ホワイトペーパー

データセンター

データ活用における FPGAの有用性の検討

インテル® FPGAを活用した文字列高速処理の実装評価

著者

株式会社エヌ・ティ・ティ・データ
技術革新統括本部
システム技術本部
デジタル技術部
近藤 賢司

目次

はじめに

近年、IoT、AI、ビッグデータといったキーワードとともに、さまざまなデータを活用したビジネスに多くの企業が注目しています。データ活用にあたっては、データの収集・蓄積・加工・分析を行う、AIなどの分析方法、応用アプリケーションなどで構成されるシステムを実現する必要があります。特に、データ量の増大に伴って、大量のデータをより速く処理するための高速処理基盤に対する需要が高まっています。このような基盤としては、Apache* Hadoop*[1] に代表される並列分散処理（1つの処理を複数のサーバーで並列に処理するテクノロジー）を導入するケースが一般的ですが、そこにはサーバー1台当たりの性能に起因する、次のような課題があります。

- **単一サーバーの性能限界による処理時間の長期化：**処理の内容により分割できるタスクの最小粒度に限界があるため、単一サーバーの性能限界に伴うタスクの処理遅延が生じてしまうと、性能要件を満たすことができません。
- **サーバー数増加に伴うコスト増大：**性能要件を満たすために大量のサーバーを導入せざるを得ない場合、コストがかさんでしまいます。

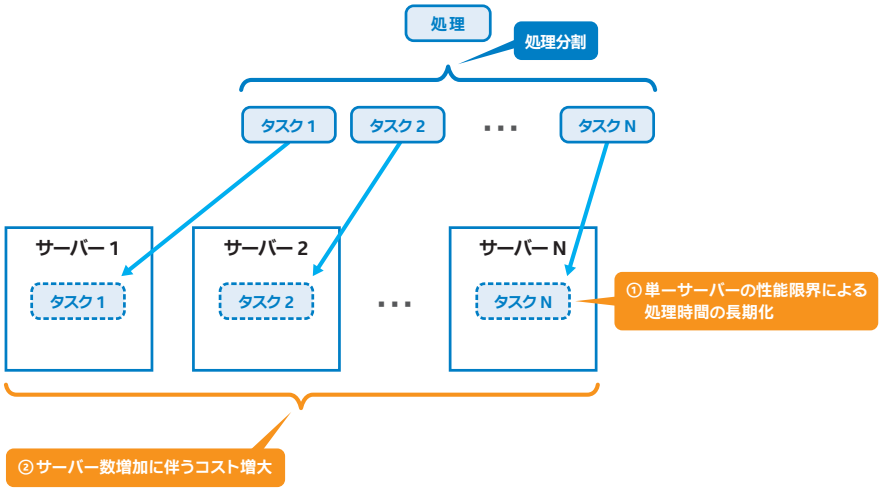


図1. データ分析基盤の課題

ハードウェアが進化を遂げた現在は、サーバー1台当たりの性能に起因するこうした課題に対処するために、処理に応じて適切なハードウェアを組み合わせるヘテロジニアス・コンピューティング環境が実現されつつあります。演算用ハードウェアには、汎用的なCPUやGPUを使用する構成が一般的です。処理特性に応じて論理回路レベルでハードウェアを最適化できるFPGAやASICの利用は、CPU、GPUに比べて技術的なハードルが高く、それほど普及していません。しかし近年では、開発ツールが進化しており、この技術的なハードルが以前に比べ低くなってきている状況にあります。そこで、株式会社エヌ・ティ・ティ・データ（NTTデータ）では、フィールドエンジニア（システム開発において、アプリケーションを実装するエンジニア）がハードウェア・レベルで処理を最適化でき、CPUと比べて性能やコストの点で優位性をもつFPGAに着目しました。

