ホワイトペーパー

FPGA
インテル® oneAPI

# oneAPI ワークロードに対する CPU、GPU、FPGA のメリットの比較

## 目 次



## はじめに

oneAPI は、中央演算処理装置 (CPU)、グラフィックス・プロセシング・ユニット (GPU) 、フィールド・プログラマブル・ゲート・アレイ (FPGA)、その他のアクセラレーターにわたって動作するデータ・セントリックなワークロードの開発と配備を簡素化するために設計されたオープンな統合プログラミング・モデルです。ヘテロジニアスなコンピューティング環境において各コンピューティング・デバイスに適切なワークロードを効果的にマッチングするには、開発者が各コンピューティング・アーキテクチャーでできることとその限界を理解している必要があります。

このドキュメントの内容は次のとおりです。

- CPU、GPU、FPGA の各アーキテクチャーの特徴の比較
- Data Parallel C++ (DPC++) 言語の構文がどのように各アーキテクチャーにマッピングされるか
- ライブラリー・サポートの違い
- 各アーキテクチャーに最適なアプリケーションの特性

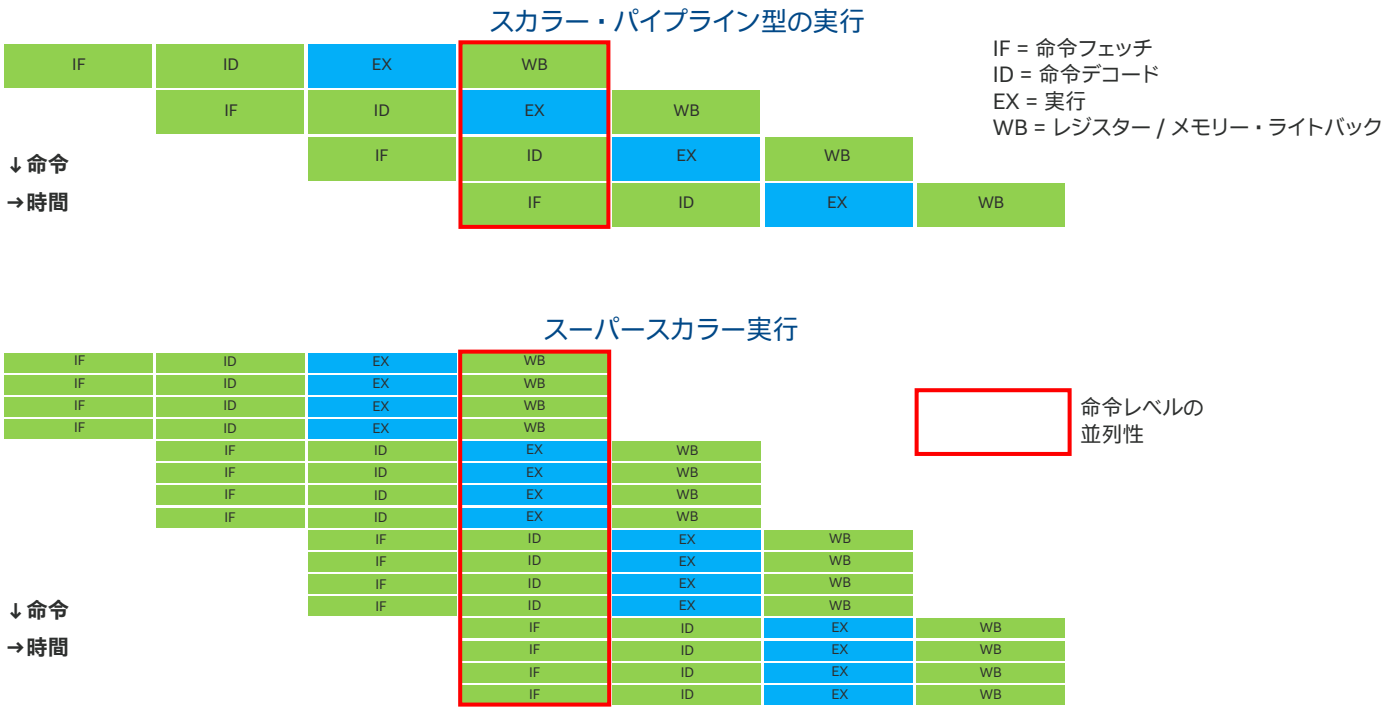
## CPU、GPU、FPGA のコンピューティング・アーキテクチャーの比較

### CPU アーキテクチャー

今回取り上げるコンピューティング・アーキテクチャーの中でも、CPU には半世紀を超える歴史があり、最も有名で、目にすることが多いアーキテクチャーです。CPU アーキテクチャーは、命令を効率的に逐次実行するように設計されているため、スカラー型アーキテクチャーと呼ばれることもあります。CPU は、多数の手法を利用して、命令レベルの並列性 (ILP) を高め、逐次プログラムをできる限り高速に実行できるよう最適化されています。

