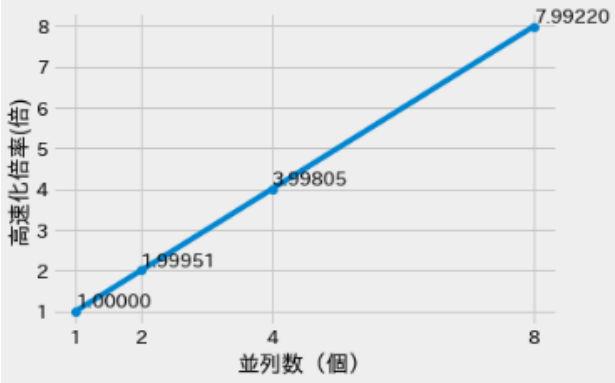

図 8: 5k データで並列回路の高速化倍率

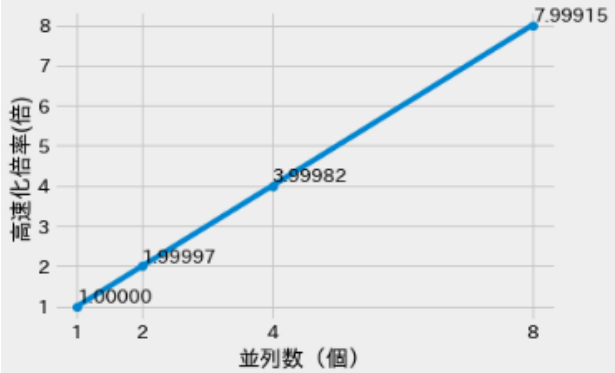

図 9: 50k データで並列回路の高速化倍率

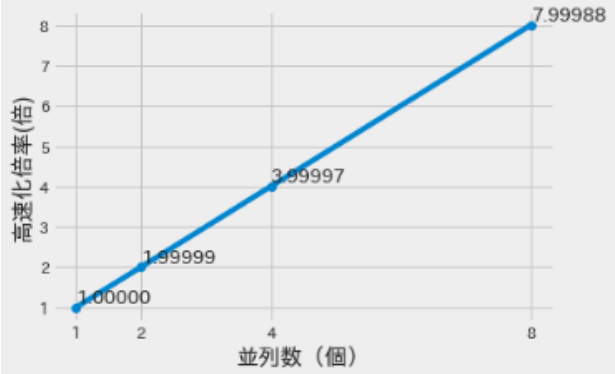

図 10: 500k データで並列回路の高速化倍率