データ活用における処理速度の向上を要望する声は高まり続けていますが、実際には、データの加工（フィルタリング、アグリゲーション、ソート、フォーマット変換など）に全体の約 80% もの時間を要している場合が少なくありません。そこで、データ加工の性能向上をターゲットとした PoC を行いました。その結果、Linux* の監査ログのデータ加工に FPGA を適用した場合、従来の CPU のみの処理に比べて 4 倍以上の高速化が可能になることが実証されました。

FPGA 検討の背景

多品種少量開発に対応できるハードウェア

業種や業界が共通の場合は、システムに要求される処理も類似する傾向がありますが、完全に同一ということはなく、システムごとに多少の差異のある部分が存在します。そのため、業種や業界に特化したハードウェアをゼロベースで作成した場合、高い費用対効果を望むことはできません。一方で、この差異のある特化した処理の部分こそが、競合他社に対する企業固有の差別化要因であったり、歴史的経緯を含む変えがたい部分であったり、企業の競争力を生み出す根源となっている場合もあります。したがって、このような多品種少量開発に対応する FPGA を活用することは、システム開発を支援し、企業活動の改善にもつながると考えられます。

なぜ今 FPGA に注目するのか

FPGA は、組込み分野で豊富な実績のある広く普及したテクノロジーである一方、システム開発においてはそれほど浸透していませんでしたが、テクノロジーの進歩により課題が解決しつつあります。それには、次の 2 つの要素が挙げられます。

- **インテル® FPGA プログラマブル・アクセラレーション・カード（インテル® FPGA PAC）[2] などの登場：** サーバー上で FPGA を利用する場合、FPGA をどのように設置するかをハードウェア設計の中で検証する必要があり、それが FPGA の利用を難しくする一因となっていました。しかし、インテル® FPGA PAC などの登場により、企業でも扱いやすくコモディティー化した PCI Express* に接続可能な FPGA を利用できる環境が整いつつあります。
- **High Level Synthesis（HLS）開発ツールの進化：** 従来、FPGA 上で動作するアプリケーションの開発には Verilog などの Hardware Description Language（HDL）による記述が主流でしたが、HDL はソフトウェア・エンジニアにとって身近なものではなく、アプリケーション開発のハードルが高まる要因となっていました。しかし、OpenCL*[3] をはじめ、C 言語に近い記述方式を用いて FPGA 上で動作するアプリケーションを開発できる HLS テクノロジーが進化するにつれて、ソフトウェア・エンジニアでも FPGA アプリケーションの開発に取り組みやすいツールが整備されてきています。

こうしたテクノロジーの進歩によってこれまでの課題も克服されつつあることから、FPGA を活用したシステム開発への注目はさらに高まっています。

データ活用領域への FPGA の導入で期待される効果

試行錯誤サイクルの高速化によるビジネス貢献

データ活用の処理には、主に 2 つの方法がありますが、高い性能目標値が課されると、処理それぞれに固有の難しさが生じます。

- **大規模バッチ：** 蓄積された大容量データを高速に処理して目的のフォーマットへとデータ加工する処理や、未知の法則の発見を目的とした数理モデルの作成処理などが挙げられます。大規模バッチによる処理では、数分から数時間と、処理時間が長くなる傾向があります。
- **ストリーム：** 逐次発生するデータに対するリアルタイムでのデータ加工や、あらかじめ定められた計算ロジックを当てはめて何らかの判定を行う処理などが挙げられます。ストリームによる処理では、数ミリ秒から数秒と短時間での処理が求められます。

ストリームについては、高頻度で自動売買を行う高頻度取引（HFT）やネットワーク・パケット監視など、FPGA を用いた実例もすでに多数存在しますが、大規模バッチについては、FPGA の効果を検証した事例は今のところ多くありません。

これらの課題を解決する技術的なアプローチの 1 つとして、FPGA を用いて処理特性に応じた適切なハードウェアを作成することで、高速処理の実現が可能になります。最終的にはデータ活用における試行錯誤のサイクルを短縮できるようになります。