スカラー・パイプライン型の CPU コアは、依存関係がない場合、複数のステージに分割された命令を最高 1 IPC (クロックサイクル当たりの命令実行数) で実行できます。最近の CPU コアでは、パフォーマンスを向上するため、高度なメカニズムを使用して命令レベルの並列性を見つけ、1 クロックサイクルで複数のアウトオブオーダー命令を実行できるマルチスレッドのスーパースカラー型プロセッサーになっています。多数の命令を同時にフェッチし、これらの命令間の依存グラフを特定し、高度な分岐予測メカニズムを利用して命令を並列実行します (通常、その性能は IPC で比較するとスカラー型プロセッサーの 10 倍に及びます)。

以下の図は、スカラー・パイプライン型 CPU とスーパースカラー型 CPU の実行動作を簡略化して示したものです。



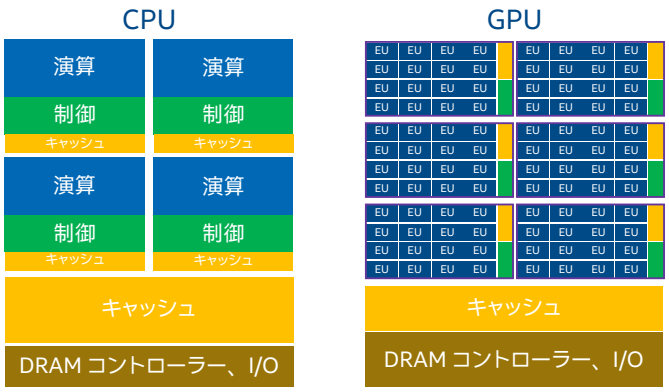

CPU コンピューティングは GPU や FPGA などのコプロセッサーにオフロードするのと比較して多くのメリットがあります。まず、データをオフロードする必要がないため、データ転送オーバーヘッドが最小限で済み、レイテンシーが短くなります。周波数の高い CPU はスカラー実行を最適化するためにチューニングされており、ほとんどのソフトウェア・アルゴリズムは逐次的な性質を持つことから、最近の CPU では容易に高いパフォーマンスを発揮できます。一部のベクトル並列化が可能なアルゴリズムに対しては、最近の CPU はシングル・インストラクション・マルチ・データ (SIMD) 命令をサポートしています。たとえば、インテル® アドバンスト・ベクトル・エクステンション 512 などがその例です。その結果、CPU は幅広いワークロードに適しています。超並列ワークロードの場合でも、分岐が分散するアルゴリズムや命令レベルの並列性が高いアルゴリズムでは (特にデータ量に対して演算量が多い場合)、CPU の方がアクセラレーターよりも優れた性能を発揮します。

CPU のメリット :

- アウトオブオーダーのスーパースカラー実行
- 極めて大規模な命令レベルの並列性を抽出できる高度な制御
- 高精度な分岐予測
- 逐次的コードの自動並列化
- サポートされる命令の多さ
- オフロード・アクセラレーションと比較して短いレイテンシー
- 逐次的なコード実行により開発が容易

**GPU アーキテクチャー**

GPU は、高性能 CPU と比べて超並列、小型、かつ特定用途向けコアで構成されるプロセッサーです。GPU アーキテクチャーは、個々のスレッドのレイテンシーとパフォーマンスよりも、すべてのコアの合計スループットを確保することに最適化されています。GPU アーキテクチャーはベクトルデータ (複数の数値から成る配列) を効率的に処理するため、ベクトル・アーキテクチャーと呼ばれることがよくあります。 GPU は、コンピューティングにより多くのシリコン面積を割り当て、一方、キャッシュや制御系は削減されています。その結果、GPU ハードウェアは命令レベルの並列性を引き出すことにはそれほど重点を置かず、ソフトウェア側による並列化によってパフォーマンスと効率性を実現します。GPU はインオーダー・プロセッサーで、高度な分岐予測はサポートしていません。その代わりに、たくさんの算術論理ユニット (ALU) と深いパイプラインを備えています。独立した大規模なデータをマルチスレッドで実行することで、簡素な制御系と小さなキャッシュサイズの対価を補ってあまりあるパフォーマンスが実現されます。GPU はマルチスレッディングと SIMD の両方を活用するシングル・インストラクション・マルチ・スレッド (SIMT) 実行モデルを採用しています。SIMT モデルでは、複数のスレッド (ワークアイテム、または SIMD レーン演算のシーケンス) が同じ SIMD 命令ストリーム内でロックステップで実行されます。複数の SIMD 命令ストリームが 1 つの実行ユニット (EU) にマッピングされ、1 つのストリームがストールした場合、GPU の EU はこれらの SIMD 命令ストリーム間でコンテキスト・スイッチできます。




上図は、CPU と GPU の違いを示しています。 EU は、GPU 上の基本処理単位です。 各 EU は複数の SIMD 命令ストリームを処理できます。 シリコン面積が同じ場合、GPU の方が CPU より多くのコア / EU を持ちます。 GPU は階層構造を持ちます。 複数の EU が合わさって、シェアードローカルメモリーと同期メカニズムをもつ演算ユニットを形成します (紫色で囲んだ部分。サブスライスまたはストリーミング・マルチプロセッサー)。 これらの演算ユニットが合わさって GPU を構成します。

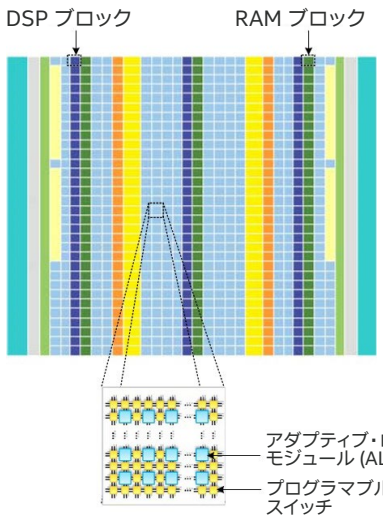
GPU のメリット :

- 超並列の最大数千個の小型で効率的な SIMD コア / EU
- データ並列コードの効率的な実行
- 高いダイナミック・ランダム・アクセス・メモリー (DRAM) 帯域幅

### FPGA アーキテクチャー

ソフトウェアプログラマブルな固定アーキテクチャーである CPU や GPU と違い、FPGA はリコンフィギャラブルで、その演算エンジンはユーザーによって定義されます。FPGA をターゲットとするソフトウェアを記述すると、コンパイルされた各命令は空間内の FPGA ファブリックに配置されるハードウェア・コンポーネントとなり、各コンポーネントをすべて並列に実行できます。そのため、FPGA アーキテクチャーは空間的アーキテクチャーと呼ばれることもあります。

FPGA は、最大数百万個に及ぶプログラマブルな 1 ビットのアダプティブ・ロジック・モジュール (それぞれが 1 ビット ALU のように動作可能)、最大何万個にも及ぶコンフィギャラブルなメモリーブロック、可変精度浮動小数点および固定小数点演算をサポートする何万個もの算術演算エンジン (デジタル信号処理 (DSP) ブロック) で構成される小規模なプロセシング・ユニットの大規模アレイです。これらすべてのリソースは、必要に応じてアクティブ化可能なメッシュ状のプログラマブルな配線で接続されます。




FPGA 上でソフトウェアが「実行」されるとき、コンパイルおよびアセンブルされた命令が CPU や GPU 上で実行されるのとは違った形で実行されます。データは、FPGA 上の、ソフトウェアに記述された演算に合わせてカスタマイズされた深いパイプラインを通過します。データフロー・パイプライン・ハードウェアがソフトウェアに合わせこまれているため、制御オーバーヘッドがなくなり、パフォーマンスと効率性が向上します。CPU と GPU の場合、命令のステージがパイプライン化され、クロックサイクルごとに新しい命令の実行が開始されます。FPGA の場合は、演算がパイプライン化され、クロックサイクルごとに別々のデータに対して動作する新しい命令ストリームの実行が開始されます。



FPGA における並列処理の基本的形態はパイプライン並列化です。パイプライン並列化は、他の並列化手法と組み合わせることができます。たとえば、データ並列化 (SIMD)、タスク並列化 (複数のパイプライン)、スーパースカラー実行 (複数の独立した命令を並列実行) をパイプライン並列化と併用して最高のパフォーマンスを実現できます。

FPGA のメリット :

- 効率性 : データ・プロセシング・パイプラインがソフトウェアのニーズに合わせてチューニングされます。制御ユニット、命令フェッチユニット、レジスター・ライトバック、その他の実行オーバーヘッドは必要ありません。
- カスタムの命令 : CPU/GPU がネイティブでサポートしていない命令も、FPGA に簡単に実装でき、効率的に実行できます (ビット操作など)。

- 並行した作業にまたがるデータ依存性を、パイプラインをストールさせることなく解消できます。
- 柔軟性 : FPGA は、様々な関数およびデータ型 (非標準のデータ型を含む) に対応するよう再構成可能です。
- アルゴリズムに合わせてチューニングされたカスタムのオンチップ・メモリー・トポロジー : 広帯域のオンチップ・メモリーを内蔵しており、ストールを最小限に抑える (または解消する) アクセスパターンに対応できます。
- 豊富な I/O: FPGA コアは、さまざまなネットワーク、メモリー、カスタム・インターフェイス、プロトコルと直接通信できるため、レイテンシーが短く、確定的になります。

## oneAPI および DPC++ の CPU、GPU、FPGA に対する マッピングについて

各アーキテクチャーの基本を説明したところで、このセクションでは、oneAPI および DPC++ の実行が CPU、GPU、FPGA の実行ユニットとどのようにマッピングされるかを見ていきます。

NDRange カーネル

```
auto total = range{N};
auto wg_size = range{256};

queue{}.submit([&](handler& h) {
   accessor out{input_buf, h};
   accessor in{output_buf, h};

   h.parallel_for(nd_range{total, wg_size},
                              [=](auto i) {
      // kernel
      out[i] = process(in[i]);
   });
});
```