性能向上によるコスト削減

データ活用の課題として、システムコスト削減を要望する声は常に聞こえてきますが、FPGA を利用して処理性能を引き上げることで、次のようなコストメリットを期待できます。ただし、FPGA の導入に際しては増加するコストも減少するコストもあるため、それぞれを評価しながら適用を検討する必要があります。

- **ハードウェア台数の削減：** 従来、大量サーバーの並列分散処理で高速処理を実現していた場合、個々のサーバーの処理能力を向上させることで、サーバー台数を減らしてコストを削減することができます。サーバー集約はハードウェアの取得および構築コストのみならず、運用保守、スペース、電力といったランニングコストにも寄与します。

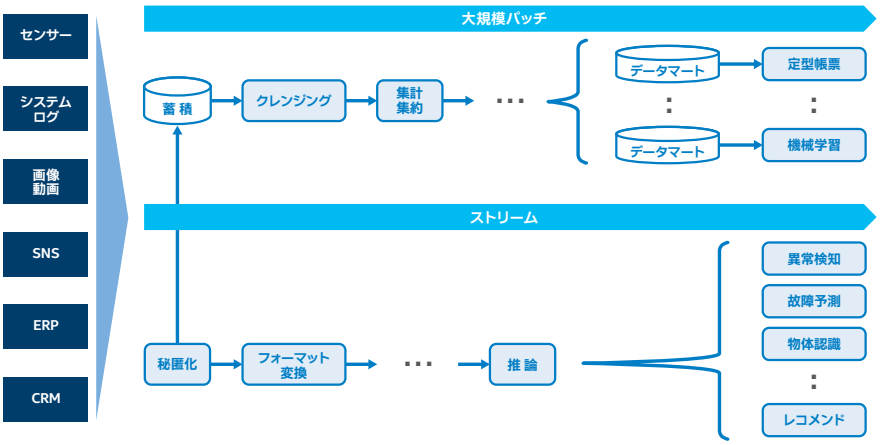


図 2. データ活用の処理方法

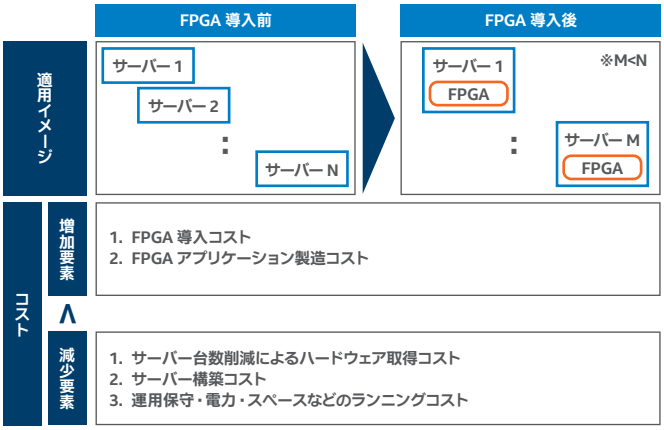


図 3. ハードウェア台数の削減によるコストメリット

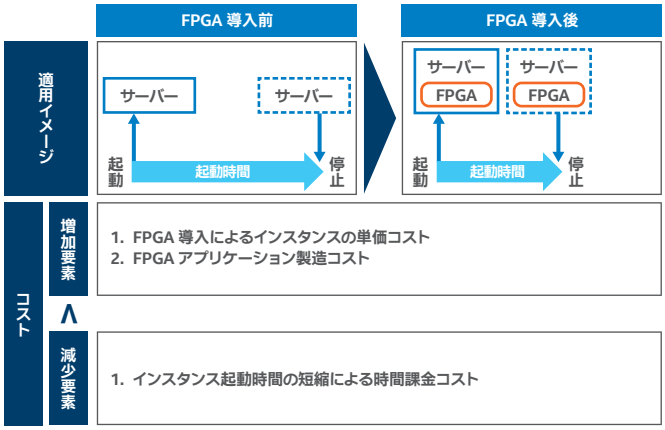


図 4. インスタンス使用時間の短縮によるコストメリット

- **クラウド・インスタンスの使用時間の短縮：**時間単位で課金されるクラウド環境では、処理時間の短縮がそのまま使用時間の短縮につながり、コスト削減効果を期待できます。