シングル・タスク・カーネル

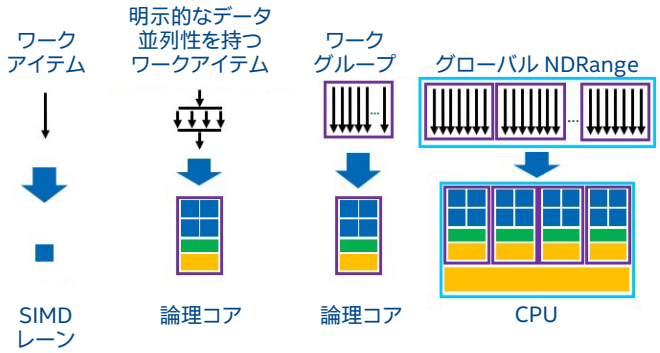
```
queue{}.submit([&](handler& h) {
   accessor out{input_buf, h};
   accessor in{output_buf, h};
   h.single_task( [=] {
      // kernel
      for (int i=0; i<N; i++) {
         out = process(in[i], out);
      }
   });
});
```

DPC++ の詳細については、本稿末尾に記載する DPC++ の参考資料を参照してください。上に示すのは NDRange カーネルとシングル・タスク・カーネルの 2 種類の DPC++ カーネルです。このセクションでは、これら 2 つのカーネルとその並列処理が各種のアーキテクチャー上でどのように実行されるかについて考察します。左側に示す NDRange カーネルのラムダ式は、parallel_for によって N 個のワークアイテム (スレッド) にわたるデータ並列処理として開始され、それぞれ 256 個のワークアイテムから成る N/256 個のワークグループに分割されます。NDRange カーネルは、本質的にデータ並列です。右側では、ラムダ式を実行するのに 1 つのスレッドのみ開始されます。ただし、カーネル内にループが存在しデータ要素が反復されます。single_task により、ループの各イタレーションから並列性が得られます。

### CPU の実行

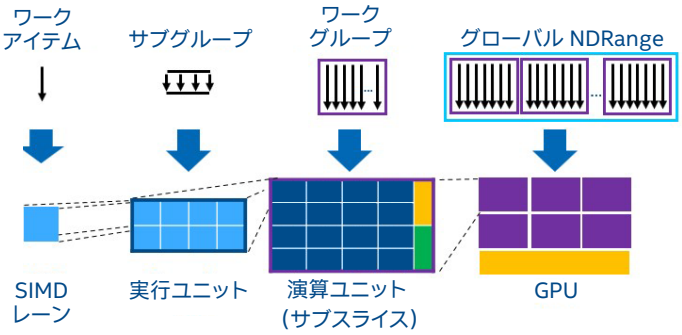
CPU は複数の種類の並列化に基づいてパフォーマンスを達成します。SIMD のデータ並列性、命令レベルの並列性、スレッドレベルの並列性 (複数のスレッドを別々の論理コアで実行) をすべて活用できます。これらの種類の並列性は DPC++ で次の方法で実現できます。

1. SIMD データ並列性 :
   - (同じワークグループに属する) 各ワークアイテムは、1 つの CPU SIMD レーンにマッピングできます。ワークアイテム (サブグループ) は、SIMD 形式で同時に実行されます。
   - ベクトルデータ型を使用して SIMD 演算を明示的に指定できます。
   - コンパイラーはループベクトル化により SIMD コードを生成できます。ループの 1 回のイタレーションが 1 つの CPU SIMD レーンにマッピングされます。ループの複数回のイタレーションは SIMD 形式で同時に実行されます。
2. CPU コアおよびハイパースレッド並列性 :
   - 異なるワークグループを異なる論理コア上で並列に実行できます。
     - コア数が 16、ハイパースレッド数が 32 のマシンでは、32 個のワークグループを並列実行できます。
3. 従来の命令レベルの並列性 :
   - 従来の逐次的ソフトウェア実行と同様に、高度な CPU 制御によって実現されます。



### GPU の実行

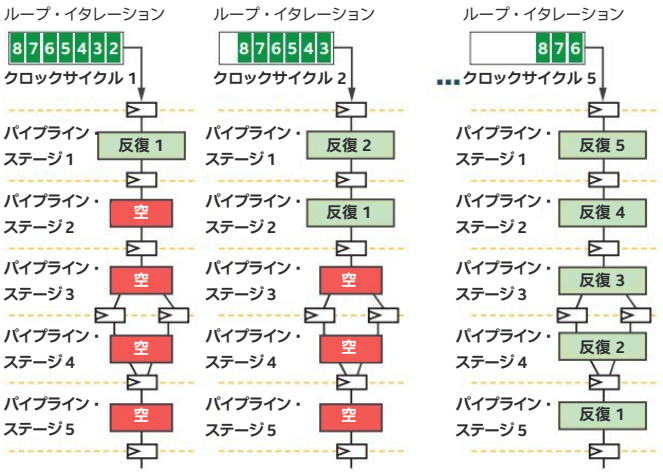
GPU は大規模なデータ並列ワークロードに基づいてパフォーマンスを達成します。その結果、シングル・タスク・カーネルが利用されることはほとんどありません。GPU の深い実行パイプラインを完全に利用するには、NDRange カーネルが必要です。カーネルを GPU 上で実行するとき、個々のワークアイテムが 1 つの SIMD レーンにマッピングされます。SIMD 形式で実行されるワークアイテムによりサブグループが形成され、サブグループは GPU EU にマッピングされます。ワークグループ (同期およびローカルデータを共有可能なワークアイテムを含む) は、実行のために演算ユニット (ストリーミング・マルチプロセッサーまたはサブスライスとも呼ぶ) に割り当てられます。そして、ワークアイテムの全グローバル NDRange が GPU 全体にマッピングされます。



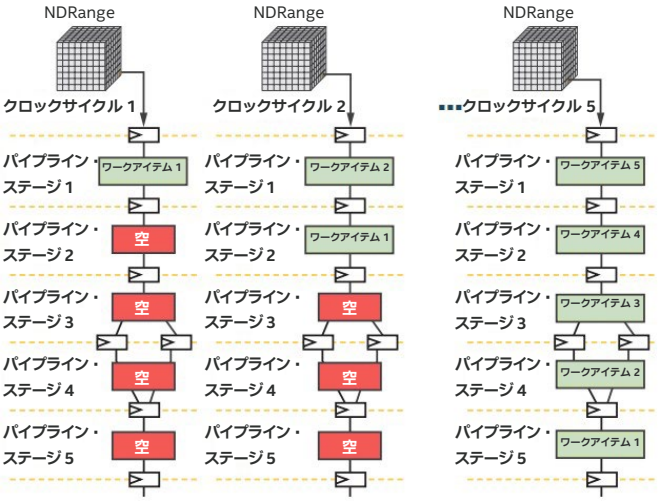

### FPGA の実行

カーネルを FPGA 用にコンパイルすると、カーネル内の演算は空間的に配置されます。依存関係にあるカーネル演算は深いパイプラインとして構成され、独立した演算は並列実行されます。パイプライン化された実装の重要なメリットは、データをあるパイプライン・ステージから前の方のステージにルーティングするだけで、パイプラインをストールさせることなく、隣接するワークアイテム間の依存性を解消できることです。パフォーマンスの鍵を握るのは、深いパイプラインを常に満たすことです。FPGA では NDRange カーネルでもシングル・タスク・カーネルでも高いパフォーマンスを引き出すことが可能ですが、FPGA で良好に動作するのはシングル・タスク・カーネルであると一般に言われています。

シングル・タスク・カーネルでは、FPGA はループの実行をパイプライン化しようとします。各クロックサイクルで、連続するループ・イタレーションがパイプラインの 1 つ目のステージに入ります。ソフトウェアに記述されたループ・イタレーション間の依存性は、あるステージの出力を次のステージの入力にルーティングすることによってハードウェア的に解消できます。




NDRange カーネル実行の場合、各クロックサイクルで、異なるワークアイテムがカスタム演算パイプラインの 1 つ目のステージに入ります。




その他のアーキテクチャーと異なり、カスタム FPGA パイプライン演算ユニットでは、FPGA リソースとスループットのトレードオフにより、パイプラインの SIMD 幅を広くすることでより多くのワークアイテムを並列実行するという選択が可能です。たとえば、ループ・イタレーションを展開 (アンロール) して、データ並列化を実現できます。

## ライブラリー・サポート

CPU、GPU、FPGA で高いパフォーマンスを発揮するソフトウェアを最速で作成する方法の 1 つは、oneAPI ライブラリーが提供する機能を利用することです。oneAPI は、演算負荷およびデータ負荷の高いさまざまな分野 (ディープラーニング、科学計算、動画分析、メディア・プロセシングなど多数) 向けのライブラリーを提供しています。ライブラリー・サポートは、アルゴリズムに最適なデバイスを特定するのに役立ちます。一般的に、最も広範なライブラリー・サポートを持つのが CPU で、その次に来るのが GPU です。一方、手作業での実装を最も多く必要とするのが FPGA です。ライブラリー関数が複数のアーキテクチャーでサポートされている場合、処理対象のデータ量やデータの場所 (処理対象のデータが格納されているデバイス) などの各実行の特徴によって、使用するのに最適なデバイスが決定されます。このセクションでは、各種 oneAPI ライブラリーとそのデバイスサポートについて見ていきます。ライブラリー・サポートは常に進化しています。最新情報は、最新のライブラリー・ドキュメント (英語) を参照してください。

### インテル ® oneAPI DPC++ ライブラリー (oneDPL) (CPU、GPU、FPGA)

oneDPL は、複数のデバイスをターゲットとする並列アプリケーションを開発する DPC++ 開発者の生産性を向上できます。oneDPL は並列 C++ Standard Template Library (STL) の機能に加えて、追加の並列アルゴリズム、イテレーター、範囲ベースの API、乱数ジェネレーター、検証済み標準 C++ API、およびその他の機能へのアクセスを提供します。oneDPL は、CPU、GPU、FPGA をサポートしています。