仕様変更や機能追加に対する柔軟性の確保

システム開発の現場では、仕様変更や機能追加への柔軟な対応も必要です。ハードウェア調達後の仕様変更は、性能やコストの点で大きなインパクトをもたらします。FPGA であれば、ハードウェア納品後でも論理回路構成を変更できるため、ルールや制度の改変などに伴う避けがたい仕様変更にも柔軟に対応することが可能です。また、OpenCL* を用いることで、修正の工数自体もソフトウェア修正と大きく変わらない程度に抑えられる場合もあります。

FPGA におけるアプリケーション開発

FPGA アクセラレーションとは

FPGA アクセラレーションとは、CPU で実装していた一部の処理を FPGA にオフロードして処理を高速化することです。FPGA アクセラレーションでは、FPGA と CPU それぞれの特性を理解したうえで、適切に使い分ける必要があります。FPGA 上で構成できる論理回路のエリアには限りがあるので、システム上必要な機能のうち、どの機能を FPGA で実装するかの選定が重要です。図 5 に示すように、処理の中でボトルネックになっている部分を FPGA で実装して高速化させる使用方法が、FPGA アクセラレーションの主流となります。

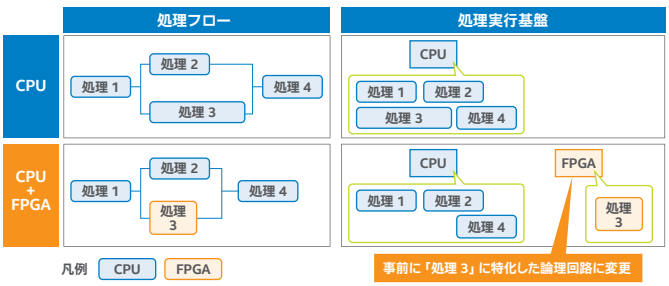


図 5. FPGA アクセラレーション処理のイメージ

PoC 対象選定について

データ活用において、フィルタリング、アグリゲーション、ソート、フォーマット変換などはデータ加工の代表的な処理です。今回実施する PoC では、Linux* の監査ログ出力機能である auditd の出力する監査ログファイルを加工して、ログイン / ログオフ時間を把握するためのデータを出力する処理をサンプルとして取り上げることにしました。これには、監査ログを入力として、フィルタリング、アグリゲーション、ソートといったデータ加工で必要になる基本的な処理が含まれており、ベースとなる数値データを取得する事例に適していると判断しました。

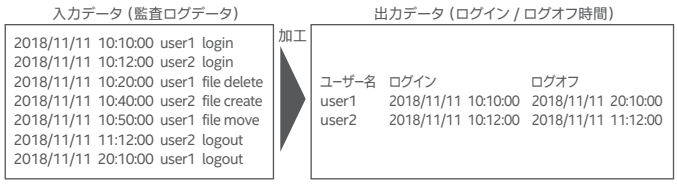


図 6. PoC 処理の概要

開発手法

FPGA アプリケーションの開発には、主に 2 つの手法があります。

- **Hardware Description Language（HDL）：**VHDL や Verilog HDL という IEEE 規格のハードウェア記述専用の言語で論理回路を記述します。設計者の意図した論理回路を確実に実現できる一方で、ハードウェアの知識が必要になるため、フィールドエンジニアにはハードルの高い開発手法です。
- **High Level Synthesis（HLS）：**ソフトウェア・プログラミングで使用される高位言語（C、C++ など）による記述から HDL を生成（高位合成）します。高位合成コンパイラーが回路を生成するため、設計者の意図しない HDL となる可能性もありますが、HDL や論理回路を直接扱わずに済むため、FPGA 開発におけるハードルが下がります。

  システム開発の現場では、高位言語に精通しているエンジニアの数と比較して、ハードウェア記述言語を習得しているエンジニアの数は多くありません。そのため、今回の PoC では開発体制の容易さを考慮して、FPGA アクセラレーターはインテル® プログラマブル・アクセラレーション・カード（インテル® Arria® 10 GX FPGA 搭載版）を採用し、開発手法には HLS を採用しました。

インテル® FPGA PAC の処理を高位合成で開発する場合に選択が可能なコンパイラーには次の 2 種類があります。

- **インテル® FPGA SDK for OpenCL™[4]:** Khronos Group が標準化している OpenCL* を実行するための SDK。C 言語ベースの並列コンピューティングのための言語です。
- **インテル® HLS コンパイラー[5]:** C/C++ による入力をもとに、インテル® FPGA に最適化された製品レベル品質の論理回路コードを生成する、高位合成(HLS)ツールです。入出力部分のハードウェア設計が必要になるため、OpenCL* に比べて開発のハードルは高くなります。

今回の PoC では、フィールドエンジニアが開発することを想定して、インテル® FPGA SDK for OpenCL™ を用いて開発を行いました。

処理フロー

今回の PoC の処理フローは次のように、処理の大部分を FPGA で実装しています。

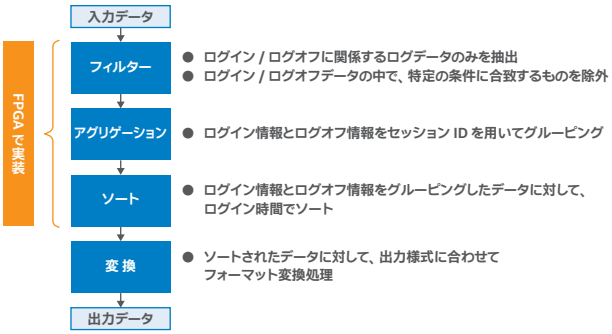


図 7. PoC 処理フロー図

インテル® FPGA SDK for OpenCL™ では、次の 2 種類のプログラムを作成する必要があります。

- **ホストプログラム:** CPU で動作させる処理を記述します。また、カーネルプログラムを実行するための処理を記述します。
- **カーネルプログラム:** FPGA で動作させる処理を記述します。

今回の PoC で実装した内容を以下に示します。複雑な処理部分を FPGA に実装して、FPGA の効果を検証します。

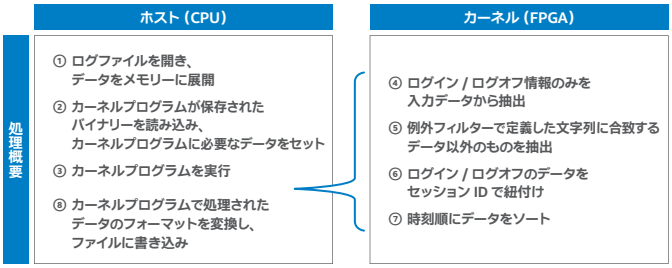


図 8. プログラム概要

また、今回の PoC におけるデータフローを図 9 に示します。ストレージにあるデータを CPU を使ってメモリーに展開し、メモリー上にあるデータに対して FPGA が処理を実行する流れになります。PCI Express* 経由でのデータのやり取りになるため、インテル® プログラマブル・アクセラレーション・カード (インテル® Arria® 10 GX FPGA 搭載版) のアクセラレーション・スタックでは、セキュリティー上の観点により FPGA からストレージなどの PCI Express* デバイスへの直接アクセスが制限されているため、このようなフローをとる必要があります。