### インテル ® oneAPI マス・カーネル・ライブラリー (oneMKL) (CPU、GPU)

oneMKL は、大規模な演算問題を解く算術演算ルーチンによってパフォーマンスを向上します。oneMKL は BLAS および LAPACK 線形代数ルーチン、高速フーリエ変換、ベクトル化された数学関数、乱数生成関数、その他の機能を提供します。現在、oneMKL は CPU と GPU をサポートしていますが、将来的にその他のアクセラレーターのサポートが追加される可能性があります。

### インテル ® oneAPI スレッディング・ビルディング・ブロック (oneTBB) (CPU)

oneTBB は、DPC++ では提供されない CPU 向けの特定の論理並列演算アルゴリズムを提供する C++ テンプレート・ライブラリーです。このライブラリーを DPC++ と組み合わせて使用することで、コードの実行を CPU と GPU に分割することができます。

### インテル ® oneAPI データ・アナリティクス・ライブラリー (oneDAL) (CPU、部分的に GPU をサポート)

oneDAL は、バッチ処理、オンライン処理、分散処理の演算において、高度に最適化されたアルゴリズム・ビルディング・ブロック (前処理、変換、分析、モデリング、検証、意思決定) をデータ分析のあらゆる段階 に提供する、ビッグデータの解析を高速化するのに役立つライブラリーです。CPU 向けに設計されましたが、oneDAL は一部のアルゴリズムで GPU の利用を可能にする DPC++ API 拡張を提供しています。

### インテル ® oneAPI ディープ・ニューラル・ネットワーク・ライブラリー (oneDNN) (CPU、GPU)

oneDNN にはディープラーニング・アプリケーションおよびフレームワークのためのビルディング・ブロックが含まれます。ブロックには、たたみ込み、プーリング、LSTM、LRN、ReLU などが含まれています。このライブラリーは CPU と GPU の両方をサポートしています。

### インテル ® oneAPI コレクティブ・コミュニケーション・ライブラリー (oneCCL) (CPU、GPU)

oneCCL は、ディープラーニング・アプリケーションで発生する通信パターン用に最適化されたプリミティブをサポートしています。開発者や研究者がより新しく、より深いモデルをより高速にトレーニングするのに役立ちます。

### インテル ® oneAPI ビデオ・プロセシング・ライブラリー (oneVPL) (CPU)

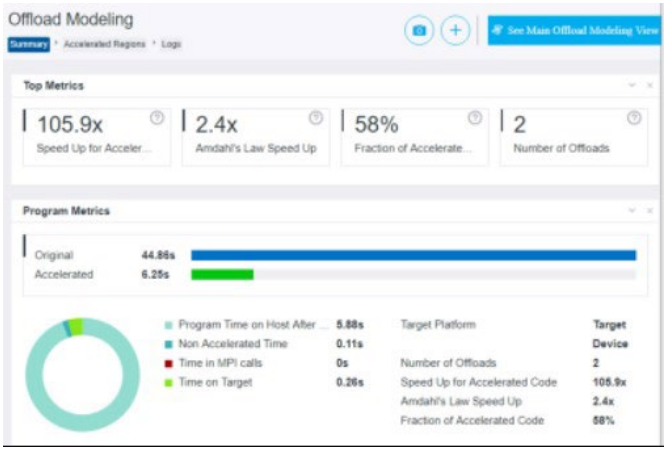
oneVPL は、メディア・パイプラインを構築するビデオのデコーディング、エンコーディング、プロセシングのためのプログラミング・インターフェイスです。現在、このライブラリーは CPU への配備をサポートしています。将来的に、その他のハードウェア・アクセラレーターもサポートする予定です。

**現時点の oneAPI ライブラリー・サポートのまとめ**

| | oneDPL | oneMKL | oneTBB | oneDAL | oneDNN | oneCCL | oneVPL |
|---|---|---|---|---|---|---|---|
| CPU | O | O | O | O | O | O | O |
| GPU | O | O | | 部分的 | O | O | |
| FPGA | O | | | | | | |

## GPU オフロード用のコード領域の特定

ヘテロジニアスなコンピューティング環境では、アルゴリズムからオフロード・アクセラレーションに適した部分を特定するのが困難な場合があります。アルゴリズムの特性を知っていたとしても、パーティショニングおよびアクセラレーター最適化の実行前にアクセラレーター上でのパフォーマンスを予測するのは大変です。インテル ® oneAPI ベース・ツールキットには、そのために役立つインテル ® アドバイザー のオフロード・アドバイザーが含まれています。




インテル ® アドバイザーは、効率的なスレッディング、ベクトル化、メモリー使用、およびオフロードによってアプリケーションのパフォーマンスを向上する設計および分析ツールです。C、C++、DPC++、Fortran、OpenMP*、Python をサポートしています。インテル ® アドバイザー のオフロード・アドバイザー機能は、GPU オフロードにより効果が見込めるコード部分の特定を支援するために設計されています。GPU 実行用にコードをリファクタリングする前に、オフロード・アドバイザーを使うと以下のことが可能です。

1. オフロードできるコードを特定する
2. オフロードによって見込まれるパフォーマンス高速化を定量化する
3. データ転送にかかるコストを見積もり、データ転送最適化に関するガイダンスを提供する
4. ボトルネックを特定し、潜在的なパフォーマンスの見積もりを提供する