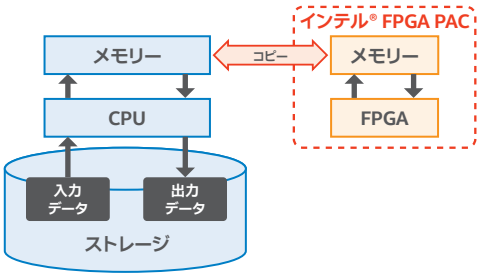


図 9. データフロー図

検証条件

ハードウェア構成およびソフトウェア構成を以下に示します。

表 1. ハードウェア構成

CPU	インテル® Xeon® Gold 6126 プロセッサー 2.60GHz、12 コア /24 スレッド× 2
メモリー	128GB
ストレージ	2TB × 2 (HDD)、480GB × 2 (SSD)
ネットワーク	10Gbps × 2、1Gbps × 2
FPGA	インテル® FPGA プログラマブル・アクセラレーション・カード (インテル® Arria® 10 GX FPGA 搭載版) × 1

表 2. ソフトウェア構成

CentOS*	7.5.1804
インテル® FPGA SDK for OpenCL™	17.1.1
Gcc	4.8.5 (OS 同梱)
g++	4.8.5 (OS 同梱)

PoC で使用する監査ログのデータ量について以下に示します。まずは基礎数値を取るために 1 台のサーバーで処理できるデータ量で検証しました。日々のログデータを参考に、5GB、10GB、20GB の 3 種類のデータ量を想定しました。

検証結果

処理時間

各試験データに対して、CPU 単体での処理時間と CPU+FPGA での処理時間の比較を以下に示します。いずれのデータ量においても CPU+FPGA の処理性能が CPU 単体での処理性能を上回っています。ターンアラウンド時間（処理を実行し完了するまでの時間）で最大 4.17 倍、CPU 時間（CPU で処理をしている時間で、ストレージ I/O などは含まれない）で最大 36.9 倍の性能差を確認することができました。

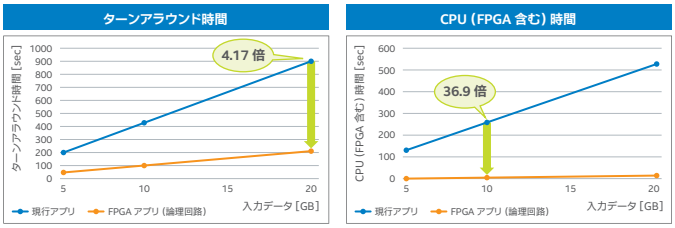


図 10. 検証結果（処理時間）

リソース使用量

現行アプリは CPU を一定時間占有しています。全部で 48 スレッドのため、1 スレッド占有の場合は約 2% の使用率となりますが、FPGA に処理をオフロードした場合、CPU 使用率は低下しています。今回の PoC では、FPGA の使用により、CPU 使用率のピーク値を 81% 低減できることが実証されました。FPGA によって CPU 使用率が低下すれば、最適な CPU モデルの選択が可能になります。これにより、コスト削減効果も期待できます。

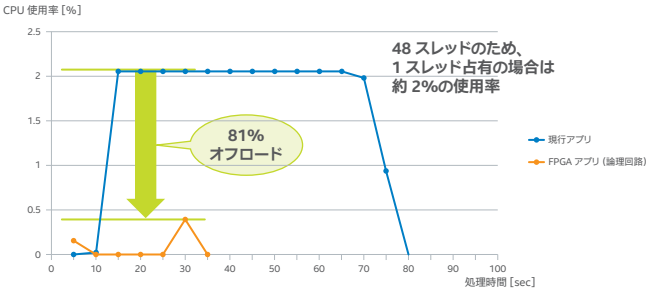


図 11. 検証結果（リソース使用状況）

チューニングのポイント

今回の PoC では FPGA 上でのアプリケーション開発に OpenCL* を採用していますが、FPGA の処理性能を引き出すアプリケーションを実装するにはいくつかのポイントがあります。実際にそれらのポイントを意識しないアプリケーション（チューニングなし）では、FPGA の特性に合わせた実装とそうでない実装とで、処理時間は大幅に異なります。ここでは FPGA の特性を意識すべきポイントを 2 点挙げます。なお、このチューニングは入力データ量 5GB のものを対象に実施しました。

- **メモリー領域：** OpenCL* では、4 種類のメモリー領域を使うことができます。グローバル領域を使ってしまうと処理時間が長くなりがちですが、プライベート領域をうまく活用することで処理時間の短縮が可能になります。C 言語などではメモリー領域に区別がないので、メモリー領域を意識したプログラミングをすることはありませんが、OpenCL* では性能を向上させるために、処理特性に応じて適切な領域を選定することが重要となります。また、データをメモリー領域間でコピーする時間とデータの処理にかかる時間を正確に見極めることも重要です。グローバル領域は低速であるため、多数の処理を実行する場合は一度プライベート領域にコピーするほうが、処理全体としては高速に動作する場合があります。

表 3. OpenCL* でのメモリー

1	Global memory	低速
2	Constant memory	↓
3	Local memory	↓
4	Private memory	高速

- **論理回路生成：** FPGA は CPU に比べてクロック周波数が低いため、CPU 以上の性能を発揮するためには、演算器を大量に配置し、1 クロックで多数のデータを同時に処理できる論理回路を生成する必要があります。論理回路を多数生成する方法はいくつかありますが、代表的な例を以下に示します。OpenCL* 内で、「#pragma unroll」を記述することで、コンパイラーがループをインライン展開します。これにより、複数回路が生成され、1 クロックで同時並列に処理を実行することが可能となり、高速化につながります。一方で回路が多数生成されることで、FPGA 内のエリア使用率が高くなるため、使用にあたっては、コンパイラーが生成するレポートを参照するといった注意も必要です。

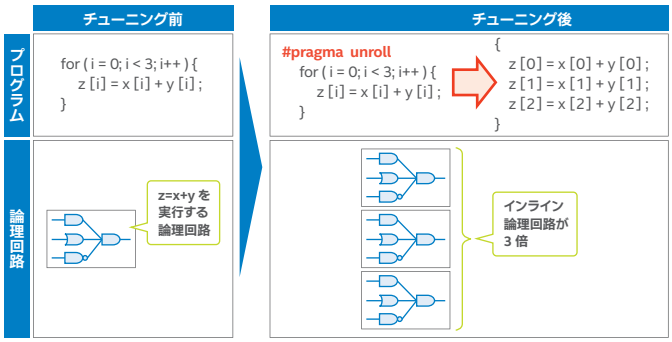
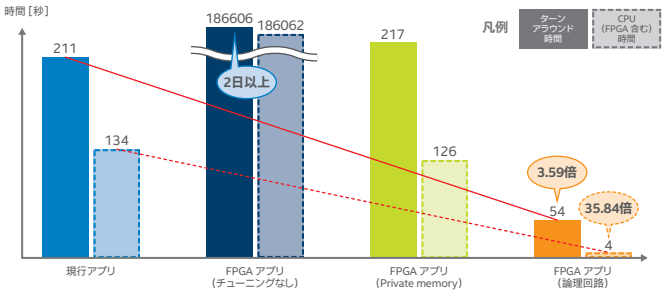


図 12. OpenCL* での論理回路複製例

チューニング実施前と実施後の処理時間の推移を以下に示します。チューニングを実施する前は、現行アプリよりも処理時間が長くなっていましたが、FPGA の特性を意識してチューニングを実施することで、現行アプリと比較して 3 倍以上の処理時間の短縮を確認することができました。また、ターンアラウンド時間にはストレージ I/O などの FPGA での高速化の適用範囲以外の時間も含まれます。それらを除いた CPU 時間と CPU+FPGA 時間の比較では、35 倍以上の性能向上を確認できました。



※入力データ 5GB のケース

図 13. チューニング結果

FPGA を使ったアクセラレーションのポイント

FPGA には得意な処理と不得意な処理があるため、高速化を実現するためには、FPGA に適した処理に対して適切なアルゴリズムを採用することが重要です。以下に典型的な例を挙げます。