CPU は本質的に GPU とは異なるため、CPU コードを移植して理想的な GPU コードにするには、多大な労力が必要になることがあります。オフロード・アドバイザーは、移植作業に着手する前に、GPU オフロードに適したアルゴリズムを特定するのに大きな威力を発揮する生産性向上ツールです。

## アプリケーション例

このセクションでは、各アーキテクチャーのアプリケーション例を見ていきます。どのアクセラレーター・アーキテクチャーにどのアルゴリズムが適しているかの予測は、アプリケーションの特性とシステムのボトルネックによって異なります。

### GPU ワークロードの例

GPU は超データ並列のアクセラレーターです。データセットの規模がデータ転送のオーバーヘッドを上回り、データ間の依存性のほとんどが回避可能で、データフローの大半が非分岐の場合、最適なアクセラレーターは GPU です。

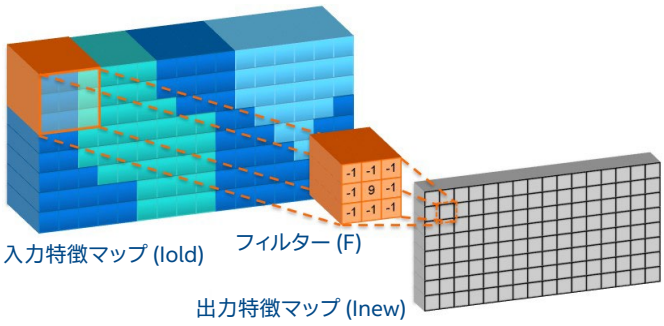
一般に、GPU のメリットをフルに活用できるワークロードには次のような特性があります。

- データ並列性を持つ (同じ演算がデータ全体に適用される)
- 問題の規模が大きく、多数の GPU プロセシング・エレメントが利用される
  - メイン・コンピューター・メモリーから GPU メモリーへのデータ転送時間が無視できる規模
- 処理対象のデータに依存性が少ない (またはない)
  - あるデータポイントの処理結果の出力が別のデータポイントの出力に影響を与えない
- 大半が非分岐の制御フロー (最小限の分岐およびループ分散)
  - 再帰関数は多くの場合書き直しが必要
- GPU でサポートされるデータ型と一致
  - たとえば、文字列の処理には時間がかかる場合がある
- インオーダーのデータアクセス
  - GPU は連続的な読み出しと書き込みに最適化されている
  - ランダムな順序でデータにアクセスするアルゴリズムは多くの場合書き直しが必要

たとえば、GPU に理想的なワークロードに画像処理があります。通常、非常に多くのピクセルを処理します。各ピクセルに同じ基本的演算が適用され、出力されるピクセルは互いに依存しません。

GPU でよく使用されるその他のワークロードは、ディープラーニング・アプリケーションです。コンピューター・ビジョン・アプリケーションにおいて、たたみ込みニューラル ネットワーク (CNN) のたたみ込み層の 1 つの出力特徴マップを計算する式を以下に示します。

$$I_{new}[x][y] = \sum_{x'=-1}^{1} \sum_{y'=-1}^{1} \sum_{z'=0}^{2} I_{old}[x+x'][y+y'][z'] \times F[x'][y'][z']$$



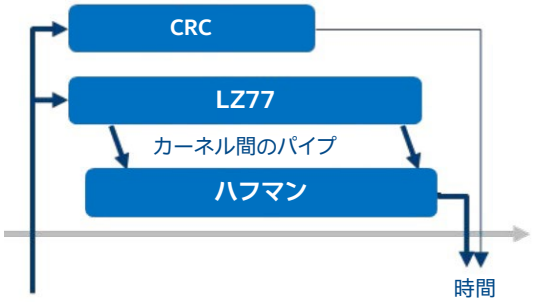
出力特徴マップを計算するには、3D フィルターをマルチチャネル入力特徴マップの幅および高さにわたってたたみ込み、スライディング・ドット積を計算します。1 つの CNN 層内で数百の特徴マップが計算されることがあり、さらに CNN の順方向パスで数百の層を計算する必要がある可能性があります。1 つの層のドット積は独立しており、計算は分岐を伴わず、各層の計算量が多く、転送が必要な入出力データ量よりも演算量の高速化の度合いが大きいため、GPU は CNN を効率的に処理できます。

GPU の典型的なアプリケーションは、他にも AI、データ分析、気候モデリング、遺伝学、物理学など、さまざまな分野で見ることができます。

### FPGA ワークロードの例

FPGA アーキテクチャーの効率的なカスタムの深いパイプラインは、シリアルのコードで容易に表現でき、データエレメント間に依存関係がある可能性のあるアルゴリズムに適しています。このようなタイプのアルゴリズムは、GPU のデータ並列アーキテクチャーではうまく実行できません。

その 1 つの例は Gzip 圧縮です。Gzip は 3 つのカーネルで処理できます。1 つは LZ77 データ圧縮カーネルで、ファイル内の重複するパターンを検索して取り除きます。2 つ目のカーネルはハフマン符号を実行し、LZ77 の出力をエンコードして結果を生成します。最後に、他の 2 つのカーネルとは独立して、CRC32 エラー検出カーネルが入力に対して動作します。




このアルゴリズムが FPGA での実行に適している理由は複数あります。

1. 各カーネルに専用のカスタム演算パイプラインがあるため、FPGA では 3 つのカーネルを同時に実行できます。
2. LZ77 の場合、重複を特定するシーケンシャル検索は、依存性があるために NDRange カーネルでは容易に並列化できません (たとえば、ファイルの残り半分の検索結果が最初の半分に依存します)。これはファイル全体をシンボルごとに反復するシングル・タスク・カーネルで実装できます。その場合、反復間のカスタム・ルーティングにより依存性を解決できます。
3. FPGA のカスタムのオンチップ・メモリー・アーキテクチャーでは、ストールを起こすことなく大量のディクショナリー・ルックアップを同時に実行できます。
4. ハフマン符号はバイト境界にアラインされないため、エンコードにはビット操作が必要ですが、これは FPGA で効率的に実装できます。
5. FPGA では、同時実行されるカーネル間で個々のデータをパイプ接続できます。これにより、LZ77 カーネルとハフマン・エンコーディング・カーネルを同時に実行できます。

この実装の詳細とサンプルコードは、「Accelerating Compression on Intel® FPGAs（英語）」の記事にあります。

同様の理由で、他にも多くワークロードが FPGA に適しています。画像のロスレス圧縮、ゲノム配列決定、データベース分析アクセラレーション、機械学習、金融コンピューティングなどはすべて、FPGA アクセラレーションの有望な候補です。

### CPU とヘテロジニアス・ワークロード

CPU は、コンピューティングで最も広く使用されている汎用的なプロセッサーです。コンピューティング・アクセラレーションに GPU または FPGA を利用するすべてのアプリケーションで、依然としてタスクのオーケストレーションを処理するために CPU は必要です。CPU は、アルゴリズムが GPU や FPGA の能力を効率的に引き出すことができない場合のデフォルトの選択肢です。CPU は、GPU ほど演算密度は高くなく、FPGA ほど演算効率は高くないものの、ベクトル、メモリー、およびスレッド最適化を施すことで、演算アプリケーションにおいて優れたパフォーマンスを発揮できます。オフローディングの要件が満たされていない場合は特にそうです。

ヘテロジニアス環境では、メインの演算はアクセラレーターで行いながら、CPU でコードの他の部分のパフォーマンスを向上することが可能です。oneAPI では、1 つのアプリケーションで複数のアーキテクチャーをまたぐパフォーマンスを引き出し、CPU、GPU、および FPGA のメリットを活用できます。

## まとめ

現在、CPU、GPU、FPGA、およびその他のアクセラレーターを含むコンピューティング・システムのヘテロジニアス化が進んでいます。アーキテクチャーごとに特性はさまざまで、最高のパフォーマンスを発揮するために特定のワークロードに適するアーキテクチャーは異なります。複数の種類のコンピューティング・アーキテクチャーを使用すると、アーキテクチャーごとに異なるプログラミングおよび最適化が必要になります。oneAPI と DPC++ は、各アーキテクチャーに合わせたソフトウェアの開発に有効なプログラミング・モデルを、直接プログラミング手法またはライブラリーを通して提供します。

最も演算密度が高いのは GPU アーキテクチャーです。カーネルがデータ並列で、シンプルで、多くの演算が必要な場合、GPU が最適な可能性が高いでしょう。最も演算効率が高いのは FPGA アーキテクチャーです。FPGA 、そして生成されたカスタムの演算パイプラインはほとんどすべてのカーネルの高速化に使用できますが、FPGA 実装が空間的性質を持つため、利用可能な FPGA リソースにには限りがあります。最後に、最も柔軟性が高く、最も広範なライブラリー・サポートを受けられるのが CPU アーキテクチャーです。最近の CPU は多くの種類の並列性をサポートしており、効果的に使用することで他のアクセラレーターを補完できます。

本稿で述べた各コンピューティング・アーキテクチャーを体感したい方は、インテル® DevCloud（英語）を使用して最新の CPU、GPU、FPGA を含むヘテロジニアスな環境で oneAPI および DPC++ をお試しください。最新のインテル® ハードウェアおよびソフトウェア（英語）のクラスター上で、無償でワークロードを開発、テスト、実行できます。

## 参考資料

- Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL（英語）
- oneAPI Specification and Documentation（英語）(on eapi.com)
- インテル ® oneAPI プログラミング・ガイド（英語）
- インテル ® oneAPI DPC++ FPGA 最適化ガイド（英語）
- Optimize Your GPU Application with Intel® oneAPI Base Toolkit（英語）
- oneAPI GPU 最適化ガイド（英語）
- インテル ® アドバイザー（英語）
- インテル ® DevCloud（英語）
- インテル ® oneAPI ベース・ツールキット（英語）

### 製品およびパフォーマンスに関する情報

[1] 実際の性能は使用方法、構成、その他の要因によって異なります。詳細は www.Intel.com/PerformanceIndex（英語）を参照してください。