- **パイプライン化できないシーケンシャル処理：**データ処理の順序性があらかじめ決まっている場合や、前の処理結果を次の処理が必要とする場合は、並列実行やパイプライン処理が著しく困難になることがあります。こうした場合、高速化のためにはアルゴリズムの全面的な見直しが必要になります。また、現状の FPGA は CPU に比べてクロック周波数が低いため、演算器に各クロックデータが入力されない場合は性能を十分に発揮できない場合があります。
- **集合演算処理：**リレーショナル・データベースが得意とするアグリゲーションやソート処理を FPGA でアクセラレーションする場合、通常ソフトウェア処理で使われるものとは異なるアルゴリズムを適用する必要があります。関連する論文なども多数ありますが、各プロジェクトにおいて当該アルゴリズムを実装するにはかなりの工数を要する可能性があります。

このような作業をフィールドエンジニアが行うことにはノウハウや工数の面で困難な場合があるため、ライブラリーやリファレンス・デザインの拡充が望まれます。

まとめ

今回の PoC では、データ活用でしばしば見られる典型的な処理の一部を FPGA にオフロードすることで、処理性能が 4 倍以上向上することを確認できました。また、リソース使用量の削減により、コストが軽減されることも確認できました。これにより、今後のデータ活用基盤の構築に関しても、サーバー台数の削減や CPU モデルの最適化によって、さらなる低コスト化を実現できる可能性を期待できます。

今回の PoC のユースケースでは、FPGA を効果的に利用するには以下の 2 点が重要なポイントとなります。

- **FPGA にオフロードする処理の選定：**順序性や依存関係を考慮した、FPGA が得意な処理のみを厳選してオフロードできるだけの処理構造に対する分析力が必要とされます。
- **高位合成設計やハードウェア構成を理解したプログラミング：**プログラムを記述する際に、データがハードウェア上をどのように流れ、帯域幅や各種容量をどの程度利用できるか計算しておかなければ、FPGA の並列処理の効果が発揮されない可能性があります。また、FPGA の高位合成設計の特徴を十分に理解していない場合、処理結果は等価でも、演算性能の低い論理回路が生成され、性能要件が満たされない可能性が高くなります。

言い換えるならば、この 2 点で示すように、FPGA の特性を十分理解したうえで利用することで、FPGA に備わる並列処理の効果を最大限に活用することができます。FPGA を活用したシステム開発を行う場合は、このような FPGA の特徴を理解した人材をプロジェクトに参画させることで、データ活用に不可欠なシステム基盤を短期間で開発し、コスト改善効果と高いパフォーマンスの達成をなし得ることができるでしょう。

NTT データはさまざまな業界・業種でのシステム開発で得られたビジネスロジックの豊富な知識と本検証で得られた FPGA に関するノウハウを組み合わせ、ハードウェアの特性を生かした高速な処理を含むデータ活用を実現していきます。

NTT データについて

株式会社エヌ・ティ・ティ・データは、情報技術で、世界中の新しい「しくみ」や「価値」を創造し、より豊かで調和のとれた社会の実現に貢献することを目指しています。お客様との間に長期にわたるゆるぎない関係を築き上げ、お客様の夢を実現することが、私たちの使命です。私たちは、グローバルでのシームレスなサービスと、地域ごとのきめ細やかな対応力をもって、お客様の変革をサポートしています。

お問い合わせ先：
Satoshi.Kondou@nttdata.com

詳細情報

インテル® FPGA アクセラレーション・ハブのウェブページ
https://www.intel.co.jp/fpgaacceleration/

[1] https://hadoop.apache.org/

[2] https://www.intel.co.jp/content/www/jp/ja/programmable/products/boards_and_kits/dev-kits/altera/acceleration-card-arria-10-gx/overview.html

[3] https://www.khronos.org/opencl/

[4] https://www.intel.co.jp/content/www/jp/ja/software/programmable/sdk-for-opencl/overview.html

[5] https://www.intel.co.jp/content/www/jp/ja/software/programmable/quartus-prime/hls-compiler.html

343905-001JA